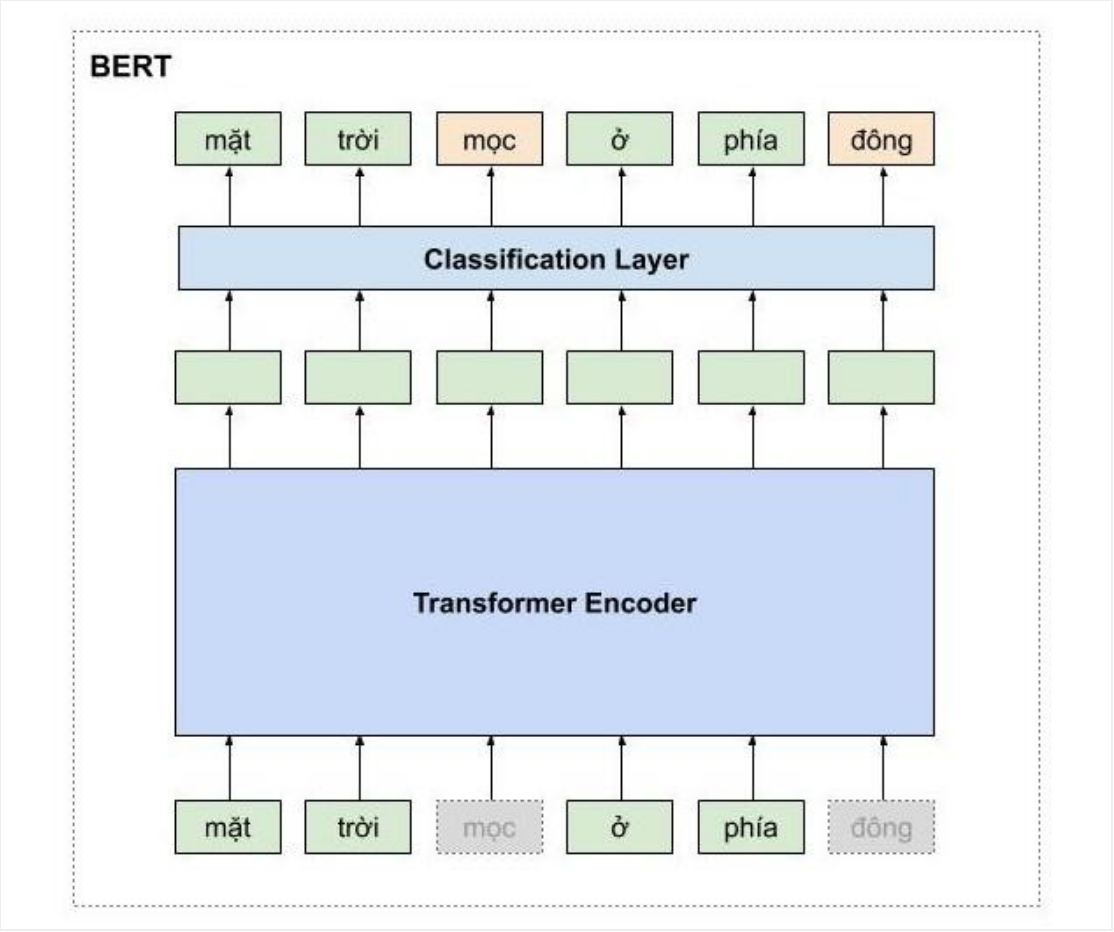
Tìm hiểu mô hình Transformer - Ngươi Không Phải Là Anh Hùng, Ngươi Là Quái Vật Nhiều Đầu.

Trong blog này, mình sẽ trình bày chi tiết cách mô hình Transformer hoạt động, cũng như là cách cài đặt mô hình chi tiết cho những bạn mới có kiến thức cơ bản về deep learning như CNN hoặc LSTM cũng có thể hiểu được.

Sự nổi tiếng của mô hình Transformer thì không cần phải 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.



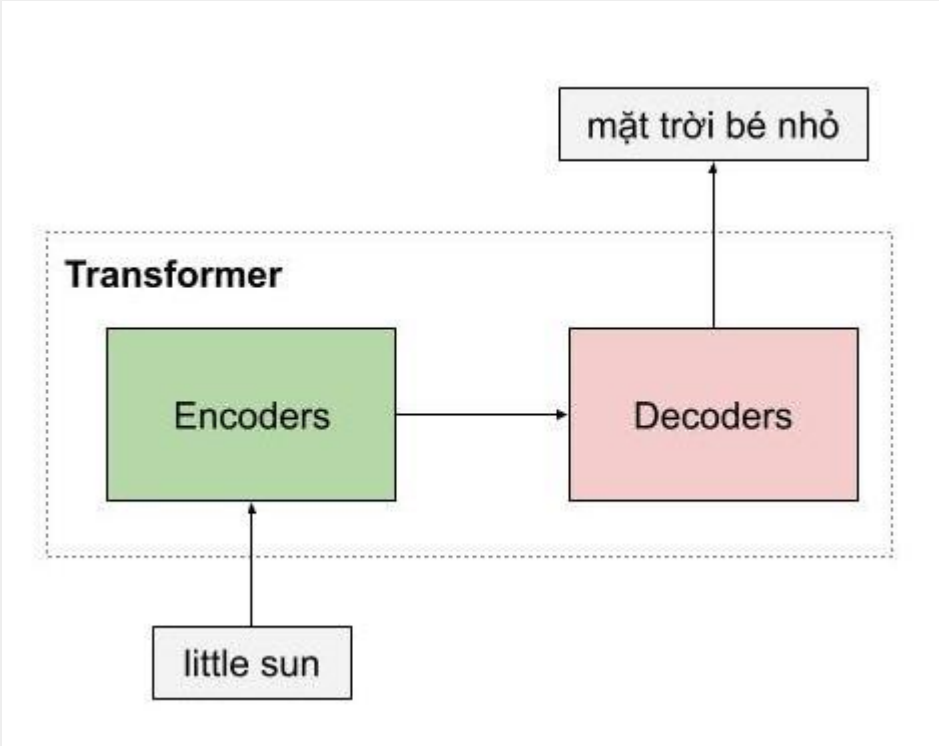
Ý tưởng chủ đạo của Transformer vẫn là áp dụng cơ thể Attention, những ở mức phức tạp hơn và thật sự là thú vị hơn so với cách được đề xuất trước đó trong một bài báo của tác giả Lương Minh Thắng, một người Việt rất nổi tiếng trong cộng đồng deep learning.

