



Machine trans - Nein

Xử lý ngôn ngữ tự nhiên (Trường Đại học Công nghệ thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh)



Scan to open on Studocu

Vietnam National University HCMC
University Of Information Technology



BÁO CÁO ĐỒ ÁN MÔN HỌC

CS221.L22.KHCL

MACHINE TRANSLATION

Thành viên nhóm

19520197 Lê Đoàn Thiện Nhân
19520592 Dương Huỳnh Huy
19521281 Trương Minh Châu

Giảng viên phụ trách

Nguyễn Thị Quý

Mục lục

1 Giới thiệu	3
1.1 Dịch Máy (Machine Translation)	3
1.2 Mô hình dịch máy	3
2 Phân kỳ và phân loại ngôn ngữ	3
2.1 Phân loại thứ tự từ	4
2.2 Sự phân kỳ từ ngữ	5
2.3 Phân loại hình thái học	6
2.4 Mật độ tham chiếu (Referential density)	6
3 Mô hình Encoder-Decoder	6
4 Ứng dụng mạng RNN vào mô hình Encoder-Decoder	7
4.1 Khái quát	7
4.2 Mạng RNN Encoder-Decoder	8
4.3 Huấn luyện mô hình Encoder-Decoder	10
5 Cơ chế Attention	11
6 Thuật toán Beam Search	13
6.1 Lý do sử dụng thuật toán Beam Search	13
6.2 Ý tưởng thuật toán	14
6.2.1 Chọn ra những từ có xác suất lớn nhất	14
6.2.2 Chọn ra những câu có xác suất lớn nhất	15
6.3 Pseudo Code	15
7 Một số chi tiết thực tế trong xây dựng hệ thống Dịch Máy	17
7.1 Sự mã hóa (Tokenization)	17
7.2 Kho văn bản của Dịch Máy	17
7.2.1 Parallel Corpus (kho ngữ liệu song song)	17
7.2.2 Sentence alignment	18
7.3 Back-translation (Dịch ngược)	20
8 Đánh giá mô hình dịch máy	20
8.1 Sử dụng con người để đánh giá	20
8.2 Phương pháp đánh giá tự động: BLEU	20
8.2.1 Chỉ số đánh giá BLEU cơ sở	20
8.2.2 Độ chính xác n-gram	21
8.2.3 Độ chính xác n-gram được sửa đổi	22
8.2.4 Chi tiết thuật toán BLEU	22
8.2.5 Những hạn chế của BLEU	23
8.3 Phương pháp dựa trên Embedding	23
9 Các vấn đề thiên vị và đạo đức trong dịch máy	23
10 Demo code	24
10.1 The Encoder (Bộ mã hoá)	28
10.2 The decoder (Bộ giải mã)	29
10.2.1 Simple Decoder	29
10.2.2 Cơ chế Attention (Attention Mechanism)	30

10.2.3 Training model	32
11 Cài tiến mô hình dựa trên Transformer	38
11.1 Giới thiệu Transformer	38
11.2 Tổng quan mô hình	38
11.3 Embedding Layer với Position Encoding	40
11.4 Phương pháp đề xuất sinusoidal position encoding	41
11.5 Encoder	44
11.5.1 Self Attention Layer	45
11.5.2 Multi Head Attention	47
11.6 Residuals Connection và Normalization Layer	49
11.7 Decoder	50
11.7.1 Masked Multi Head Attention	50
11.8 Cài đặt Encoder	51
11.9 Cài đặt Decoder	52
11.10 Cài đặt Transformer	53
11.11 Data Loader	57
11.12 Optimizer	59
11.13 Label Smoothing	61
11.14 Visualize	66
11.14.1 Visualize Encoder	66
11.14.2 Visualize Decoder	67

1 Giới thiệu

1.1 Dịch Máy (Machine Translation)

Dầu tiên dịch thuật là thay đổi ngôn ngữ nước ngoài sang ngôn ngữ mẹ đẻ và ngược lại, do đó **dịch máy** (MT) là việc sử dụng máy tính để dịch văn bản. Một số cho rằng dịch máy có thể dịch các tài liệu chuyên ngành trong các lĩnh vực như văn học nhưng điều đó vẫn chưa thể thực hiện được. Lý do chính là khó dịch chính xác, nói rõ hơn là máy móc không thể dịch đúng ngữ pháp, chọn từng chữ phù hợp ngữ cảnh của bản dịch và không thể hiện được bất kỳ sự sáng tạo nào của con người.

Hiện tại, dịch máy chỉ tập trung vào các nhiệm vụ thực tế như truy cập thông tin hoặc hỗ trợ người phiên dịch. Sử dụng dịch máy để truy cập thông tin có lẽ là một trong những ứng dụng phổ biến nhất của công nghệ NLP và Google Dịch là công cụ đã thực hiện tất cả các tác vụ dịch tự động hàng trăm tỷ từ mỗi ngày giữa hơn 100 ngôn ngữ. Bên cạnh đó, dịch máy cũng hỗ trợ người dịch nâng cao năng suất và chất lượng bản dịch bằng cách tạo bản dịch nháp trong giai đoạn chỉnh sửa hậu kỳ, từ đó tạo ra một sản phẩm cho từng ngôn ngữ riêng biệt. Nhiệm vụ này được gọi là **dịch hỗ trợ máy tính** (CAT).

Ở thời điểm hiện tại, ứng dụng của dịch máy chính là đáp ứng nhu cầu giao tiếp của con người như dịch lời nói và dịch hình ảnh (bằng cách sử dụng OCR¹ để quét và trích xuất văn bản thành chuỗi làm đầu vào cho hệ thống dịch máy để dịch).

1.2 Mô hình dịch máy

Thuật toán tiêu chuẩn cho dịch máy là mạng **Sequence-to-Sequence** (hay còn gọi là mạng **Encoder-Decoder**), sử dụng kiến trúc mạng RNN hoặc Transformer để thực hiện phân loại (gán một câu với các thẻ cảm xúc tích cực hoặc tiêu cực để phân tích cảm xúc) hoặc gắn nhãn trình tự (gán từng từ trong một câu đầu vào với mỗi từ loại hoặc với một thẻ thực thể riêng). Đối với phần của gắn thẻ từ loại, mỗi thẻ được liên kết trực tiếp với một từ đầu vào, vì vậy chúng ta có một mô hình lấy đầu vào là một chuỗi các vectơ x_n và trả về đầu ra là các vectơ y_n tương ứng. Các từ đầu ra không thể chỉ được tạo ra từ các từ đầu vào riêng lẻ.

English:	<i>He wrote a letter to a friend</i>
Japanese:	<i>tomodachi ni tegami-o kaita</i>

friend to letter wrote

Với hệ thống dịch máy, nhiệm vụ chính là dịch các từ sang ngôn ngữ đích mà không cần phải giống với ngôn ngữ nguồn theo số lượng hoặc thứ tự. Hay nói cách khác là thứ tự từ ngữ của ngôn ngữ đầu vào không nhất thiết giống với ngôn ngữ đầu ra. Mô hình encoder-decoder thành công trong việc xử lý các loại trường hợp phức tạp của việc gán chuỗi, do đó không chỉ áp dụng cho dịch máy mà còn cho tóm tắt (tóm tắt một văn bản dài thành tiêu đề hoặc tóm tắt), đối thoại (tự động trả lời câu trả lời cho cuộc đối thoại của người dùng) và phân tích ngữ nghĩa (chuyển đổi một chuỗi từ thành biểu diễn ngữ nghĩa như dạng logic).

Trước khi bước vào tìm hiểu về các mô hình được sử dụng, thảo luận cách đánh giá mô hình cũng như là số liệu BLEU thì ở phần tiếp theo chúng ta phải tìm hiểu khái quát nền tảng ngôn ngữ học đối với dịch máy: những điểm khác biệt chủ yếu giữa các ngôn ngữ cần phải chú ý khi xem xét nhiệm vụ dịch thuật.

2 Phân kỳ và phân loại ngôn ngữ

Hầu hết các ngôn ngữ của con người đều có một số khía cạnh phổ biến và những tính phổ biến này phát sinh từ vai trò chức năng chính của ngôn ngữ là một hệ thống giao tiếp của con người. Vì vậy mọi ngôn ngữ dường như

¹Optical Character Recognition

đều có những từ để chỉ người, để nói về các hoạt động như ăn, uống, ngủ và vân vân; cũng có những cấu trúc ngôn ngữ phổ biến gồm tính từ, danh từ, động từ và trạng từ để đặt câu hỏi WH- hoặc ra lệnh.

Càng đào sâu vào tìm hiểu các ngôn ngữ, sự khác biệt sẽ dần xuất hiện nhiều hơn. Sự hiểu biết về những nguyên nhân gây ra sự khác biệt trong dịch thuật sẽ giúp xây dựng mô hình MT tốt hơn. Ví dụ như sự khác biệt về ngôn ngữ và từ vựng phải được xử lý từng cái một, trong khi sự khác biệt về hệ thống như ngữ pháp có thể mô hình hóa một cách tổng quát. Việc nghiên cứu những điểm tương đồng và khác biệt giữa các hệ thống ngôn ngữ được gọi là **phân loại ngôn ngữ** và phần này sẽ chỉ ra một số phân loại ảnh hưởng đến mô hình MT.

2.1 Phân loại thứ tự từ

Trong ngôn ngữ học, **phân loại thứ tự từ** là nghiên cứu trật tự của các cú pháp thành phần của một ngôn ngữ và cách các ngôn ngữ khác nhau sử dụng các trật tự khác nhau. Xem xét trong các câu tường thuật đơn giản, các ngôn ngữ sẽ có sự khác nhau về thứ tự từ cơ bản của động từ, chủ ngữ và tân ngữ. Ví dụ như tiếng Anh, Pháp, Đức và Trung đều là ngôn ngữ **SVO** (Subject-Verb-Object), có nghĩa là động từ có xu hướng nằm giữa chủ ngữ và tân ngữ. Trong khi đó tiếng Nhật và Hàn là ngôn ngữ **SOV** với động từ có xu hướng được đặt ở cuối các mệnh đề đơn giản, còn trật tự từ của tiếng Ả Rập và Ireland là **VSO**. Hai ngôn ngữ có cùng chung kiểu trật tự từ cơ bản thường có những điểm tương đồng khác như là các ngôn ngữ **VO** sẽ có các giới từ (**preposition**) còn ngôn ngữ **OV** lại có các hậu từ (**postposition**).

English: *He wrote a letter to a friend*

Japanese: *tomodachi ni tegami-o kaita*
friend to letter wrote

Arabic: *katabt risāla li ṣadq*
wrote letter to friend

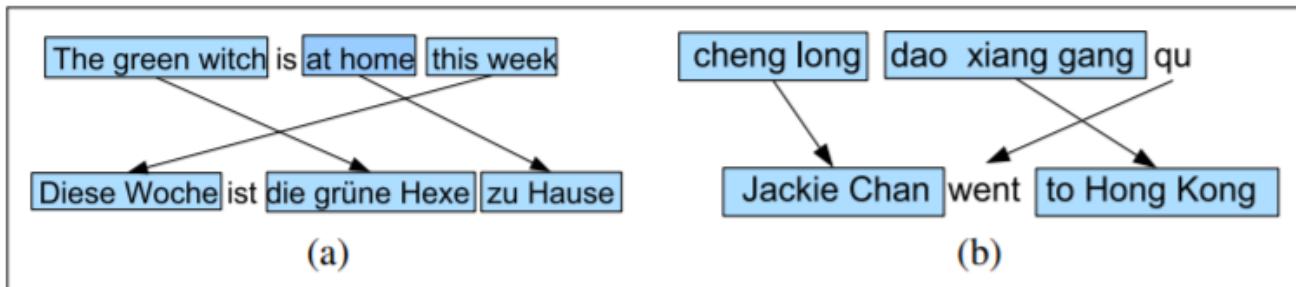
Ở ví dụ này trong câu tiếng anh, theo sau động từ *wrote* là tân ngữ *a letter* và cụm tính từ *to a friend*, trong đó giới từ *to* được theo sau bởi đối số (**argument**) của nó *a friend*. Thế nhưng trong câu tiếng Nhật ở trên, từng thứ tự từ bị đảo ngược; các đối số *friend* *to a letter* được đặt phía trước động từ *wrote* và ở đây *to* được xem là hậu từ đứng sau đối số *friend*. Cuối cùng là Tiếng Ả Rập, với thứ tự VSO, có động từ trước tân ngữ và giới từ.

Các loại thứ tự ưu tiên khác thay đổi từ ngôn ngữ này sang ngôn ngữ khác. Trong một số ngôn ngữ như tiếng Anh hoặc Trung, tính từ có xu hướng xuất hiện ở phía trước động từ. Trong khi những ngôn ngữ khác như Tây Ban Nha và Do Thái hiện đại thì tính từ ở phía sau danh từ.

Spanish *bruja verde*

English *green witch*

Hình dưới đây là các ví dụ cho thấy sự khác biệt về trật tự từ. Tất cả những khác biệt về trật tự từ này giữa các ngôn ngữ có thể gây ra vấn đề cho quá trình dịch, đòi hỏi hệ thống phải thực hiện nhiều sự tái sắp xếp cấu trúc để tạo ra kết quả đầu ra.



2.2 Sự phân kỳ từ ngữ

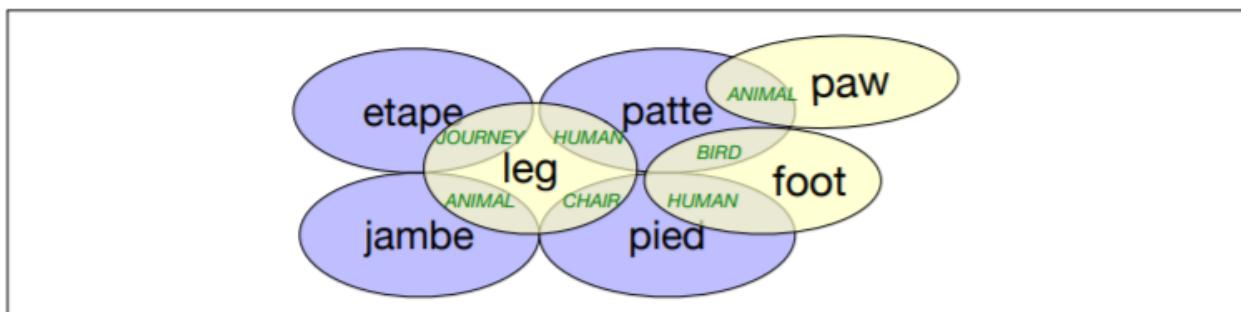
Đối với việc dịch thuật, chúng ta sẽ cần dịch từng chữ riêng lẻ từ ngôn ngữ này sang ngôn ngữ khác bởi vì từ thích hợp sẽ được thay đổi tùy theo ngữ cảnh của bản dịch đó.

- Từ *bass* trong tiếng Anh nếu xuất hiện trong tiếng Tây Ban Nha có thể được xem như là con cá *lubina* hoặc là một nhạc cụ *bajo*.
- Trong tiếng Đức sẽ dùng hai từ riêng biệt mà tiếng Anh gọi là bức tường *wall*: *wand* cho các bức tường bên trong tòa nhà và *mauer* cho các bức tường bên ngoài tòa nhà.
- Nếu như trong tiếng anh sử dụng *brother* để gọi cho bất kỳ mối quan hệ anh em ruột thì tiếng Trung và nhiều ngôn ngữ khác có các từ khác biệt dành cho anh trai và em trai tương ứng với *gēge* và *dìdi*.

Trong tất cả những trường hợp này, việc dịch *bass*, *wall* hoặc *brother* từ tiếng Anh sẽ yêu cầu một loại chuyên môn, phân biệt các cách sử dụng khác nhau của một từ. Vì lý do này, các trường của MT và Word Sense Disambiguation được liên kết chặt chẽ với nhau.

Đôi khi một ngôn ngữ đặt ra nhiều ràng buộc ngữ pháp đối với việc lựa chọn từ ngữ hơn. Ví dụ như nếu như tiếng Anh phân loại các danh từ nếu chúng là số ít hay số nhiều và đếm được hay không đếm được, còn tiếng Trung thì không. Thêm một dẫn chứng là tiếng Pháp hoặc Tây Ban Nha phân loại giới tính ngữ pháp trên tính từ vì vậy nếu bản dịch từ tiếng Anh sang Pháp yêu cầu xác định giới tính của tính từ.

Cách mà các ngôn ngữ khác nhau về từ vựng phân chia không gian khái niệm có thể là phức tạp hơn vấn đề dịch one-to-many, dẫn đến các ánh xạ many-to-many. Ví dụ, Hình sau đây tóm tắt một số phức tạp được thảo luận bởi Hutchins và Somers (1992) trong việc dịch *leg*, *foot* và *paw* từ tiếng Anh sang tiếng Pháp. Ví dụ, khi *leg* được sử dụng cho một con vật, nó được dịch sang tiếng Pháp là *jambe*; nhưng về *leg* của một cuộc hành trình sẽ được gọi là *etape* và nếu *leg* của một chiếc ghế, chúng ta sẽ dùng *pied*.



Ngoài ra thì một ngôn ngữ cũng có thể có một khoảng trống về mặt từ vựng (**lexical gap**), có nghĩa là sẽ không có từ hoặc cụm từ nào ngắn gọn trong ngôn ngữ đó để chú thích, giải thích hay diễn đạt chính xác ý nghĩa của một từ trong ngôn ngữ kia. Ví dụ trong tiếng Anh không có từ đơn nào tương ứng để có thể diễn tả *guāi* trong tiếng Trung và *oyakōkō* trong tiếng Nhật mà thay vào đó sẽ dùng các cụm từ như *filial piety, loving child, good son/daughter*.

Cuối cùng, các ngôn ngữ khác nhau một cách hệ thống trong việc các thuộc tính khái niệm của một sự kiện được ánh xa vào các từ cụ thể như thế nào. Talmy (1985, 1991) lưu ý rằng ngôn ngữ có thể được biểu thị đặc biệt bởi hướng và cách thức chuyển động được đánh dấu trên động từ hoặc trên "vệ tinh" (**satellite**): các trợ từ, cụm giới từ, hoặc cụm từ trạng ngữ. Trong ngôn ngữ học đó được gọi là **verb-framing** và **satellite-framing**, là những mô tả diễn hình về cách mà các cụm động từ trong một ngôn ngữ có thể mô tả đường đi của chuyển động hoặc cách thức của chuyển động, tương ứng.

