

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO CUỐI KÌ

MÔN HỌC SÂU

Người hướng dẫn: PGS.TS LÊ ANH CƯỜNG

Người thực hiện: TRẦN NHỰT QUANG – 52100298

PHẠM DUY KHOA – 52100901

NGUYỄN KHẮC ANH TÀI - 52100306

Lớp : 21050301

Khoá : 25

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO CUỐI KÌ

MÔN HỌC SÂU

Người hướng dẫn: **PGS.TS LÊ ANH CƯỜNG**

Người thực hiện: **TRẦN NHỰT QUANG**

PHẠM DUY KHOA

NGUYỄN KHẮC ANH TÀI

Lớp : **21050301**

Khoá : **25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Là một sinh viên khoa Công nghệ thông tin, em xin chân thành cảm ơn những thầy cô trong Khoa đã mang đến cho em những kiến thức bổ ích, không chỉ về chuyên môn mà còn là những kỹ năng mềm thiết yếu. Cùng những tinh hoa kiến thức đó, em đã cải thiện được bản thân cả trong học tập lẫn đời sống.

Trong bộ môn Học Sâu, em đã từng gặp nhiều vấn đề về lý thuyết trên lớp và kỹ năng thực hành lập trình. Nhưng nhờ có sự giảng dạy tận tình của thầy Lê Anh Cường mà mọi khó khăn trong việc học của em đã được giải đáp. Với báo cáo giữa kỳ này, từ những kiến thức đã được bồi dưỡng trên lớp, em đã thực hiện bằng hết sức mình để có thể đáp lại công ơn giảng dạy của thầy, dù có thể còn gặp nhiều sai sót không đáng. Lời cuối, em xin kính chúc thầy có nhiều sức khỏe để có thể tiếp tục đồng hành cùng chúng em trên con đường học vấn.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH

TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của PGS.TS LÊ ANH CƯỜNG. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 1 tháng 4 năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

Trần Nhựt Quang

Phạm Duy Khoa

Nguyễn Khắc Anh Tài

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Đề tài báo cáo được chia ra thành 2 phần chính là:

- Tìm hiểu về Transformer based encoder và GPT model

Phần đầu yêu cầu tìm hiểu về mô hình mã hóa dựa trên Transformer và mô hình GPT. Cụ thể, bạn cần nghiên cứu cấu trúc, cơ chế hoạt động, các thành phần chính như self-attention, feed-forward layers, positional encoding và cách các mô hình này được áp dụng trong xử lý ngôn ngữ tự nhiên. Sau đó thực hiện pretrain fineturning với một trong hai mô hình.

- Triển khai mô hình Transformer cho bài toán dịch thuật tiếng anh

Triển khai mô hình Transformer để thực hiện dịch thuật từ tiếng Anh sang tiếng Việt. Bạn sẽ cần xây dựng và huấn luyện mô hình Transformer, bao gồm cả encoder và decoder, sử dụng một bộ dữ liệu dịch thuật song ngữ. Sau khi mô hình được huấn luyện, bạn sẽ đánh giá hiệu suất của nó dựa trên các chỉ số đánh giá chất lượng dịch như BLEU score, đồng thời phân tích các lỗi phổ biến và đề xuất cách cải thiện mô hình.

MỤC LỤC

ĐỒ ÁN ĐƯỢC HOÀN THÀNH.....	2
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG.....	2
CHƯƠNG 1 – TRANSFORMER BASED ENCODER MODELS	2
1.1 BERT (Bidirectional Encoder Representations from Transformers).....	2
1.2 So sánh BERT với RNN.....	2
1.3 Kiến trúc của BERT	3
CHƯƠNG 2 – ÚNG DỤNG TRANSFORMER TRONG BÀI TOÁN DỊCH THUẬT TIẾNG ANH SANG TIẾNG VIỆT	7
2.1 Giới thiệu về Transformer	7
2.2 Tổng quan về mô hình.....	8
2.3 Embedding Layer with Position Encoding	9
2.4 Encoder.....	10
2.5 Self Attention Layer	11
2.6 Multi Head Attention	14
2.7 Residuals Connection và Normalization Layer	15
2.8 Decoder	15
CHƯƠNG 3: KẾT QUẢ THỰC NGHIỆM	17
3.1 Kết quả huấn luyện	17
3.2 Kết quả dự đoán	17
3.3 Kết quả hiển thị Encoder và Decoder với attention	17
TÀI LIỆU THAM KHẢO.....	19

CHƯƠNG 1 – TRANSFORMER BASED ENCODER MODELS

1.1 BERT (Bidirectional Encoder Representations from Transformers)

BERT là một model biểu diễn ngôn ngữ (Language Model- LM) được google giới thiệu vào năm 2018. Trước khi BERT ra đời thì các tác vụ như: phân loại cảm xúc văn bản (tốt hay xấu, tích cực hay tiêu cực), sinh văn bản, dịch máy,... đều sử dụng kiến trúc RNN. Kiến trúc này có nhiều nhược điểm như train chậm, mất quan hệ giữa các từ xa nhau.

Tại thời điểm công bố, BERT đã nhanh chóng trở thành bá đạo trong mảng NLP bởi những cải tiến chưa từng có ở những model trước đó như: tăng độ chính xác, GLUE score (General Language Understanding Evaluation score)

1.2 So sánh BERT với RNN

Lý do chính là bởi nó có nhúng thêm ngữ cảnh (Context) vào trong các vector embedding. Ngữ cảnh là một thứ vô cùng quan trọng trong ngôn ngữ. Với các ngữ cảnh khác nhau thì các từ trong câu được hiểu theo ý nghĩa hoàn toàn khác nhau, các LM bỏ qua ngữ cảnh thì khó có thể đạt được chất lượng tốt.

Ngược về quá khứ ta sẽ thấy được sự phát triển của các phương pháp nhúng từ (word embedding) qua thời gian.

Đầu tiên là món embedding không ngữ cảnh

Ví dụ ta có 2 câu:

- Hôm nay em đi **chơi** bóng đá
- Thằng kia nó **chơi** em anh ạ

Hai cái từ chơi kia rõ ràng nghĩa khác nhau. Nhưng với việc nhúng từ không ngữ cảnh thì cả 2 từ này sẽ đều ánh xạ ra chung 1 vector word embedding và điều này đã làm giảm đi sự mềm mại và đa nghĩa của ngôn ngữ.

Tiếp theo là món embedding có ngữ cảnh một chiều

Người ta sử dụng các kiến trúc mạng RNN để có thể tạo ra mối quan hệ thứ tự giữa các từ trong câu, từ đó tạo ra vector nhúng từ có ngữ cảnh. Tuy nhiên việc này chỉ thực hiện được theo một chiều left-to-right hoặc right-to-left mà thôi. Một số mạng phúc tạp hơn đã sử dụng BiLSTM để chạy dọc theo câu theo 2 hướng ngược nhau nhưng 2 hướng này lại độc lập, chả liên quan gì đến nhau nên có thể xem là một chiều mà thôi.

Ví dụ một case mà chiêu nhúng từ một chiều này fail nhé. Giả sử ta có bài toán như sau:

- Câu văn gốc: “Hôm nay Nam đưa bạn gái đi chơi”
- Sau đó chúng ta che từ bạn gái đi và câu trên trở thành “Hôm nay Nam đưa [mask] đi chơi”.
- Yêu cầu bài toán là dự đoán ra từ đã được masking.

Rồi, với mặt thường của chúng ta thì ngay khi nhìn thấy câu này thì nghĩ ngay đến từ bạn gái. Nhưng model thì không như vậy, do nó chỉ được training một chiều, nó sẽ dự đoán [mask] từ các word trước đó là “Hôm nay Nam đưa” và thế là kết quả [mask] có thể sẽ được dự là: “tiền”, “mắt”, “hàng”.... (tùy vào corpus của chúng ta).