Tổng Quan Mô Hình

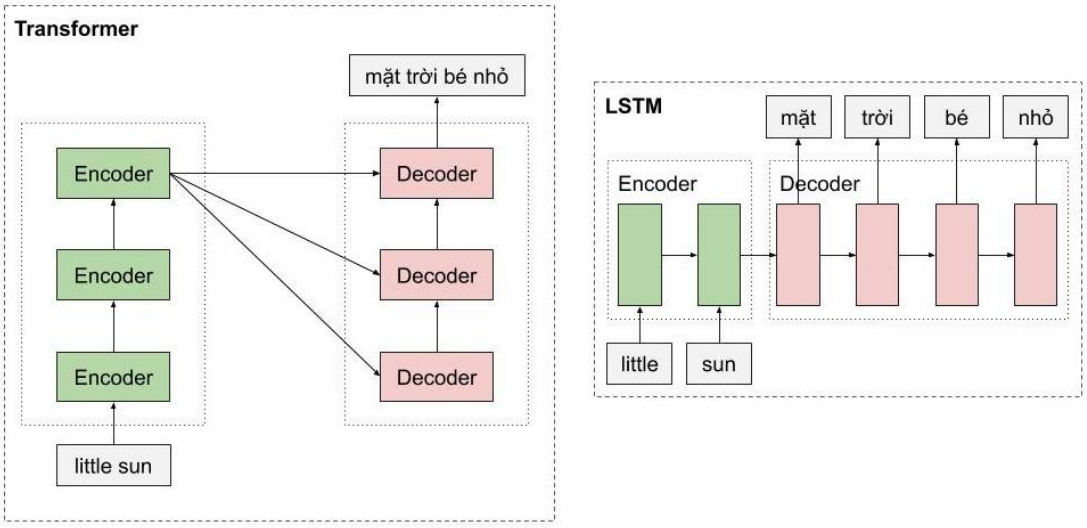
Để cho dễ cảm nhận được cách mà mô hình hoạt động, mình sẽ trình bày trước toàn bộ kiến trúc mô hình ở mức high-level và sau đó sẽ đi chi tiết từng phần nhỏ cũng như công thức toán của nó.

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.

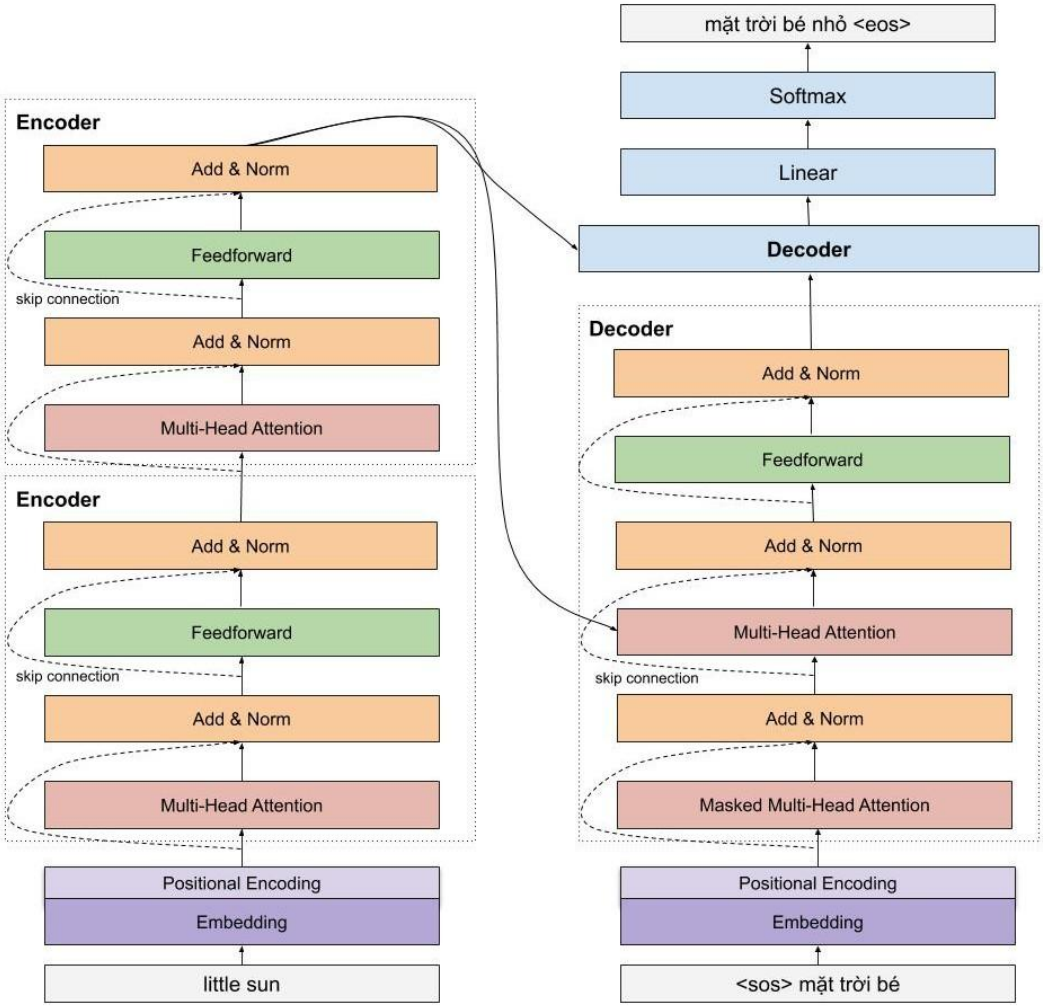
Trong ví dụ ở dưới, encoder của mô hình transformer nhận một câu tiếng anh, và encode thành một vector biểu diễn ngữ nghĩa của câu *little sun*, sau đó mô hình decoder nhận vector biểu diễn này, và dịch nó thành câu tiếng việt *mặt trời bé nhỏ*



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 transformer 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 hình LSTM nữa như trong kiến trúc Bidirectional LSTM.



Một cái nhìn vừa tổng quát và chi tiết sẽ giúp ích cho các bạn. Mình sẽ đi vào chi tiết một số phần cực kì quan trọng như sinusoidal position encoding, multi head attention của encoder, còn của decoder thì các bạn thấy được kiến trúc rất giống với của encoder, do đó mình sẽ chỉ đi nhanh qua mà thôi.