English: *The bottle floated out.*

Spanish: La botella salió flotando.
The bottle exited floating.

Ở câu tiếng anh trên chúng ta dễ dàng thấy hướng của cái chai trôi ra ngoài được thể hiện trên trợ từ *out* còn với tiếng Tây Ban Nha được đánh dấu trên động từ *salió*. Từ đó các ngôn ngữ cũng được chia thành hai loại: ngôn ngữ verb-framing và ngôn ngữ satellite-framing. Với các ngôn ngữ verb-framing như tiếng Nhật, Tamil và nhiều ngôn ngữ thuộc họ ngôn ngữ Rôman, Semitic và Maya thể hiện hướng chuyển động trên động từ (để các "vệ tinh" thể hiện cách thức chuyển động) và hai loại chuyển động trên được thể hiện ngược lại với các ngôn ngữ satellite-framing như tiếng Trung và các ngôn ngữ hệ Án-Âu không thuộc nhóm Rôman như tiếng Anh, Thụy Điển, Nga.

2.3 Phân loại hình thái học

Về mặt hình thái học, ngôn ngữ có thể được mô tả dựa trên 2 hướng khác nhau. Hướng thứ nhất là số lượng hình vị ở mỗi từ, những ngôn ngữ isolating như tiếng Việt và tiếng Quảng Đông thì thông thường chỉ có một hình vị cho một từ đơn lẻ, còn đối với những ngôn ngữ polysynthetic như tiếng Eskimo thì một từ lại có rất nhiều hình vị có thể tương đương với 1 câu trong tiếng Anh. Ở hướng thứ hai, mức độ mà các hình vị có thể phân đoạn được, từ các ngôn ngữ tương tự như tiếng Thổ Nhĩ Kỳ, các hình vị có ranh giới tương đối rõ ràng, sang các ngôn ngữ như tiếng Nga, một phụ tố duy nhất có thể kết hợp nhiều morpheme.

Việc dịch giữa các ngôn ngữ có hình thái phong phú đòi hỏi phải xử lý cấu trúc dưới cấp độ từ và vì lý do này, các hệ thống hiện đại thường sử dụng các mô hình từ khóa con như mô hình wordpiece hoặc BPE.

2.4 Mật độ tham chiếu (Referential density)

Những ngôn ngữ có thể bỏ đại từ được gọi là **pro-drop language**, các ngôn ngữ có xu hướng sử dụng nhiều đại từ hơn có mật độ tham chiếu cao hơn.

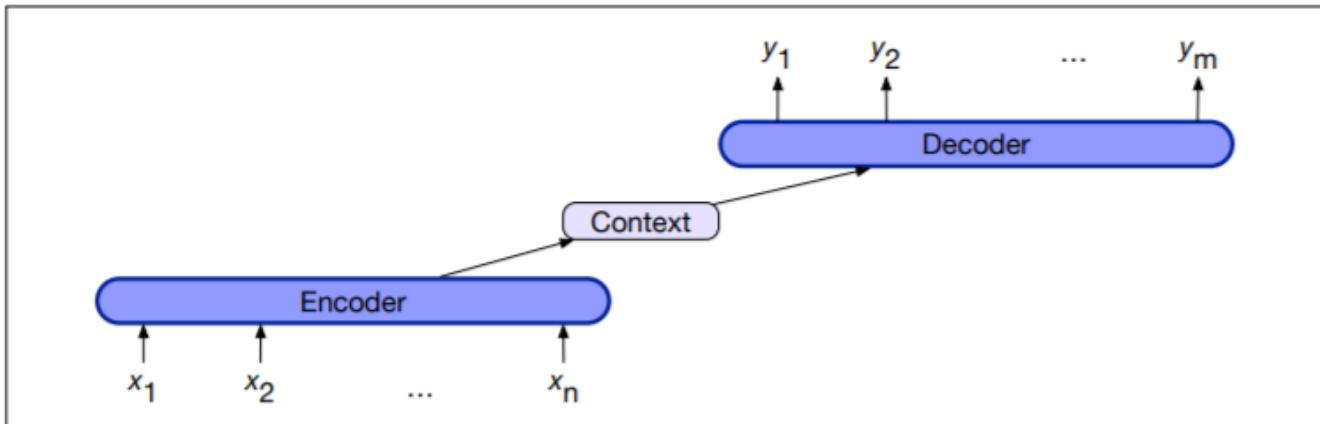
Việc dịch từ các ngôn ngữ pro-drop sang các ngôn ngữ không pro-drop như tiếng Anh có thể gặp nhiều khó khăn vì mô hình phải tìm ra những chỗ bị lược bỏ và cố gắng khôi phục lại đại từ sao cho chính xác nhất.

3 Mô hình Encoder-Decoder

Mạng encoder-decoder hay mạng sequence-to-sequence là những mô hình có thể tạo ra các chuỗi đầu ra phù hợp với ngữ cảnh với độ dài tùy ý. Mạng encoder-decoder đã được áp dụng cho một loạt các ứng dụng bao gồm dịch

máy, tóm tắt, trả lời câu hỏi và hội thoại.

Ý tưởng cơ bản chính của các mạng này là việc sử dụng mạng encoder để lấy chuỗi đầu vào và tạo ra một biểu diễn theo ngữ cảnh của nó, thường được gọi là ngữ cảnh (**context**). Biểu diễn này sau đó được chuyển đến decoder để tạo ra một chuỗi đầu ra cụ thể của nhiệm vụ. Hình 11.3 dưới đây sẽ minh họa kiến trúc này.



Mô hình điển hình của mạng encoder-decoder này gồm 3 thành phần chính:

- **Encoder** nhận một chuỗi đầu vào x_1^n và sau đó tạo ra một chuỗi biểu diễn được ngữ cảnh hóa tương ứng h_1^n . Các loại mạng như LSTM, GRU, mạng phức hợp, Transformer được sử dụng làm encoder.
- **Context vector** (vectơ ngữ cảnh) c là hàm trạng thái ẩn h_1^n có chức năng gói gọn các thông tin, bản chất của các phần tử đầu vào rồi truyền tải sang decoder.
- **Decoder** nhận c làm đầu vào và tạo ra một chuỗi độ dài tùy ý của các trạng thái ẩn h_1^m chính nó cũng như thu được một chuỗi đầu ra y_1^m tương ứng.

4 Ứng dụng mạng RNN vào mô hình Encoder-Decoder

4.1 Khái quát

Mở đầu cho phần này chúng ta sẽ mô tả mô hình encoder-decoder dựa trên một cặp mạng neural truy hồi (RNN). Nhớ lại cách tính xác suất của một chuỗi đầu ra y của mô hình ngôn ngữ RNN có điều kiện đã được giới thiệu trong quá trình học, thì với bất kì mô hình ngôn ngữ nào khác, chúng ta có thể chia nhỏ xác suất như sau:

$$p(y) = p(y_1)p(y_2 | y_1)p(y_3 | y_1, y_2) \dots p(y_m | y_1, \dots, y_{m-1}) \quad (1)$$

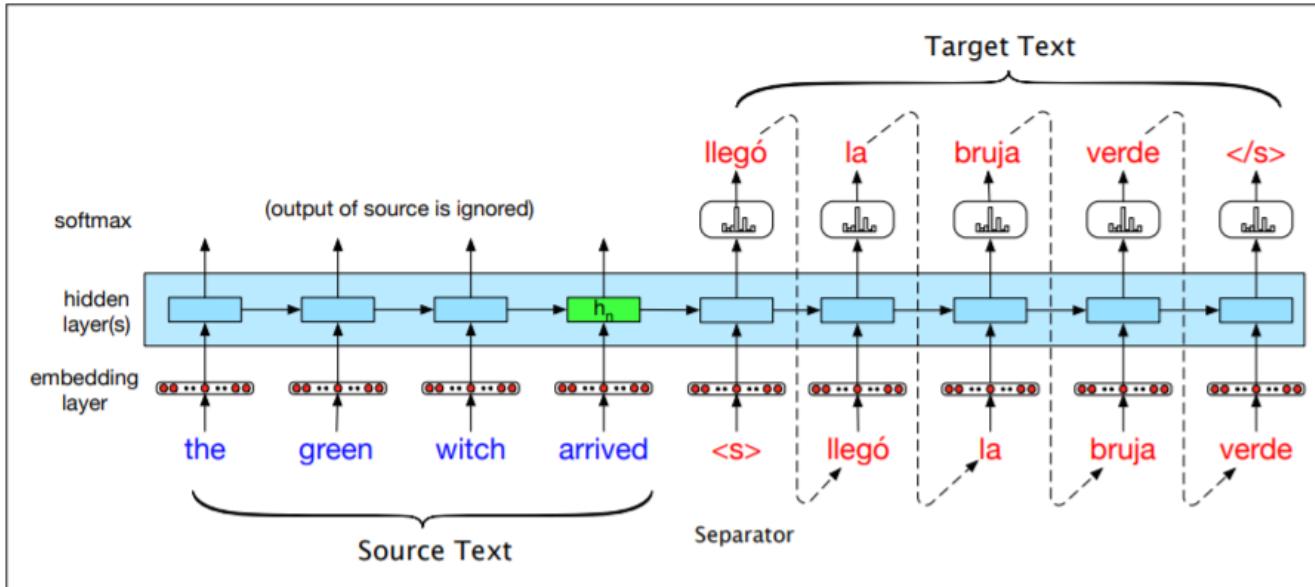
Tại một thời điểm cụ thể t , trạng thái ẩn trước đó $h^{(t-1)}$ được tính bằng cách chuyển các mã thông báo $t-1$ của tiền tố (**prefix**) qua mô hình ngôn ngữ rồi sử dụng phương pháp Forward Inference để tạo ra một chuỗi trạng thái ẩn kết thúc bởi trạng thái ẩn của từ cuối cùng của tiền tố. Sau đó sử dụng trạng thái ẩn cuối cùng ấy $h^{(t-1)}$ làm điểm bắt đầu để tạo mã thông báo tiếp theo.

Về mặt hình thức, nếu g là một hàm kích hoạt như *tanh* hoặc *ReLU*, một hàm của đầu vào ở thời điểm t và trạng thái ẩn ở thời điểm $t-1$ cùng với f là hàm Softmax trên tập hợp các mục từ vựng có thể có, thì ở tại thời điểm t trạng thái ẩn $h^{(t)}$ và đầu ra y_t được tính như sau:

$$\begin{aligned} h_t &= g(h_{t-1}, x_t) \\ y_t &= f(h_t) \end{aligned} \quad (2)$$

Để chuyển mô hình ngôn ngữ với tính năng tự hồi quy tạo ra dữ liệu (**Auto-regressive Generation**) trở thành một mô hình dịch thuật để có thể phiên dịch từ văn bản nguồn sang văn bản đích trong 1 giây chỉ cần thêm dấu phân tách câu ở cuối văn bản **nguồn** rồi nối với văn bản **đích**. Ý tưởng mã thông báo phân tách câu (**Sentence Separator Token**) đã được giới thiệu ngắn gọn khi xem xét sử dụng mô hình ngôn ngữ transformer để thực hiện tóm tắt, bằng cách huấn luyện mô hình ngôn ngữ có điều kiện. Giả sử gọi source text là x và target text là y , chúng ta tính được xác suất $p(y | x)$ như sau:

$$p(y) = p(y_1 | x) p(y_2 | y_1, x) p(y_3 | y_1, y_2, x) \dots p(y_m | y_1, \dots, y_{m-1}, x) \quad (3)$$

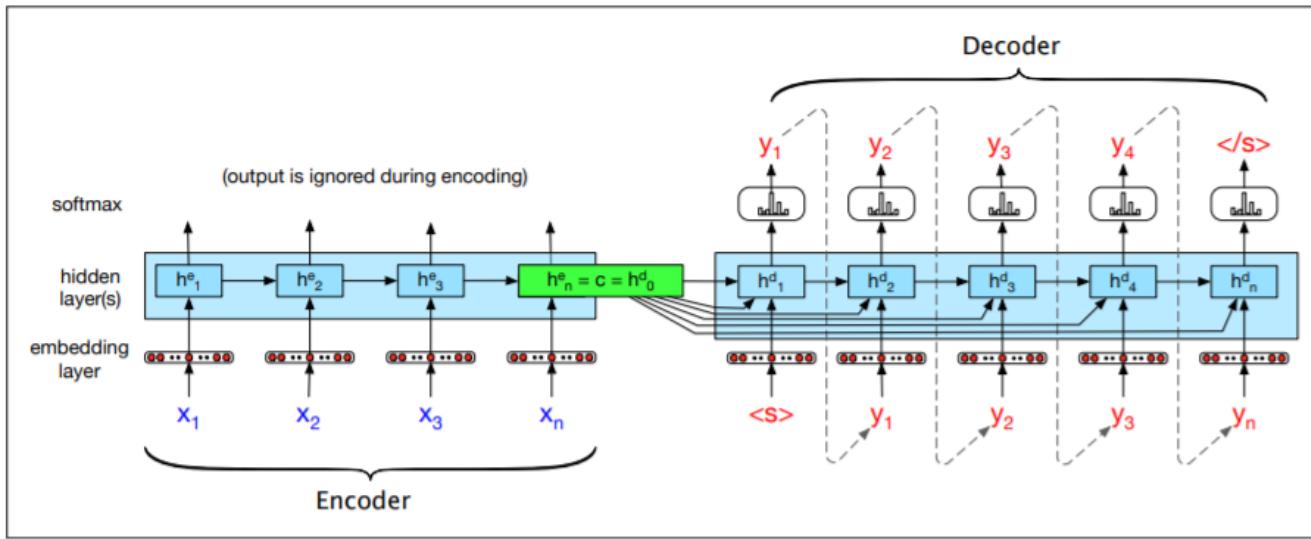


4.2 Mạng RNN Encoder-Decoder

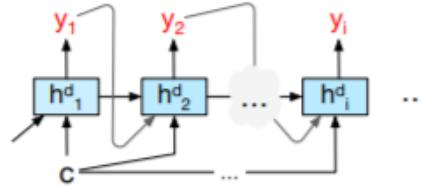
Theo như hình minh họa ở dưới đây, các phần tử của mạng ở bên trái xử lý chuỗi đầu vào x được xem là encoder. Mặc dù mô hình đơn giản chỉ hiển thị một lớp mạng duy nhất cho encoder, nhưng các kiến trúc xếp chồng là hàm quy chuẩn (**norm**), nơi các trạng thái đầu ra từ lớp trên cùng của ngăn xếp được lấy làm đại diện cuối cùng. Thiết kế encoder được dùng rộng rãi sử dụng các biLSTM xếp chồng lên nhau, nơi các trạng thái ẩn từ các lớp trên cùng từ Forward và Backward được nối với nhau để cung cấp các biểu diễn theo ngữ cảnh cho mỗi bước thời gian.

Mục đích chính của encoder là tạo ra một biểu diễn đầu vào được ngữ cảnh hóa, biểu diễn này được gọi là vectơ ngữ cảnh c . Vectơ ngữ cảnh thu được từ encoder đọc chuỗi đầu vào và tạo ra biểu diễn không gian liên tục. Vectơ ngữ cảnh thu được chính là trạng thái ẩn cuối cùng của encoder, h_n^e ² và rồi chuyển đến decoder.

²Các ký tự e và d được sử dụng để phân biệt trạng thái ẩn của encoder và decoder



Mạng decoder ở phía bên phải của mô hình 11.5 lấy trạng thái vừa được chuyển tới để khởi tạo trạng thái ẩn đầu tiên của decoder. Nghĩa là ô đầu tiên của RNN decoder sử dụng c làm trạng thái ẩn đầu h_d^0 . Decoder tự động tạo ra một chuỗi các đầu ra, từng phần tử ở từng thời điểm cho đến khi một điểm đánh dấu kết thúc chuỗi được tạo ra. Mỗi trạng thái ẩn đều tùy thuộc vào trạng thái ẩn trước và đầu ra được tạo ở trạng thái trước đó.