Và BERT đã đến và làm điều chúng ta cần

Ngay trong cái tên của BERT đã thấy ngay chữ Bidirectional (2 chiều) rồi. Tóm lại là một từ trong câu sẽ được biểu diễn một cách có liên quan đến cả từ trước lẫn từ sau, hay nói cách khác là liên quan đến tất cả các từ còn lại trong câu. Do đó khi ta che 1 từ trong câu đi, ví dụ như từ “bạn gái” bên trên thì lập tức model có thể predict ra khá chính xác vì dựa vào cả đoạn “Hôm nay Nam đưa” và “đi chơi”.

1.3 Kiến trúc của BERT

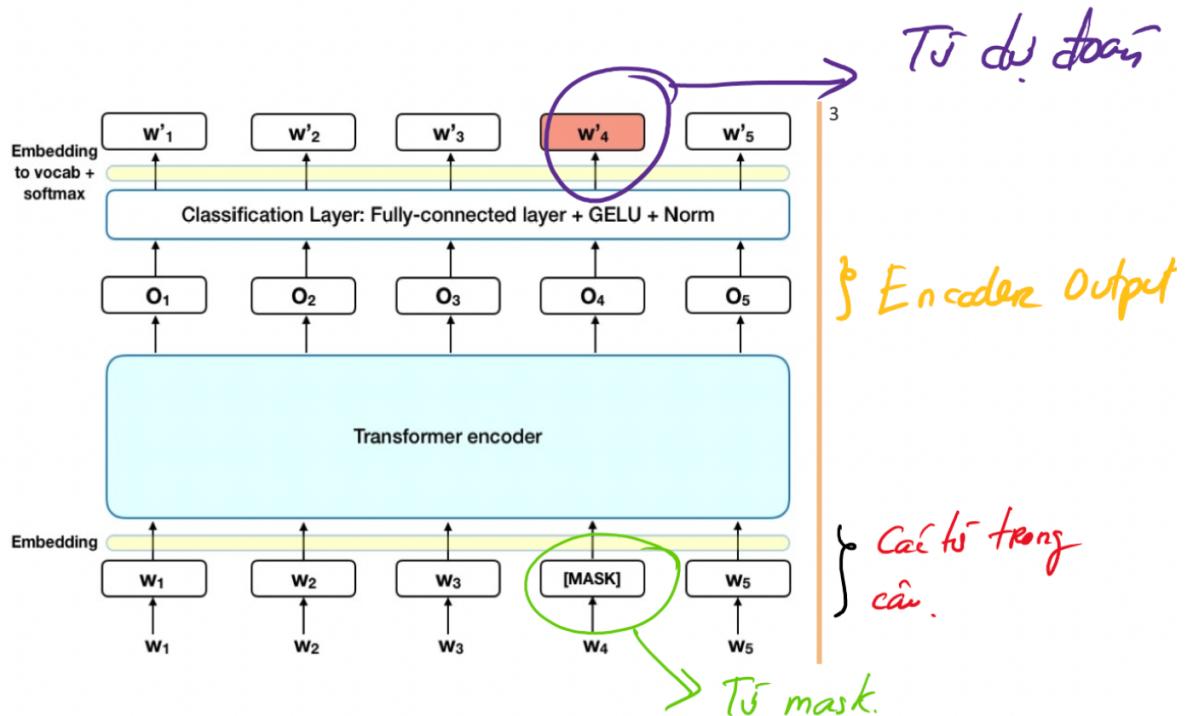
BERT được train đồng thời 2 task gọi là **Masked LM** (để dự đoán từ thiếu trong câu) và **Next Sentence Prediction** (NSP – dự đoán câu tiếp theo câu hiện tại). Hai món này được train đồng thời và loss tổng sẽ là kết hợp loss của 2 task và model sẽ cố gắng minimize loss tổng này. Chi tiết 2 task này như sau:

Masked LM

Với task này, ta train sẽ thực hiện che đi tầm 15% số từ trong câu và đưa vào model. Và ta sẽ train để model predict ra các từ bị che đó dựa vào các từ còn lại (đúng như câu Nam đưa bạn gái đi chơi ở trên đó).

Cụ thể là:

- Thêm một lớp classification lên trên encoder output
- Đưa các vector trong encoder output về vector bằng với vocab size, sau đó softmax để chọn ra từ tương ứng tại mỗi vị trí trong câu.
- Loss sẽ được tính tại vị trí masked và bỏ qua các vị trí khác (để đánh giá xem model dự đoán từ mask đúng/sai ntn mà, các từ khác đâu có liên quan).



Next Sentence Prediction (NSP)

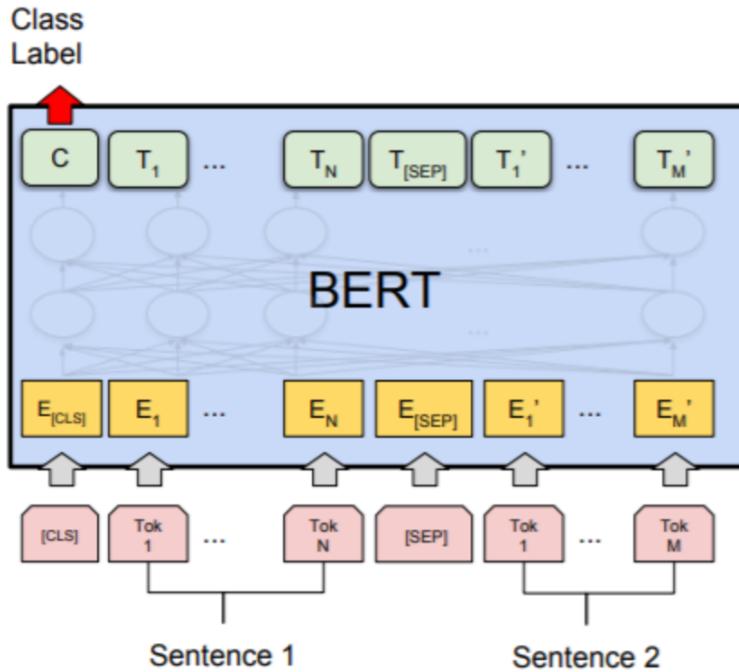
Với task này thì model sẽ được feed cho một cặp câu và nhiệm vụ của nó là output ra giá trị 1 nếu câu thứ hai đúng là câu đi sau câu thứ nhất và 0 nếu không phải. Trong quá trình train, ta chọn 50% mẫu là Positive (output là 1) và 50% còn lại là Negative được ghép linh tinh (output là 0).

Cụ thể cách train như sau:

- Bước 1: Ghép 2 câu vào nhau và thêm 1 số token đặc biệt để phân tách các câu. Token [CLS] thêm vào đầu câu Thứ nhất, token [SEP] thêm vào cuối mỗi câu. Ví dụ ghép 2 câu “Hôm nay em đi học” và “Học ở trường

rất hay” thì sẽ thành [CLS] Hôm nay em đi học [SEP] Học ở trường rất vui [SEP]