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.

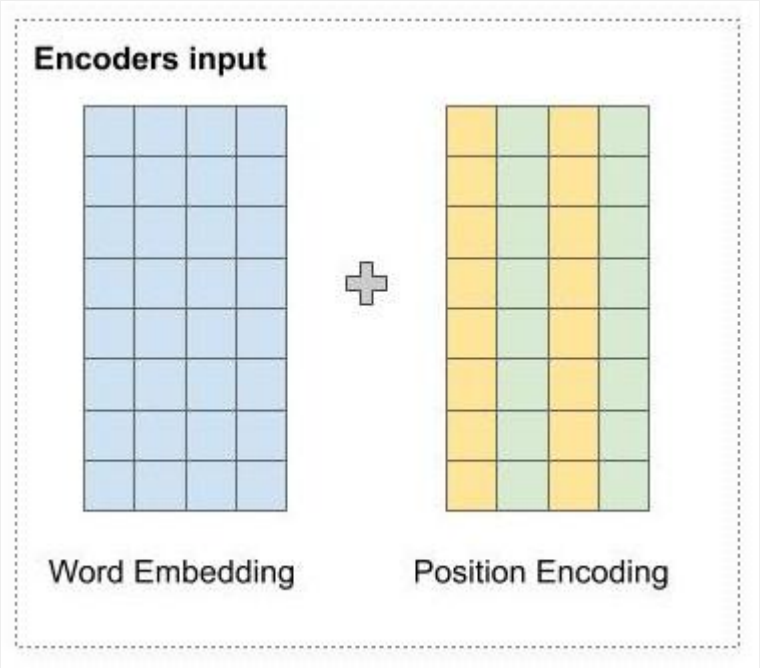
Đầu tiên, các từ được biểu diễn bằng một vector sử dụng một ma trận word embedding có số dòng bằng kích thước của tập từ vựng. Sau đó các từ trong câu được tìm kiếm trong ma trận này, và được nối nhau thành các dòng của một ma trận 2 chiều chứa ngữ nghĩa của từng từ riêng biệt. Nhưng như các bạn đã thấy, 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 băng 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.

Phương pháp đề xuất sinusoidal position encoding

Phương pháp của tác giả đề xuất không gặp những hạn chế mà chúng ta vừa nêu. Vị trí của các từ được mã hóa bằng một vector có kích thước bằng word embedding và được cộng trực tiếp vào word embedding.



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.

$$p_t^i = f(t)^i = \begin{cases} sin(w_k * t) \text{ if } i = 2k \\ cos(w_k * t) \text{ if } i = 2k+1 \end{cases}$$

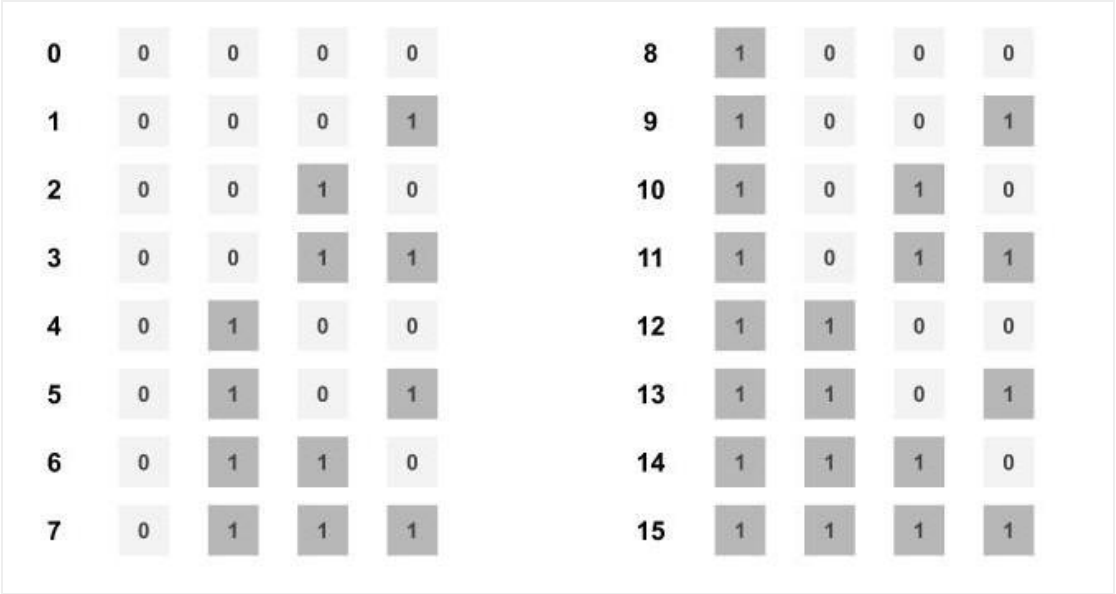
Trong đó

$$w_k = \frac{1}{10000^{2k/d}}$$

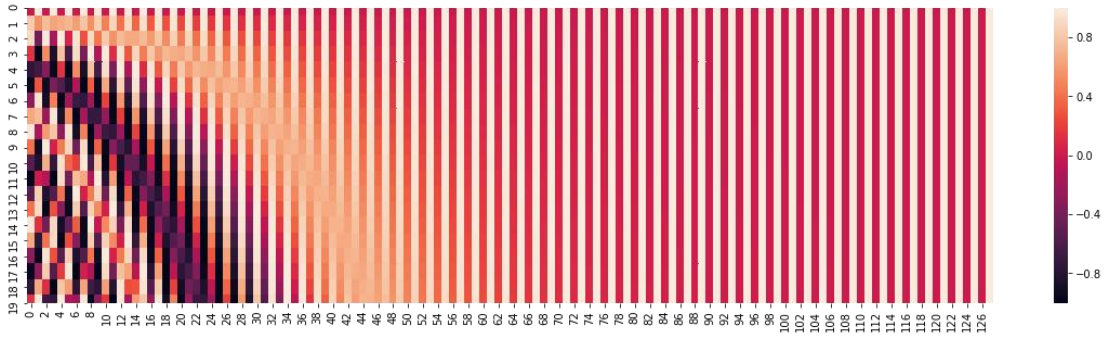
Trong hình dưới này, mình minh họa cho cách tính position encoding của tác giả. Giả sử chúng ta có word embedding có 6 chiều, thì position encoding cũng có tương ứng là 6 chiều. Mỗi dòng tương ứng với một từ. Giá trị của các vector tại mỗi vị trí được tính toán theo công thức ở hình dưới.

