Course Natural Language Process
2021II_INT3406_1

TRANSFORMERS IN SEQUENCE TO SEQUENCE TRANSLATION

Report

Thành viên:

Nguyễn Kim Đức 19020006
Hoàng Gia Anh Đức 19020005
Trần Thanh Trà 19020461
Phạm Thanh Đạt 19020247

MỤC LỤC

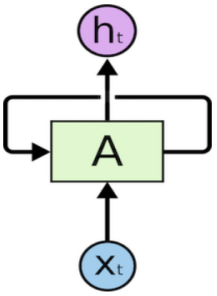
1. Đặt vấn đề:

Xử lý ngôn ngữ tự nhiên (NLP) là một lĩnh vực rộng, đa dạng với nhiều tác vụ đặc biệt. Trong các tác vụ đó, có mô hình Sequence-to-Sequence nhận input là một sequence và trả lại output cũng là một sequence. Ví dụ bài toán dịch máy, input là câu "He is a good person." và output là câu "Anh ấy là một người tốt. ". Có nhiều phương pháp đã được sử dụng để tiếp cận và xử lý, truyền thống thì có Recurrent Neural Networks (RNN), Long-short Term Memory (LSTM), gần đây thì phương pháp được sử dụng hiệu quả nhất là Transformers. Bài báo cáo này sẽ tập trung tìm hiểu về mô hình Transformers và ứng dụng trong xử lý ngôn ngữ tự nhiên.

2. Các cách tiếp cận và giải quyết:

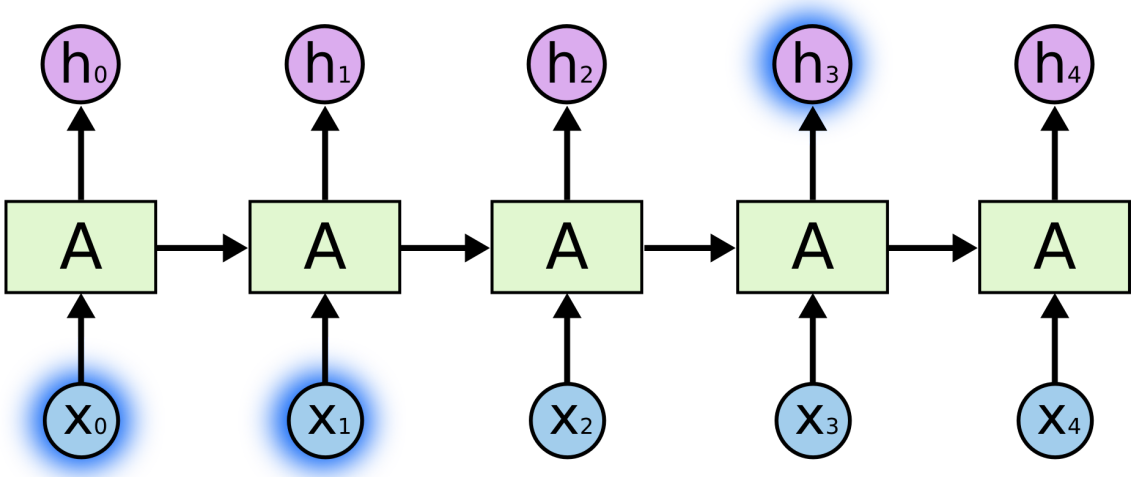
2.1. RNN:

Phương pháp truyền thống cho mô hình Seq2Seq là sử dụng RNN cho cả phần encoder (mã hóa input) và phần decoder (giải mã input đưa ra output tương ứng)



The input is represented as x_t

Mỗi phần tử của mạng RNN nhận đầu vào là x_t, đầu ra h_t, đồng thời thông tin xử lý ở mỗi bước (mỗi từ) là đầu vào cho bước xử lý sau (từ tiếp sau)



Tính chất này giúp cho RNN có thể xử lý các đầu vào liên quan đến chuỗi hoặc danh sách. Theo cách này, nếu ta muốn dịch một văn bản, đầu vào sẽ là các từ xuất hiện

trong văn bản đó. Thông tin từ đầu chuỗi có thể được truyền và sử dụng khi xử lý các từ tiếp theo.

Vấn đề về sự phục thuộc dài hạn:

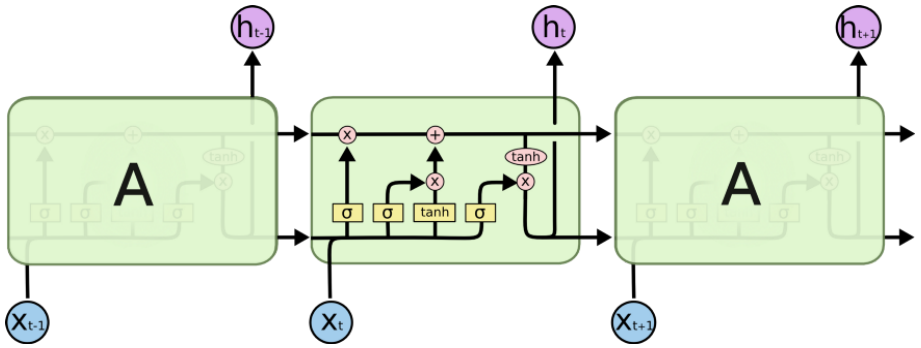
Khi phải dịch một câu quá dài mà trong đó nghĩa của một từ lại phụ thuộc vào từ được dịch trước đó rất xa, RNN không đem lại hiệu quả. Đó là hệ quả của hiện tượng Vanishing Gradient - thông tin được truyền qua càng nhiều bước thì sẽ càng bị mất mát về mặt giá trị và mạng cũng không thể cập nhật được thông tin ban đầu. Để giải quyết vấn đề này, phương pháp LSTM là phiên bản cải tiến của RNN được sinh ra.

2.2. LSTM (Long-Short Term Memory):

Trong cuộc sống hàng ngày, khi sắp xếp công việc, chúng ta thường ưu tiên những việc quan trọng. Các việc không phục vụ mục tiêu của chúng ta có thể bỏ qua được. Nhưng RNN không làm như vậy, toàn bộ đầu vào đều được xử lý và biến đổi qua mọi bước mà không quan tâm thông tin nào là quan trọng, thông tin nào là không.

LSTM sử dụng một vài phép nhân chia trong kiến trúc của mình để giải quyết vấn đề này. LSTM cell có thêm một nhánh C cho phép toàn bộ thông tin đi qua cell, giúp duy trì thông tin cho những câu dài. Với LSTM, thông tin được truyền qua một cơ chế mang tên ô nhớ. Nó có thể lựa chọn các thông tin quan trọng và quên đi các thông tin khác.

Cụ thể bên trong một phần tử LSTM được mô tả như hình dưới đây:



Mỗi phần tử nhận đầu vào là x_t (từ tiếp theo trong câu nguồn), trạng thái của phần tử trước và đầu ra của h_t. Những dữ liệu đầu vào này được tính toán và đầu ra là từ được dịch trong câu đích và trạng thái của bước vừa thực hiện.

Với việc lưu lại trạng thái ở mỗi bước, những thông tin được coi là quan trọng khi dịch từ trong một câu xác định sẽ được truyền đi xuyên suốt chuỗi LSTM.

Nhưng LSTM vẫn còn tồn tại vấn đề như đã nêu ở RNN, là khi câu quá dài thì LSTM vẫn sẽ đưa ra kết quả không chính xác. Lý do là bởi trạng thái của mỗi LSTM khi truyền đi xa vẫn sẽ bị giảm thông tin theo hàm mũ.

Một vấn đề khác của RNN và LSTM là chỉ dịch word by word mà không thể xử lý song song được.

2.3. Attention:

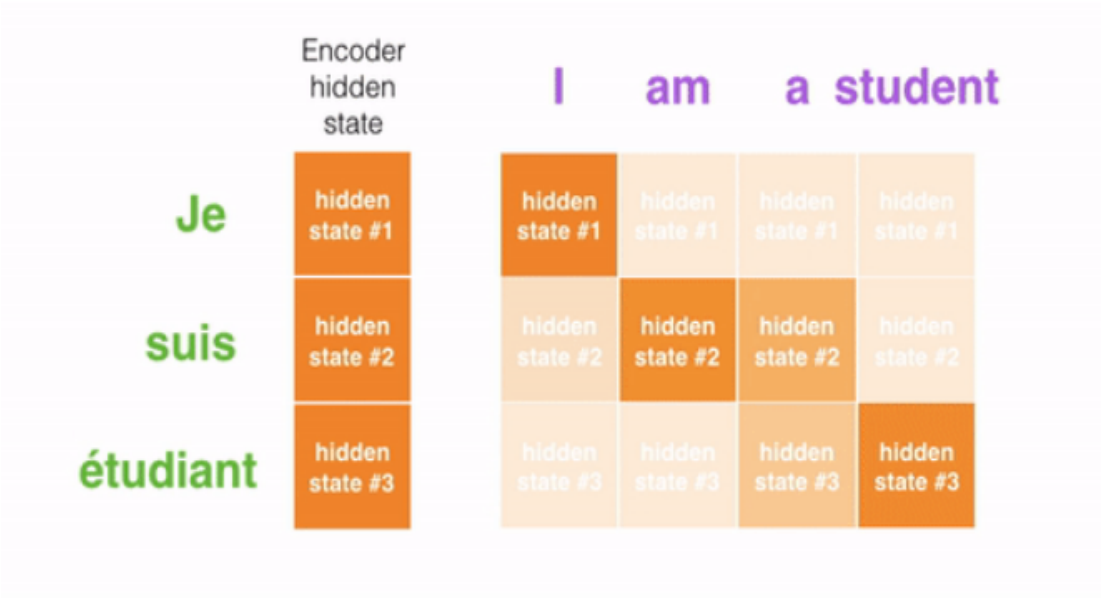
Để giải quyết vấn đề đó, các nhà nghiên cứu tạo ra một cơ chế giúp mô hình tập trung chú ý vào các từ cụ thể. Điều này khá phù hợp với cách làm việc của con người. Cơ chế này mang tên Attention.

Khi dịch một câu, chúng ta sẽ chú ý tới từng từ đang được dịch. Vậy thì neural network có thể đạt được điều này bằng cơ chế 'attention', tập trung vào một phần thông tin đưa ra trong câu. Ở mỗi bước khác nhau, nó sẽ tập trung vào nhiều vị trí khác nhau của các RNN khác

Với RNN, thay vì mã hóa toàn bộ câu vào một trạng thái ẩn, mỗi từ có một trạng thái ẩn riêng. Sau khi mã hóa, tất cả chúng được truyền vào decoder.