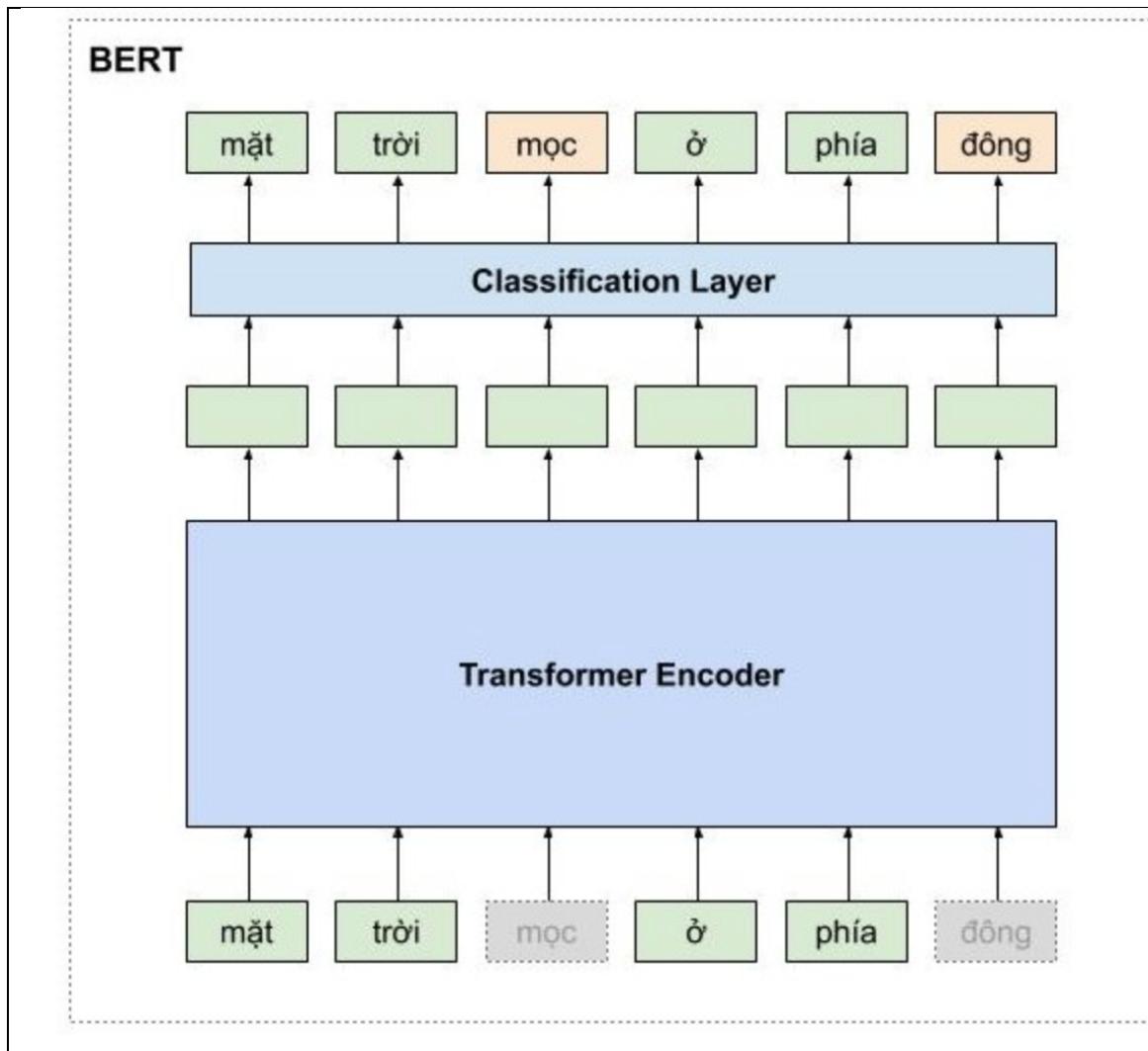
- Bước 2. Mỗi token trong câu sẽ được cộng thêm một vector gọi là Sentence Embedding, thực ra là đánh dấu xem từ đó thuộc câu Thứ nhất hay câu thứ 2 thôi. Ví dụ nếu thuộc câu Thứ nhất thì cộng thêm 1 vector toàn số “0” có kích thước bằng Word Embedding, và nếu thuộc câu thứ 2 thì cộng thêm một vector toàn số “1”.
- Bước 3. Sau đó các từ trong câu đã ghép sẽ được thêm vector Positional Encoding vào để đánh dấu vị trí từng từ trong câu đã ghép (Bạn nào chưa biết thì xem lại bài về Transformer nhé).
- Bước 4. Đưa chuỗi sau bước 3 vào mạng.
- Bước 5. Lấy encoder output tại vị trí token [CLS] được transform sang một vector có 2 phần tử $[c1 \ c2]$.
- Bước 6. Tính softmax trên vector đó và output ra probability của 2 class: Đi sau và Không đi sau. Để thể hiện câu thứ hai là đi sau câu thứ nhất hay không, ta lấy argmax là được



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

CHƯƠNG 2 – ỨNG DỤNG TRANSFORMER TRONG BÀI TOÁN DỊCH THUẬT TIẾNG ANH SANG TIẾNG VIỆT

2.1 Giới thiệu về Transformer



Sự nổi tiếng của mô hình Transformer thì không cần bàn cãi, vì nó chính là nền tảng của rất nhiều mô hình khác mà nổi tiếng nhất là BERT (Bidirectional Encoder Representations from Transformers) một mô hình dùng để học biểu diễn

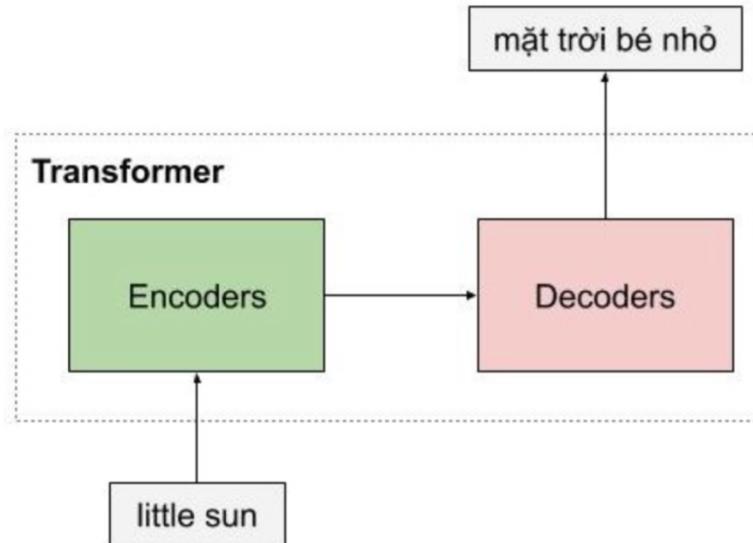
của các từ tốt nhất hiện tại và đã tạo ra một bước ngoặt lớn cho động đồng NLP trong năm 2019. Và chính Google cũng đã áp dụng BERT trong cỗ máy tìm kiếm của họ. Để hiểu BERT, các bạn cần phải nắm rõ về mô hình Transformer.

2.2 Tổng quan về mô hình

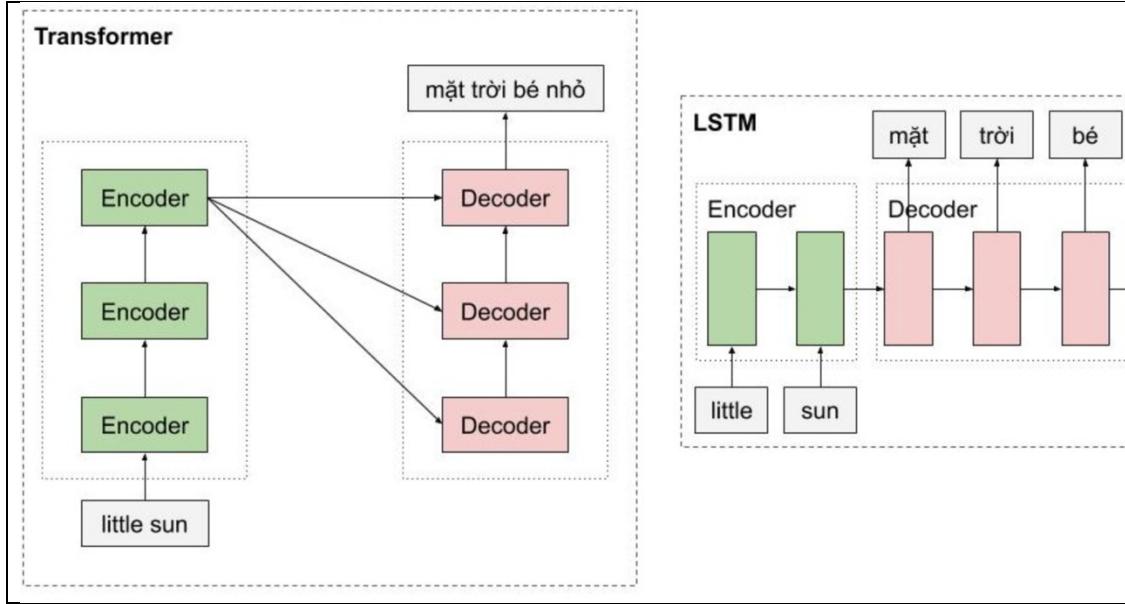
Để cho dễ cảm nhận được cách mà mô hình hoạt động, mình sẽ trình bày trước toàn bộ kiến trúc mô hình ở mức high-level và sau đó sẽ đi chi tiết từng phần nhỏ cũng như công thức toán của nó.