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

Lúc này một số bạn sẽ thắc mắc tại sao với cách biểu diễn vị trí như tác giả đề xuất lại có thể mã hóa thông tin vị trí của từ? Hãy tưởng tượng bạn có các số từ 0-15. Các bạn có thể thấy rằng bit ngoài cùng bên phải thay đổi nhanh nhất mỗi 1 số, và sau đó là bit bên phải thứ 2, thay đổi mỗi 2 số, tương tự cho các bit khác.

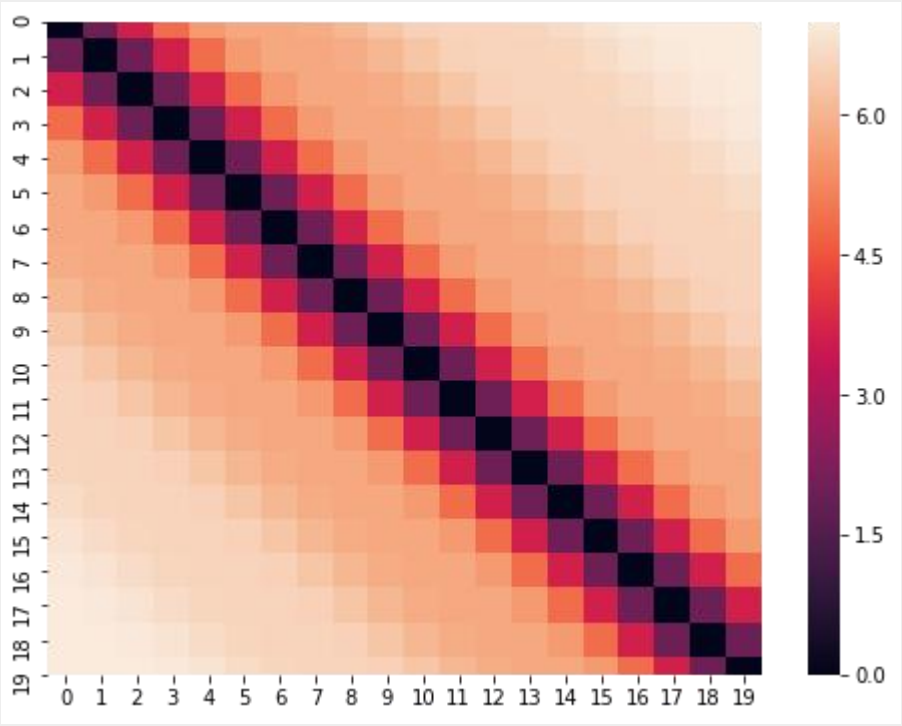


Trong công thức của tác giả đề xuất, các bạn cũng thấy rằng, hàm sin và cos có dạng đồ thị tần số và tần số này giảm dần ở các chiều lớn dần. Các bạn xem hình dưới, ở chiều 0, giá trị thay đổi liên tục tương ứng với màu sắc thay đổi liên tục, và tần số thay đổi này giảm dần ở các chiều lớn hơn.



Nên chúng ta có thể cảm nhận được việc biểu diễn của tác giả khá tương tự như cách biểu diễn các số nguyên trong hệ nhị phân, cho nên chúng ta có thể biểu diễn được vị trí các từ theo cách như vậy.

Chúng ta cũng có thể xem ma trận khoảng cách của các vector biểu diễn vị trí như hình dưới. Rõ ràng, 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.



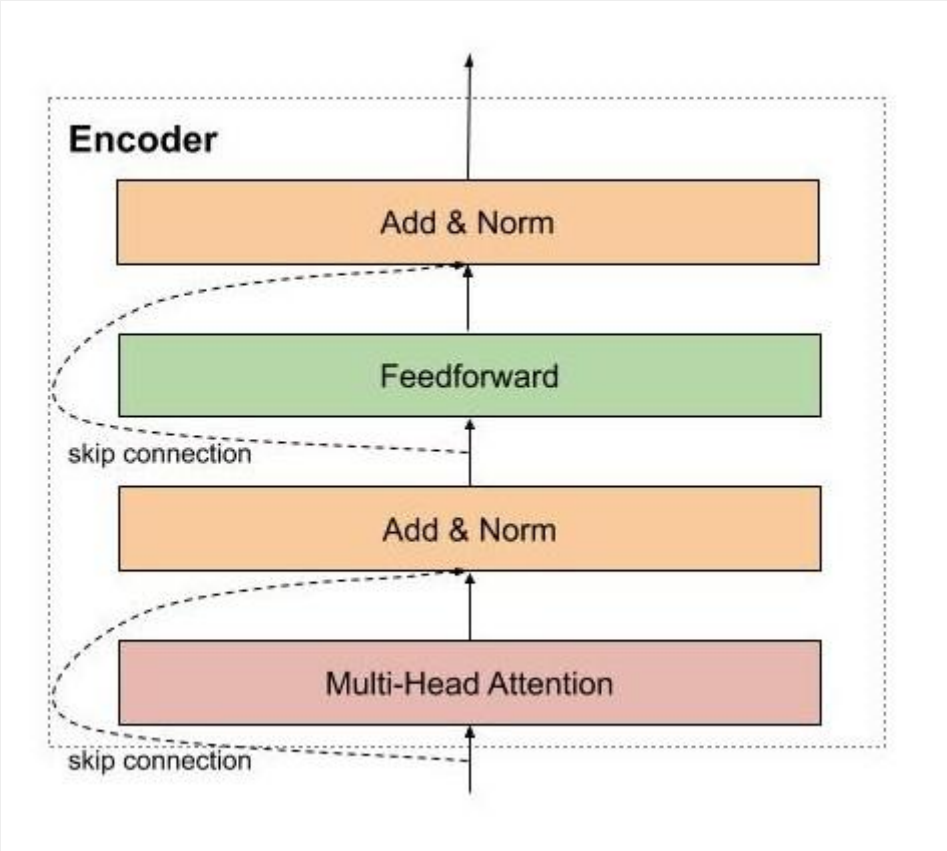
Ngoài ra, một tính chất của phương pháp tác giả đề xuất là nó cho phép mô hình dễ dàng học được mối quan hệ tương đối giữa các từ. Cụ thể, biểu diễn vị trí của từ t + offset có thể chuyển thành biểu diễn vị trí của từ t bằng một phép biến đổi tuyến tính dựa trên ma trận phép quay.