Một nhược điểm của cách tiếp cận trên chính là sự ảnh hưởng của vectơ ngữ cảnh c , sẽ suy yếu khi chuỗi đầu ra được tạo ra. Một giải pháp khắc phục là làm vectơ ngữ cảnh c có sẵn ở mỗi bước trong quá trình giải mã chuỗi đầu vào bằng cách thêm vào phương trình tham số c để tính toán trạng thái ẩn hiện tại:

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c) \quad (4)$$

Dưới đây là toàn bộ các phương trình khái quát quá trình thực thi của decoder trong mô hình encoder-decoder cơ bản với ngữ cảnh có sẵn ở từng thời điểm giải mã. Đã được đề cập ở phía trên g là một hàm thay thế cho một số thành phần (**flavour**) của RNN và $y_{(t-1)}$ là phép nhúng cho đầu ra được lấy mẫu từ hàm softmax ở bước trước:

$$\begin{aligned} c &= h_n^e \\ h_0^d &= c \\ h_t^d &= g(\hat{y}_{t-1}, h_{t-1}^d, c) \\ z_t &= f(h_t^d) \\ y_t &= \text{softmax}(z_t) \end{aligned} \quad (5)$$

Cuối cùng, như đã trình bày trước đó, đầu ra y tại mỗi bước thời gian bao gồm một phép tính softmax trên tập hợp các đầu ra có thể có (từ vựng, trong trường hợp mô hình hóa ngôn ngữ hoặc dịch máy). Chúng ta tính toán

đầu ra có khả năng xảy ra nhất tại mỗi bước thời gian bằng cách lấy argmax trên đầu ra softmax:

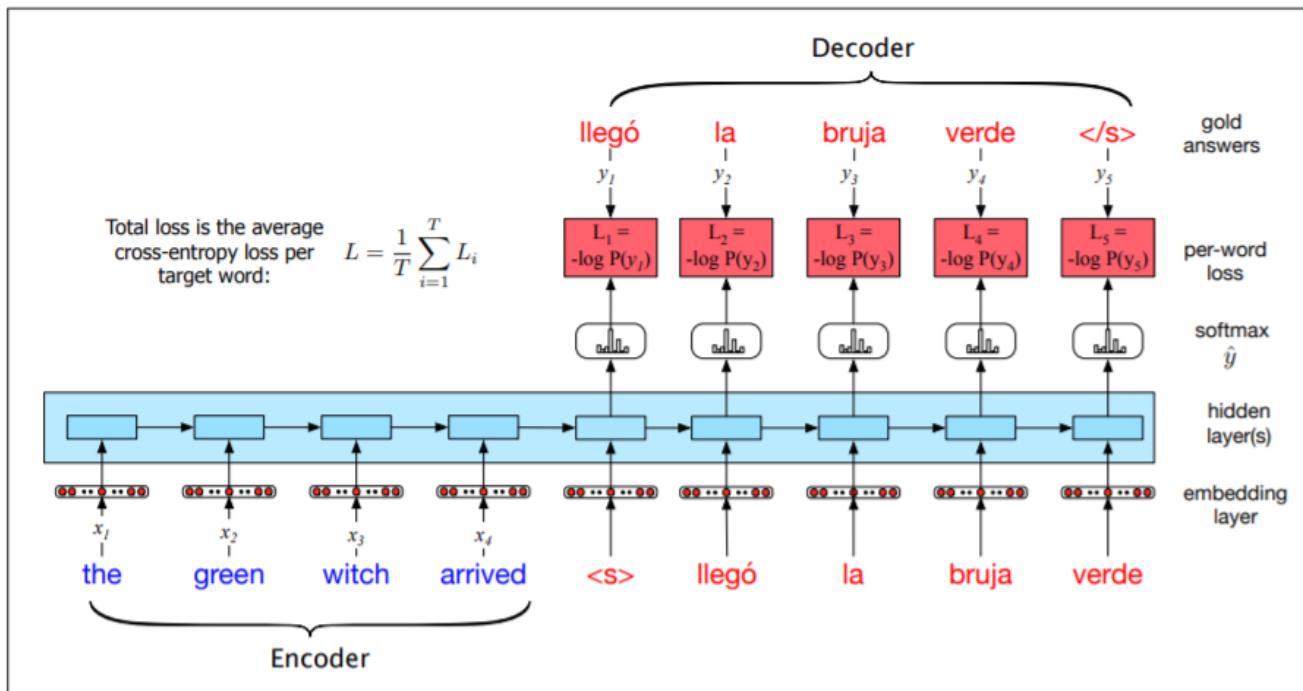
$$\hat{y}_t = \text{argmax}_{w \in V} P(w | x, y_1 \dots y_{t-1}) \quad (6)$$

Cũng có nhiều cách khác nhau để làm cho mô hình mạnh mẽ hơn một chút. Ví dụ chúng ta có thể giúp mô hình theo dõi những gì đã được tạo và những gì chưa bằng cách điều chỉnh lớp đầu ra y không chỉ trên trạng thái ẩn h_t^d và ngữ cảnh c mà còn trên đầu ra $y_{(t-1)}$ được tạo ra tại bước thời gian trước:

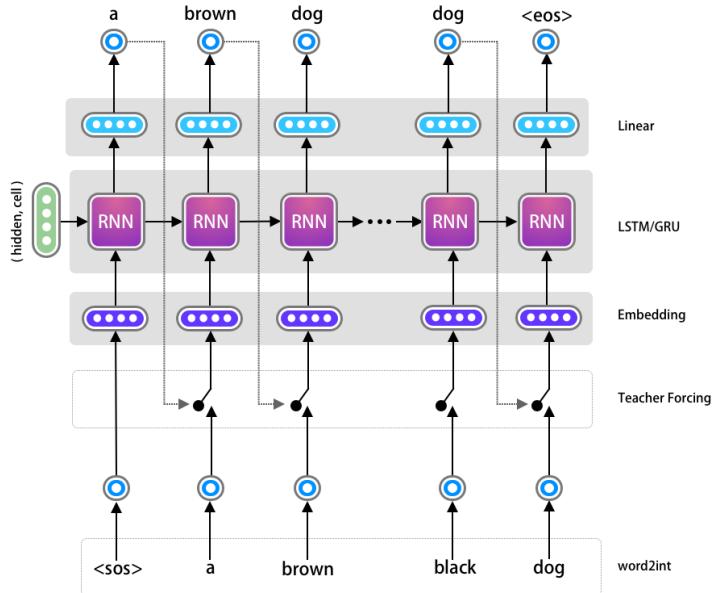
$$y_t = \text{softmax}(\hat{y}_{t-1}, y_t, c) \quad (7)$$

4.3 Huấn luyện mô hình Encoder-Decoder

Kiến trúc encoder-decoder được huấn luyện từ đầu đến cuối, giống như với các mô hình ngôn ngữ RNN. Mỗi ví dụ huấn luyện là một bộ chuỗi được ghép nối, một nguồn và một đích. Được kết hợp với mã thông báo phân tách, các cặp nguồn-đích này có thể đóng vai trò là dữ liệu huấn luyện. Đối với dịch máy, dữ liệu huấn luyện thường bao gồm các bộ câu và bản dịch của chúng. Chúng có thể được rút ra từ bộ dữ liệu chuẩn của các cặp câu được sắp xếp thành từng dòng, và chúng ta sẽ thảo luận tiếp trong Phần 11.7.2. Sau khi có một bộ huấn luyện, quá trình huấn luyện sẽ tự tiến hành như với bất kỳ mô hình ngôn ngữ dựa trên RNN nào. Mô hình được cung cấp văn bản nguồn và sau đó bắt đầu với mã phân tách (**Separator Token**) được huấn luyện tự động để dự đoán từ tiếp theo, như được hiển thị trong hình dưới đây.



Sự khác biệt giữa việc huấn luyện (Hình 7) và việc suy luận (Hình 4) đối với kết quả đầu ra ở mỗi bước thời gian. Decoder trong quá trình suy luận sử dụng đầu ra được ước tính của chính nó \hat{y}_t làm đầu vào cho bước x_{t+1} trong thời gian tiếp theo. Khi một trong các đầu ra dự đoán sai từ thì đầu ra tiếp theo không học được vì nó nhận được đầu vào sai và decoder sẽ có xu hướng ngày càng di chêch khỏi câu mục tiêu với phần còn lại của các mã thông báo chuỗi. Điều này dẫn đến sự hội tụ của mô hình rất chậm. Do đó, trong huấn luyện, việc sử dụng **Teacher Forcing** trong decoder là phổ biến hơn.



Teacher Forcing có nghĩa là chúng ta buộc hệ thống sử dụng mã thông báo gold target từ việc huấn luyện làm đầu vào tiếp theo x_{t+1} , thay vì cho phép nó dựa vào đầu ra (có thể sai) của decoder \hat{y}_t . Điều này tăng tốc độ huấn luyện và hội tụ nhanh hơn.

5 Cơ chế Attention

Trong cơ chế Attention, vectơ ngữ cảnh c trong mô hình encoder-decoder vanilla là một vectơ duy nhất, là hàm của các trạng thái ẩn của encoder: $c = f(h_1^e)$. Thay vì phải sử dụng trực tiếp toàn bộ vector trạng thái ẩn của encoder như là ngữ cảnh cho decoder vì kích thước đầu vào ảnh hưởng trực tiếp đến số lượng trạng thái ẩn, ta có thể tạo một vectơ c có 1 độ dài cố định duy nhất bằng cách lấy tổng trọng số của tất cả các trạng thái ẩn của encoder h_1^e .

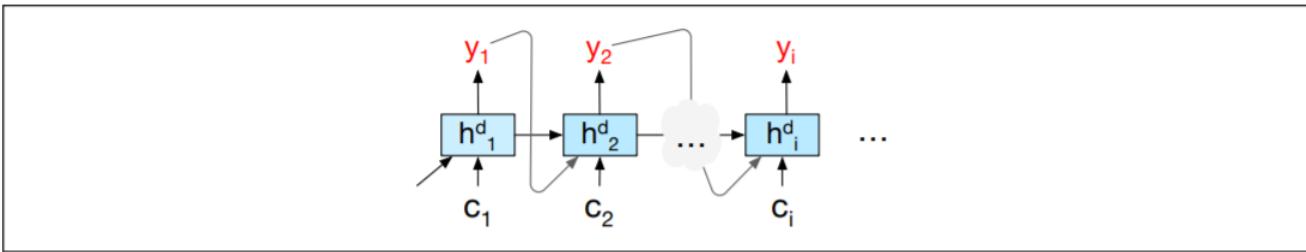
Trọng số được sử dụng để tập trung vào một phần cụ thể của bộ văn bản đầu vào có liên quan trực tiếp đến mã thông báo (**token**) đang được tạo ra từ decoder. Vector ngữ cảnh được tạo ra bởi cơ chế Attention cho nên nó rất cơ động, khác nhau trong từng mã thông báo trong quá trình giải mã.

Static context vector được thay bởi một vector động được trích ra từ trạng thái ẩn của encoder ở từng điểm trong suốt quá trình giải mã bởi cơ chế Attention. Vector ngữ cảnh c_i được khởi tạo mới với từng giai đoạn giải mã i và xem xét tất cả trạng thái ẩn của encoder trong quá trình đạo hàm. Sau đó làm cho vector ngữ cảnh có giá trị trong suốt quá trình giải mã bằng cách tùy chỉnh quá trình tính toán của các trạng thái ẩn ở decoder hiện tại trên nó (bao gồm cả trạng thái ẩn đầu tiên và đầu ra trước đó được tạo bởi decoder). Phương trình dưới đây thể hiện điều đó:

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i) \quad (8)$$

Phương trình trên của cơ chế Attention thể hiện sự cho phép mỗi trạng thái ẩn của decoder nhìn thấy một ngữ cảnh động khác nhau, đây là hàm của tất cả các trạng thái ẩn ở encoder.

Bước đầu tiên trong quá trình tính toán vector c_i là tính toán mức độ tập trung vào mỗi trạng thái của encoder, mức độ liên quan của mỗi trạng thái của decoder với trạng thái của encoder được ghi lại trong h_{i-1}^d . Chúng ta ghi lại sự liên quan bằng cách tính ở từng trạng thái i trong suốt quá trình giải mã – có một điểm (h_{i-1}^d, h_j^e) cho từng trạng thái decoder j .



Sự đơn giản nhất như điểm số, được gọi là dot-product attention, thể hiện mức độ liên quan như sự tương đồng: ước lượng mức độ tương tự nhau của trạng thái ẩn decoder đối với trạng thái ẩn encoder bằng cách tính toán dot-product giữa chúng:

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e \quad (9)$$

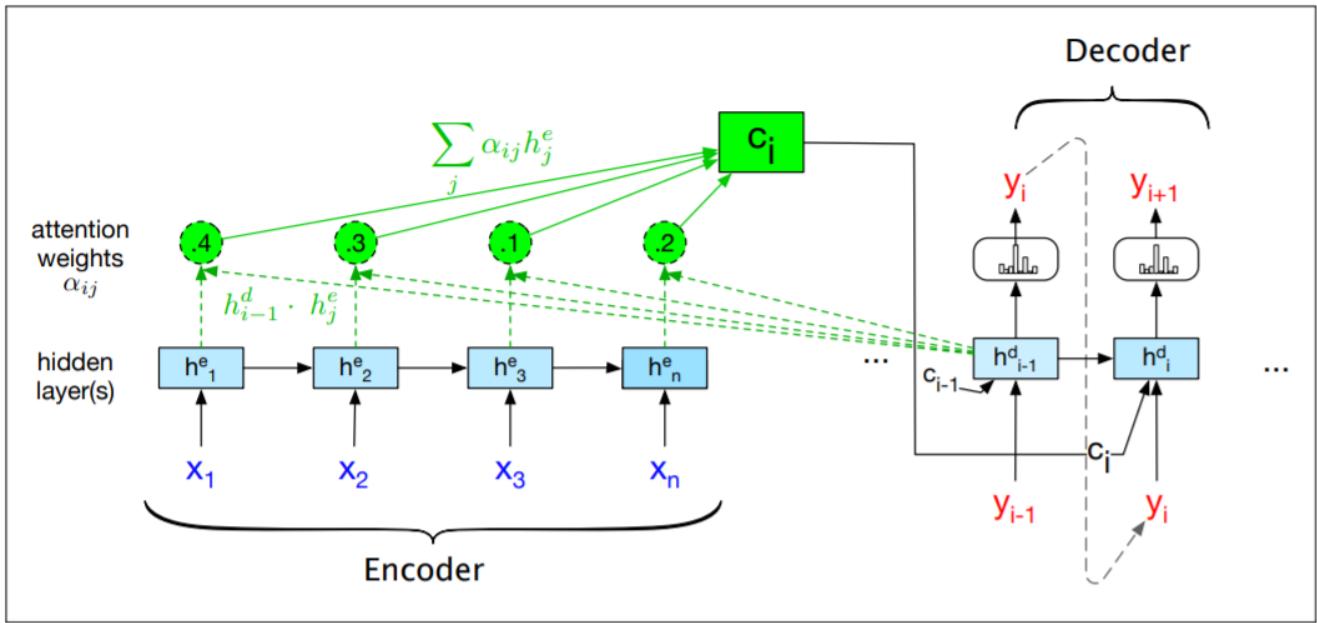
Điểm số thu được từ dot-product là một đại lượng vô hướng phản ánh mức độ tương đồng giữa hai vectơ. Vectơ của những điểm số này chạy ngang qua tất cả trạng thái ẩn encoder, cung cấp cho ta mức độ liên quan của từng trạng thái encoder với bước hiện tại của decoder. Để tận dụng những điểm số này, ta sẽ chuẩn hóa chúng bằng hàm Softmax để tạo vectơ có trọng số, α_{ij} , cho chúng ta biết mức độ liên quan theo tỷ lệ của mỗi trạng thái ẩn của encoder j sang trạng thái ẩn đầu tiên của decoder h_{i-1}^d .

$$\begin{aligned} \alpha_{ij} &= \text{softmax} ((\text{score}(h_{i-1}^d, h_j^e) \forall j \in e)) \\ &= \frac{\exp(\text{score}(h_{i-1}^d, h_j^e))}{\sum_k \exp(\text{score}(h_{i-1}^d, h_k^e))} \end{aligned} \quad (10)$$

Cuối cùng, với phân phối theo α , chúng ta có thể tính toán vector ngữ cảnh có độ dài cố định cho trạng thái decoder hiện tại bằng cách lấy giá trị trung bình có trọng số trên tất cả trạng thái ẩn encoder.

$$c_i = \sum_j \alpha_{ij} h_j^e \quad (11)$$

Như vậy, chúng ta đã hoàn tất việc tạo ra một vector ngữ cảnh có độ dài cố định có tính đến lượng thông tin từ toàn bộ trạng thái encoder được cập nhật một cách chủ động để phản ánh sự yêu cầu của decoder tại mỗi bước giải mã. Hình dưới đây minh họa mô hình encoder-decoder sử dụng attention, tập trung vào tính toán một vector ngữ cảnh c_i .



Hình trên phác thảo bộ mô hình encoder-decoder sử dụng attention, tập trung vào tính toán một vector ngữ cảnh c_i . Giá trị của vector ngữ cảnh c_i là một trong các đầu vào để tính toán h_i^d . Nó được tính bằng cách lấy tổng trọng số của các trạng thái ẩn encoder với mỗi trạng thái được đo do số lượng dot-product của chúng và trạng thái ẩn đầu tiên của decoder h_{i-1}^d .

Và tất nhiên có thể tạo ra nhiều hàm tính điểm phức tạp cho mô hình Attention. Thay vì dùng dạng đơn giản dot-product của Attention, ta có thể tạo ra một hàm mạnh mẽ hơn nhiều, có thể tính toán mức độ liên quan của từng trạng thái ẩn encoder với trạng thái ẩn decoder bằng cách tham số hoá điểm số với trọng số của riêng nó, W_s .

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d W_s h_j^e \quad (12)$$

Trọng số W_s sau khi được huấn luyện bình thường từ đầu đến cuối, cung cấp một mạng lưới có khả năng tìm hiểu khía cạnh nào tương đồng giữa trạng thái của encoder và decoder là quan trọng đối với ứng dụng hiện tại. Mô hình song tuyến này còn cho phép encoder-decoder sử dụng các vector có chiều khác nhau, trong khi mô hình dot-product đơn giản của Attention yêu cầu các trạng thái ẩn của encoder-decoder có cùng số chiều.

6 Thuật toán Beam Search

Thuật toán beam search là một thuật toán tìm kiếm heuristic. Nó được sử dụng trong các bài toán như dịch máy, nhận dạng giọng nói, tóm tắt văn bản và vân vân. Đó là các bài toán xử lý ngôn ngữ tự nhiên có đầu ra liên quan đến việc tạo một chuỗi các từ.

6.1 Lý do sử dụng thuật toán Beam Search

Trong các bài toán xử lý ngôn ngữ tự nhiên cụ thể là bài toán dịch máy yêu cầu đầu ra của mô hình là chuỗi các từ có trong từ điển.

Thường thì mỗi từ trong chuỗi từ mà mô hình của các bài toán như trên dự đoán sẽ đi kèm theo một phân phối xác suất tương ứng. Khi đó, bộ decoder của mô hình sẽ dựa trên phân bố xác suất đó để tìm ra chuỗi từ phù hợp nhất.