Giống như những mô hình dịch máy khác, kiến trúc tổng quan của mô hình transformer bao gồm 2 phần lớn là encoder và decoder. Encoder dùng để học vector biểu của câu với mong muốn rằng vector này mang thông tin hoàn hảo của câu đó. Decoder thực hiện chức năng chuyển vector biểu diễn kia thành ngôn ngữ đích.

Trong ví dụ ở dưới, encoder của mô hình transformer nhận một câu tiếng việt, và encode thành một vector biểu diễn ngữ nghĩa của câu little sun, sau đó mô hình decoder nhận vector biểu diễn này, và dịch nó thành câu tiếng việt mặt trời bé nhỏ



Một trong những ưu điểm của transformer là mô hình này có khả năng xử lý song song cho các từ. Như các bạn thấy, Encoders của mô hình transfromer là một dạng feedforward neural nets, bao gồm nhiều encoder layer khác, mỗi encoder layer này xử lý đồng thời các từ. Trong khi đó, với mô hình LSTM, thì các từ phải được xử lý tuần tự. Ngoài ra, mô hình Transformer còn xử lý câu đầu vào theo 2 hướng mà không cần phải stack thêm một mô hình LSTM nữa như trong kiến trúc Bidirectional LSTM.



2.3 Embedding Layer with Position Encoding

Trước khi đi vào mô hình encoder, chúng ta sẽ tìm hiểu cơ chế rất thú vị là Position Encoding dùng để đưa thông tin về vị trí của các từ vào mô hình transformer.

Đầu tiên, các từ được biểu diễn bằng một vector sử dụng một ma trận word embedding có số dòng bằng kích thước của tập từ vựng. Sau đó các từ trong câu được tìm kiếm trong ma trận này, và được nối nhau thành các dòng của một ma trận 2 chiều chứa ngữ nghĩa của từng từ riêng biệt. Nhưng như các bạn đã thấy, transformer xử lý các từ song song, do đó, với chỉ word embedding mô hình không thể nào biết được vị trí các từ. Như vậy, chúng ta cần một cơ chế nào đó để đưa thông tin vị trí các từ vào trong vector đầu vào. Đó là lúc positional encoding xuất hiện và giải quyết vấn đề của chúng ta. Tuy nhiên, trước khi giới thiệu cơ chế position encoding của tác giả, các bạn có thể giải quyết vấn đề bằng một số cách naive như sau:

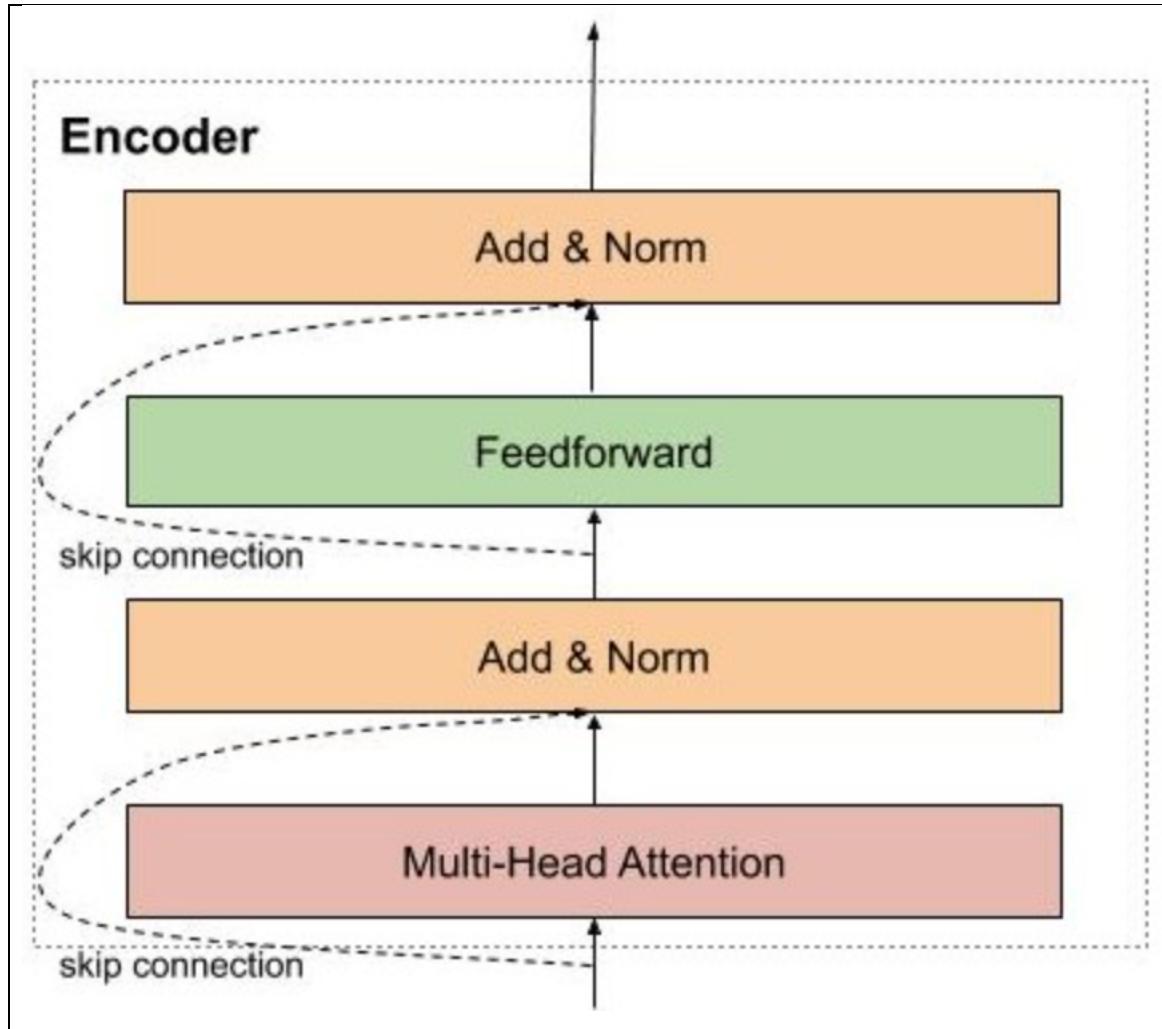
Biểu diễn vị trí các từ bằng chuỗi các số liên tục từ 0,1,2,3 ..., n. Tuy nhiên, chúng ta gặp ngay vấn đề là khi chuỗi dài thì số này có thể khá lớn, và mô hình sẽ gặp khó khăn khi dự đoán những câu có chiều dài lớn hơn tất cả các câu có trong tập huấn luyện. Để giải quyết vấn đề này, các bạn có thể chuẩn hóa lại cho chuỗi số này nằm trong đoạn từ 0-1 bằng cách chia cho n nhưng mà chúng ta sẽ gặp vấn đề khác là khoảng cách giữa 2 từ liên tiếp sẽ phụ thuộc vào chiều dài của

chuỗi, và trong một khoản cố định, chúng ta không hình dùng được khoản đó chứa bao nhiêu từ. Điều này có nghĩa là ý nghĩa của position encoding sẽ khác nhau tùy thuộc vào độ dài của câu đó.