Để dễ hình dung phương pháp của tác giả đề xuất lại hoạt động tốt, các bạn có thể tưởng tượng, hàm sin, và cos, giống như là kim giây và kim phút trên đồng hồ. Với 2 kim này, chúng ta có thể biểu diễn được 3600 vị trí. Và đồng thời có thể hiểu được ngay tại sao biểu diễn của từ t + offset và từ t lại có thể dễ dàng chuyển đổi cho nhau.

Encoder

Encoder của mô hình transformer có thể bao gồm nhiều encoder layer tượng 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.

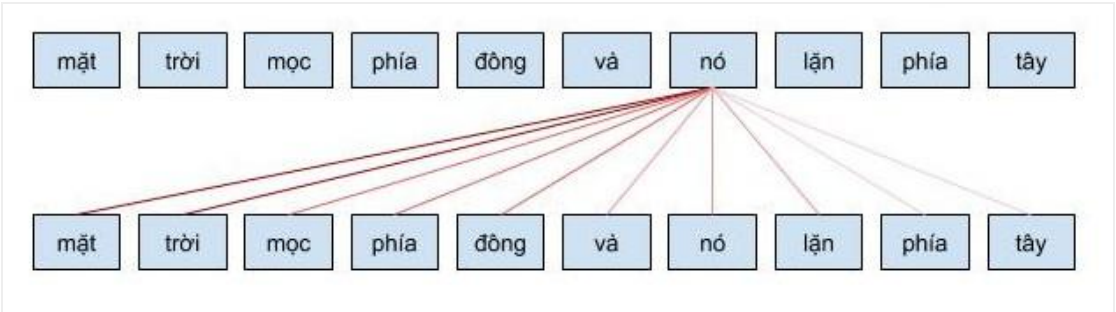
Trong 2 thành phần chính này, các bạn sẽ hứng thú nhiều hơn về multi-head attention vì đó là một layer mới được giới thiệu trong bài báo này, và chính nó tạo nên sự khác biệt giữa mô hình LSTM và mô hình Transformer mà chúng ta đang tìm hiểu.



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.

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 **nó** 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.

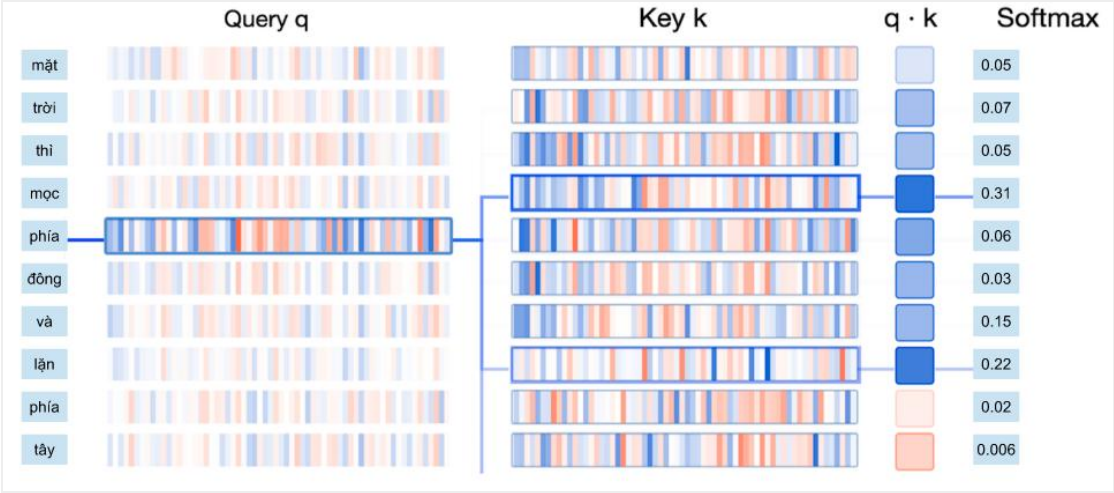


Bạn có thể tưởng tượng cơ chế self attention giống như cơ chế tìm kiếm. Với một từ cho trước, cơ chế này sẽ cho phép mô hình tìm kiếm trong cách từ còn lại, từ nào "giống" để sau đó thông tin sẽ được mã hóa dựa trên tất cả các từ trên.

Đầu tiên, với môi 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.

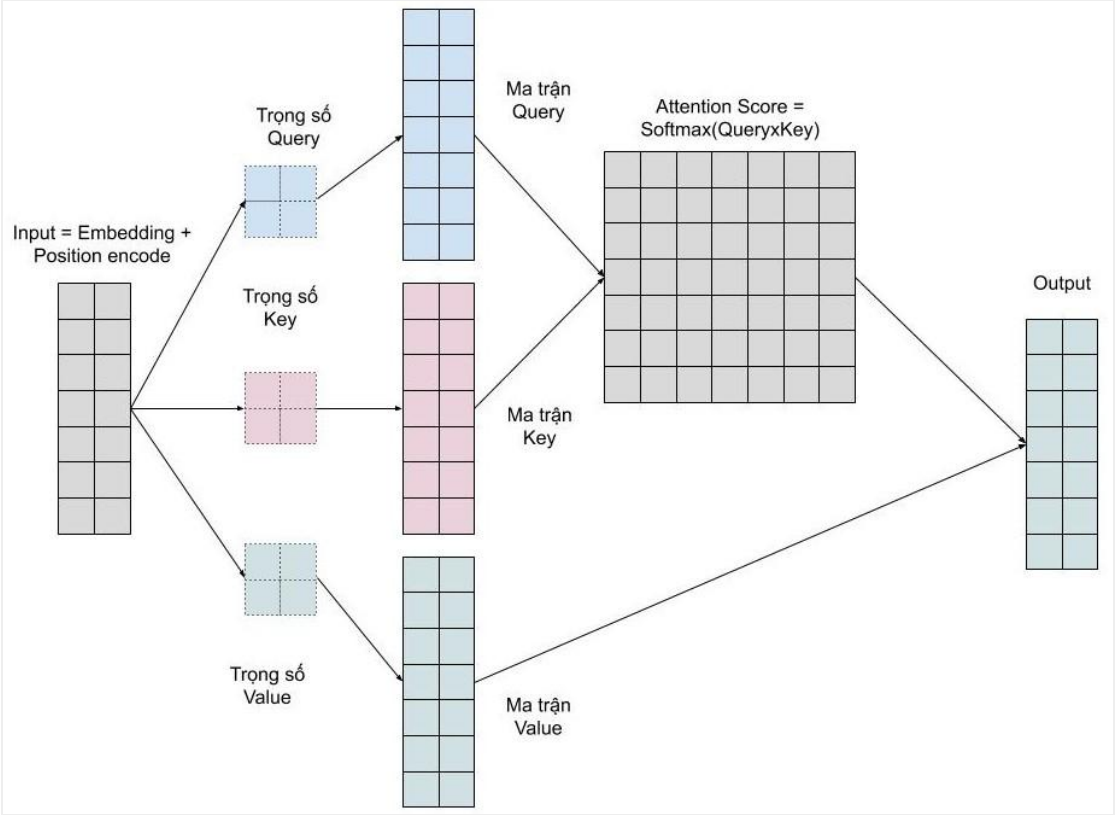
- query vector: vector dùng để chứa thông tin của từ được tìm kiếm, so sánh. Giống như là câu query của google search.
- key vector: vector dùng để biểu diễn thông tin các từ được so sánh với từ cần tìm kiếm ở trên. Ví dụ, như các trang web mà google sẽ so sánh với từ khóa mà bạn tìm kiếm.
- value vector: 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 !!!