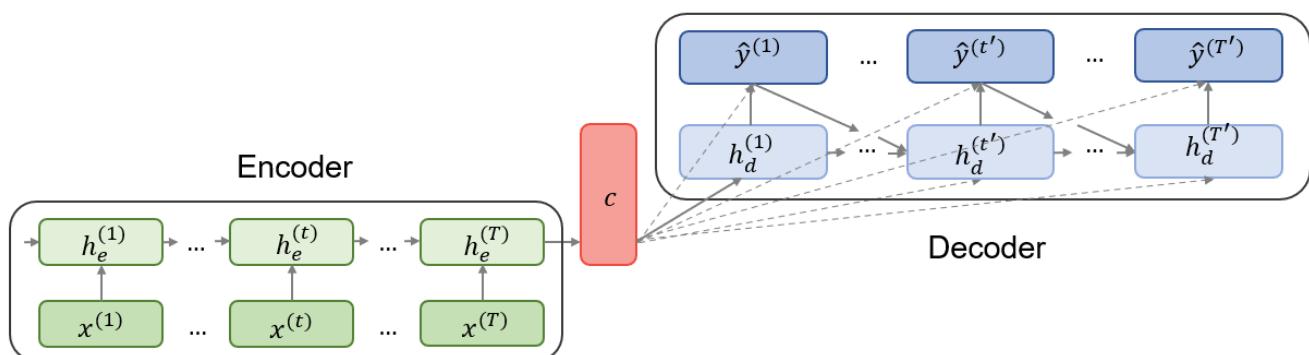
Tìm kiếm chuỗi từ phù hợp nhất yêu cầu chúng ta cần duyệt qua tất cả các chuỗi từ có thể có từ dự đoán của mô hình. Thường thì từ điển của chúng ta sẽ có kích thước rất lớn, với các bài toán về tiếng Việt thì khoảng trên dưới 30.000 từ khác nhau (bao gồm các từ tiếng Anh phổ biến, tiếng Việt thường thì ít hơn) hoặc nếu làm với dữ liệu tiếng Anh thì kích thước bộ từ điển còn lớn hơn nhiều. Khi đó, không gian tìm kiếm của chúng ta là kích thước từ điển lũy thừa với độ dài của chuỗi.

Do không gian tìm kiếm là quá lớn. Trên thực tế, chúng ta thường dùng một thuật toán tìm kiếm heuristic để có được một kết quả tìm kiếm đủ tốt cho mỗi dự đoán thay vì phải tìm kiếm toàn cục.

Mỗi chuỗi từ sẽ được gán một số điểm dựa trên xác suất phân bố của chúng và thuật toán tìm kiếm sẽ dựa trên điểm số này để đánh giá các chuỗi từ này.

6.2 Ý tưởng thuật toán

Trong mô hình cơ bản của Seq2Seq, tại bước giải mã, vector đầu ra của RNN sẽ được đưa qua một hàm softmax để tìm kiếm từ có xác suất xuất hiện lớn nhất. Từ này sẽ là vector đầu vào để dự đoán từ tiếp theo, thế nhưng có một câu hỏi được đặt ra là: "Liệu xác suất của từ cao nhất liệu có luôn là tốt nhất?".



6.2.1 Chọn ra những từ có xác suất lớn nhất

Thuật toán Beam Search có một tham số được gọi là **beam width**. Tại mỗi bước dự đoán, thay vì chọn ra từ có xác suất lớn nhất, ta sẽ chọn ra beam width kết quả có xác suất cao nhất. Ví dụ chúng ta cần dịch một câu từ tiếng Pháp sang tiếng Anh như sau:



- Bước 1: Chọn beam width = 3. Ta sẽ có 3 từ có xác suất lớn nhất:

$$y^{<1>} = \{in, Jane, September\}$$

- Bước 2: Lúc này, xác suất mỗi từ xuất hiện sẽ là $P(y^{<1>} | x)$. Dựa lần lượt những từ này vào bước tiếp theo và tiếp tục chọn ra 3 từ có xác suất lớn nhất:

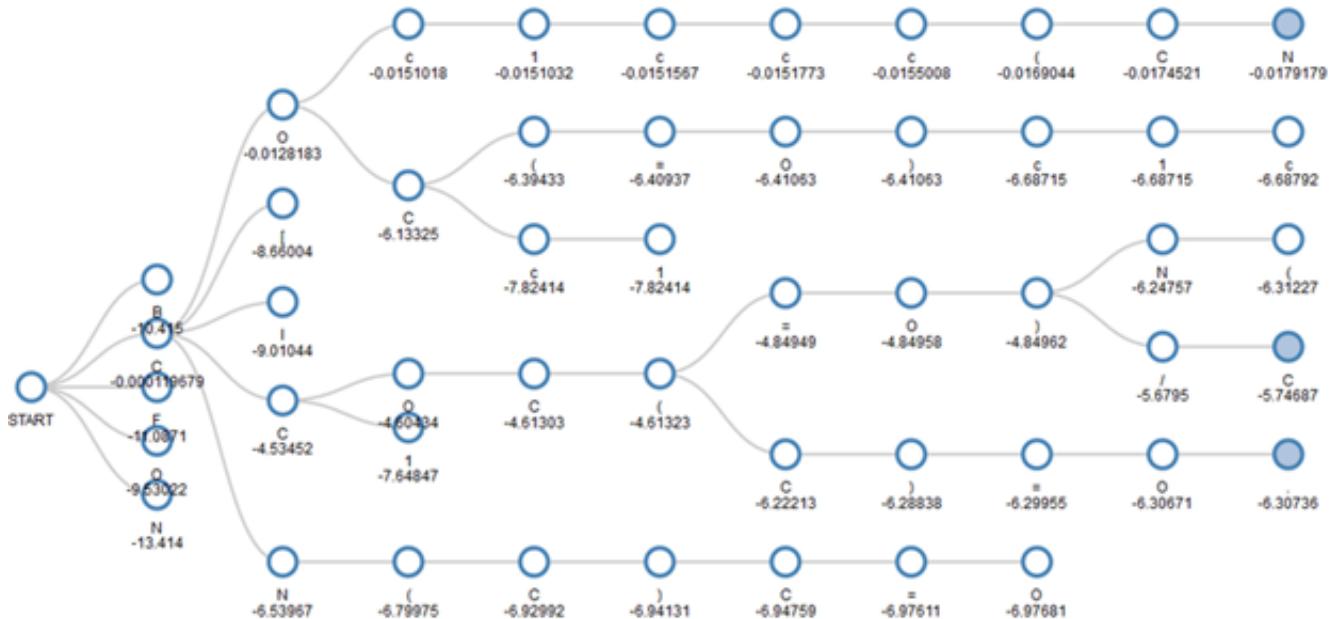
$$\begin{aligned} in \Rightarrow y^{<2>} &= \{September, \dots\} \\ Jane \Rightarrow y^{<2>} &= \{is, visits, \dots\} \end{aligned}$$

- Chúng ta hoàn toàn có thể sử dụng 1 mức threshold để lựa chọn những từ có xác suất đạt chuẩn trong 3 từ kia, nếu không có từ nào thỏa mãn thì chúng ta có thể dừng việc xét nhánh từ đó. Xác suất của chuỗi từ lúc này sẽ là:

$$P(y^{<2>} | x, y^{<1>}) \quad (13)$$

6.2.2 Chọn ra những câu có xác suất lớn nhất

Cứ tiếp tục tính toán các bộ xác suất như vậy cho tới khi kí tự EOS (hết câu) xuất hiện. Lúc này ta sẽ có một cây xác suất:



Công việc lúc này là nhân tất cả các xác suất của cả chuỗi có điều kiện lại:

$$P = P(y^{<1>} | x) P(y^{<2>} | x, y^{<1>}) \dots P(y^{<T_y>} | x, y^{<1>} \dots y^{<T_y-1>}) \quad (14)$$

Output sẽ là câu có kết quả xác suất lớn nhất. Ta sẽ dùng logarit để tính toán vì kết quả xác suất thường nhỏ hơn 1 và nếu câu dài thì kết quả có thể sẽ rất nhỏ.

6.3 Pseudo Code

```

function BEAMDECODE( $c$ ,  $beam\_width$ ) returns best paths
     $y_0, h_0 \leftarrow 0$ 
     $path \leftarrow 0$ 
     $complete\_paths \leftarrow ()$ 
     $state \leftarrow (c, y_0, h_0, path)$  ;initial state
     $frontier \leftarrow \langle state \rangle$  ;initial frontier

    while  $frontier$  contains incomplete paths and  $beamwidth > 0$ 
         $extended\_frontier \leftarrow \langle \rangle$ 
        for each  $state \in frontier$  do
             $y \leftarrow DECODE(state)$ 
            for each word  $i \in Vocabulary$  do
                 $successor \leftarrow NEWSTATE(state, i, y_i)$ 
                 $new\_agenda \leftarrow ADDTOBEAM(successor, extended\_frontier, beam\_width)$ 

            for each  $state$  in  $extended\_frontier$  do
                if  $state$  is complete do
                     $complete\_paths \leftarrow APPEND(complete\_paths, state)$ 
                     $extended\_frontier \leftarrow REMOVE(extended\_frontier, state)$ 
                     $beam\_width \leftarrow beam\_width - 1$ 
                 $frontier \leftarrow extended\_frontier$ 

            return  $completed\_paths$ 

function NEWSTATE( $state$ ,  $word$ ,  $word\_prob$ ) returns new state

function ADDTOBEAM( $state$ ,  $frontier$ ,  $width$ ) returns updated frontier

    if LENGTH( $frontier$ )  $< width$  then
         $frontier \leftarrow INSERT(state, frontier)$ 
    else if SCORE( $state$ )  $>$  SCORE(WORSTOF( $frontier$ ))
         $frontier \leftarrow REMOVE(WORSTOF(frontier))$ 
         $frontier \leftarrow INSERT(state, frontier)$ 
    return  $frontier$ 

```

7 Một số chi tiết thực tế trong xây dựng hệ thống Dịch Máy

7.1 Sự mã hóa (Tokenization)

Tokenization là nhiệm vụ tách văn bản thành các mã thông báo sau đó được chuyển đổi thành số. Những con số này lần lượt được sử dụng bởi các mô hình học máy để xử lý và đào tạo thêm. Tokenization có thể được thực hiện cho các từ hoặc câu riêng biệt. Nếu văn bản được chia thành các từ bằng cách sử dụng một số kỹ thuật tách thì nó được gọi là mã hóa từ và việc tách tương tự được thực hiện cho các câu được gọi là mã hóa câu.

Trong hệ thống dịch máy sẽ sử dụng một tập từ vựng (**Vocabulary**³) cố định và một cách phổ biến để tạo ra tập từ vựng này bằng cách sử dụng mã hóa từ phụ (**Subword Tokenization**). Có rất nhiều loại mã hóa từ phụ nhưng trong quá trình học chúng ta đã đề cập đến hai loại phổ biến đó là **BPE** và **WordPiece**. Tập từ vựng được dùng chung cho cả ngôn ngữ đầu vào và đầu ra với mục đích dễ sao chép mã thông báo từ nguồn tới đích cho nên kho từ vựng (**Lexicon**⁴) WordPiece/BPE được xây dựng trên một kho ngữ liệu có chứa cả dữ liệu ngôn ngữ nguồn và ngôn ngữ đích.

WordPiece là một thuật toán phân đoạn từ khóa con và sử dụng một ký hiệu đặc biệt ở đầu mỗi mã thông báo. Đây là kết quả mã hóa từ hệ thống Google MT (Wu và cộng sự, 2016):

```
words: Jet makers feud over seat width with big orders at stake
wordpieces: _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake
```

Tiếp theo đây mô tả chi tiết quá trình thực thi của thuật toán WordPiece được cung cấp kho ngữ liệu đào tạo và kích thước từ vựng mong muốn V:

- **Bước 1:** Khởi tạo kho đơn vị từ với tất cả các ký tự trong văn bản (tách từ thành chuỗi ký tự).
- **Bước 2:** Xây dựng mô hình ngôn ngữ dựa trên dữ liệu bước 1.
- **Bước 3:** Chọn đơn vị từ mới trong số tất cả các đơn vị có thể làm tăng khả năng xuất hiện trên dữ liệu đào tạo nhiều nhất khi được thêm vào mô hình.
- **Bước 4:** Lặp lại bước 3 cho đến khi đạt đến kích thước từ vựng của từ khóa phụ được xác định trước đó hoặc khả năng tăng giảm xuống dưới một ngưỡng nhất định.

WordPiece đã trở nên phổ biến vì đã được chọn làm tokenizer cho BERT, tiếp theo là Electra. WordPiece có kỹ thuật tương tự như BPE, sử dụng tần suất xuất hiện của các ký tự để xác định các hợp nhất tiềm năng nhưng đưa ra quyết định cuối cùng dựa trên cặp ký tự tối đa hóa khả năng dữ liệu huấn luyện. Việc tìm cặp ký tự ấy được tính bằng cách lấy xác suất của cặp đó chia cho xác suất ký tự đầu tiên và xác suất của ký tự theo sau. Kết quả trả về giá trị lớn nhất trong tất cả các cặp ký tự khác thì hai ký tự ấy sẽ được hợp nhất và thêm vào tập từ vựng. Lấy ví dụ như ký tự "o" theo sau là ký tự "n" sẽ chỉ được hợp nhất nếu như xác suất "on" chia cho "o", "n" sẽ lớn hơn bất kỳ cặp ký hiệu nào khác.

7.2 Kho văn bản của Dịch Máy

7.2.1 Parallel Corpus (kho ngữ liệu song song)

Parallel Corpus là một tập hợp các văn bản gốc và bản dịch sang một hay nhiều ngôn ngữ khác từ ngôn ngữ nguồn tương ứng. Trường hợp đơn giản và phổ biến nhất là kho dữ liệu song ngữ chỉ chứa dữ liệu của hai ngôn ngữ. Một số kho ngữ liệu song song bao gồm:

³Là tất cả các từ được sử dụng bởi một người cụ thể hoặc tất cả các từ tồn tại trong một ngôn ngữ hoặc chủ đề cụ thể

⁴Là danh sách tất cả các từ được sử dụng trong một ngôn ngữ hoặc chủ đề cụ thể, hoặc từ điển

- Ngữ liệu của nhiều ngôn ngữ (Multilingual Languages).
- Ngữ liệu của ngôn ngữ này được dịch trực tiếp từ ngôn ngữ kia hoặc thông qua một ngôn ngữ trung gian khác để dịch (Bilingual Languages).

Ngoài ra thì hướng dịch không cần phải cố định. Có nghĩa là có một số văn bản trong kho ngữ liệu sẽ được dịch từ ngôn ngữ A sang B hoặc một số khác cũng sẽ dịch từ A sang B và dịch ngược lại từ B sang A. Thường sẽ có 3 hướng phổ biến sau:

- Bản dịch một chiều (Unidirectional Translation).
- Bản dịch hai chiều (Bidirectional Translation).
- Bản dịch đa chiều (Multidirectional Translation).

Các mô hình dịch máy được đào tạo trên kho ngữ liệu song song này, hay còn gọi là **Bitext** có chứa một hoặc nhiều ngôn ngữ. Hiện nay có một lượng lớn kho ngữ liệu song song sẵn có để dùng. Một trong số đó tạo từ văn bản của chính phủ như tập ngữ liệu **Europarl** được trích xuất từ các kỷ yếu (proceedings) của Quốc Hội châu Âu gồm 400,000 đến 2 triệu câu từ 21 ngôn ngữ châu Âu hay tập **The United Nations Parallel Corpus** chứa 10 triệu câu bằng sáu ngôn ngữ chính thức của Liên hợp quốc (tiếng Ả Rập, Trung, Anh, Pháp, Nga, Tây Ban Nha). Ngoài ra các kho ngữ liệu song song khác đã được tạo từ phụ đề phim và TV như tập **OpenSubtitles**, hoặc tạo từ văn bản web nói chung như tập **ParaCrawl** với 223 triệu cặp câu giữa 23 ngôn ngữ thuộc khối Liên minh châu Âu và tiếng Anh được trích xuất từ CommonCrawl.

7.2.2 Sentence alignment

Sentence alignment là bài toán làm rõ ràng các mối quan hệ tồn tại giữa các câu của hai văn bản được cho là bản dịch lẫn nhau. Kho ngữ liệu tiêu chuẩn cho việc huấn luyện mô hình dịch máy là các cặp bản dịch tương ứng của mỗi ngôn ngữ phải được đặt song song với nhau hay có thể hiểu là các cặp câu được gióng thảng hàng với nhau. Khi tạo kho ngữ liệu mới cho ngôn ngữ không có sẵn nhiều tư liệu (Under-resourced Languages) hay các lĩnh vực (Domain) mới, điều cần phải làm là tạo các cặp câu song song. Có nhiều mức độ gióng hàng cho kho ngữ liệu như:

- Gióng hàng văn bản (Text Alignment).
- Gióng hàng đoạn văn (Paragraph Alignment).
- Gióng hàng câu (Sentence Alignment).
- Gióng hàng ngữ (Phrase Alignment).
- Gióng hàng từ (Word Alignment).

Dưới đây là một ví dụ mẫu về gióng hàng câu giữa 2 ngôn ngữ Anh và Pháp, với các câu được trích từ Antoine de Saint-Exupery's Le Petit Prince và một bản dịch giả định tương ứng. Quá trình gióng hàng câu được thực hiện bằng cách tìm các tập tối thiểu của các câu là bản dịch của nhau.

E1: "Good morning," said the little prince.	F1: -Bonjour, dit le petit prince.
E2: "Good morning," said the merchant.	F2: -Bonjour, dit le marchand de pilules perfectionnées qui apaisent la soif.
E3: This was a merchant who sold pills that had been perfected to quench thirst.	F3: On en avale une par semaine et l'on n'éprouve plus le besoin de boire.
E4: You just swallow one pill a week and you won't feel the need for anything to drink.	F4: -C'est une grosse économie de temps, dit le marchand.
E5: "They save a huge amount of time," said the merchant.	F5: Les experts ont fait des calculs.
E6: "Fifty-three minutes a week."	F6: On épargne cinquante-trois minutes par semaine.
E7: "If I had fifty-three minutes to spend?" said the little prince to himself.	F7: "Moi, se dit le petit prince, si j'avais cinquante-trois minutes à dépenser, je marcherais tout doucement vers une fontaine..."
E8: "I would take a stroll to a spring of fresh water"	

Gióng hàng câu là một chủ đề nghiên cứu phổ biến trong những ngày đầu của dịch máy thông kê, nhưng ít được chú ý đì khi kho ngữ liệu song song tiêu chuẩn của cặp câu được tạo song song xuất hiện. Tuy nhiên môi quan tâm về thiểu tư liệu dịch máy đã làm cho các phương pháp thu thập dữ liệu trỗi dậy trở lại trong những năm gần đây và các nhà nghiên cứu đã tìm thấy sự hạn chế trong bài toán gióng hàng câu song ngữ. Về cơ bản bài toán gióng thẳng hàng câu tự động được chia làm 2 phần nhỏ:

- **Phần 1:** Hàm chi phí sẽ lấy một hoặc nhiều câu nguồn liền kề nhau và làm tương tự như vậy đối với các câu đích, sau đó trả về một điểm số cho biết khả năng các câu ấy là bản dịch của nhau.
- **Phần 2:** Một thuật toán gióng hàng sẽ lấy những điểm số ở phần trên, cũng đồng thời nhận vào 2 tài liệu và sau đó trả về một bản gióng hàng câu giả định.