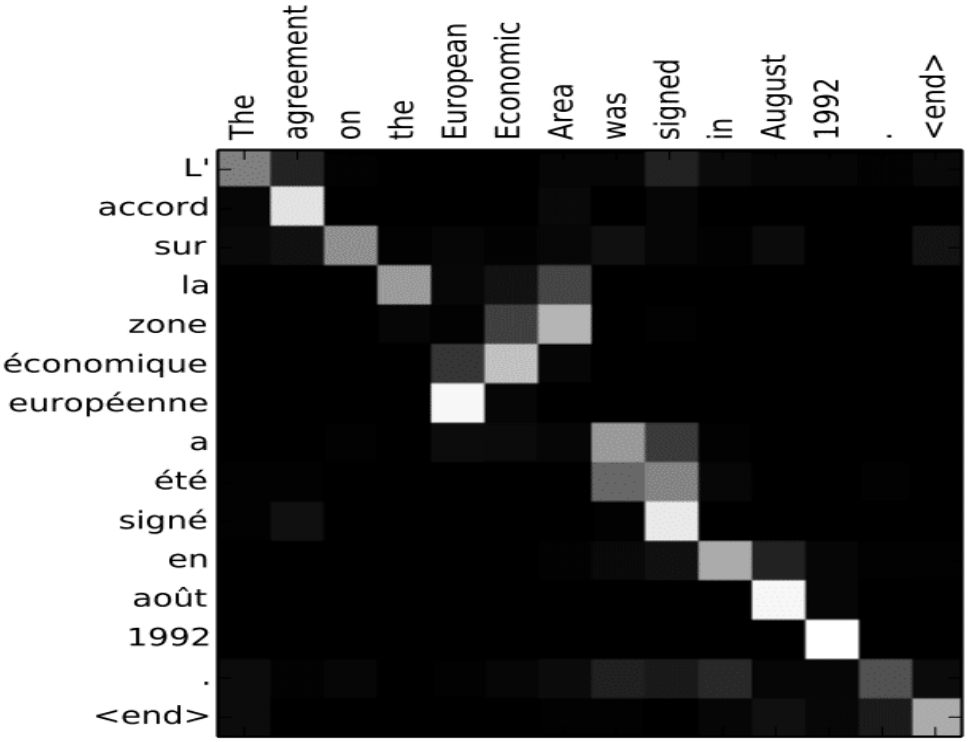
Ý tưởng đằng sau kỹ thuật này là thông tin liên quan có thể xuất hiện ở bất kỳ vị trí nào trong câu, không phụ thuộc vào khoảng cách. Do đó để giải mã một cách chính xác, nó cần tính đến tất cả các từ và sử dụng attention. Hình ở dưới thể hiện cách decoder tập trung vào các từ với các mức độ chú ý khác nhau.

Ví dụ khi dịch câu "**Je suis étudiant"** sang tiếng Anh, mô hình cần nhìn vào các từ khác nhau để dịch nó.



Màu càng đậm thì mức độ chú ý càng lớn.

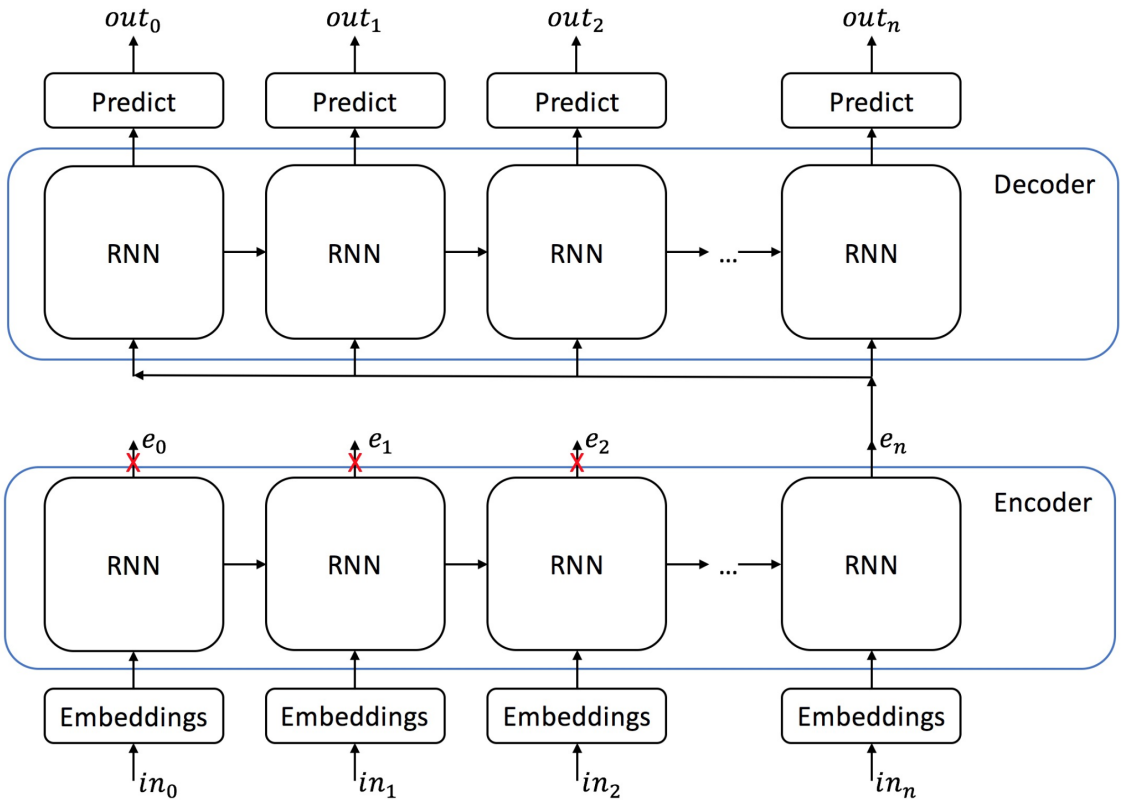
Hình phía dưới thể hiện mức độ chú ý của mô hình khi dịch “L’accord sur la zone économique européenne a été signé en août 1992.” từ tiếng Pháp sang tiếng Anh. Các ô càng sáng biểu thị rằng 1 từ A từ ngôn ngữ E1 "chú ý" hay có tương quan hơn (correlation) với 1 từ B từ ngôn ngữ E2



Cụ thể về Attention

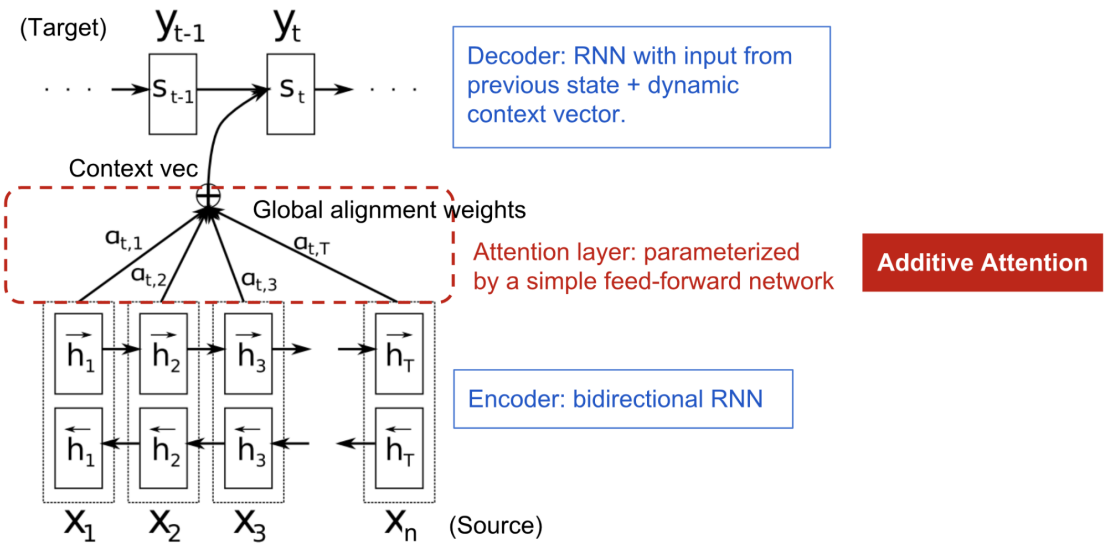
Attention sử dụng tất cả các output của từng cell qua từng timestep, kết hợp với hidden state của từng cell để "tổng hợp" ra 1 context vector (attention vector) và dùng nó làm đầu vào cho từng cell trong Decoder

Sau đây là minh họa cơ chế RNN khi không áp dụng Attention:



Mô hình encoder phải nén tất cả thông tin của một câu lại thành một vector biểu diễn duy nhất, chứa toàn bộ thông tin cần thiết để mô hình decoder có thể dịch thành câu đích. Vấn đề nằm ở chỗ, những câu dài sẽ không được dịch chính xác vì thông tin không được lưu trữ đủ trong một vector biểu diễn duy nhất.

Và khi áp dụng Attention:

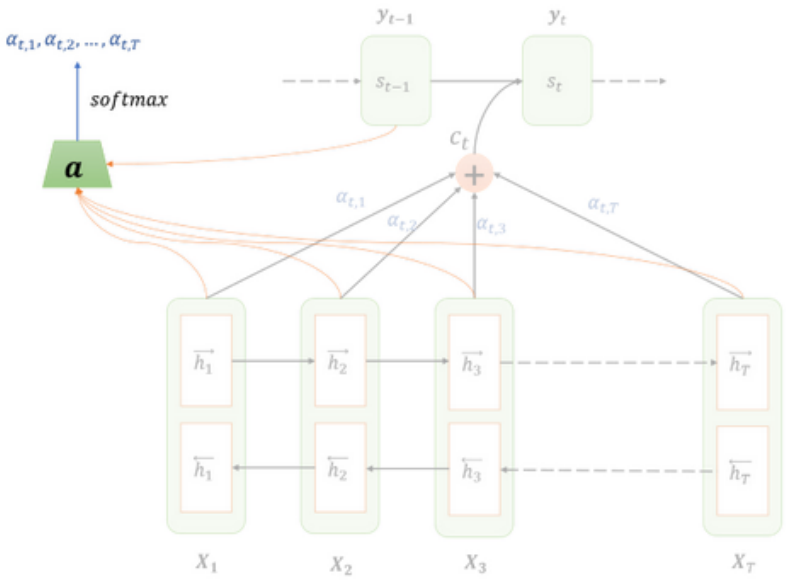


Ta sử dụng vector context để tổng hợp sử dụng thông tin của các vector x tại các vị trí khác nhau khi cần mà không mất mát thông tin.

$$\alpha_{t'} = NeuralNet([s_{t-1}, h_{t'}]),\ t' = 1 \ldots T_x$$

$$context = \sum_{t'=1}^{T_x} \alpha(t')h(t')$$

Calculating attention weights and creating the context vector using those attention values with encoder state outputs

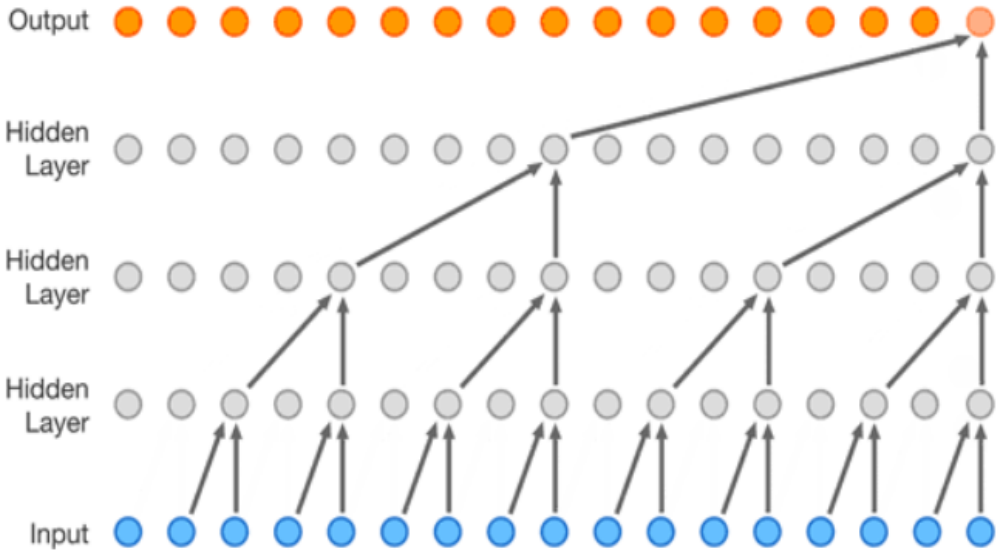


Mặc dù Attention đã giải quyết được vấn đề phụ thuộc giữa các từ trong RNN, nó vẫn chưa giúp tính toán song song. Đây là bất lợi đáng kể khi làm việc với các tập dữ liệu lớn và không tận dụng được tính toán song song của GPU.