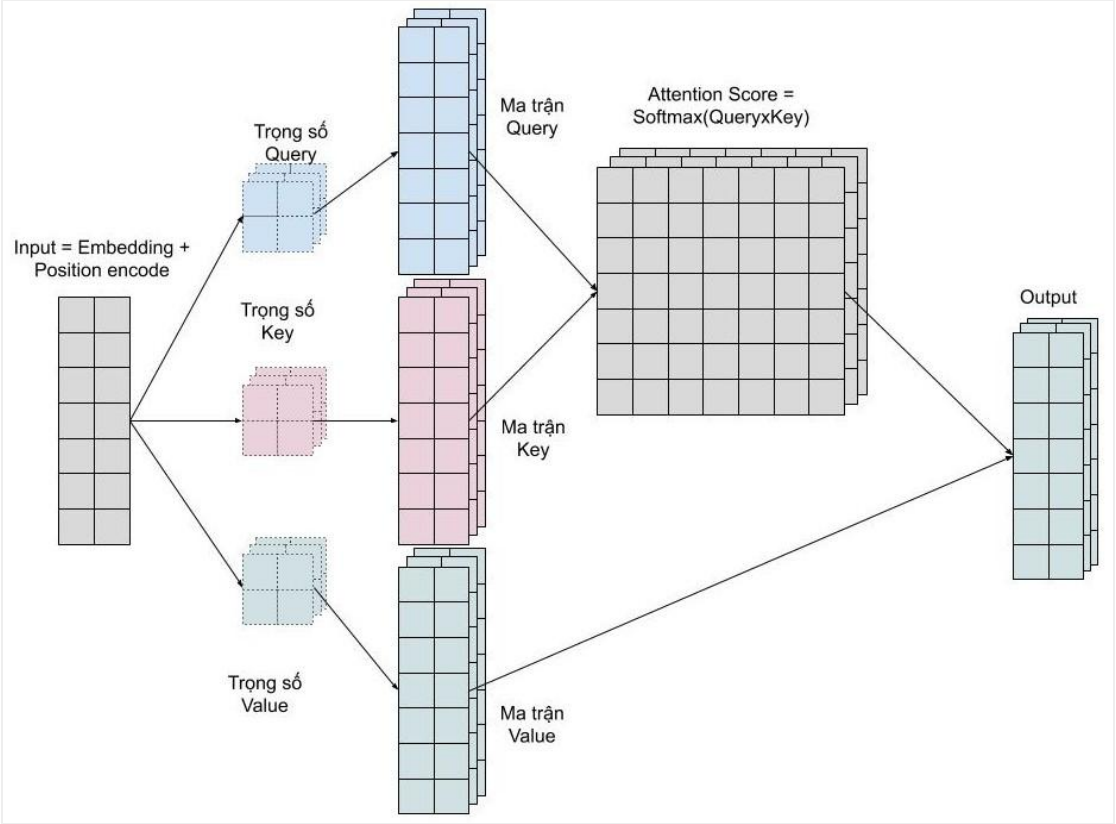
Cụ thể hơn, quá trình tính toán attention vector có thể được tóm tắt làm 3 bước như sau:

- 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.



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. 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.

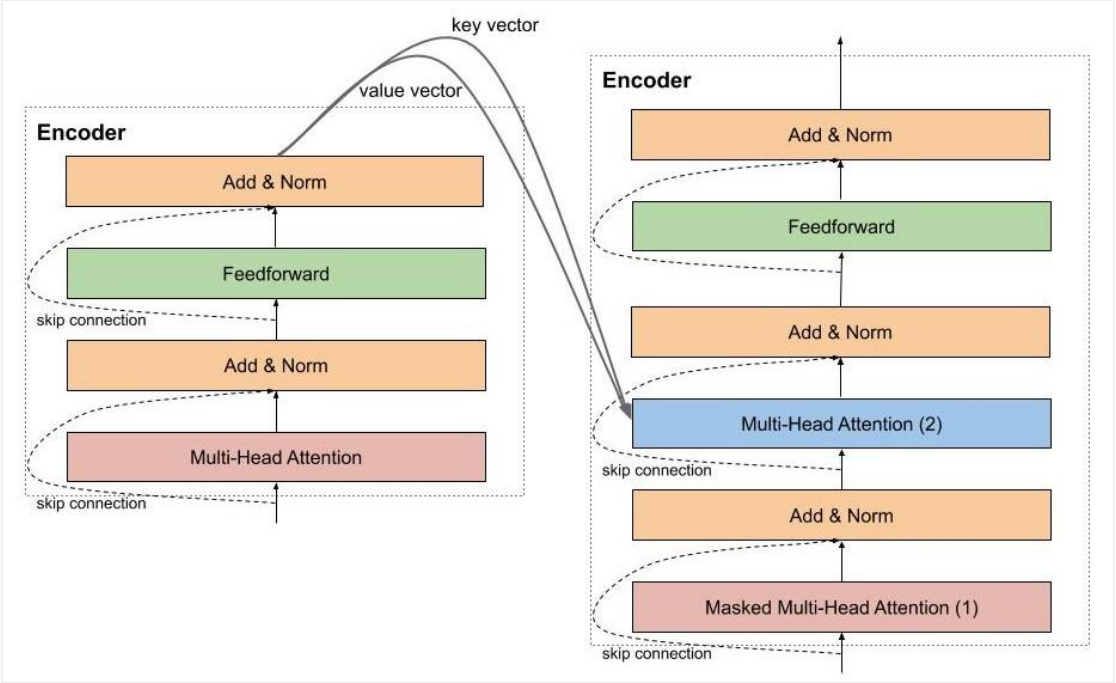
- Chú ý đến từ kế trước của một từ
- Chú ý đến từ kế sau của một từ
- Chú ý đến những từ liên quan của một từ