Dộ tương tự cosine (**Similarity Cosine**) được lựa chọn sử dụng như một hàm tính điểm để đo độ tương tự giữa các vecto biểu diễn câu đa ngôn ngữ [Artetxe and Schwenk, 2019] vì độ tương tự nhúng câu (Sentence Embedding Similarity) đã được chứng minh là có hiệu quả trong việc lọc ra các câu không song song [Hassan et al., 2018]; [Chaudhary et al., 2019] cũng như là xác định vị trí các câu song song trong kho ngữ liệu có thể so sánh [Guo et al., 2018].

Tuy nhiên, [Thompson and Koehn (2019)] cho thấy rằng hàm điểm được đề xuất của họ hoạt động vượt trội hơn các phương pháp tiên tiến trước đây, và các ước tính thực hiện trong việc gióng hàng có độ chính xác đáng kể. Họ đề xuất phương pháp chuẩn hóa thay vì các vecto biểu diễn được chọn một cách ngẫu nhiên bởi vì phương pháp này có độ phức tạp tuyển tính. Trong sentence alignment, ta sẽ muốn những cặp song song tối thiểu tuy nhiên phương pháp DP với độ tương tự cosine thường ưu tiên tìm ra các cặp nhiều-nhiều chẳng hạn như 3-3 . Để khắc phục vấn đề này, họ chia tỷ lệ chi phí theo số lượng câu nguồn và câu đích đang được xem xét từ tài liệu nguồn và tài liệu đích tương ứng:

$$c(x, y) = \frac{1 - \cos(x, y))}{\sum_{s=1}^S 1 - \cos(x, y_s) + \sum_{s=1}^S 1 - \cos(x_s, y)} \quad (15)$$

Trong đó x, y là một hay nhiều câu liên tiếp từ tài liệu nguồn và đích tương ứng; $\cos(x, y_s)$ là độ tương tự cosine giữa 2 vecto biểu diễn x, y; $nSents()$ biểu thị số lượng câu và x_1, \dots, x_S với y_1, \dots, y_S được lấy mẫu thống nhất từ tài liệu đã cho.

Cuối cùng, sẽ hữu ích nếu thực hiện một số thao tác dọn dẹp ngữ liệu bằng cách loại bỏ các cặp câu gây nhiễu. Điều này có thể liên quan đến các quy tắc viết tay để xóa các cặp có độ chính xác thấp như xóa các câu quá dài, quá ngắn; có các URL khác nhau hoặc thậm chí các cặp quá giống nhau (cho thấy rằng chúng là bản sao chứ không phải bản dịch). Ngoài ra các cặp có thể được xếp hạng theo điểm cosin nhúng đa ngôn ngữ và các cặp có điểm thấp bị loại bỏ.

7.3 Back-translation (Dịch ngược)

Dịch ngược (Back-translation) là một phương pháp để làm giàu kho ngữ liệu huấn luyện. Trong thực tế, chúng ta thường thiếu các kho ngữ liệu song song cho một vài ngôn ngữ hoặc lĩnh vực ít được biết đến, nên ta thường sử dụng một kho ngữ liệu đơn ngữ dồi dào để bổ sung cho kho ngữ liệu nhỏ hơn. Trong kỹ thuật dịch ngược ta có thể dùng kho ngữ liệu đơn ngữ lớn của ngôn ngữ mục tiêu để làm một bước dịch từ ngôn ngữ mục tiêu ra ngôn ngữ nguồn rồi sau đó thêm các cặp đã được dịch này vào kho ngữ liệu song song cho việc dịch từ ngôn ngữ nguồn sang ngôn ngữ mục tiêu. Ví dụ ta có một kho ngữ liệu song song Việt-Anh nhỏ và mục tiêu của ta là dịch từ Việt sang Anh, áp dụng kỹ thuật dịch ngược trước hết ta sẽ dùng một kho ngữ liệu tiếng Anh (vì tiếng Anh rất phổ biến và có nhiều kho ngữ liệu) dịch ra tiếng Việt rồi sau đó thêm lại các cặp câu đã dịch vào kho ngữ liệu song ngữ Việt-Anh ban đầu.

8 Đánh giá mô hình dịch máy

Các bản dịch có thể được đánh giá bằng 2 tiêu chí là: sự đầy đủ và sự trôi chảy:

- **Sự đầy đủ (adequacy):** bản dịch nắm bắt được ý nghĩa chính xác của câu nguồn tốt như thế nào.
- **Sự trôi chảy (fluency):** bản dịch trôi chảy như thế nào trong ngôn ngữ đích (nó có đúng ngữ pháp, rõ ràng không đọc, tự nhiên).

8.1 Sử dụng con người để đánh giá

Để đánh giá một cách chính xác nhất chúng ta sẽ sử dụng con người để đánh giá dựa trên 2 tiêu chí ở trên sử dụng. Thông thường chúng ta sẽ sử dụng các cộng đồng trực tuyến được thuê chỉ dành cho việc đánh giá.

Ví dụ, theo khía cạnh của sự trôi chảy, chúng ta có thể hỏi mức độ dễ hiểu, mức độ rõ ràng, mức độ dễ đọc hoặc mức độ tự nhiên của đầu ra MT (văn bản mục tiêu). Chúng ta có thể cho người đánh giá một thang điểm, ví dụ, từ 1 (hoàn toàn khó hiểu) đến 5 (hoàn toàn dễ hiểu, hoặc 1 đến 100, và yêu cầu họ đánh giá từng câu hoặc đoạn của đầu ra MT.

Chúng ta cũng có thể đánh giá sự trôi chảy cũng dựa trên thang điểm như vậy. Nếu chúng ta có người có thể đánh giá song ngữ ta sẽ đưa họ câu nguồn và câu đích để họ đánh giá. Nếu chúng ta có chỉ có người có thể đánh giá đơn ngữ thì có thể cho họ đối chiếu các bản dịch chuẩn của con người và bản dịch của máy để từ đó đánh giá mức độ thông tin được lưu giữ.

Một cách thứ hai là đánh giá dựa trên việc xếp hạng, có nghĩa là đưa ra hai bản dịch và đánh giá bản dịch nào tốt hơn.

Tuy việc đánh giá bằng con người sẽ cho ra các kết quả chính xác nhất tuy nhiên đây là một công việc tốn nhiều thời gian và công sức cho nên chúng ta sẽ đến với các phương pháp sử dụng các đánh giá tự động, tuy nhiên các phương pháp này sẽ làm giảm độ chính xác xuống một chút nhưng cũng không phải là quá ảnh hưởng đến kết quả đánh giá.

8.2 Phương pháp đánh giá tự động: BLEU

8.2.1 Chỉ số đánh giá BLEU cơ sở

BLEU (Đánh giá song ngữ) là thuật toán phổ biến nhất để đánh giá chất lượng của văn bản đã được dịch bằng máy từ ngôn ngữ tự nhiên này sang ngôn ngữ tự nhiên khác. BLEU cùng với nhiều chỉ số đánh giá khác (NIST, TER, Precision và Recall, và METEOR) dựa trên một ý tưởng đơn giản bắt nguồn từ công trình tiên phong của [Miller and Beebe-Center (1958)]: một bản dịch máy được đánh giá tốt sẽ có xu hướng chứa các từ và ngữ

xuất hiện trong bản dịch của con người trong cùng một câu, hay nói cách khác là bản dịch máy càng gần với bản dịch của con người thì càng tốt.

Điểm được tính cho các phân đoạn được dịch riêng lẻ (thường là các câu) bằng cách xem xét kho ngữ liệu song song, so sánh chúng với một tập các bản dịch đích chất lượng tốt. Những điểm số ấy sẽ được tính trung bình trên toàn bộ ngữ liệu bằng hàm **n-gram precision** với một **brevity penalty** trên toàn bộ ngữ liệu, để đưa ra con số ước tính về chất lượng tổng thể của bản dịch. Tính dễ hiểu (Intelligibility) và tính đúng ngữ pháp (Grammatical Correctness) không được tính vào.

Đầu ra của BLEU luôn là một số trong khoảng từ 0 đến 1. Giá trị này cho biết văn bản ứng cử (**Candidate Text**) tương tự như thế nào với văn bản tham chiếu (**Reference Text**). Các giá trị gần bằng 1 đại diện cho thấy nhiều văn bản tương tự nhau hơn. Rất ít bản dịch của con người có thể đạt điểm 1, bởi vì số 1 sẽ thể hiện bản dịch ứng cử giống hệt với một trong các bản dịch tham chiếu. Chính vì thế không nhất thiết phải đạt điểm 1 do có nhiều cơ hội để so sánh được và việc thêm các bản dịch tham khảo bổ sung sẽ làm tăng điểm BLEU.

Ý tưởng cho thuật toán BLEU trong hình dưới đây: cho 2 bản dịch đề cử của một câu Tây Ban Nha nguồn. Trong đó bản dịch Candidate 1 có chung nhiều cụm n-gram và đặc biệt là các cụm n-gram dài hơn bản dịch Candidate 2 khi đối chiếu với bản dịch tham khảo.

Source

la verdad, cuya madre es la historia, émula del tiempo, depósito de las acciones, testigo de lo pasado, ejemplo y aviso de lo presente, advertencia de lo por venir.

Reference

truth, whose mother is history, rival of time, storehouse of deeds, witness for the past, example and counsel for the present, and warning for the future.

Candidate 1

truth, whose mother is history, voice of time, deposit of actions, witness for the past, example and warning for the present, and warning for the future

Candidate 2

the truth, which mother is the history, émula of the time, deposition of the shares, witness of the past, example and notice of the present, warning of it for coming

8.2.2 Độ chính xác n-gram

Mở rộng số liệu unigram này cho toàn bộ ngữ liệu gồm nhiều câu như sau. Độ chính xác unigram cho một kho ngữ liệu là phần trăm mã thông báo unigram trong bản dịch ứng cử cũng xuất hiện trong bản dịch tham chiếu. Độ chính xác n-gram này được tính cho unigrams, bigrams, và trigrams. Do đó, độ chính xác n-gram $prec_n$ của toàn bộ kho ngữ liệu cho các bản dịch ứng cử là:

$$prec_n = \frac{\sum_{C \in \{ Candidates \}} \sum_{n-gram \in C} Count_{match}(n-gram)}{\sum_{C' \in \{ Candidates \}} \sum_{n-gram' \in C'} Count(n-gram')} \quad (16)$$

Lấy ví dụ như Candidate 1 trong hình trên có 19 unigram duy nhất, một trong số đó xuất hiện nhiều lần nên có tổng số 26 mã thông báo. Trong số này có 16 unigram duy nhất và tổng cộng 23 mã thông báo cũng xuất hiện trong bản dịch tham chiếu (không tính: thoại, chữ ký và hành động). Do đó, độ chính xác unigram cho ngữ liệu Candidate 1 là $23/26 = 0,88$.

8.2.3 Độ chính xác n-gram được sửa đổi

Trong hệ thống MT có thể tạo ra quá nhiều từ trùng lặp với nhau, tạo ra một lỗ hổng khi sử dụng Eq.[16] để tính toán. Các bản dịch đề cử vẫn sẽ nhận một điểm số rất lớn cho các từ lặp lại ấy như hình minh họa dưới đây.

Candidate:	the	the	the	the	the	the	the
Reference 1:	the	cat	is	on	the	mat	
Reference 2:	there	is	a	cat	on	the	mat

Về phía trực quan, vấn đề ở đây rõ ràng là một từ tham chiếu nên được xem như đã đổi chiếu xong sau khi một từ trong Candidate phù hợp, được xác định và không được đếm ra so sánh lại. Nhưng khi áp dụng Eq.[16] trên chúng ta có thể đếm được độ chính xác lúc này của Candidate là $7/7$. Để khắc phục điều này, số lượng cắt bớt và độ chính xác được sửa đổi đã được đề xuất.

Độ chính xác n-gram được thay đổi sẽ tính tương tự cho bất kì n. Đối với mỗi n-gram duy nhất sẽ tính tần suất tối đa của từng n-gram trong mỗi câu tham chiếu. Số đếm đặc biệt nhỏ nhất và số đếm ban đầu được gọi là số đếm được cắt bớt. Có nghĩa là, số lượng được cắt bớt không lớn hơn số lượng ban đầu. Sau đó sử dụng số đếm được cắt bớt này thay cho số lượng ban đầu để tính toán độ chính xác đã sửa đổi.

$$\text{prec}_n = \frac{\sum_{C \in \{\text{Candidates}\}} \sum_{n\text{-gram} \in C} \text{Count}_{\text{match_clipped}}(n\text{-gram})}{\sum_{C' \in \{\text{Candidates}\}} \sum_{n\text{-gram}' \in C'} \text{Count}(n\text{-gram}')} \quad (17)$$

8.2.4 Chi tiết thuật toán BLEU

BLEU lấy giá trị trung bình nhân (**Geometric Mean**) của điểm chính xác đã sửa đổi của kho tài liệu thử nghiệm và sau đó nhân kết quả với hệ số hình phạt ngắn gọn theo cấp số nhân. Hiện tại, quá trình chuyển đổi ký tự hoa/thường (**Case Folding**) là cách chuẩn hóa văn bản duy nhất được thực hiện trước khi tính toán độ chính xác. Cụ thể công thức:

$$\text{BLEU} = \text{BP} \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right) \quad (18)$$

Trong đó, p_n là độ chính xác đã được sửa đổi cho n-grams; \log được tính theo cơ số e; w_n là trọng số trong khoảng (0,1) cho $\log p_n$ và $\sum_{n=1}^N$; và BP là mức phạt ngắn gọn (Brevity Penalty) để phạt các bản dịch máy ngắn.

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ \exp \left(1 - \frac{r}{c} \right) & \text{if } c \leq r \end{cases} \quad (19)$$

Với c là số unigrams (độ dài) trong tất cả các câu ứng cử và r là độ dài phù hợp nhất cho mỗi câu tương ứng trong ngữ liệu. Ở đây độ dài đối sánh tốt nhất là độ dài câu tham chiếu gần nhất với các câu ứng viên.

Thông thường, BLEU được đánh giá hoạt động tốt hơn khi dựa trên ngữ liệu nơi có nhiều câu ứng cử được

dịch từ các văn bản nguồn khác nhau và mỗi câu có một số câu tham khảo. Khi đó c là tổng số unigrams (độ dài) trong tất cả các câu đề xuất và r là tổng độ dài phù hợp nhất cho mỗi câu đề cử trong ngữ liệu. Và đối với BP, mỗi câu ứng cử sẽ được chọn cho câu tham chiếu có độ dài gần nhất để tính toán tham chiếu. Nhưng trên thực tế, hầu hết kho ngữ liệu dịch chỉ có một bản dịch duy nhất của con người để so sánh.

Không khó để nhận thấy rằng BLEU luôn có giá trị từ 0 đến 1. Đó là bởi vì BP, w_n , và p_n luôn nằm trong khoảng từ 0 đến 1, và thậm chí là:

$$\begin{aligned} \exp \left(\sum_{n=1}^N w_n \log p_n \right) &= \prod_{n=1}^N \exp (w_n \log p_n) \\ &= \prod_{n=1}^N [\exp (\log p_n)]^{w_n} \\ &= \prod_{n=1}^N p_n^{w_n} \\ &\in [0, 1] \end{aligned} \tag{20}$$

Không chỉ vậy, mà BLEU cũng thường sử dụng $N = 4$ và $w_n = \frac{1}{N}$. Cuối cùng, việc triển khai BLEU yêu cầu tiêu chuẩn hóa nhiều chi tiết về smoothing và mã hóa. Vì lý do này nên được đề xuất sử dụng các triển khai tiêu chuẩn như SACREBLEU [Post, 2018] thay vì cố gắng triển khai BLEU từ đầu.

8.2.5 Những hạn chế của BLEU

Mặc dù các chỉ số tự động như BLEU rất hữu ích nhưng chúng có những hạn chế cần lưu ý. BLEU rất cục bộ: một cụm từ lớn được di chuyển xung quanh có thể không thay đổi điểm số của BLEU và BLEU không thể đánh giá các thuộc tính câu chéo của một tài liệu như tính mạch lạc diễn ngôn (Discourse Coherence). BLEU và các chỉ số đánh giá tự động tương tự cũng kém hiệu quả trong việc so sánh các loại hệ thống rất khác nhau như so sánh bản dịch có sự hỗ trợ của con người với bản dịch máy hoặc các kiến trúc dịch máy khác nhau với nhau [Callison-Burch et al., 2006]. Các chỉ số đánh giá tự động như vậy có lẽ thích hợp nhất khi đánh giá các thay đổi đối với một hệ thống đơn lẻ.

8.3 Phương pháp dựa trên Embedding

Fương pháp đánh giá BLEU có một nhược điểm là các từ đồng nghĩa hay có thể thay thế được cho các từ trong câu dịch chuẩn lại không được đánh giá một cách chuẩn xác. Một trong các phương pháp đầu tiên được đề xuất để giải quyết vấn đề này là METEOR và gần đây là các phương pháp dựa trên BERT.

Các phương pháp này sẽ nhận đầu vào là các danh sách các tuple có dạng (x, \tilde{x}, r) trong đó $x = (x_1, x_2, \dots, x_n)$ là các câu dịch chuẩn, $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n)$ là các câu dịch máy và $r \in R$ là điểm đánh giá của \tilde{x} so với x . Với dữ liệu như vậy các thuật toán như BLEURT hay BERTSCORE nhận vào x và \tilde{x} và trả về r .

9 Các vấn đề thiên vị và đạo đức trong dịch máy

Dịch máy cũng hay gặp phải các vấn đề về đạo đức và thiên vị. Chẳng hạn trong các ngôn ngữ sử dụng các đại từ không phân biệt giới tính như tiếng Hungary, việc dịch từ tiếng Hungary sang tiếng Anh sẽ vấp phải những nhập nhằng hay thiên kiến đến từ văn hoá.