2.4. Convolutional Neural Network

Mạng tích chập CNN là giải pháp cho tính toán song song. Một số mạng nổi tiếng trong biến đổi chuỗi sử dụng CNN có thể kể đến như Wavenet và Bytenet.

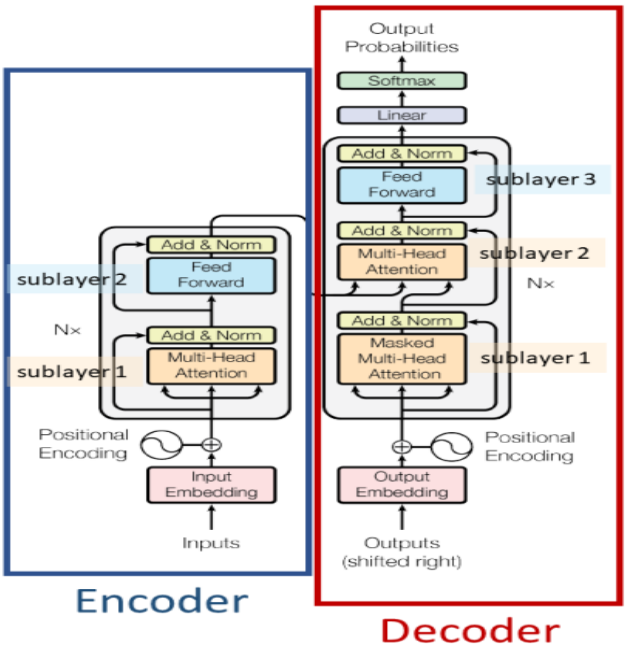
Lý do CNN có thể tính toán song song là các từ được xử lý cùng lúc và không cần chờ đợi nhau. Không chỉ có vậy, khoảng cách giữa một từ đầu ra với một từ đầu vào là log(N), thay vì N như trong RNN. Ta có thể thấy rõ trong mô hình của Wavenet tại hình dưới đây.



Wavenet có mô hình là Convolutional Neural Network (CNN).

Vấn đề của CNN là nó không giải quyết bài toán phụ thuộc trong câu. Đó là lý do Transformer được tạo ra, kiến trúc này kết hợp CNN với Attention.

3. Phân tích và xây dựng mô hình Transformer

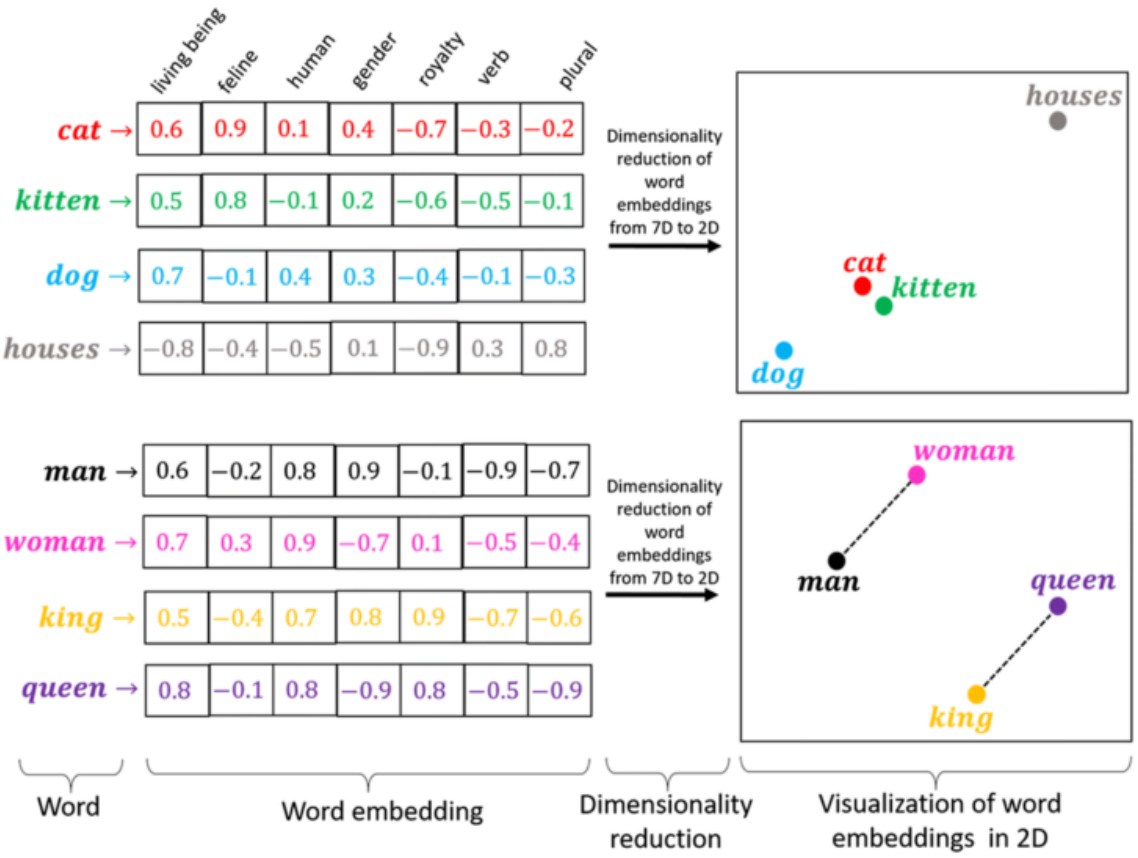


Hình 2.1. Kiến trúc mô hình Transformers

3.1. Word Embedding

Trước hết, về cơ chế word embedding - một cơ chế sử dụng vector được tính toán để biểu diễn quan hệ tương đồng giữa các từ.

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.



3.2. Positional Encoding

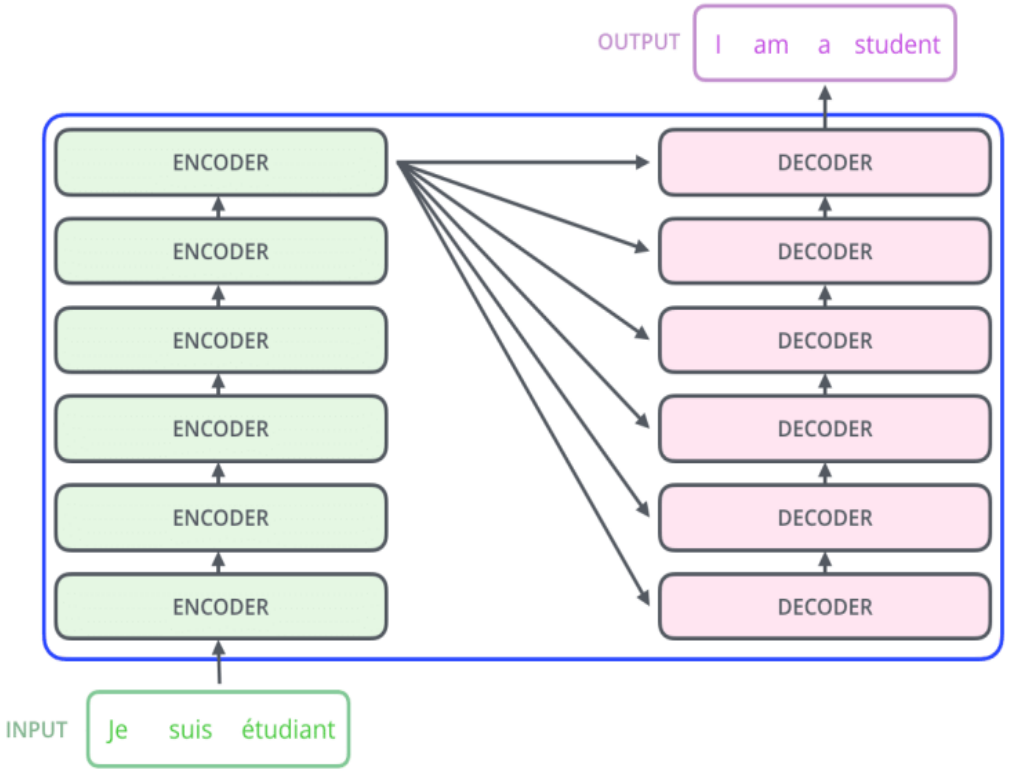
Một chi tiết quan trọng khác của Transformer là mã hóa vị trí (positional encoding). Encoder của Transformer không có sự lặp lại tuần tự như RNN, chúng ta phải đưa thông tin về vị trí vào vectơ đầu vào. Các tác giả của Transformer thực hiện một mánh rất thông minh sử dụng sin và cos.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

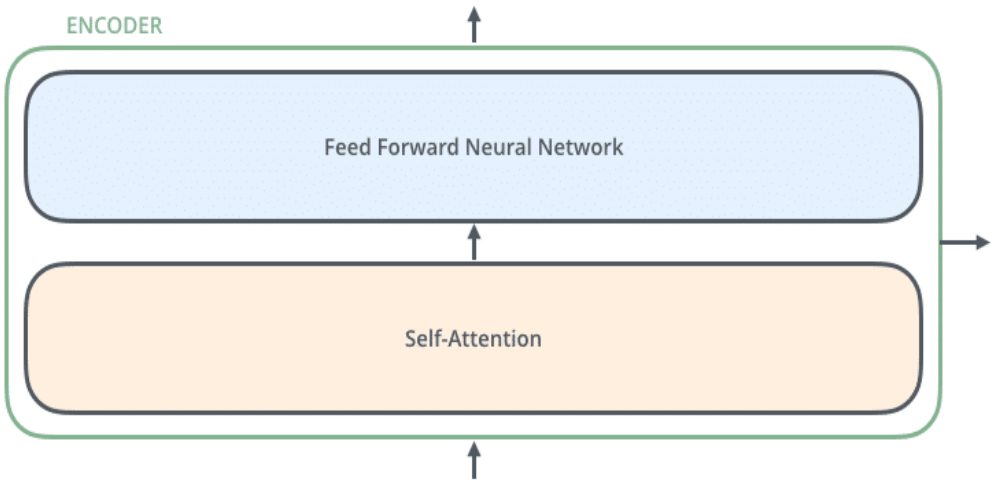
Trong đó: pos là vị trí của từ trong câu, PE là giá trị phần tử thứ i trong embeddings có độ dài d$model$ Chúng ta không đi sâu vào khía cạnh toán học của kỹ thuật này, nhưng về cơ bản, với các bước thời gian lẻ, ta dùng hàm cos, với bước thời gian chẵn, ta dùng hàm sin. Sau đó ta cộng các vectơ vào embedding của đầu vào. Việc này giúp mô hình nhận biết được vị trí của mỗi véctơ. Kết hợp của hàm sin và cos có những thuộc tính tuyến tính mà mô hình có thể dễ dàng học được.