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 hội tụ hơn và trách mất mát thông tin trong quá trình huấn luyện mô hình, ví dụ như là thông tin của vị trí các từ được mã hóa.

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 giữa từ đang được dịch với các từ nguồn.

Final Fully Connected Layer, Softmax và Loss function

Giống như nhiều mô hình khác, chúng ta cần thêm một fully connected layer để chuyển output từ layer phía trước thành ma trận có chiều băng số từ mà các bạn cần dự đoán. Sau đó thì đến softmax để các bạn tính được xác suất của từ xuất hiện tiếp theo là bao nhiêu.

Loss function thì tất nhiên là cross-entropy mà thôi, giống như ở các mô hình phân loại khác mà các bạn đã làm quen.

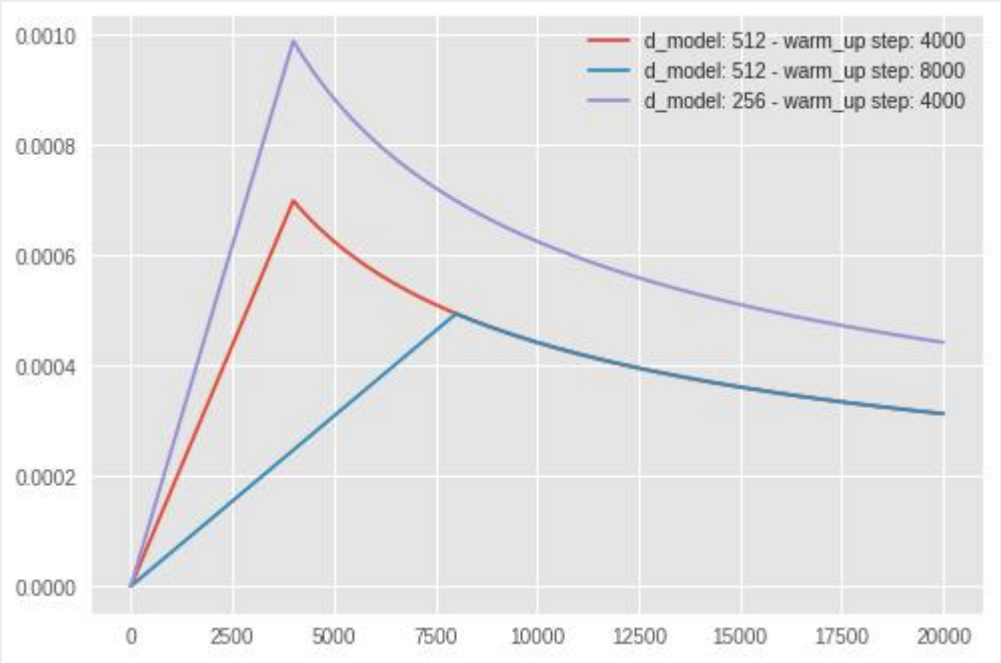
Các kỹ thuật đặc biệt để huấn luyện Transformer

Để huấn luyện mô hình transformer, các bạn cần phải biết đến 2 kỹ thuật rất thú vị này. Nếu không sử dụng kỹ thuật đầu tiên về optimizer thì mô hình transformer sẽ **không hội tụ** được luôn đấy :))

Optimizer

Để huấn luyện mô hình transformer, các bạn vẫn sử dụng Adam, tuy nhiên, learning rate cần phải được điều chỉnh trong suốt quá trình học theo công thức sau

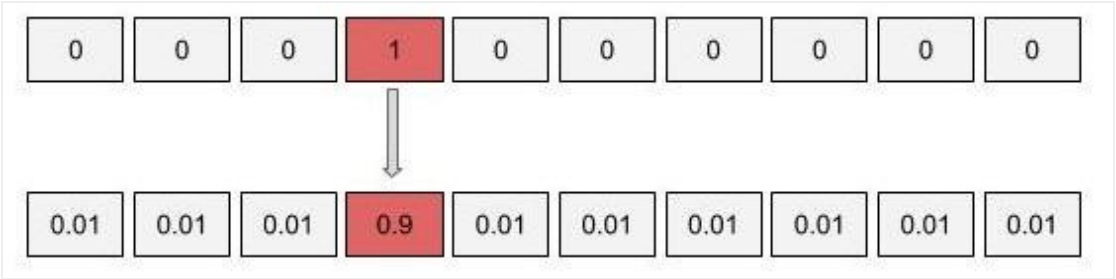
$$lr\_rate = d_{d\_model}^{-0.5} * min(step\_num^{-0.5},\ step\_num * warmup\_step^{-1.5})$$



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.

Label Smoothing

Với mô hình nhiều triệu tham số của transformer, thì việt overfit là chuyện dễ dàng xảy ra. Để hạn chế hiện tượng overfit, các bạn 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, các bạn sẽ thay đổi nhãn này một chút bằng cách phân bố một tí xác suất vào các trường hợp còn lại.



Giờ thì các bạn sẽ an tâm khi có thể để số epoch lớn mà không lo rằng mô hình sẽ overfit nặng nề.

Implementation

Với mình lý thuyết mà không có thực hành có nghĩa là vẫn chưa hiểu rõ được lý thuyết đó. Cho nên mình đã cài đặt và mô tả chi tiết các bước làm trong notebook tại đây.