2.4 Encoder

Encoder của mô hình transformer có thể bao gồm nhiều encoder layer tương tự nhau. Mỗi encoder layer của transformer lại bao gồm 2 thành phần chính là multi head attention và feedforward network, ngoài ra còn có cả skip connection và normalization layer.

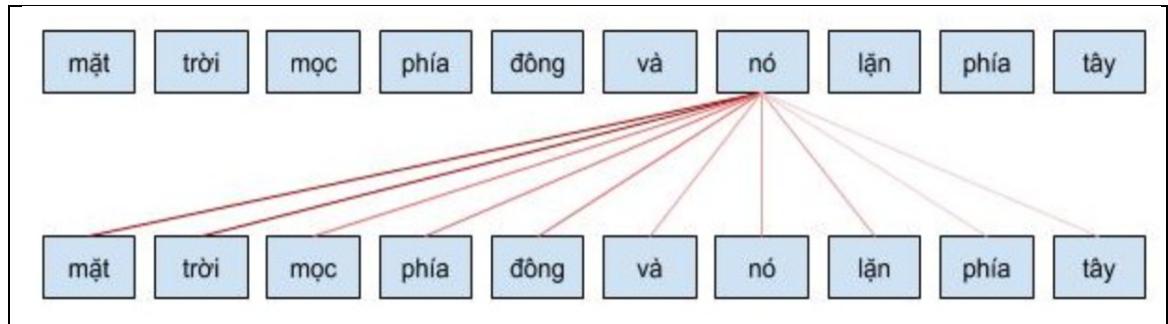
Trong 2 thành phần chính này, các bạn sẽ hứng thú nhiều hơn về multi-head attention vì đó là một layer mới được giới thiệu trong bài báo này, và chính nó tạo nên sự khác biệt giữa mô hình LSTM và mô hình Transformer mà chúng ta đang tìm hiểu.



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. Multi Head Attention thật chất là self-attention, nhưng mà để mô hình có thể có chú ý nhiều pattern khác nhau, tác giả đơn giản là sử dụng nhiều self-attention.

2.5 Self Attention Layer

Self Attention cho phép mô hình khi mã hóa một từ có thể sử dụng thông tin của những từ liên quan tới nó. Ví dụ khi từ đó được mã hóa, nó sẽ chú ý vào các từ liên quan như là mặt trời. Cơ chế self attention này có ý nghĩa tương tự như cơ chế attention mình đã chia sẻ ở bài trước và những công thức toán học cũng tương ứng với nhau.



Bạn 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 cách từ còn lại, từ nào “giống” để sau đó thông tin sẽ được mã hóa dựa trên tất cả các từ trên.

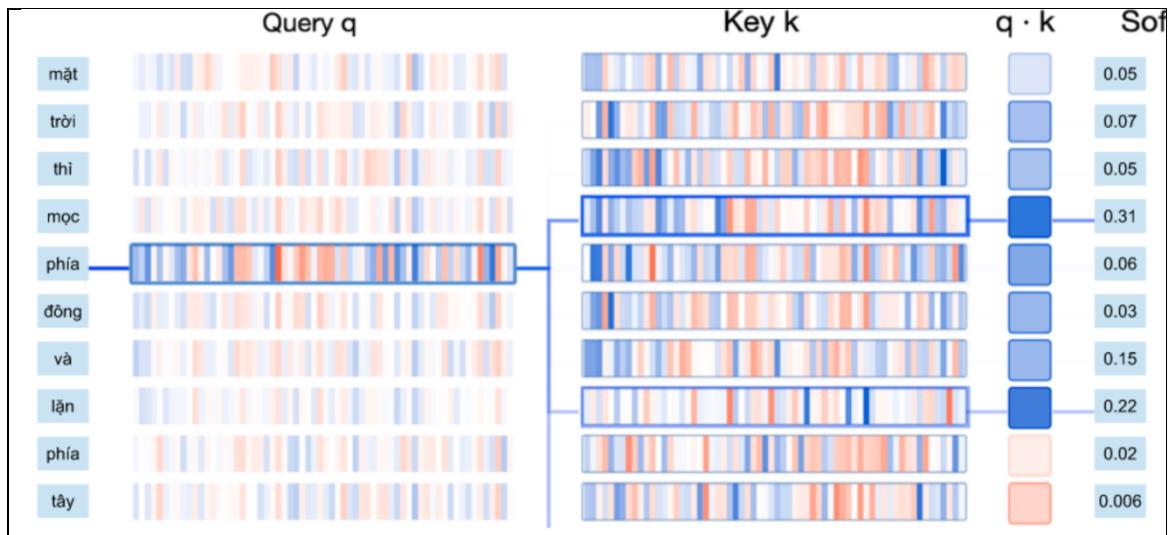
Đầu tiên, với mỗi từ chúng ta cần tạo ra 3 vector: query, key, value vector bằng cách nhân ma trận biểu diễn các từ đầu vào với ma trận học tương ứng.

* query vector: vector dùng để chứa thông tin của từ được tìm kiếm, so sánh. Giống như là câu query của google search.

* key vector: vector dùng để biểu diễn thông tin các từ được so sánh với từ cần tìm kiếm ở trên. Ví dụ, như các trang webs mà google sẽ so sánh với từ khóa mà bạn tìm kiếm.

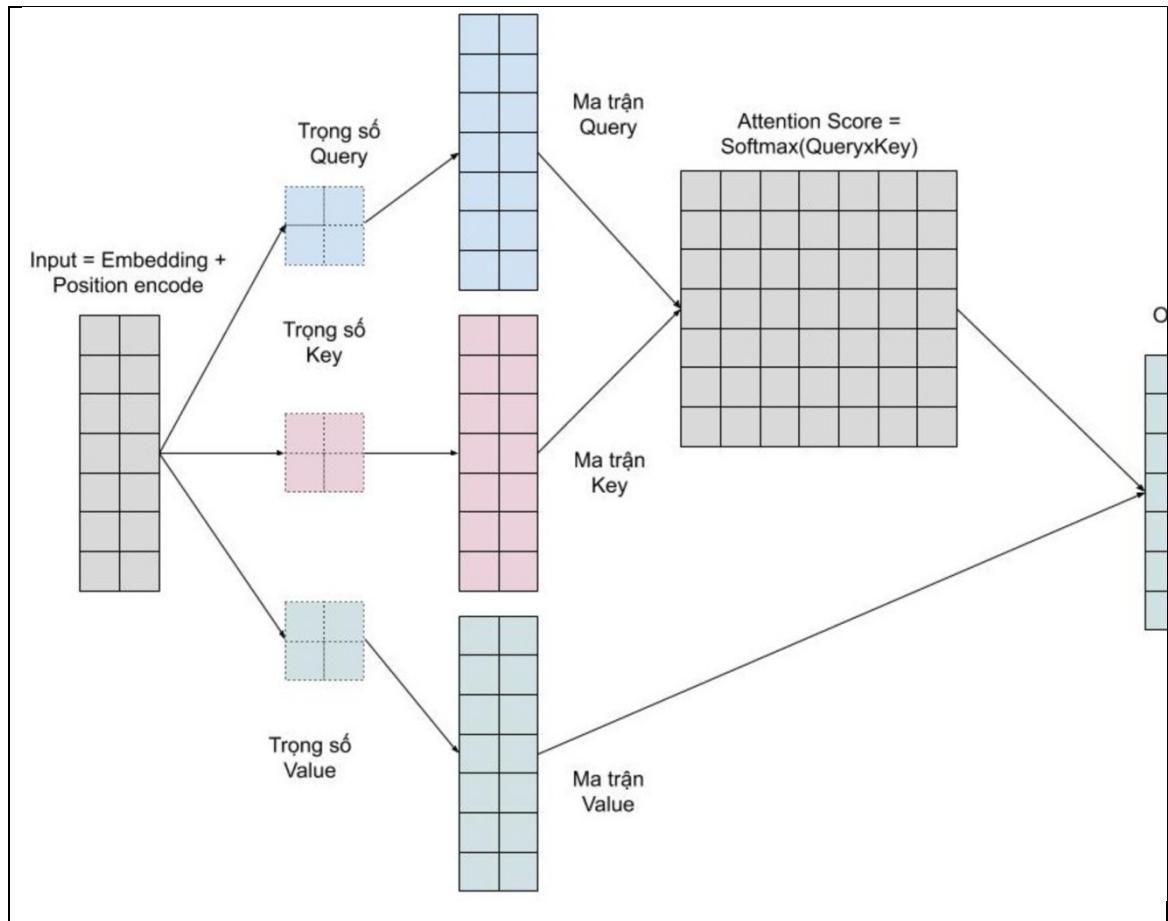
* value vector: vector biểu diễn nội dung, ý nghĩa của các từ. Các bạn có thể tượng tượng, nó như là nội dung trang web được hiển thị cho người dùng sau khi tìm kiếm.