Như đã giới thiệu ở trên, Transformer kết hợp sức mạnh của CNN và Attention. Cụ thể hơn, kiến trúc này sử dụng self-attention. Trong kiến trúc đang xét, Transformer chứa 6 encoder và 6 decoder.

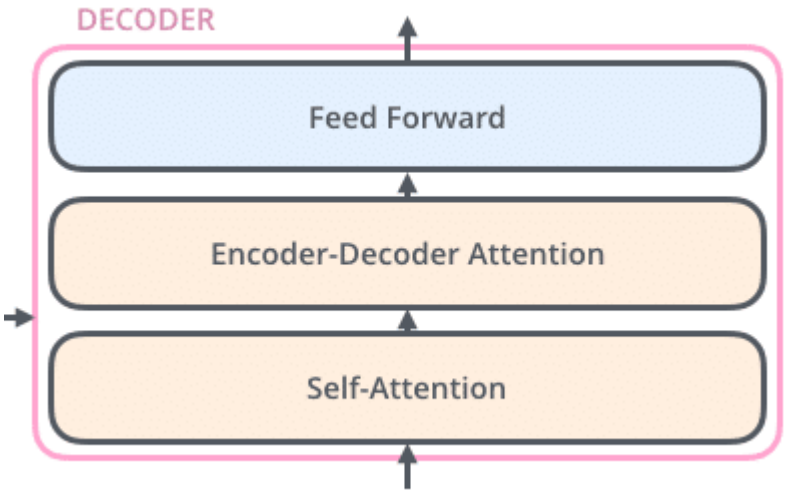


3.3. Encoder

Các encoder đều rất giống nhau, có cùng kiến trúc. Mỗi encoder chứa hai lớp: Self-attention và mạng truyền thẳng (FNN).



Self-attention giúp encoder nhìn vào các từ khác trong lúc mã hóa một từ cụ thể. Đầu ra của self-attention được truyền vào một mạng nơ ron truyền thẳng (feed-forward). Tất cả các vị trí khác nhau đều sử dụng chung một mạng truyền thẳng. Các decoder cũng có kiến trúc giống như vậy nhưng giữa chúng có một lớp attention để nó có thể tập trung vào các phần liên quan của đầu vào.



3.3.1 Self-Attention

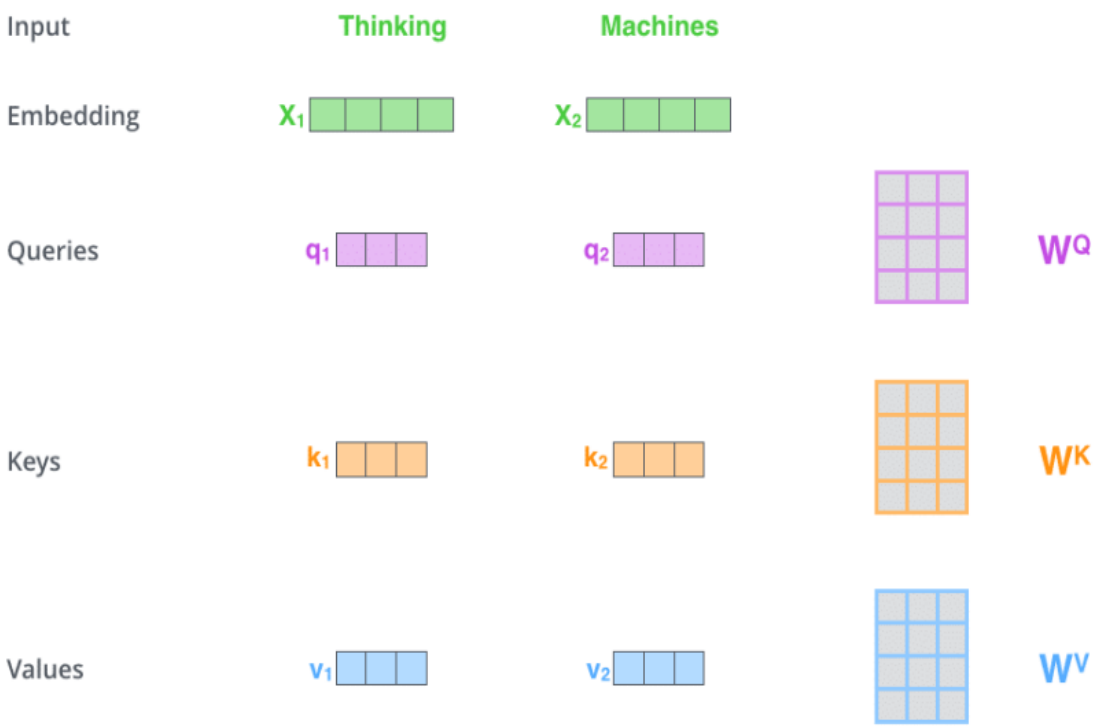
Bước đầu tiên để tính self-attention là tạo ra bộ 3 vectơ từ các vectơ đầu vào của encoder. Tại encoder đầu tiên, vectơ đầu vào là word embedding của từ. Như vậy với mỗi từ, ta sẽ có 3 vectơ

Query - vector dùng để chứa thông tin của từ được tìm kiếm, so sánh

Key - vector dùng để biểu diễn thông tin các từ được so sánh với từ cần tìm kiếm ở trên

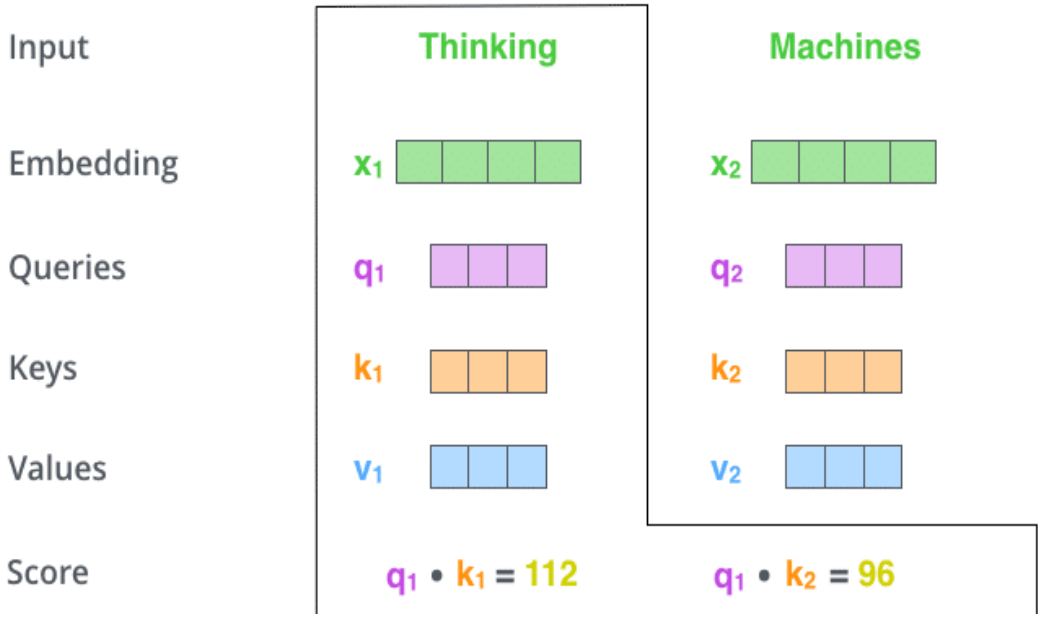
Value - vector biểu diễn nội dung, ý nghĩa của các từ.

Các vectơ này được tạo nên bởi phép nhân ma trận giữa véctơ đầu vào và 3 ma trận trọng số chúng ta sử dụng trong quá trình huấn luyện. 3 vectơ này đóng vai trò khác nhau và đều quan trọng đối với attention.

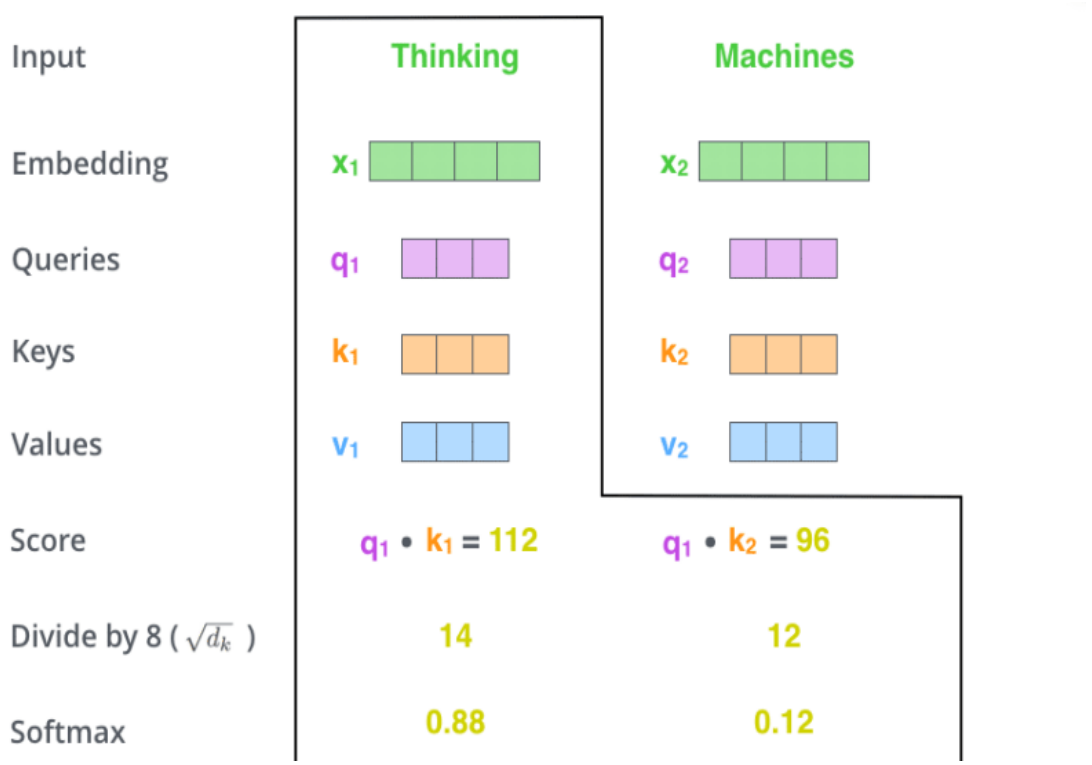


Bước thứ hai là tính điểm. Với mỗi từ, ta cần tính điểm của các từ khác trong câu đối với từ này. Giá trị này giúp quyết định từ nào cần chú ý và chú ý bao nhiêu khi mã hóa một từ.