Dataset song ngữ anh-việt

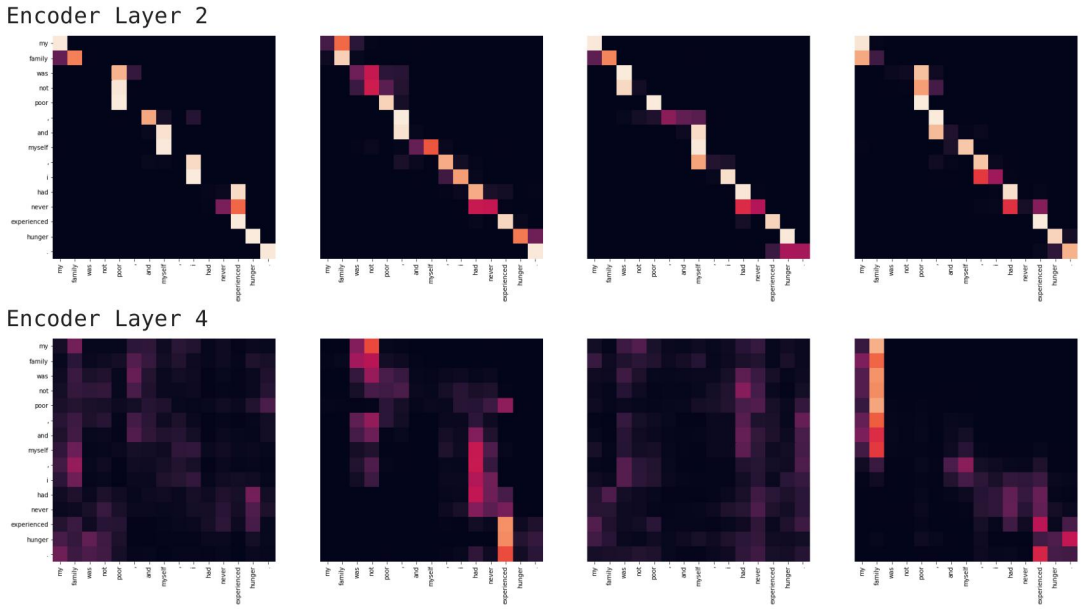
Đồng thời mình cũng cung cấp bộ dữ liệu song ngữ được thu thập trên TED bao gồm hơn 600k câu song ngữ anh-việt tại đây. Với bộ dữ liệu mình hy vọng các bạn sẽ không gặp phải khó khăn khi thử nghiệm những kiến thức mới.

Visualization

Visualize trong số của các mô hình sử dụng cơ chế attention thực sự rất thú vị. Trong mô hình transformer, chúng ta visualize tại encoder và tại decoder. Các bạn có thể visualize đồng thời tại các heads của multi-head attentions, và tại layers khác nhau.

Encoder Visualize

Các bạn có thể dùng heatmap để visualize giá trị attention, sẽ cho chúng ta biết khi encode một câu mô hình chú ý từ gì ở lân cận

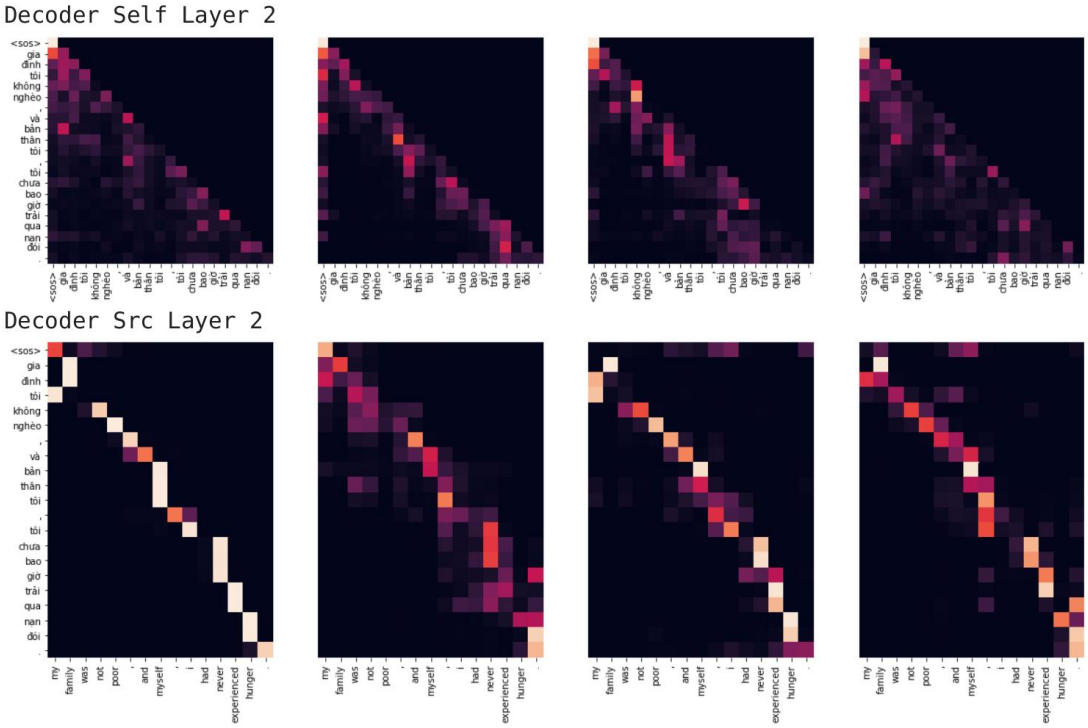


Ở đây mình visualize giá trị attention của encoder layer số 2 và 4, tại các head 0,1,2,3 (trong cài đặt các bạn có tổng cộng 6 encoder layer và 8 heads nhé). Nhìn vào các heatmaps ở trên, các bạn có thể thấy được rằng khi encode một từ mô hình sẽ nhìn vào các từ liên quan xung quanh. Ví dụ từ **family** có thể được mã hóa bằng 2 từ liên quan như **my** và **family**.

Decoder Visualize

Ở decoder, các bạn có 2 loại visualization

- self attention: giá trị attention khi mô hình decoder mã hóa câu đích lúc dịch
- src attention: giá trị attention khi mô hình decoder sử dụng câu src lúc dịch



Ở ví dụ này mình visualize decoder layer số 2, tại 4 heads 0,1,2,3. Các bạn có thể quan sát được khi encode từ **đình** mô hình sẽ nhìn vào các từ kế cạnh là **gia** và **tôi**, (và còn nhiều kiểu pattern khác nữa nhé). Còn khi dự đoán từ **tôi** mô hình sẽ nhìn vào từ **my**.