Để tính tương quan, chúng ta đơn giản chỉ cần tính tích vô hướng dựa các vector query và key. Sau đó dùng hàm softmax để chuẩn hóa chỉ số tương quan trong đoạn 0-1, và cuối cùng, tính trung bình cộng có trọng số giữa các vector values sử dụng chỉ số tương quan mới tính được. Quá dễ !!!



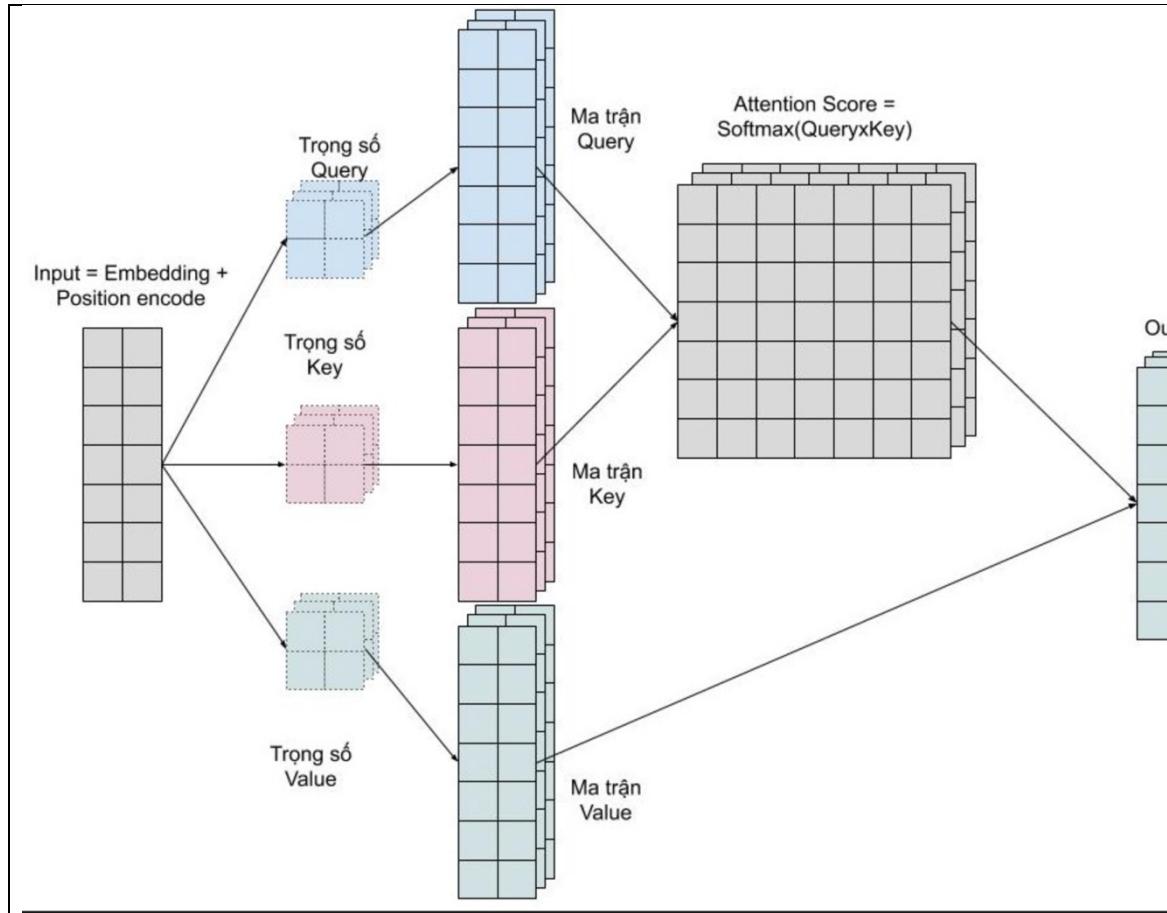
Cụ thể hơn, quá trình tính toán attention vector có thể được tóm tắt làm 3 bước như sau:

- * Bước 1: Tính ma trận query, key, value bằng cách khởi tạo 3 ma trận trọng số query, key, vector. Sau đó nhân input với các ma trận trọng số này để tạo thành 3 ma trận tương ứng.
- * Bước 2: Tính attention weights. Nhân 2 ma trận key, query vừa được tính ở trên với nhau để với ý nghĩa là so sánh giữa câu query và key để học mối tương quan. Sau đó thì chuẩn hóa về đoạn [0-1] bằng hàm softmax. 1 có nghĩa là câu query giống với key, 0 có nghĩa là không giống.
- * Bước 3: Tính output. Nhân attention weights với ma trận value. Điều này có nghĩa là chúng ta biểu diễn một từ bằng trung bình có trọng số (attention weights) của ma trận value.



2.6 Multi Head Attention

Chúng ta muốn mô hình có thể học nhiều kiểu mối quan hệ giữ 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. Tức là chúng ta cần nhiều ma trận query, key, value mà thôi. Giờ đây ma trận trọng số key, query, value sẽ có thêm 1 chiều depth nữa.



Multi head attention cho phép mô hình chú ý đến đồng thời những pattern dễ quan sát được như sau.