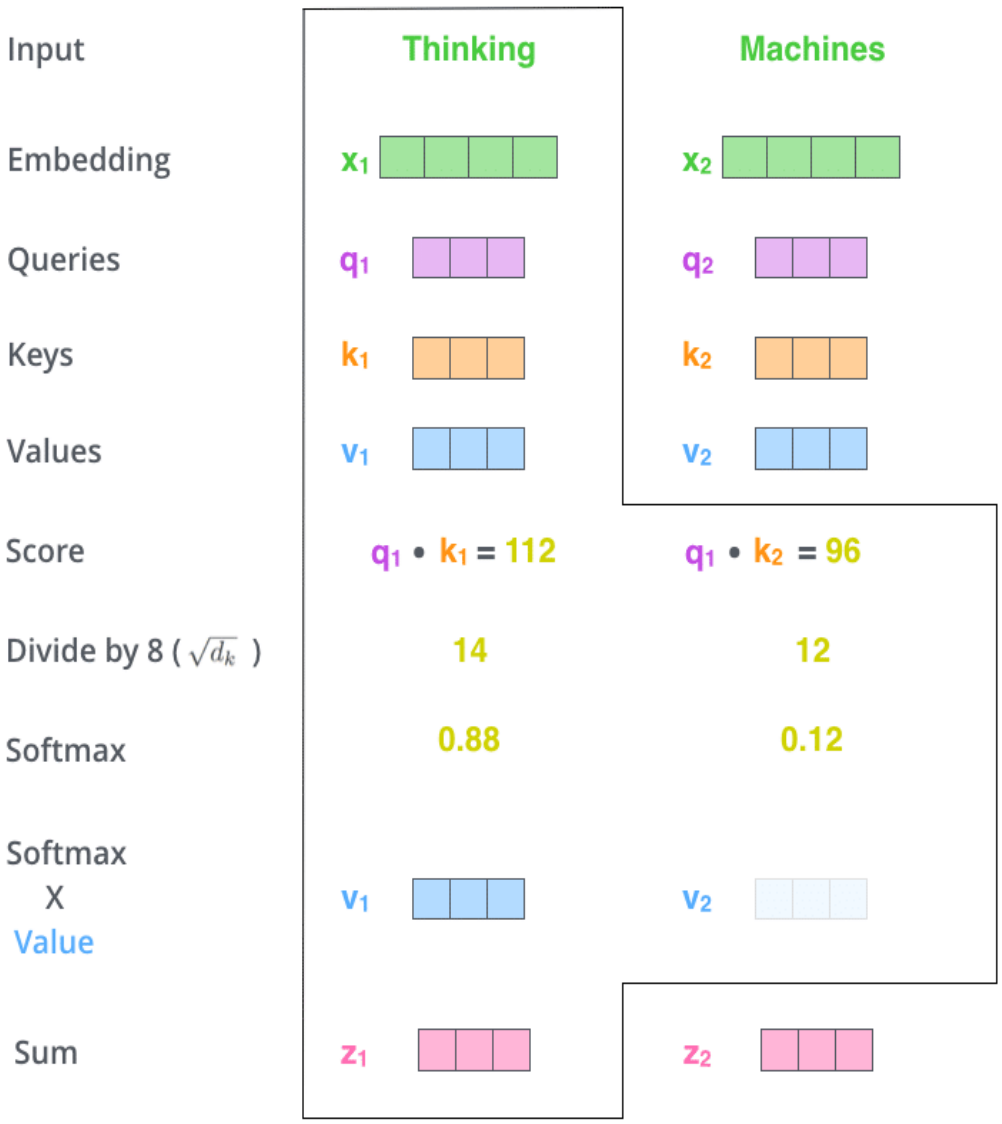
Điểm được tính bằng tích vô hướng giữa véctơ *Query* của từ đang xét với lần lượt các vectơ *Key* của các từ trong câu. Ví dụ, khi ta tính self-attention trên từ có vị trí 1, điểm của nó với chính nó là q1.k1, điểm của nó với từ thứ hai là q1.k2, v..v..



Bước tiếp theo là chuẩn hóa điểm. Trong bài báo gốc, điểm được chia cho 8 (căn bậc 2 của 64 – số chiều của vectơ *Key*). Điều này giúp cho độ dốc trở nên ổn định hơn. Tiếp theo, giá trị này được truyền qua hàm softmax để đảm bảo các giá trị điểm đều dương và có tổng không vượt quá 1.



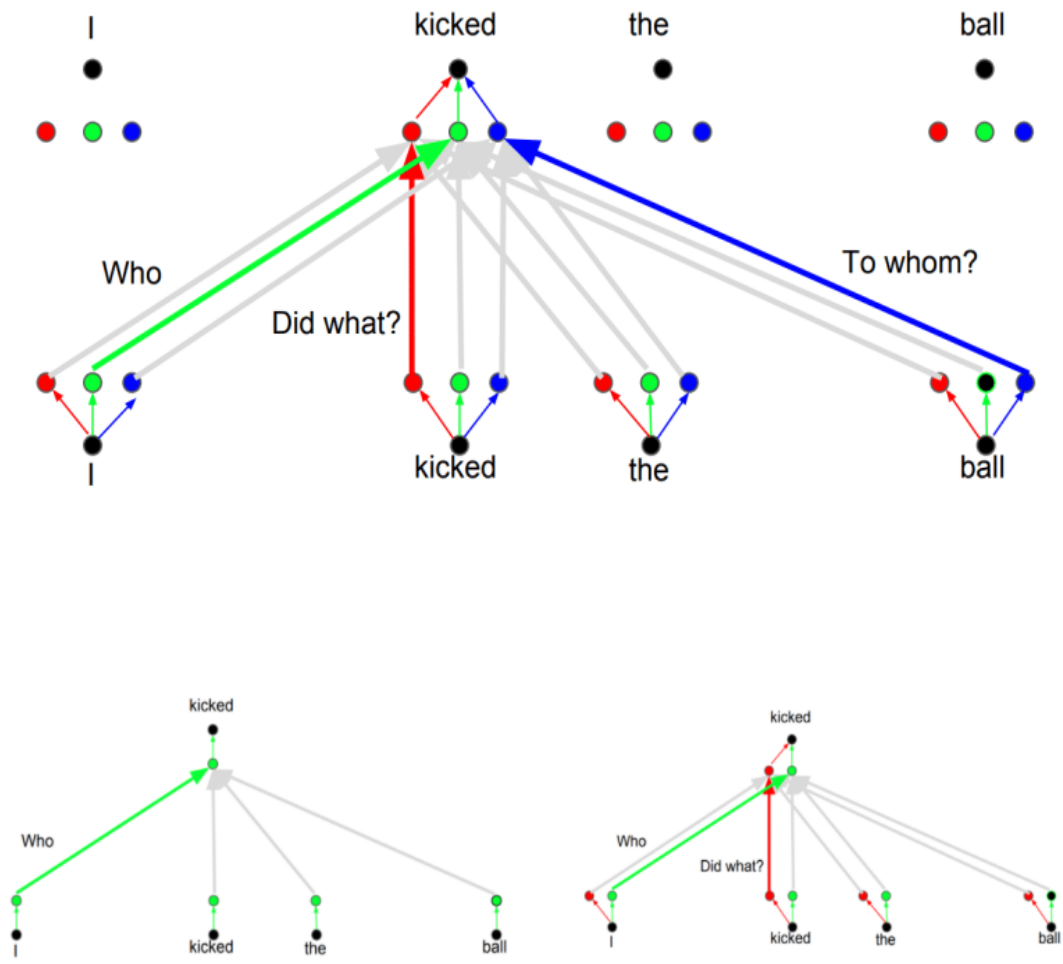
Bước tiếp theo là nhân vectơ *Value* với mỗi giá trị điểm đã tính phía trên rồi cộng lại với nhau. Ý đồ của việc này là bảo toàn giá trị vectơ của các từ cần được chú ý và loại bỏ vectơ của các từ không liên quan (bằng cách nhân nó với một số rất nhỏ, ví dụ như 0.001).



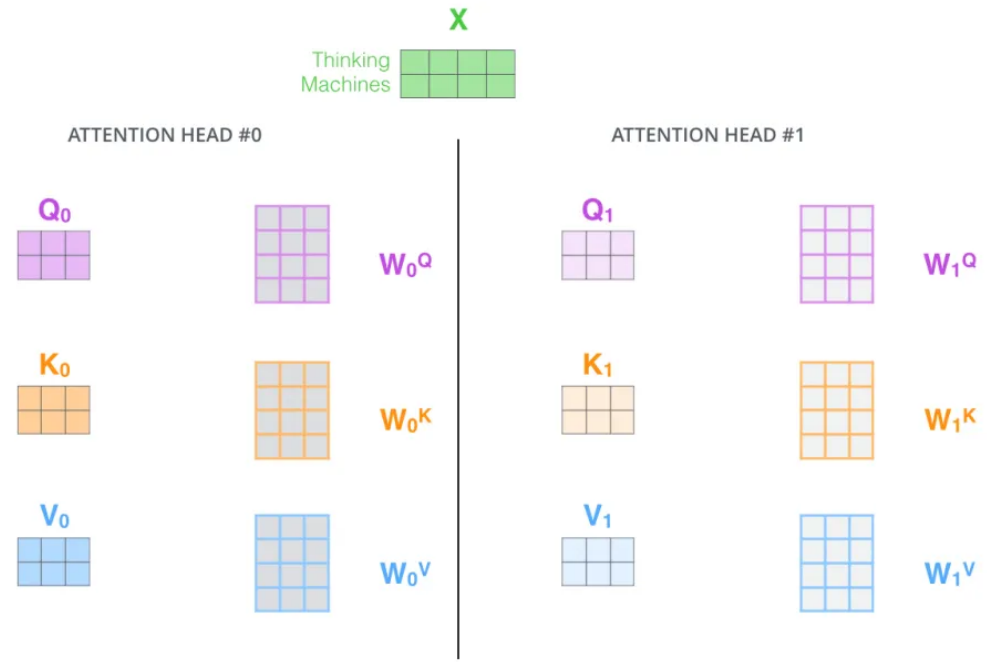
3.3.2. Multihead attention

Trên thực tế, có một vài chi tiết nữa giúp cho kiến trúc này hoạt động hiệu quả hơn. Ví dụ như thay vì sử dụng một bộ giá trị self-attention, mô hình có thể sử dụng nhiều bộ QKV khác nhau. Kỹ thuật này mang tên Multihead attention.

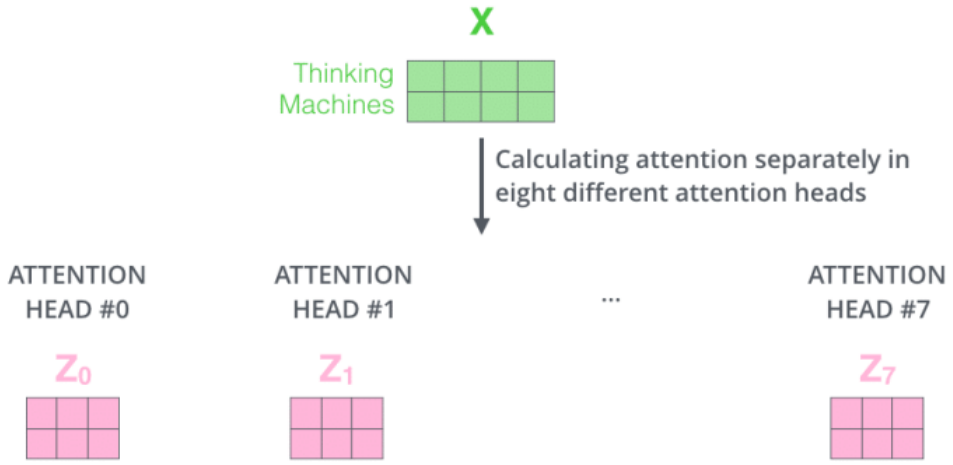
Ý tưởng đằng sau kỹ thuật này là một từ có thể có nhiều nghĩa hoặc nhiều cách thể hiện khác nhau khi dịch ra một ngôn ngữ khác. Ngoài ra, mức độ liên hệ giữa các từ có thể thay đổi khi ta quan tâm đến các khía cạnh khác nhau của một câu nói.