Hungarian (gender neutral) source	English MT output
ő egy ápoló	she is a nurse
ő egy tudós	he is a scientist
ő egy mérnök	he is an engineer
ő egy pék	he is a baker
ő egy tanár	she is a teacher
ő egy vesküvőszervező	she is a wedding organizer
ő egy vezérigazgató	he is a CEO

Figure 11.19 When translating from gender-neutral languages like Hungarian into English, current MT systems interpret people from traditionally male-dominated occupations as male, and traditionally female-dominated occupations as female (Prates et al., 2019).

Ta có thể thấy rằng nghề nghiệp là y tá (nurse) thì một đại từ trung tính trong tiếng Hung khi dịch sang tiếng Anh sẽ được gán với đại từ chỉ nữ nhưng đáng chú ý hơn là kể cả các nghề nghiệp vốn trung tính như CEO cũng được gán cho đại từ chỉ nam. Các nghiên cứu cũng cho thấy là các thuật toán dịch máy có hiệu suất tệ hơn khi gặp phải các câu trung tính về mặt giới tính hay không theo một thiên kiến xã hội nào.

Khá nhiều vấn đề về đạo đức đã được đưa ra và cần nghiên cứu thêm. Điều cần thiết nhất là cách đánh giá mô hình để tìm ra những điều mô hình chưa thể đánh giá chính xác. Vì trong các trường hợp khẩn cấp thiếu người phiên dịch ta có thể áp dụng các mô hình dịch máy, để tránh việc bản dịch "lầm hại" đến người dùng đặc biệt trong các lĩnh vực y tế ta cần xác định **mức độ tin cậy** với bản dịch để cho người dùng biết rằng các mối nguy hiểm tiềm tàng.

Một vấn đề nữa là sự cần thiết cho một thuật toán dịch máy cần ít tài nguyên và có thể dịch nhiều ngôn ngữ khác nhau. Điều này cũng đưa đến một vấn đề khác là các nghiên cứu thường tập trung vào tiếng Anh mà thiếu đi các nghiên cứu dành cho các ngôn ngữ thiểu số khác.

10 Demo code

Ta import các thư viện cần thiết trong quá trình training như:

- + Từ __future__: ta import các thư viện con như:
 - + unicode_literals:
 - + print_function:
 - + division:
- + Từ io: ta import thư viện con:
 - + open: để mở file data có đuôi dạng “txt”, “tsv”
- + unicodedata:
- + string:
- + re:
- + random:

+ Từ thư viện torch: ta import thư viện con như:

- + optim:
 - + nn: từ torch.nn ta import các hàm trong nó để thực hiện các lệnh cần làm.
- VD: functional (torch.nn.functional)

Sau đó ta sẽ dùng biến tên “device” để chọn chạy code bằng nhân của cpu hay nhân cuda của card rồi thông qua kiểm tra xem có card rời trong máy hay không nếu không có thì ta sẽ sử dụng nhân của cpu để chạy code.

Ta sẽ chuẩn bị file dữ liệu để train cho mô hình dưới đây. File dữ liệu sẽ là dịch câu tiếng Anh sang câu tiếng Việt với tên file là vi-en.tsv

Với function đầu tiên:

```
1 SOS_token = 0
2 EOS_token = 1
3
4 class Lang:
5     def __init__(self, name):
6         self.name = name
7         self.word2index = {}
8         self.word2count = {}
9         self.index2word = {0: "SOS", 1: "EOS"}
10        self.n_words = 2 #Count SOS and EOS
11
12    def addSentence(self, sentence):
13        for word in sentence.split(' '):
14            self.addWord(word)
15
16    def addWord(self, word):
17        if word not in self.word2index:
18            self.word2index[word] = self.n_words
19            self.word2count[word] = 1
20            self.index2word[self.n_words] = word
21            self.n_words += 1
22        else:
23            self.word2count[word] += 1
```

Function trên, ta sẽ trích từng từ trong câu được đặt vào trong function và sau đó đếm số lần xuất hiện của nó. Ở function “__init__” tạo 1 loại dữ liệu cho biến có tên “name” (là tên theo biến được gọi ở các hàm khác, ví dụ như input_lang = Lang(lang2). Ở đây “lang2” là cái “name” và “lang2” chứa các thông tin như: “name”, “word2index : dict”, “word2count: dict”, “index2word: dict”, “n_words: int”). Hàm “addSentence” có nhiệm vụ tách Sentence được đưa vào thành các từ riêng biệt và gọi hàm “addWord” để tìm xem từ được tách có xuất hiện trong câu hiện tại hay chưa. Với function “addWord”, nó sẽ kiểm tra xem từ đó có xuất hiện trong “word2index” hay chưa. Nếu chưa thì từ đó sẽ được thêm vào trong word2index và khởi tạo các dữ liệu như sau:

- + word2index[“từ đang kiểm tra”] = n_words: với n_words là từ thứ n trong dữ liệu ta lưu. Sở dĩ nó bằng 2 là vì trước đó ta đã lưu “SOS” và “EOS” như là 2 từ nên “từ đang kiểm tra” sẽ đứng kế tiếp các từ này.
- + word2count[“từ đang kiểm tra”] = 1: vì đây là lần đầu nó xuất hiện nên default sẽ là 1.
- + index2word[“vị trí của “từ đang kiểm tra” trong kiểu dữ liệu này”] = “từ đang kiểm tra”: ta thêm “từ đang kiểm tra” vào trong biến để sau đó nếu nó xuất hiện ta chỉ việc count nó lên.

+ `n_words += 1`: update vị trí cho từ tiếp theo nếu từ tiếp theo được thêm vào kiểu dữ liệu này.

Nếu “từ đang kiểm tra” đã có trong “word2index” ta chỉ cần tăng số lần xuất hiện của nó thêm 1 đơn vị.

+ `word2count["từ đang kiểm tra"] += 1`

Sau khi thực thi hết câu, ta sẽ được data là 1 dict chứa các từ trong câu và số lần xuất hiện của từng từ trong câu.

Function dưới đây sẽ đọc file data ta đã chuẩn bị trước. Với từng hàng trong file data, mỗi hàng sẽ có hai câu với hai thứ tiếng, và hai câu trong hàng sẽ được tách thành mảng con [câu 1, câu 2] lưu vào trong mảng cha có tên là “pairs”. Trong quá trình sẽ tách từng hàng và trong từng hàng thì tách thành 2 câu, ta sẽ normalize câu đó lại thành câu chỉ toàn kí tự thường, xoá các khoảng trắng không cần thiết và thay các kí tự như “!?” Thành “\ ” để dễ quản lý và tạo sự đồng nhất để thực thi các function sau thông qua việc gọi hàm “normalizeString” theo từng câu được tách trong từ hàng trong file data.

```
1 def normalizeString(s):
2     s = s.lower().strip()
3     s = re.sub(r"([.!?])", r" \1", s)
4     return s
5
6 def readLangs(lang1, lang2, reverse=False):
7     print("Reading lines...")
8
9     #Read the file and split into lines
10    lines = open('%s-%s.tsv' % (lang1, lang2), encoding = 'utf-8').
11        read().strip().split('\n')
12
13    #Split every line into pairs and normalize
14    pairs = [[normalizeString(s) for s in l.split('\t')] for l in lines]
15
16    #Reverse pairs, make Lang instances
17    if reverse:
18        pairs = [list(reversed(p)) for p in pairs]
19        input_lang = Lang(lang2)
20        output_lang = Lang(lang1)
21    else:
22        input_lang = Lang(lang1)
23        output_lang = Lang(lang2)
24
25    return input_lang, output_lang, pairs
```

Sau khi thực hiện function trên thì:

+ “`input_lang`” sẽ là câu cần translate trong file data.

+ “`output_lang`” sẽ là câu kết quả chính xác trong file data.

+ “`pairs`” là 1 mảng 2 chiều với từng chiều mang 2 câu trên cùng 1 dòng trong file data.

Và các câu được lưu trong cả 3 kiểu dữ liệu trên đều đã được normalize.

Lưu ý phần điều kiện `reverse` chỉ để cho biết rằng ta đang muốn translate giữa 2 ngôn ngữ từ ngôn ngữ nào sang ngôn ngữ nào mà thôi.

Tiếp theo ta sẽ đặt điều kiện cho độ dài của câu vì nếu câu quá dài, thời gian train sẽ lâu. Ở đây ta sẽ đặt điều kiện độ dài trung bình cho câu là 10, như vậy thì dữ liệu trong câu sẽ không quá phức tạp, ta sẽ train nhanh

hơn một chút. Đồng thời ta modify lại các cách viết tắt cho các từ trong tiếng Anh. Ta sẽ modify lại mảng 2 chiều “pairs”, chỉ giữ lại các cặp câu có độ dài dưới 10.

```
1 MAX_LENGTH = 10
2
3 eng_prefixes = (
4     "i am ", "i'm ",
5     "he is", "he's ",
6     "she is", "she's ",
7     "you are", "you're ",
8     "we are", "we're ",
9     "they are", "they're "
10 )
11
12 def filterPair(p):
13     return len(p[0].split(' ')) < MAX_LENGTH and
14         len(p[1].split(' ')) < MAX_LENGTH
15
16 def filterPairs(pairs):
17     return [pair for pair in pairs if filterPair(pair)]
```

Function kế tiếp chỉ có việc gọi lại các hàm đã kể trên và hiển thị thông tin, quá trình sau khi data đã được chuẩn bị.

```
1 def prepareData(lang1, lang2, reverse=False):
2     input_lang, output_lang, pairs = readLangs(lang1, lang2, reverse)
3     print("Read %s sentence pairs" % len(pairs))
4     pairs = [[p[1], p[3]] for p in pairs]
5     pairs = filterPairs(pairs)
6     print("Counting words...")
7
8     for pair in pairs:
9         input_lang.addSentence(pair[0])
10        output_lang.addSentence(pair[1])
11    print("Counted words:")
12    print(input_lang.name, input_lang.n_words)
13    print(output_lang.name, output_lang.n_words)
14    return input_lang, output_lang, pairs
15
16 input_lang, output_lang, pairs = prepareData('vi', 'en', False)
17 print(random.choice(pairs))
```

Kết quả sau khi thực hiện như sau:

```

 Reading lines...
Read 11356 sentence pairs
Counting words...
Counted words:
vi 2421
en 3394
['góp gió thành bão .', 'many a little makes a mickle .']

```

Tiếp theo ta sẽ tìm hiểu về mô hình seq2seq.

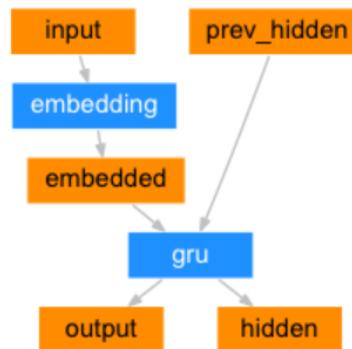
Mạng Sequence to Sequence, hoặc mạng seq2seq, hoặc mạng Encoder-Decoder, là một mô hình bao gồm hai RNN được gọi là bộ mã hóa (encoder) và bộ giải mã (decoder). Bộ mã hóa (encoder) đọc một chuỗi đầu vào và xuất ra một vectơ duy nhất, và bộ giải mã (decoder) đọc vectơ đó để tạo ra một chuỗi đầu ra.

Không giống như dự đoán trinh tự với một RNN duy nhất, trong đó mọi đầu vào tương ứng với một đầu ra, mô hình seq2seq không cần phải quá lo về độ dài câu và thứ tự của các từ trong câu, điều này chứng minh rằng seq2seq là mô hình lý tưởng cho việc dịch ngôn ngữ.

Với mô hình seq2seq, bộ mã hóa (encoder) tạo ra một vectơ duy nhất, trong trường hợp lý tưởng, mã hóa “ý nghĩa” của chuỗi đầu vào thành một vectơ duy nhất - một điểm duy nhất trong không gian N chiều của câu.

10.1 The Encoder (Bộ mã hóa)

Bộ mã hóa (encoder) của mạng seq2seq là một RNN xuất ra giá trị cho mỗi từ từ câu đầu vào. Đối với mỗi từ đầu vào, bộ mã hóa (encoder) xuất ra một vectơ và một trạng thái ẩn (hidden state), đồng thời sử dụng trạng thái ẩn (hidden state) cho từ đầu vào tiếp theo.



```

1 class EncoderRNN(nn.Module):
2     def __init__(self, input_size, hidden_size):
3         super(EncoderRNN, self).__init__()
4         self.hidden_size = hidden_size

```

```

5     self.embedding = nn.Embedding(input_size, hidden_size)
6     self.gru = nn.GRU(hidden_size, hidden_size)
7
8
9     def forward(self, input, hidden):
10        embedded = self.embedding(input).view(1, 1, -1)
11        output = embedded
12        output, hidden = self.gru(output, hidden)
13        return output, hidden
14
15    def initHidden(self):
16        return torch.zeros(1, 1, self.hidden_size, device = device)

```

Trong class này, ta sẽ định nghĩa một số biến trong phần `__init__` để thực hiện tính toán ở các hàm tiếp theo, trong đó:

- + “input_size” là một số lượng các từ trong câu được đưa vào.
- + “hidden_size” là tham số ta cho trước, là số lượng node ở lớp hidden.
- + “embedding” là ẩn dữ liệu ban đầu dưới dạng một không gian vectơ số mới.
- + “gru” là 1 lớp ẩn.

Trong “forward” chúng ta cài đặt các thao tác chuyển tiếp của mạng, trong trường hợp này (bộ mã hoá), chúng ta nhúng từ đầu vào (embedding) và chuyển nó sang lớp GRU. “initHidden” trả về một tensor chứa giá trị vô hướng với 1 hàng, `hidden_size` cột.

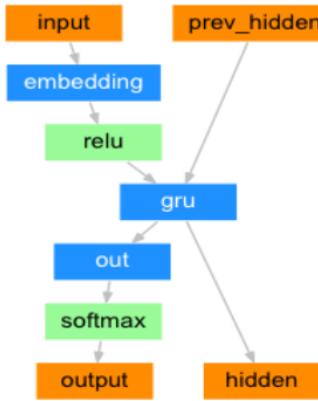
10.2 The decoder (Bộ giải mã)

Bộ giải mã là một RNN khác, nó lấy các vector đầu ra của bộ mã hoá (encoder) và xuất ra một chuỗi các từ để tạo bản dịch.

10.2.1 Simple Decoder

Chúng ta chỉ sử dụng đầu ra cuối cùng của bộ mã hoá. Và đầu ra cuối cùng này đôi khi là 1 vector ngữ cảnh (context vector) vì nó mã hoá ngữ cảnh của cả câu. Vector ngữ cảnh này được sử dụng làm trạng thái ẩn ban đầu của bộ giải mã (decoder).

Ở từng bước giải mã, bộ giải mã (decoder) được cấp một token đầu vào và trạng thái ẩn (hidden state). Token đầu vào ban đầu là token SOS bắt đầu của chuỗi và trạng thái ẩn (hidden state) đầu tiên là vector ngữ cảnh (context vector) (trạng thái ẩn (hidden state) cuối cùng của bộ mã hoá (encoder)).



```

1  class DecoderRNN(nn.Module):
2      def __init__(self, hidden_size, output_size):
3          super(DecoderRNN, self).__init__()
4          self.hidden_size = hidden_size
5
6          self.embedding = nn.Embedding(output_size, hidden_size)
7          self.gru = nn.GRU(hidden_size, hidden_size)
8          self.out = nn.Linear(hidden_size, output_size)
9          self.softmax = nn.LogSoftmax(dim = 1)
10
11     def forward(self, input, hidden):
12         output = self.embedding(input).view(1, 1, -1)
13         output = F.relu(output)
14         output, hidden = self.gru(output, hidden)
15         output = self.softmax(self.out(output[0]))
16         return output, hidden
17
18     def initHidden(self):
19         return torch.zeros(1, 1, self.hidden_size, device = device)

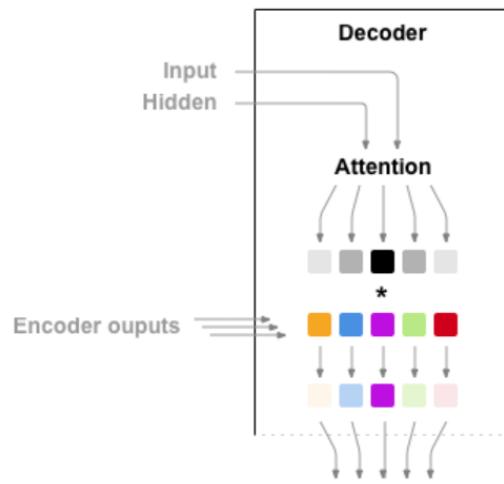
```

Với Encoder (bộ giải mã), nó không khác class Decoder quá nhiều, chỉ khác ở đầu ra output là softmax.

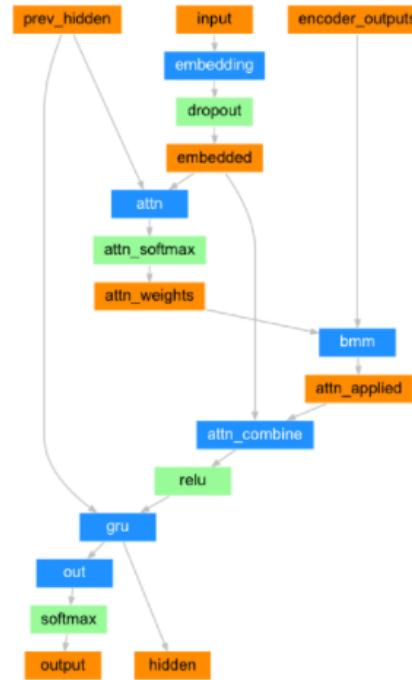
10.2.2 Cơ chế Attention (Attention Mechanism)

Nếu chỉ có vector ngữ cảnh được truyền giữa bộ mã hoá (encoder) và bộ giải mã (decoder), vector ngữ cảnh đó sẽ thực hiện cả việc mã hoá toàn bộ câu. Đó sẽ là 1 gánh nặng cho vector đó.