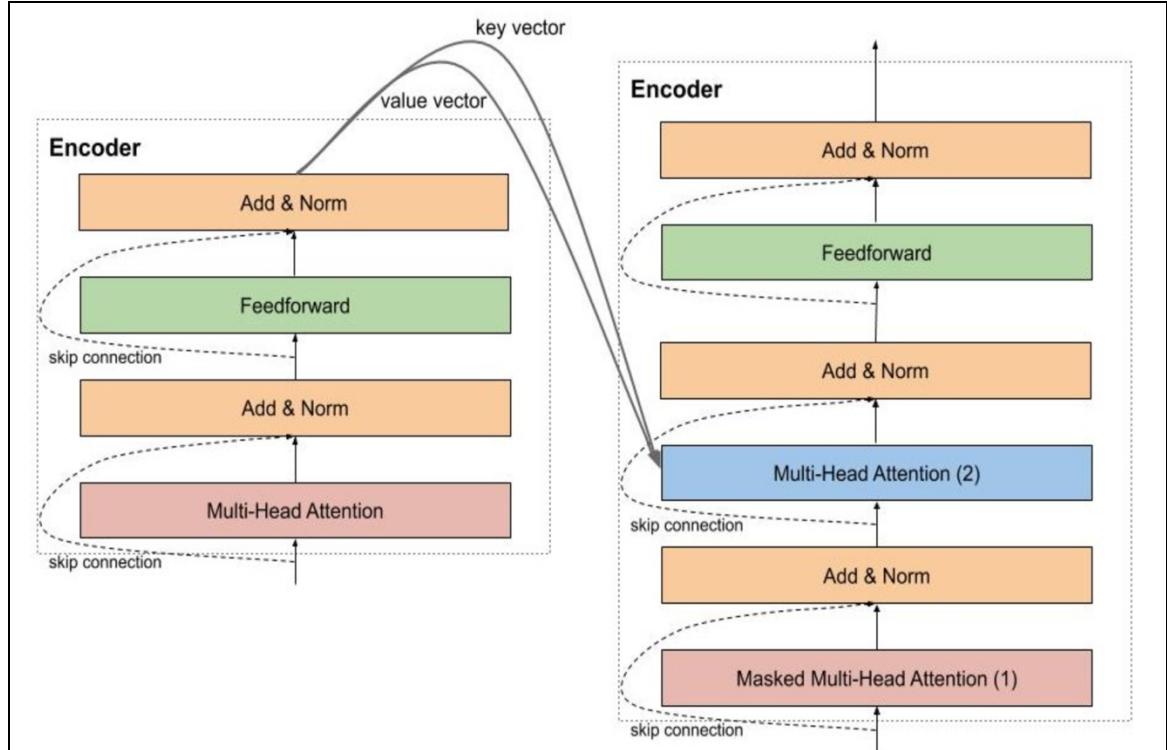
- Chú ý đến từ kế trước của một từ
- Chú ý đến từ kế sau của một từ
- Chú ý đến những từ liên quan của một từ

2.7 Residuals Connection và Normalization Layer

Trong kiến trúc của mô hình transformer, residuals connection và normalization layer được sử dụng mọi nơi, giống như tinh thần của nó. 2 kỹ thuật giúp cho mô hình huấn luyện nhanh hơn và trách mắng thông tin trong quá trình huấn luyện mô hình, ví dụ như là thông tin của vị trí các từ được mã hóa.

2.8 Decoder

Decoder thực hiện chức năng giải mã vector của câu nguồn thành câu đích, do đó decoder sẽ nhận thông tin từ encoder là 2 vector key và value. Kiến trúc của decoder rất giống với encoder, ngoại trừ có thêm một multi head attention nằm ở giữa dùng để học mối liên quan giữ từ đang được dịch với các từ được ở câu nguồn.



Masked Multi Head Attention

Masked Multi Head Attention tất nhiên là multi head attention mà chúng ta đã nói đến ở trên, có chức năng dùng để encode các từ câu câu đích trong quá trình dịch, tuy nhiên, lúc cài đặt chúng ta cần lưu ý rằng phải che đi các từ ở tương lai chưa được mô hình dịch đến, để làm việc này thì đơn giản là chúng ta chỉ cần nhân với một vector chứa các giá trị 0,1.

Trong decoder còn có một multi head attention khác có chức năng chú ý các từ ở mô hình encoder, layer này nhận vector key và value từ mô hình encoder, và output từ layer phía dưới. Đơn giản bởi vì chúng ta muốn so sánh sự tương quan giữ từ đang được dịch với các từ nguồn.

CHƯƠNG 3: KẾT QUẢ THỰC NGHIỆM

3.1 Kết quả huấn luyện

```

[+] epoch: 000 - iter: 00399 - train loss: 9.281 - time: 0.1355
epoch: 000 - iter: 00399 - train loss: 8.370 - time: 0.1263
epoch: 000 - iter: 00599 - train loss: 7.1865 - time: 0.1295
epoch: 000 - iter: 00799 - train loss: 6.4846 - time: 0.1264
epoch: 000 - iter: 00999 - train loss: 6.1972 - time: 0.1292
epoch: 000 - iter: 01199 - train loss: 6.1971 - time: 0.1359
epoch: 000 - iter: 01399 - train loss: 6.0396 - time: 0.1345
epoch: 000 - iter: 01599 - train loss: 5.9591 - time: 0.1341
epoch: 000 - iter: 01799 - train loss: 5.7359 - time: 0.1329
epoch: 000 - iter: 01999 - train loss: 5.6722 - time: 0.1329
epoch: 000 - iter: 02199 - train loss: 5.6727 - time: 0.1327
epoch: 000 - iter: 02399 - train loss: 5.3774 - time: 0.1333
epoch: 000 - iter: 02499 - valid loss: 3.976 - bleu score: 0.0114 - time: 125.6958
epoch: 001 - iter: 00399 - train loss: 5.1849 - time: 0.1263
epoch: 001 - iter: 00599 - train loss: 5.1849 - time: 0.1263
epoch: 001 - iter: 00799 - train loss: 5.0459 - time: 0.1275
epoch: 001 - iter: 00999 - train loss: 4.9286 - time: 0.1281
epoch: 001 - iter: 01199 - train loss: 4.9286 - time: 0.1288
epoch: 001 - iter: 01399 - train loss: 4.7339 - time: 0.1111
epoch: 001 - iter: 01599 - train loss: 4.6557 - time: 0.1318
epoch: 001 - iter: 01799 - train loss: 4.5259 - time: 0.1299
epoch: 001 - iter: 01999 - train loss: 4.5240 - time: 0.1226
epoch: 001 - iter: 02199 - train loss: 4.4562 - time: 0.1289
epoch: 001 - iter: 02399 - train loss: 4.2996 - time: 0.1293
epoch: 001 - iter: 02499 - valid loss: 3.1684 - bleu score: 0.1072 - time: 151.4589
epoch: 002 - iter: 00399 - train loss: 4.9451 - time: 0.1294
epoch: 002 - iter: 00599 - train loss: 4.9451 - time: 0.1327
epoch: 002 - iter: 00799 - train loss: 4.8009 - time: 0.1279
epoch: 002 - iter: 00999 - train loss: 3.9476 - time: 0.1072
epoch: 002 - iter: 01199 - train loss: 3.8881 - time: 0.1327
epoch: 002 - iter: 01399 - train loss: 3.8881 - time: 0.1324
epoch: 002 - iter: 01599 - train loss: 3.8663 - time: 0.1186
epoch: 002 - iter: 01799 - train loss: 3.8844 - time: 0.1389
epoch: 002 - iter: 01999 - train loss: 3.8849 - time: 0.1299
epoch: 002 - iter: 02399 - train loss: 3.7214 - time: 0.1257
epoch: 002 - iter: 02499 - valid loss: 2.7862 - bleu score: 0.1741 - time: 163.0977
epoch: 003 - iter: 00399 - train loss: 3.6542 - time: 0.1370
epoch: 003 - iter: 00599 - train loss: 3.6346 - time: 0.1280
epoch: 003 - iter: 00799 - train loss: 3.5539 - time: 0.1289
epoch: 003 - iter: 00999 - train loss: 3.5353 - time: 0.1354
epoch: 003 - iter: 01199 - train loss: 3.5859 - time: 0.1290
epoch: 003 - iter: 01399 - train loss: 3.5859 - time: 0.1295
epoch: 003 - iter: 01599 - train loss: 3.5233 - time: 0.1230
epoch: 003 - iter: 01799 - train loss: 3.5224 - time: 0.1263
epoch: 003 - iter: 01999 - train loss: 3.4956 - time: 0.1256
epoch: 003 - iter: 02399 - train loss: 3.4956 - time: 0.1257
epoch: 003 - iter: 02499 - valid loss: 2.6176 - bleu score: 0.2073 - time: 172.6203
epoch: 004 - iter: 00399 - train loss: 3.3587 - time: 0.1293
epoch: 004 - iter: 00599 - train loss: 3.3723 - time: 0.1255
epoch: 004 - iter: 00799 - train loss: 3.3723 - time: 0.1252
epoch: 004 - iter: 00999 - train loss: 3.3913 - time: 0.1115
epoch: 004 - iter: 01199 - train loss: 3.3439 - time: 0.1363