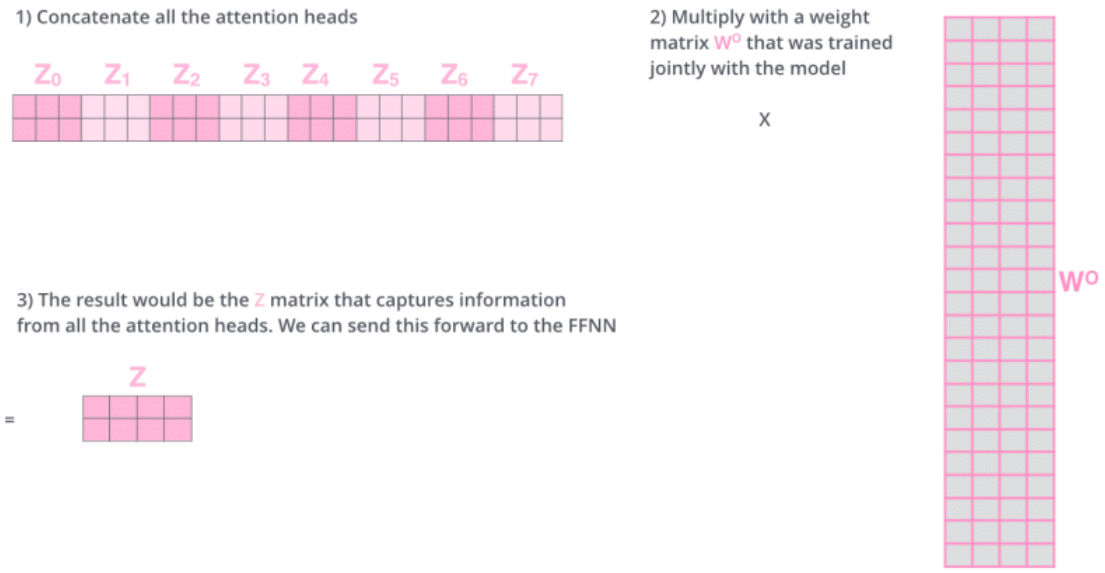
Nó mang lại cho lớp attention nhiều không gian con để biểu diễn. Với multi-headed attention chúng ta không chỉ có một mà nhiều bộ ma trận trọng số Query/Key/Value (Transformer thường sử dụng tám đầu attention, do đó ta sẽ có 8 bộ cho mỗi encoder hoặc decoder) . Mỗi bộ được khởi tạo ngẫu nhiên. Sau đó, kết thúc huấn luyện, mỗi bộ được dùng để phản ánh embedding đầu vào (hoặc vector từ các encoder/decoder phía dưới) trong một không gian con riêng biệt



Nếu ta thực hiện self-attention như đã vạch ra bên trên, với 8 lần tính với các ma trận khác nhau, ta có 8 ma trận Z khác nhau.

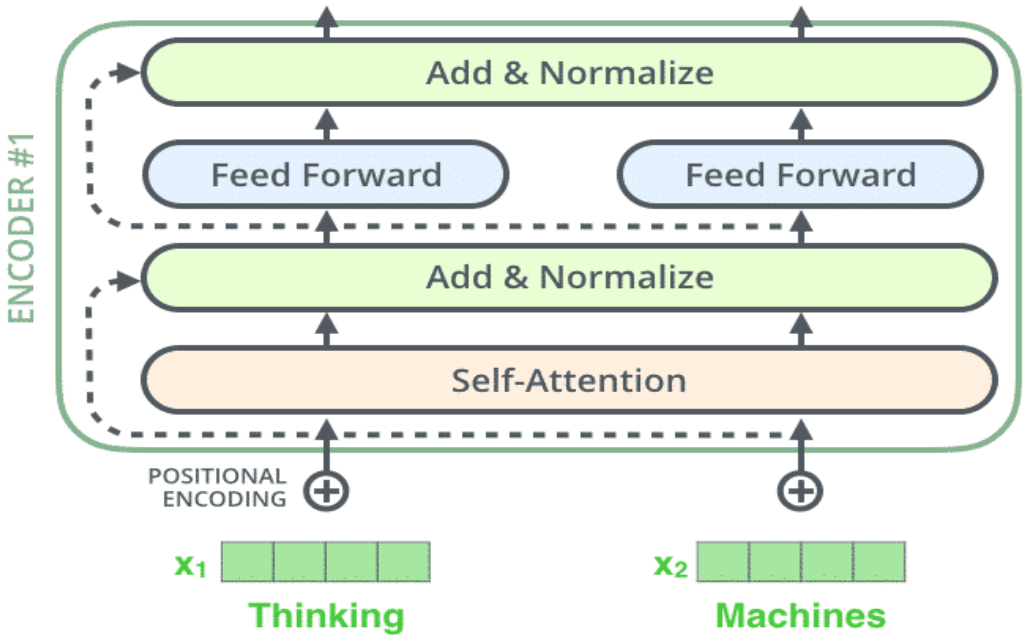


Mạng truyền thẳng không phù hợp để nhận vào 8 ma trận, thay vào đó nó cần 1 ma trận duy nhất (mỗi từ một véc tơ). Do đó, ta cần biến đổi 8 ma trận về 1 ma trận duy nhất bằng cách nối các ma trận lại và nhân chúng với một ma trận trọng số được bổ sung WO.



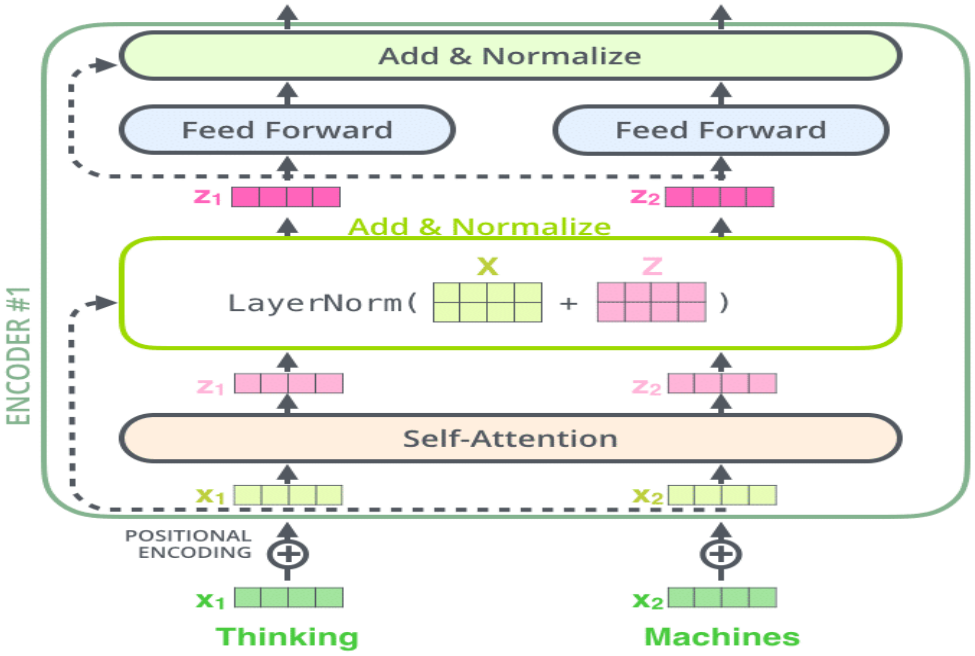
3.3.3. Residuals

Một chi tiết nữa trong kiến trúc của encoder cần phải nhắc đến là trong mỗi lớp con (self-attention, mạng truyền thẳng) của encoder có kết nối residual xung quanh chúng, và sau đó là bước chuẩn hóa lớp.

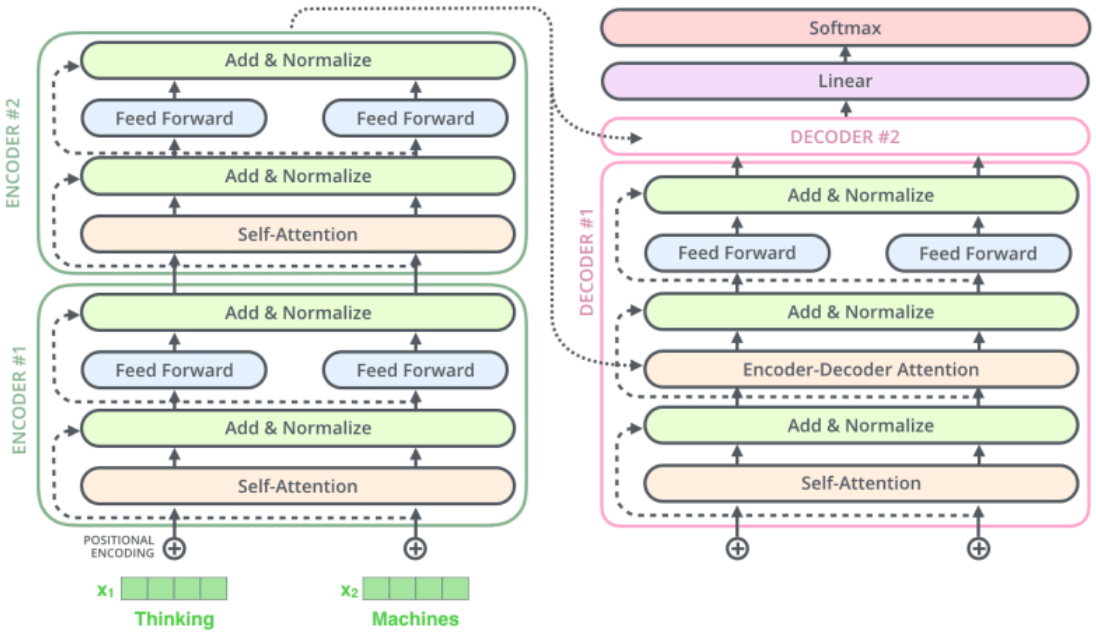


Trong mô hình, mỗi sub-layer đều là một residual block. Giống như residual blocks trong Computer Vision, skip connections trong Transformers cho phép thông tin đi qua sub-layer trực tiếp. Thông tin này (x) được cộng với attention (z) của nó và thực hiện

Layer Normalization. Nếu trực quan hóa vector và phép chuẩn hóa lớp liên quan đến self-attention, nó sẽ trông như sau:

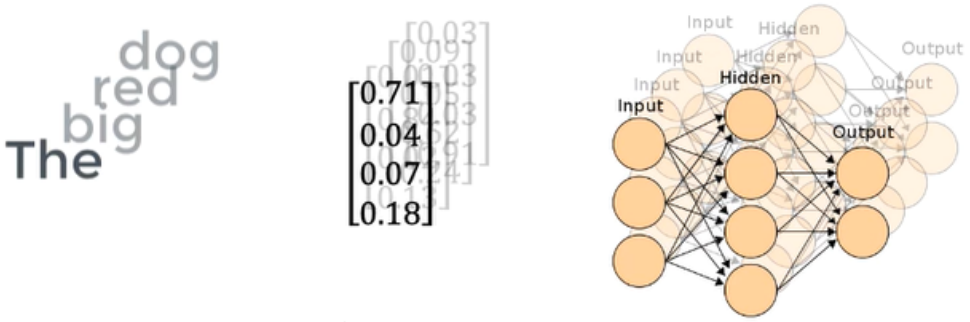


Điều này cũng được thực hiện ở decoder. Nếu ta coi Transformer là 2 ngăn xếp encoder và decoder, nó sẽ trông như sau:



3.3.4. Feed Forward

Sau khi được Normalize, các vectors z được đưa qua mạng fully connected trước khi đẩy qua Decoder. Vì các vectors này không phụ thuộc vào nhau nên có thể tận dụng được tính toán song song cho cả câu.