Cơ chế Attention cho phép mạng bộ giải mã (decoder network) “tập trung” vào một phần khác của đầu ra của bộ mã hoá (encoder) cho mỗi bước của đầu ra riêng của bộ giải mã (decoder). Trước hết, ta tính toán một tập hợp các trọng số Attention (**Attention Weights**). Chúng sẽ được nhân với các vector đầu ra của bộ mã hoá (encoder) để tạo ra một tổ hợp có trọng số (**Weighted Combination**). Kết quả (được gọi là “attn_applied” trong phần code) phải chứa thông tin về phần cụ thể đó của chuỗi đầu vào và do đó giúp bộ giải mã (decoder) chọn các từ đầu ra phù hợp.



Việc tính toán trọng số attention được thực hiện với một **attn** lớp chuyển tiếp khác, sử dụng đầu vào của bộ giải mã (decoder) và trạng thái ẩn (hidden state) làm đầu vào. Vì các câu có độ dài khác nhau trong tập data huấn luyện, để tạo và train lớp này, chúng ta phải chọn độ dài câu tối đa (độ dài đầu vào, cho đầu ra bộ mã hóa (encoder)) mà nó có thể áp dụng. Các câu có độ dài tối đa sẽ sử dụng tất cả các trọng số attention, trong khi các câu ngắn hơn sẽ chỉ sử dụng vài câu đầu tiên.



```

1 class AttnDecoderRNN(nn.Module):
2     def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
3         super(AttnDecoderRNN, self).__init__()

```

```

4     self.hidden_size = hidden_size
5     self.output_size = output_size
6     self.dropout_p = dropout_p
7     self.max_length = max_length
8
9     self.embedding = nn.Embedding(self.output_size, self.hidden_size)
10    self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
11    self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
12    self.dropout = nn.Dropout(self.dropout_p)
13    self.gru = nn.GRU(self.hidden_size, self.hidden_size)
14    self.out = nn.Linear(self.hidden_size, self.output_size)
15
16  def forward(self, input, hidden, encoder_outputs):
17      embedded = self.embedding(input).view(1, 1, -1)
18      embedded = self.dropout(embedded)
19
20      attn_weights = F.softmax(self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim = 1)
21      attn_applied = torch.bmm(attn_weights.unsqueeze(0), encoder_outputs.unsqueeze(0))
22      output = torch.cat((embedded[0], attn_applied[0]), 1)
23      output = self.attn_combine(output).unsqueeze(0)
24
25      output = F.relu(output)
26      output, hidden = self.gru(output, hidden)
27
28      output = F.log_softmax(self.out(output[0]), dim=1)
29      return output, hidden, attn_weights
30
31  def initHidden(self):
32      return torch.zeros(1, 1, self.hidden_size, device = device)

```

10.2.3 Training model

Ta phải modify lại tập dữ liệu để thuận tiện cho việc training. Ý tưởng là đối với mỗi cặp, ta sẽ cần một tensor đầu vào (vị trí của các từ trong câu đầu vào) và tensor mục tiêu (vị trí của các từ trong câu đích). Trong khi tạo các vector này, chúng tôi sẽ nối mã thông báo EOS vào cả hai chuỗi.

```

1  def indexesFromSentence(lang, sentence):
2      return [lang.word2index[word] for word in sentence.split(' ')]
3
4  def tensorFromSentence(lang, sentence):
5      indexes = indexesFromSentence(lang, sentence)
6      indexes.append(EOS_token)
7      return torch.tensor(indexes, dtype=torch.long, device=device).view(-1, 1)
8
9  def tensorsFromPair(pair):
10     input_tensor = tensorFromSentence(input_lang, pair[0])
11     target_tensor = tensorFromSentence(output_lang, pair[1])
12     return (input_tensor, target_tensor)

```

Function “indexesFromSentence” sẽ trả về một mảng (vector) 1 chiều với từng phần tử trong mảng là thứ tự của các từ trong câu trong tập dữ liệu “lang”.

Function “tensorFromSentence” trả về tensor chứa data của indexes có n chiều.

Function “tensorFromPair” trả về 1 tuple bao gồm tensor đầu vào và tensor đầu ra sau khi modify pair.

Để huấn luyện mô hình, ta chạy câu đầu vào thông qua bộ mã hóa (encoder) và theo dõi mọi đầu ra cũng như trạng thái ẩn (hidden state) mới nhất. Sau đó, bộ giải mã (decoder) được cấp mã thông báo SOS làm đầu vào đầu tiên và trạng thái ẩn (hidden state) cuối cùng của bộ mã hóa (encoder) là trạng thái ẩn (hidden state) đầu tiên của nó.

“Teacher forcing” là khái niệm sử dụng đầu ra mục tiêu thực làm mỗi đầu vào tiếp theo, thay vì sử dụng prediction của bộ giải mã (decoder) làm đầu vào tiếp theo. Sử dụng “Teacher forcing” làm cho nó hội tụ nhanh hơn nhưng khi mạng được đào tạo được khai thác, nó có thể biểu hiện sự không ổn định.

Ta có thể quan sát kết quả đầu ra của mạng do “Teacher” đọc với ngữ pháp mạch lạc nhưng nghĩa lại khá khác so với bản dịch chính xác - trực giác nó đã học cách đại diện cho ngữ pháp đầu ra và có thể “hiểu” ý nghĩa khi “teacher” nói với nó vài từ đầu tiên, nhưng Nó đã không học đúng cách tạo câu từ bản dịch ngay từ đầu.

Do sự tự do mà PyTorch’s autograd mang lại cho ta, chúng ta có thể ngẫu nhiên chọn sử dụng lệnh “Teacher forcing” hoặc không bằng một câu lệnh if đơn giản. Bật “teacher_forcing_ratio” để sử dụng nó nhiều hơn.

```
1 teacher_forcing_ratio = 0.5
2
3 def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer,
4           criterion, max_length=MAX_LENGTH):
5     encoder_hidden = encoder.initHidden()
6
7     encoder_optimizer.zero_grad()
8     decoder_optimizer.zero_grad()
9
10    input_length = input_tensor.size(0)
11    target_length = target_tensor.size(0)
12
13    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)
14
15    loss = 0
16
17    for ei in range(input_length):
18        encoder_output, encoder_hidden = encoder(
19            input_tensor[ei], encoder_hidden)
20        encoder_outputs[ei] = encoder_output[0, 0]
21
22    decoder_input = torch.tensor([[SOS_token]], device=device)
23
24    decoder_hidden = encoder_hidden
25
26    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
27
28    if use_teacher_forcing:
29        #Teacher forcing: Feed the target as the next input
30        for di in range(target_length):
31            decoder_output, decoder_hidden, decoder_attention = decoder(
32                decoder_input, decoder_hidden, encoder_outputs)
33            loss += criterion(decoder_output, target_tensor[di])
34            decoder_input = target_tensor[di] #Teacher forcing
```

```

35     else:
36         #Without teacher forcing: use its own predictions as the next input
37         for di in range(target_length):
38             decoder_output, decoder_hidden, decoder_attention = decoder(
39                 decoder_input, decoder_hidden, encoder_outputs)
40             topv, topi = decoder_output.topk(1)
41             decoder_input = topi.squeeze().detach() #Detach from history as input
42
43             loss += criterion(decoder_output, target_tensor[di])
44             if decoder_input.item() == EOS_token:
45                 break
46
47         loss.backward()
48
49         encoder_optimizer.step()
50         decoder_optimizer.step()
51
52     return loss.item() / target_length

```

Như vậy, sau khi ta train model hàm sẽ trả về tỉ lệ mất mát sau khi train và tự ghi nhớ các kết quả trong quá trình train như 1 mạng nơron.

Ta sẽ đo đặc thời gian huấn luyện để có thể hiểu rõ hơn sự nhanh hay chậm của mô hình này thông qua hàm dưới đây.

```

1 import time
2 import math
3
4 def asMinutes(s):
5     m = math.floor(s / 60)
6     s -= m * 60
7     return '%dm %ds' % (m, s)
8
9 def timeSince(since, percent):
10    now = time.time()
11    s = now - since
12    es = s / (percent)
13    rs = es - s
14    return '%s (- %s)' % (asMinutes(s), asMinutes(rs))

```

Và đương nhiên, ta cần phải có 1 hàm “trainIters” để thực hiện việc gọi hàm “Train” ta đã nói ở trên và in các thông tin sau khi “Train” để ta có thể nắm rõ tình hình “Train” và độ hiệu quả của mô hình ta đang sử dụng.

```

1 def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100, learning_rate=0.01):
2     start = time.time()
3     plot_losses = []
4     print_loss_total = 0 #Reset every print_every
5     plot_loss_total = 0 #Reset every plot_every
6
7     encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
8     decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
9     training_pairs = [tensorsFromPair(random.choice(pairs))
10                     for i in range(n_iters)]
11     criterion = nn.NLLLoss()

```

```

12
13     for iter in range(1, n_iters + 1):
14         training_pair = training_pairs[iter - 1]
15         input_tensor = training_pair[0]
16         target_tensor = training_pair[1]
17
18         loss = train(input_tensor, target_tensor, encoder,
19                      decoder, encoder_optimizer, decoder_optimizer, criterion)
20         print_loss_total += loss
21         plot_loss_total += loss
22
23         if iter % print_every == 0:
24             print_loss_avg = print_loss_total / print_every
25             print_loss_total = 0
26             print('%s (%d %d%%) %.4f' % (timeSince(start, iter / n_iters),
27                                         iter, iter / n_iters * 100, print_loss_avg))
28
29         if iter % plot_every == 0:
30             plot_loss_avg = plot_loss_total / plot_every
31             plot_losses.append(plot_loss_avg)
32             plot_loss_total = 0
33
34     showPlot(plot_losses)

```

Kết quả sau khi gọi hàm trên từng lần là:



```

1m 43s (- 24m 4s) (5000 6%) 4.4101
3m 21s (- 21m 48s) (10000 13%) 3.6988
5m 0s (- 20m 0s) (15000 20%) 3.1222
6m 40s (- 18m 22s) (20000 26%) 2.7100
8m 23s (- 16m 46s) (25000 33%) 2.3437
10m 5s (- 15m 8s) (30000 40%) 2.0156
11m 49s (- 13m 30s) (35000 46%) 1.7594
13m 33s (- 11m 51s) (40000 53%) 1.5535
15m 16s (- 10m 11s) (45000 60%) 1.3720
17m 1s (- 8m 30s) (50000 66%) 1.2268
18m 46s (- 6m 49s) (55000 73%) 1.0828
20m 29s (- 5m 7s) (60000 80%) 0.9968
22m 14s (- 3m 25s) (65000 86%) 0.8972
23m 58s (- 1m 42s) (70000 93%) 0.8203
25m 43s (- 0m 0s) (75000 100%) 0.7862

```

(Thời gian sau khi hoàn thành train | Dự đoán khoảng thời gian train xong | Số lượng Iters – phần trăm số lượng iter đã train so với tổng số iter có | Độ mất mát)

Ta sẽ dùng thư viện matplotlib để phác thảo độ mất mát sau khi huấn luyện dưới dạng đồ thị để thuận tiện cho việc so sánh.

```

1 import matplotlib.pyplot as plt
2 plt.switch_backend('agg')
3 import matplotlib.ticker as ticker
4 import numpy as np

```

```

5
6 def showPlot(points):
7     plt.figure()
8     fig, ax = plt.subplots()
9     #This locator puts ticks at regular intervals
10    loc = ticker.MultipleLocator(base=0.2)
11    ax.yaxis.set_major_locator(loc)
12    plt.plot(points)

```

Hàm “Evaluation” khá giống hàm “Train”, nhưng hàm này không có mục tiêu cụ thể, vì thế ta chỉ cần cung cấp các prediction của bộ giải mã lại cho từng bước. Mỗi lần nó dự đoán (predict) một từ, ta sẽ thêm từ đó vào chuỗi đầu ra và nếu như nó dự đoán EOS Token, chúng ta sẽ dừng tại đó. Đồng thời ta lưu đầu ra của decoder’s attention để hiển thị sau này.

```

1 def evaluate(encoder, decoder, sentence, max_length=MAX_LENGTH):
2     with torch.no_grad():
3         input_tensor = tensorFromSentence(input_lang, sentence)
4         input_length = input_tensor.size()[0]
5         encoder_hidden = encoder.initHidden()
6
7         encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)
8
9         for ei in range(input_length):
10             encoder_output, encoder_hidden = encoder(input_tensor[ei],
11                                             encoder_hidden)
12             encoder_outputs[ei] += encoder_output[0, 0]
13
14         decoder_input = torch.tensor([[SOS_token]], device=device) #SOS
15
16         decoder_hidden = encoder_hidden
17
18         decoded_words = []
19         decoder_attentions = torch.zeros(max_length, max_length)
20
21         for di in range(max_length):
22             decoder_output, decoder_hidden, decoder_attention = decoder(
23                 decoder_input, decoder_hidden, encoder_outputs)
24             decoder_attentions[di] = decoder_attention.data
25             topv, topi = decoder_output.data.topk(1)
26             if topi.item() == EOS_token:
27                 decoded_words.append('<EOS>')
28                 break
29             else:
30                 decoded_words.append(output_lang.index2word[topi.item()])
31
32             decoder_input = topi.squeeze().detach()
33
34     return decoded_words, decoder_attentions[:di + 1]

```

Ngoài ra, sau khi train, ta vẫn có thể đánh giá (evaluate) mô hình bằng cách dịch các câu ngẫu nhiên được lấy từ tập dữ liệu train, sau đó in ra câu nguyên mẫu, câu dịch chính xác đi cùng câu nguyên mẫu, và câu được dịch ra từ mô hình để có thể đánh giá mô hình một cách khách quan hơn.

```

1 def evaluateRandomly(encoder, decoder, n=10):

```

```
2 for i in range(n):
3     pair = random.choice(pairs)
4     print('>', pair[0])
5     print('=', pair[1])
6     output_words, attentions = evaluate(encoder, decoder, pair[0])
7     output_sentence = ' '.join(output_words)
8     print('<', output_sentence)
9     print('
```

Và như vậy đã xong, ta sẽ thực hiện việc train dữ liệu bằng cách gọi các câu lệnh dưới đây. Ta sẽ mặc định cho số lớp ẩn là 256 lớp và kích hoạt các hàm EncoderRNN và AttnDecoderRNN để nhận diện và modify lại lượng dữ liệu có trong tập train và gọi hàm “trainIters” để bắt đầu train.

```
1 hidden_size = 256
2 encoder1 = EncoderRNN(input_lang.n_words, hidden_size).to(device)
3 attn_decoder1 = AttnDecoderRNN(hidden_size, output_lang.n_words, dropout_p=0.1).to(device)
4
5 trainIters(encoder1, attn_decoder1, 75000, print_every=5000)
```

Ta có thể lấy vài câu ngẫu nhiên từ trong tập train ra và đưa vào model đã được train để xem kết quả train có tốt hay không hoặc ta có thể tự input 1 câu của ta và để tập train đưa ra kết quả dịch.

Đối với lấy các câu ngẫu nhiên ta sẽ thực thi câu lệnh sau:

```
1 evaluateRandomly(encoder1, attn_decoder1)
```

Kết quả sau khi ta chạy dòng lệnh trên:

👤 > tôi chắc chắn là tom sẽ chiến thắng .
= i'm certain that tom will win .
< i'm certain tom will win win . <EOS>

> bob nghĩ rất nhiều về vấn đề đó .
= bob thought deeply about that matter .
< you should be a problem . <EOS>

> tôi không muốn thổ lộ tình cảm .
= i don't want to let my emotions out .
< i don't want to let my emotions out . <EOS>

> xui quá !
= what rotten luck !
< what rotten luck ! <EOS>

Đối với câu do ta tự đưa vào sẽ có câu lệnh sau:

```
1 output_words, attentions = evaluate(
2     encoder1, attn_decoder1, " ") # " " is a place for us put a sentence with a punctuation
3     print(output_words)
```

Kết quả sau khi ta chạy dòng lệnh trên:

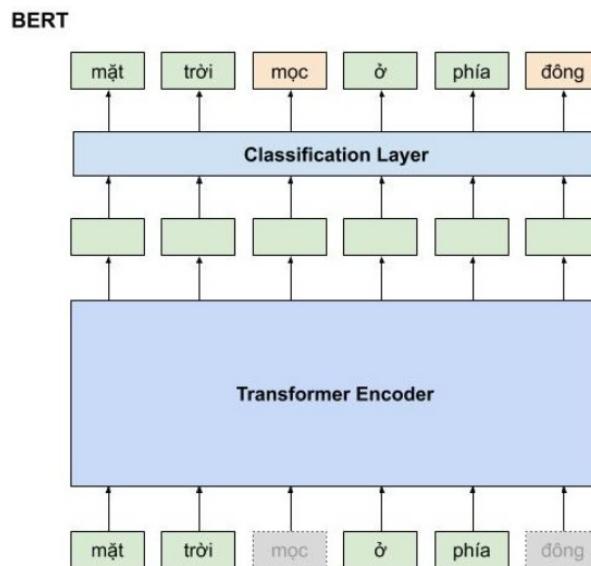
```
▶ output_words, attentions = evaluate(  
    encoder1, attn_decoder1, "anh ãy đang bị mắc kẹt .")  
print(output_words)
```

```
👤 ["he's", 'going', 'to', 'be', '.', '<EOS>']
```

11 Cải tiến mô hình dựa trên Transformer

11.1 Giới thiệu Transformer

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 cộng đồng NLP trong năm 2019. Và chính Google cũng đã áp dụng BERT trong cỗ máy tìm kiếm của họ. Để hiểu BERT, chúng ta cần phải nắm rõ về mô hình Transformer.



Ý tưởng chủ đạo của Transformer vẫn là áp dụng cơ chế 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.

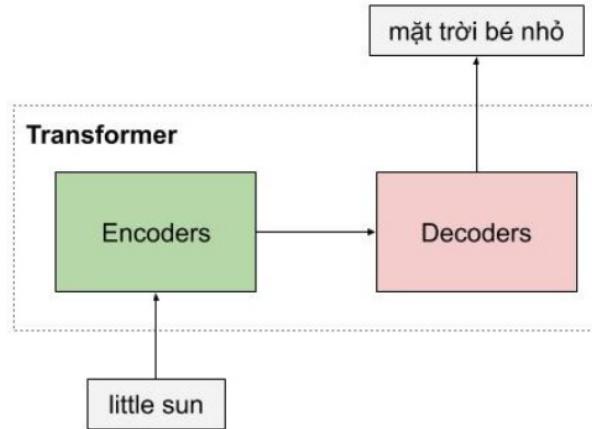
11.2 Tổng quan mô hình

Để cho dễ cảm nhận được cách mà mô hình hoạt động, ta 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.