```

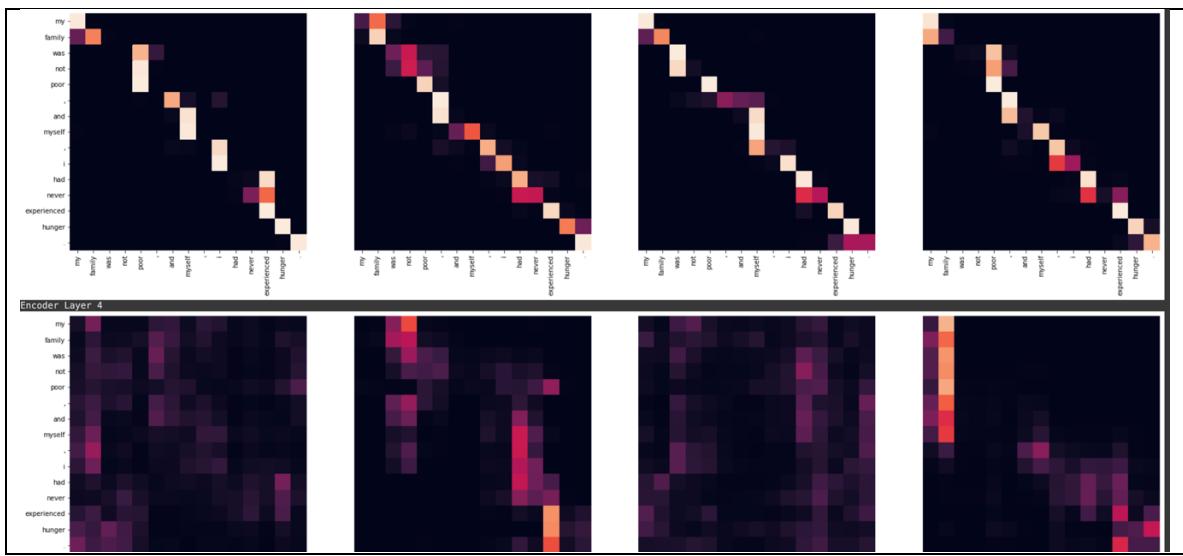
3.2 Kết quả dự đoán

```

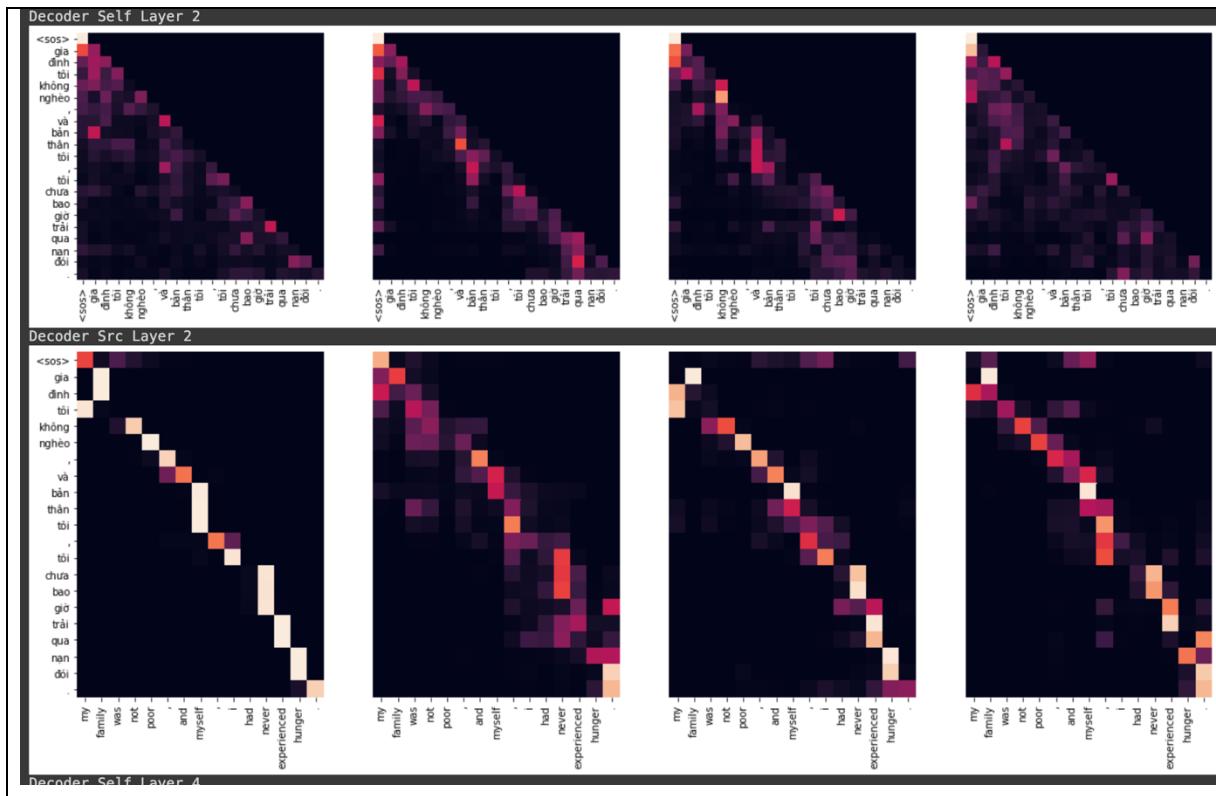
⚙️ 1 bleu(valid_src_data, valid_trg_data, model, SRC, TRG, opt['device'], opt['k'], opt['max_strlen'])
⚙️ 0.3723
[+] 1 sentence='My family was not poor , and myself , I had never experienced hunger .'
2 trans_sent = translate_sentence(sentence, model, SRC, TRG, opt['device'], opt['k'], opt['max_strlen'])
3 trans_sent
[+] 'gia đình tôi không nghèo, và bản thân tôi, tôi chưa bao giờ trải qua nạn đói.'
[+] 1 sentence='In the end, we will remember not the words of our enemies, but the silence of our friends.'
2 trans_sent = translate_sentence(sentence, model, SRC, TRG, opt['device'], opt['k'], opt['max_strlen'])
3 trans_sent
[+] 'Cuối cùng, chúng ta sẽ nhớ không những từ của kẻ thù, nhưng sự im lặng của bạn bè.'

```

3.3 Kết quả hiển thị Encoder và Decoder với attention



Visualize Encoder



Visualize Decoder

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Tìm hiểu về hog(histogram of oriented gradients) - Nguyễn Phương Lan
2. Tìm hiểu về phương pháp mô tả đặc trưng HOG (Histogram of Oriented Gradients) - Hai Ha
3. Trích đặc trưng HOG - Histograms of Oriented Gradients - Minh Nguyen
4. Bài 12 - Các thuật toán Object Detection - Phạm Đình Khanh

Tiếng Anh

5. Histograms of Oriented Gradients for Human Detection:<http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>
6. skimage Documentation : <https://scikit-image.org/docs/dev/api/skimage.feature.html?highlight=hog#skimage.feature.hog>
7. HOG (Histogram of Oriented Gradients) Features (Theory and Implementation using MATLAB and Python): <https://www.youtube.com/watch?v=QmYJCxJWdEs>