Tính toán song song cho câu

3.4. Decoder

Decoder cơ bản có các thành phần tương tự như encoder nhưng có một số điểm khác biệt như sau.

3.4.1. Masked Multi Head Attention và Encoder-decoder attention.

Masked Multi Head Attention là multi head attention, có chức năng dùng để encode các từ câu câu đích trong quá trình dịch. Lúc cài đặt cần phải che đi các từ ở tương lai chưa được mô hình dịch đến, bằng cách nhân với một vector chứa các giá trị 0,1.

Kết quả của encoder trên cùng được chuyển thành một bộ các véc tơ attention K và V. Chúng được sử dụng bởi mỗi decoder trong lớp "encoder-decoder attention" để giúp decoder tập trung vào phần quan trọng trong chuỗi đầu vào và phần đã dịch.

3.4.2. Final Fully Connected Layer, Softmax và Loss function

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

Loss function là cross-entropy, giống như ở các mô hình phân loại khác đã sử dụng.

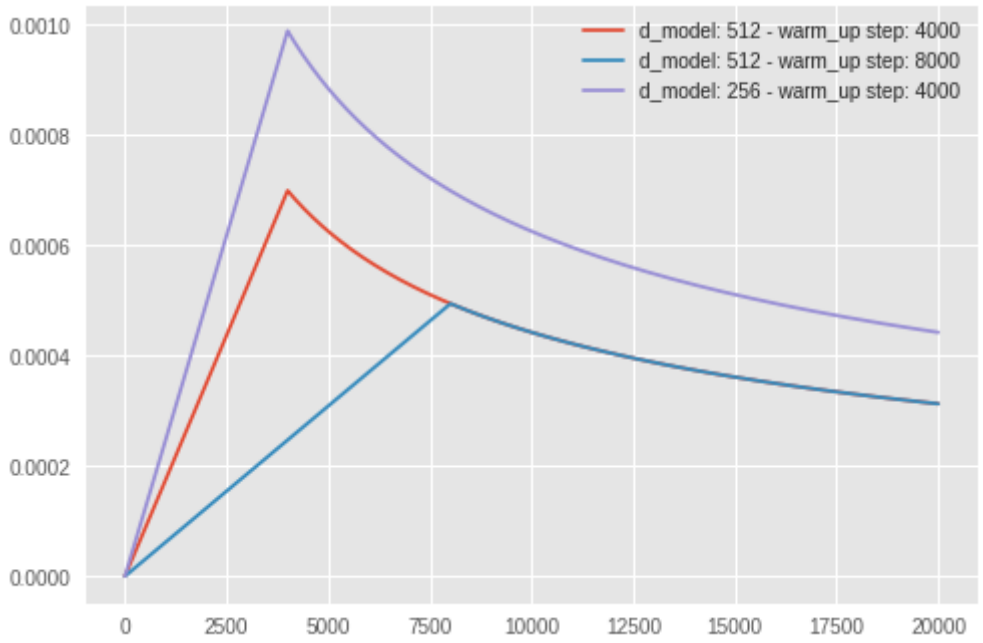
3.4.3. Các kỹ thuật đặc biệt để huấn luyện Transformer

Để huấn luyện mô hình transformer, cần optimizer và Label Smoothing để mô hình transformer hội tụ được.

- Optimizer

Để huấn luyện mô hình transformer, ta sử dụng công thức Adam, learning rate cần phải được điều chỉnh trong suốt quá trình học theo công thức sau:

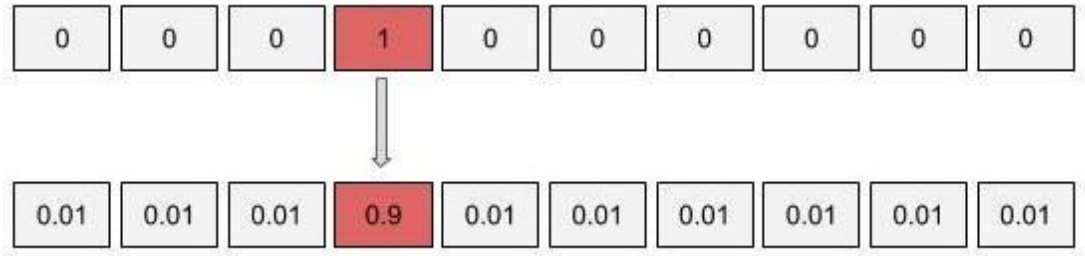
$$lr\_rate = d_{d\_model}^{-0.5} * min(step\_num^{-0.5}, step\_num * warmup\_steps^{-1.5})$$



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'. Sau đó learning rate lại giảm dần, để mô hình hội tụ.

- Label Smoothing

Với mô hình nhiều triệu tham số của transformer, để hạn chế hiện tượng overfit, có thể sử dụng kỹ thuật label smoothing. Phạt mô hình khi nó quá confident (tự tin) vào việc dự đoán. Thay vì mã hóa nhãn là một one-hot vector, sẽ thay đổi nhãn một chút bằng cách phân bố thêm xác suất vào các trường hợp còn lại.



Phạt mô hình khi có thể để số epoch lớn mô hình sẽ không overfit.

3.5. Implementation

Data: Dataset song ngữ Anh-Việt: bộ dữ liệu song ngữ được thu thập trên TED bao gồm tổng cộng 600k câu song ngữ từ bộ dữ liệu của hội nghị IWSLT2015.

Cả nhóm đã train model 8 head 6 layer và đưa ra kết quả sau:

```
epoch: 027 - iter: 01199 - train loss: 2.4071 - time: 0.2506
epoch: 027 - iter: 01399 - train loss: 2.4203 - time: 0.2245
epoch: 027 - iter: 01599 - train loss: 2.4238 - time: 0.2290
epoch: 027 - iter: 01799 - train loss: 2.4265 - time: 0.2339
epoch: 027 - iter: 01999 - train loss: 2.4151 - time: 0.2609
epoch: 027 - iter: 02199 - train loss: 2.4036 - time: 0.2573
epoch: 027 - iter: 02399 - train loss: 2.4035 - time: 0.2511
epoch: 027 - iter: 02496 - valid loss: 2.4194 - bleu score: 0.2527 - time: 409.9355
epoch: 028 - iter: 00199 - train loss: 2.3310 - time: 0.2410
epoch: 028 - iter: 00399 - train loss: 2.3890 - time: 0.2257
epoch: 028 - iter: 00599 - train loss: 2.3667 - time: 0.2464
epoch: 028 - iter: 00799 - train loss: 2.3851 - time: 0.2548
epoch: 028 - iter: 00999 - train loss: 2.3850 - time: 0.2439
epoch: 028 - iter: 01199 - train loss: 2.3836 - time: 0.2370
epoch: 028 - iter: 01399 - train loss: 2.3931 - time: 0.2543
epoch: 028 - iter: 01599 - train loss: 2.4040 - time: 0.2444
epoch: 028 - iter: 01799 - train loss: 2.3844 - time: 0.2387
epoch: 028 - iter: 01999 - train loss: 2.4059 - time: 0.2499
epoch: 028 - iter: 02199 - train loss: 2.4218 - time: 0.2455
epoch: 028 - iter: 02399 - train loss: 2.4157 - time: 0.2436
epoch: 028 - iter: 02496 - valid loss: 2.4221 - bleu score: 0.2514 - time: 400.5117
epoch: 029 - iter: 00199 - train loss: 2.3479 - time: 0.2567
epoch: 029 - iter: 00399 - train loss: 2.3540 - time: 0.2485
epoch: 029 - iter: 00599 - train loss: 2.3606 - time: 0.2516
epoch: 029 - iter: 00799 - train loss: 2.3655 - time: 0.2545
epoch: 029 - iter: 00999 - train loss: 2.3706 - time: 0.2479
epoch: 029 - iter: 01199 - train loss: 2.3825 - time: 0.2462
epoch: 029 - iter: 01399 - train loss: 2.3761 - time: 0.2447
epoch: 029 - iter: 01599 - train loss: 2.3928 - time: 0.2543
epoch: 029 - iter: 01799 - train loss: 2.3842 - time: 0.2627
epoch: 029 - iter: 01999 - train loss: 2.3751 - time: 0.2434
epoch: 029 - iter: 02199 - train loss: 2.3847 - time: 0.2568
epoch: 029 - iter: 02399 - train loss: 2.3930 - time: 0.2315
epoch: 029 - iter: 02496 - valid loss: 2.4325 - bleu score: 0.2527 - time: 400.5555
gia đình tôi không nghèo, và bản thân tôi, tôi chưa bao giờ trải qua sự đói.
```

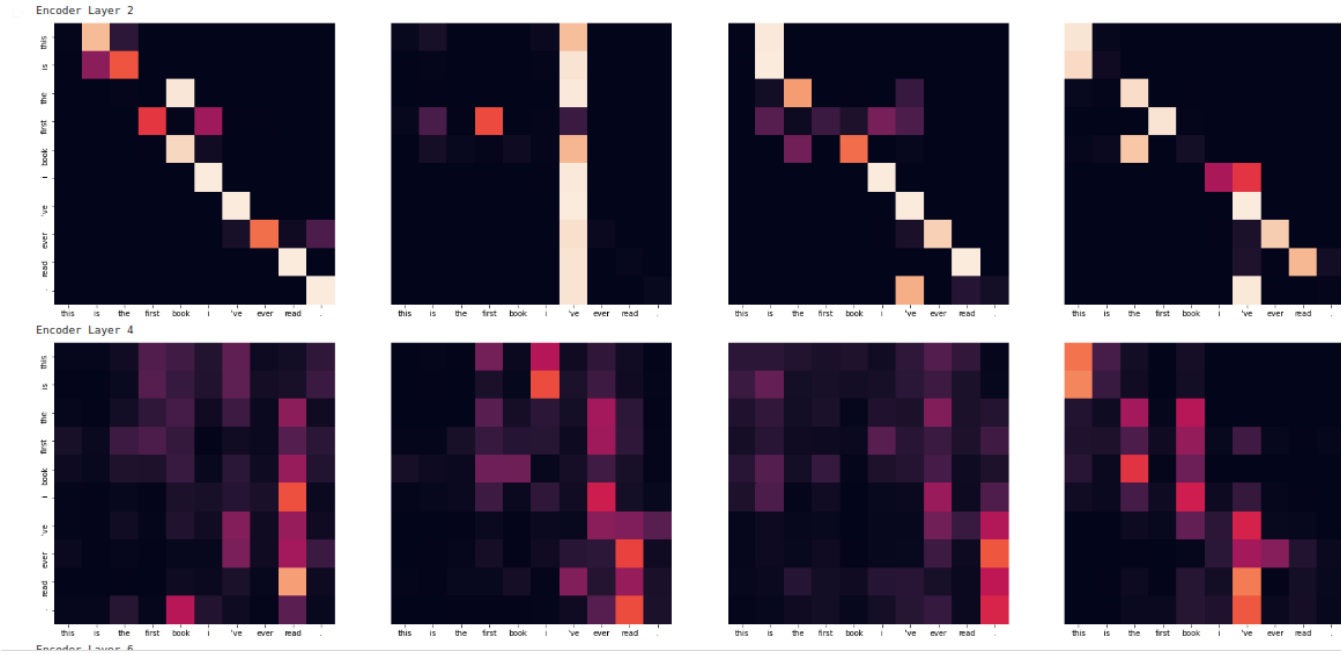
Kết quả dùng model đã train để thử câu khác.

```
[57] model = model.to(opt['device'])
     model.eval()
     sentence="this is the first book i've ever read."
     trans_sent = translate_sentence(sentence, model, SRC, TRG, opt['device'], opt['k'], opt['max_strlen'])
     trans_sent

     'đây là cuốn sách đầu tiên mà tôi đã từng đọc.'
```

Encoder Visualize

Dùng heatmap để visualize giá trị attention, sẽ cho biết khi encode một câu mô hình chú ý từ ở lân cận



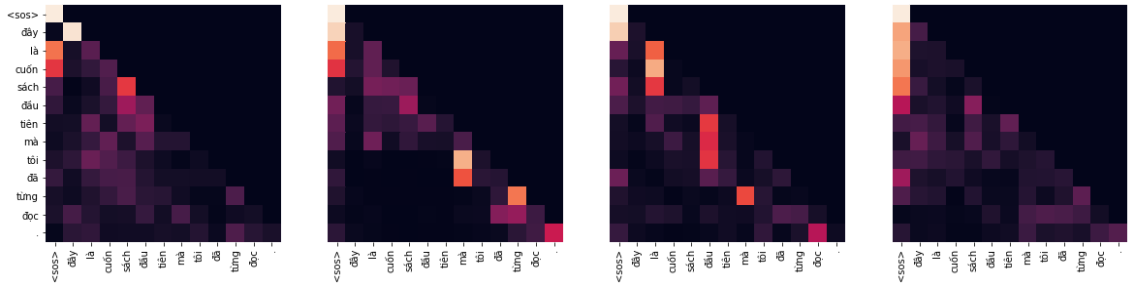
Ở đây visualize giá trị attention của encoder layer số 2 và 4, tại các head 0,1,2,3. Nhìn vào các heatmaps ở trên, ta có thể thấy được rằng khi encode một từ mô hình sẽ nhìn vào các từ liên quan xung quanh. Ví dụ từ **book** có thể được mã hóa bằng 2 từ liên quan như **the** và **book**.

Decoder Visualize

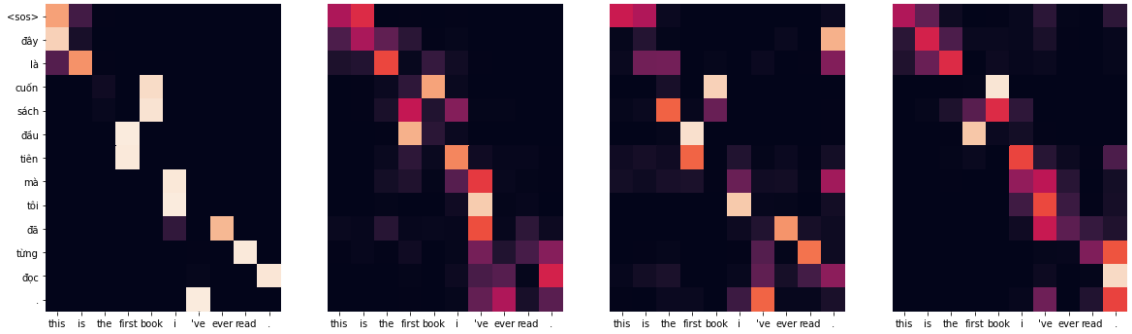
Ở decoder, có 2 loại visualization

- 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

Decoder Self Layer 2



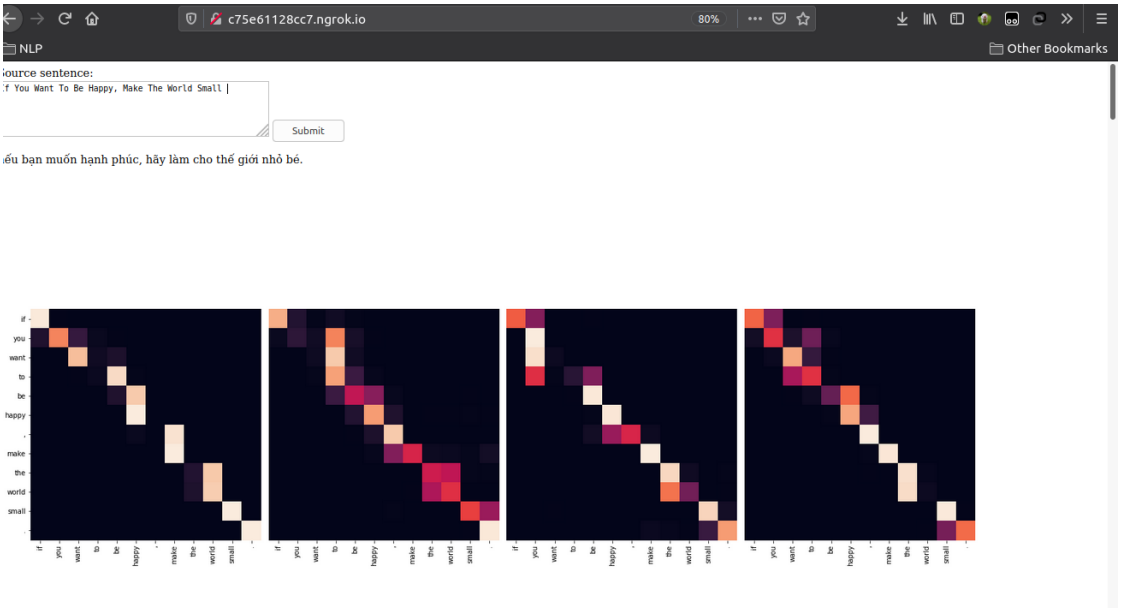
Decoder Src Layer 2



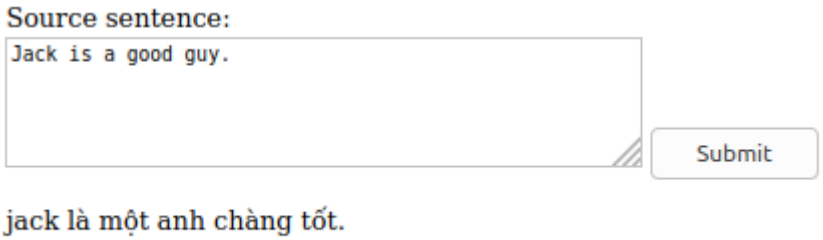
Ở ví dụ này visualize decoder layer số 2, tại 4 heads 0,1,2,3. Ta có thể quan sát được khi encode từ **sách** mô hình sẽ nhìn vào các từ kế cạnh là **đây** và **cuốn**, (và còn nhiều kiểu pattern khác nữa nhé). Còn khi dự đoán từ **tôi** mô hình sẽ nhìn vào từ **i**.

API

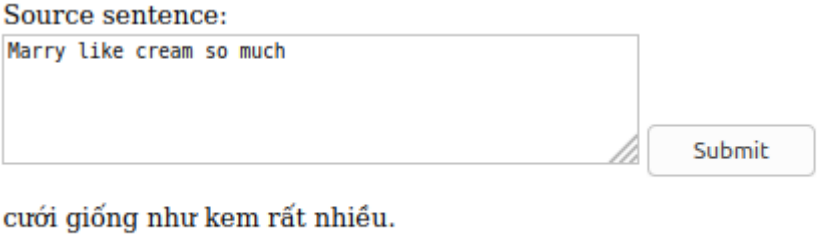
Tạo api đơn giản qua flask để dễ dàng translate câu và visualize:



Thử nghiệm với các tên riêng phổ biến như Jack thì model dịch tốt.



Nhưng khi lấy tên trùng với các danh từ khác thì không như Mary.



Mô hình cần bộ dataset riêng để nhận diện các tên hiệu quả hơn.

Đánh giá

Mô hình chạy nhanh và triển khai tốt hơn khi so sánh với RNN. Nhưng không xử lý tốt unknown words. Mô hình khi train với 16 head và 8 head cho bleu score không thực sự quá khác biệt, nên có thể khi số head đủ thì tăng số head lên cũng không thực sự tăng độ hiệu quả của việc dịch.

4. Kết luận

Trên đây là những nội dung tìm hiểu về mô hình Transformers và ứng dụng trong xử lý ngôn ngữ tự nhiên. Ứng dụng mô hình Transformers cho bài toán dịch máy – một trong các bài toán rất nổi trội và phổ biến trong xử lý ngôn ngữ tự nhiên.

Kết quả đã tìm hiểu và nắm bắt được các vấn đề về mặt mô hình, kỹ thuật trong bài toán dịch máy cho ngữ liệu song ngữ Anh-Việt. Thực nghiệm với mô hình đã xây dựng, áp dụng thay đổi số head, thử độ hiệu quả trong việc dịch tên, ngày tháng và xây dựng api web đơn giản.

Tài liệu tham khảo:

https://pbcquoc.github.io/transformer/

https://towardsdatascience.com/transformers-141e32e69591

https://github.com/huggingface/transformers/blob/master/README.md?fbclid=IwAR2Nme1sUxmRoAJriPgzFtN4Iu7pp44b7JqxC1bhAwqv_Y-0H21_n7PQeH4

https://viblo.asia/p/transformers-nguoi-may-bien-hinh-bien-doi-the-gioi-nlp-924lJPOXKPM?fbclid=IwAR3w8PeK5xDWzLC2kH4YZ8g_-uqc4MKMUfz261kw-jPldrSLuc0B3fzXdzQ

https://viblo.asia/p/scene-text-recognition-su-dung-mo-hinh-transformer-Qbq5Q0PElD8?fbclid=IwAR09jPP7EdWnq3epAXfsDUUSDkkt_svsvP-f-SZ2q6-GtjKWQtt_9YPdU1A

https://viblo.asia/p/transformerxl-leverage-transformer-for-language-modeling-aWj53WeQ56m?fbclid=IwAR1jIQ1sD6UXWJruyHHXGpuhk_HDwFbhjRfZMVKGPopuKPyTuxVp74WhtrE

https://www.youtube.com/watch?v=iDulhoQ2pro

https://www.youtube.com/watch?v=OyFJWRnt_AY

http://jalammar.github.io/illustrated-transformer/?fbclid=IwAR1oRmZOHf4DkMc4j8rO5HnFLp2W4DxP2QJWgmUyCtM4ByjRL5qo3vN-7Jk

https://trituenhantao.io/tin-tuc/minh-hoa-transformer/?fbclid=IwAR1yZerPCC-T7hPzBPAJ_oO6Xx-J4hZjHQxEVij-JCHPULmcF1duXcv4oEI