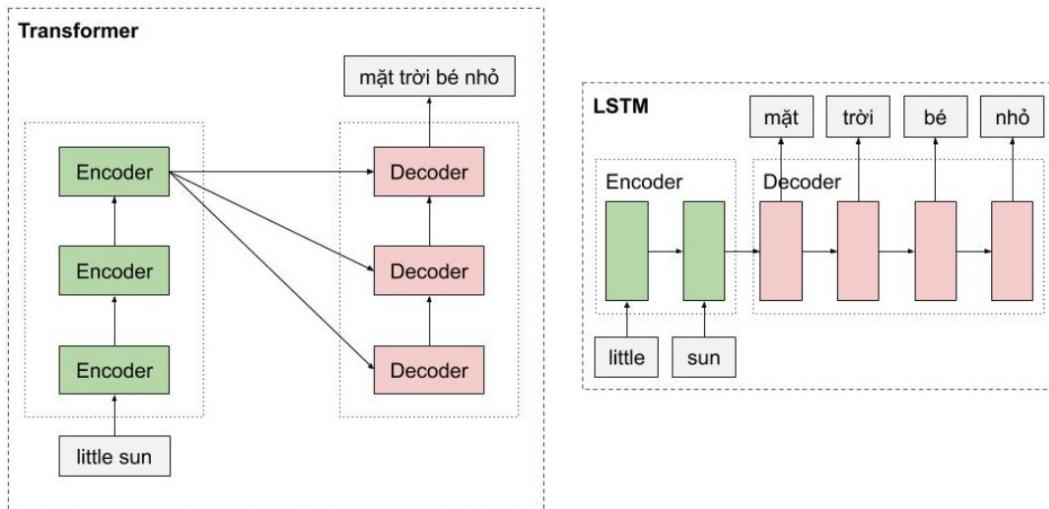
Giống như những mô hình dịch máy khác, kiến trúc tổng quan của mô hình transformer bao gồm 2 phần lớn là encoder và decoder. Encoder dùng để học vector biểu của câu với mong muốn rằng vector này mang thông tin hoàn hảo của câu đó. Decoder thực hiện chức năng chuyển vector biểu diễn kia thành ngôn ngữ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

Trong ví dụ ở dưới, encoder của mô hình transformer nhận một câu tiếng Việt, và encode thành một vector

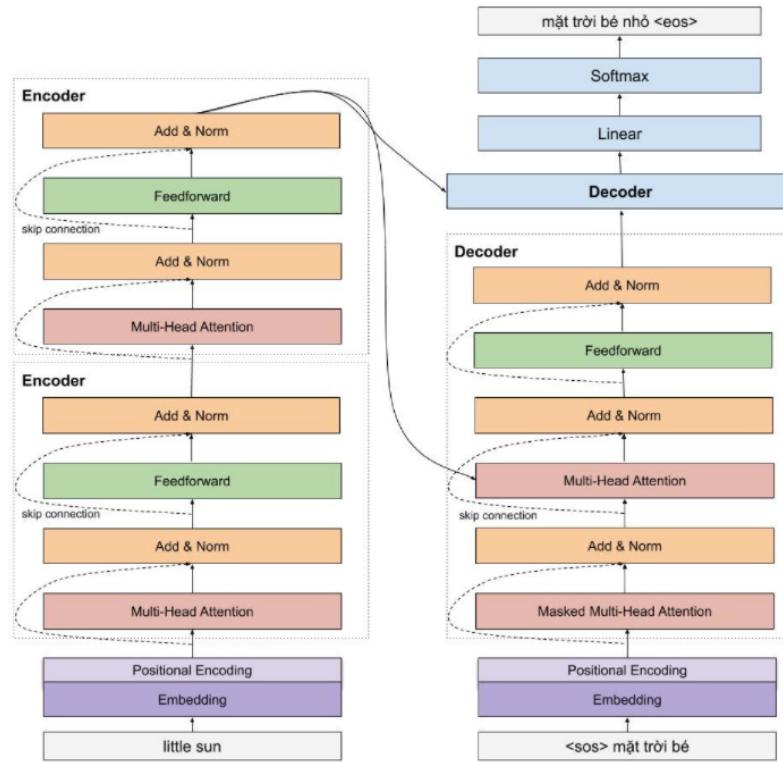
biểu diễn ngữ nghĩa của câu "little sun", sau đó mô hình decoder nhận vector biểu diễn này, và dịch nó thành câu tiếng Việt "mặt trời bé nhỏ".



Một trong những ưu điểm của Transformer là mô hình này có khả năng xử lý song song cho các từ. Như chúng ta thấy, Encoders của mô hình Transformer là một dạng Feedforward Neural Nets, bao gồm nhiều lớp encoder khác, mỗi lớp encoder 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.



Một cái nhìn vừa tổng quát và chi tiết sẽ giúp ích cho chúng ta. Ta sẽ đi vào chi tiết một số phần cực kỳ quan trọng như Sinusoidal Position Encoding, Multi Head Attention của encoder, còn của decoder thì chúng ta thấy được kiến trúc rất giống với của encoder, do đó ta sẽ chỉ đi nhanh qua mà thôi.



```

1 ! pip -q install torchtext==0.6.0
2 ! pip -q install pyvi
3
4 ! pip -q install https://github.com/trungtv/vi_spacy/raw/master/packages/
   vi_spacy_model-0.2.1/dist/vi_spacy_model-0.2.1.tar.gz
5 ! python -m spacy link vi_spacy_model vi_spacy_model
6
7 import nltk
8 nltk.download('wordnet')

```

```

1 import torch
2 import torch.nn as nn
3 from torch.autograd import Variable
4 import torch.nn.functional as F
5 import numpy as np
6 import os
7 import math

```

11.3 Embedding Layer với 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.

Dầu 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 phần giới thiệu ở trên ta có thể 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, chúng ta 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 gấp 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 gấp 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.

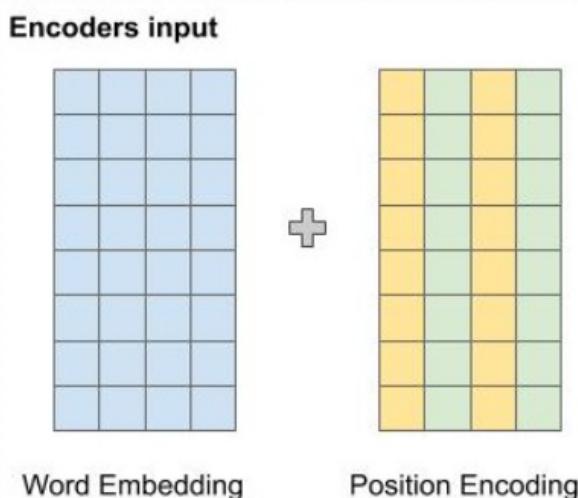
```

1 class Embedder(nn.Module):
2     def __init__(self, vocab_size, d_model):
3         super().__init__()
4         self.vocab_size = vocab_size
5         self.d_model = d_model
6
7         self.embed = nn.Embedding(vocab_size, d_model)
8
9     def forward(self, x):
10        return self.embed(x)
11
12 #Embedder(100, 512)(torch.LongTensor([1,2,3,4])).shape

```

11.4 Phương pháp để xuất sinusoidal position encoding

Phương pháp của tác giả để xuất không gấp 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 lẻ tác giả sử dụng hàm cos để tính giá trị tại chiều đó.

$$p_t^i = f(t)^i = \begin{cases} \sin(w_k * t) & \text{if } i = 2k \\ \cos(w_k * t) & \text{if } i = 2k + 1 \end{cases} \quad (21)$$

Trong đó

$$w_k = \frac{1}{10000^{2k/d}} \quad (22)$$

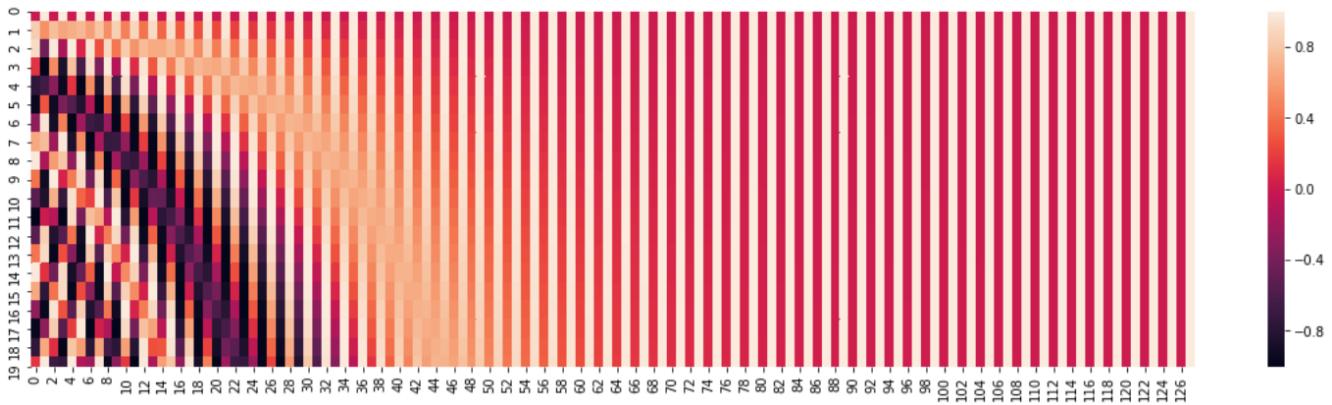
Trong hình dưới này, ta minh họa cho cách tính Position Encoding của tác giả. Giả sử chúng ta có Word Embedding có 6 chiều, thì Position Encoding cũng có tương ứng là 6 chiều. Mỗi dòng tương ứng với một từ. Giá trị của các vector tại mỗi vị trí được tính toán theo công thức ở hình dưới.

	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 chúng ta có thể 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 khi ta có các số từ 0-15. Chúng ta 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.

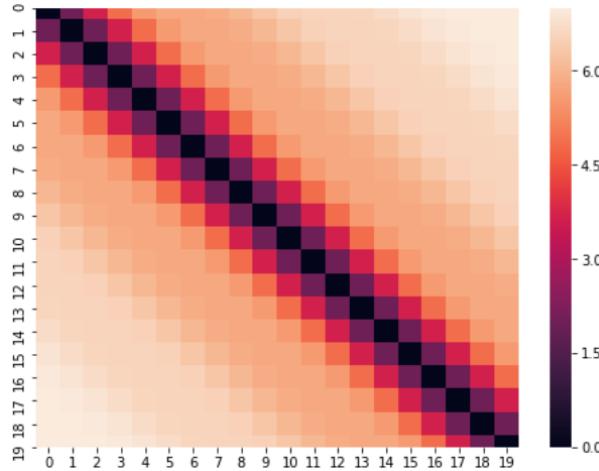
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10	1	0	1	0
3	0	0	1	1	11	1	0	1	1
4	0	1	0	0	12	1	1	0	0
5	0	1	0	1	13	1	1	0	1
6	0	1	1	0	14	1	1	1	0
7	0	1	1	1	15	1	1	1	1

Trong công thức của tác giả đề xuất, chúng ta cũng thấy rằng, hàm sin và cos có dạng đồ thị tần số và tần số này giảm dần ở các chiều lớn dần. Tiếp tục 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. Hai 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 giữ các từ. Cụ thể, biểu diễn vị trí của từ $t + \text{offset}$ có thể chuyển thành biểu diễn vị trí của từ t bằng một phép biến đổi tuyến tính dựa trên ma trận phép quay.

Dễ dẽ 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 $t + \text{offset}$ và từ t lại có thể dễ dàng chuyển đổi cho nhau.

```

1 class PositionalEncoder(nn.Module):
2     def __init__(self, d_model, max_seq_length = 200, dropout = 0.1):
3         super().__init__()
4
5         self.d_model = d_model
6         self.dropout = nn.Dropout(dropout)

```

```

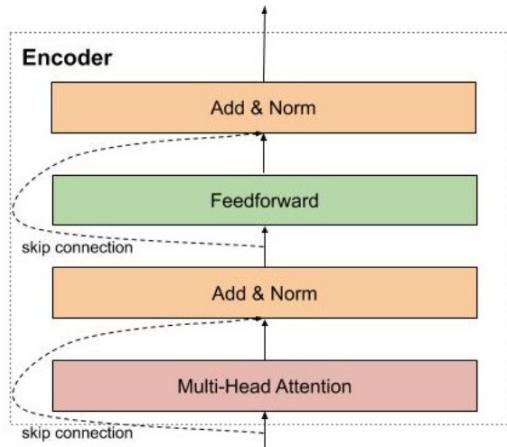
7     pe = torch.zeros(max_seq_length, d_model)
8
9
10    for pos in range(max_seq_length):
11        for i in range(0, d_model, 2):
12            pe[pos, i] = math.sin(pos/(10000**((2*i)/d_model)))
13            pe[pos, i+1] = math.cos(pos/(10000**((2*i+1)/d_model)))
14
15    pe = pe.unsqueeze(0)
16    self.register_buffer('pe', pe)
17
18
19    def forward(self, x):
20
21        x = x*math.sqrt(self.d_model)
22        seq_length = x.size(1)
23
24        pe = Variable(self.pe[:, :seq_length], requires_grad=False)
25
26        if x.is_cuda:
27            pe.cuda()
28
29        #Plus embedding vector with pe
30        x = x + pe
31        x = self.dropout(x)
32
33        return x
34
35
36 # PositionalEncoder(512)(torch.rand(5, 30, 512)).shape

```

11.5 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.

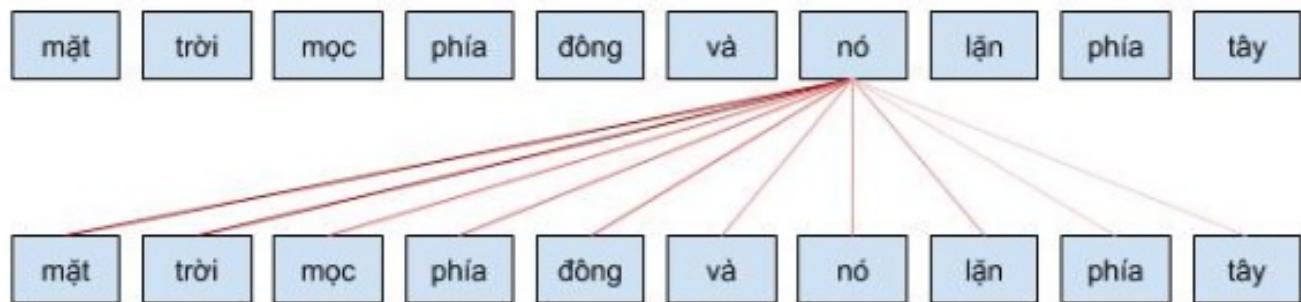
Trong 2 thành phần chính này, ta sẽ thấy hứng thú nhiều hơn về Multi-Head Attention vì đó là một layer mới được giới thiệu trong bài báo này, và chính nó tạo nên sự khác biệt giữa mô hình LSTM và mô hình Transformer mà chúng ta đang tìm hiểu.



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.

11.5.1 Self Attention Layer

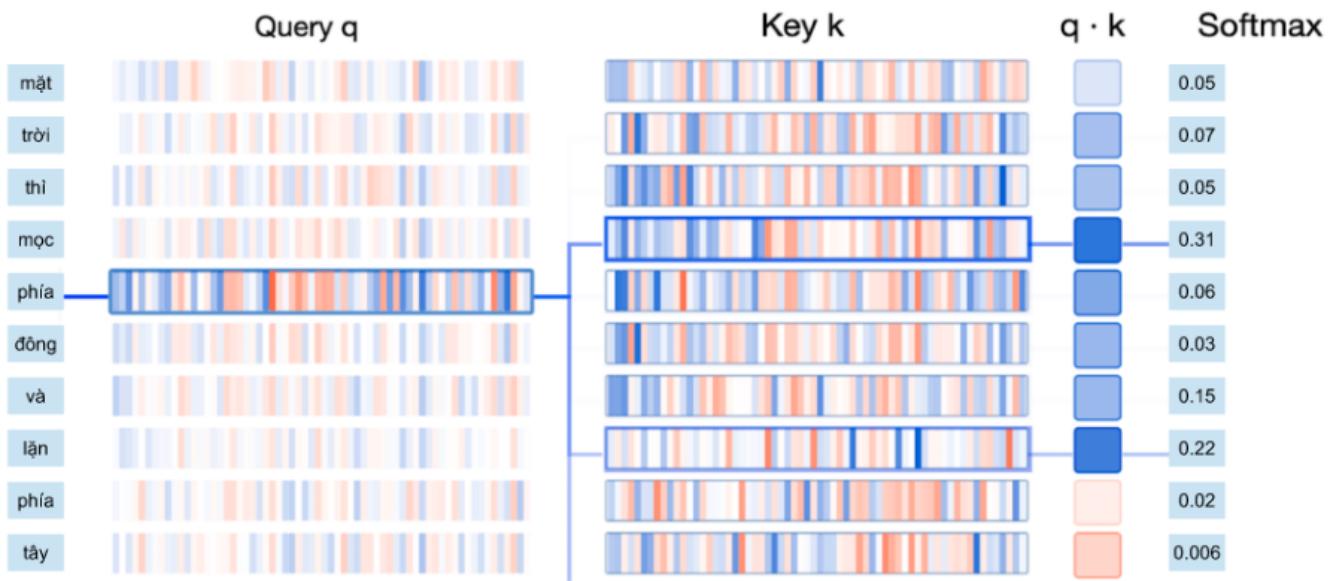
Self Attention cho phép mô hình khi mã hóa một từ có thể sử dụng thông tin của những từ liên quan tới nó. Ví dụ khi từ nó được mã hóa, nó sẽ chú ý vào các từ liên quan như là mặt trời. Cơ chế Self Attention này có ý nghĩa tương tự như cơ chế Attention đã được chia sẻ ở bài trước và những công thức toán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



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

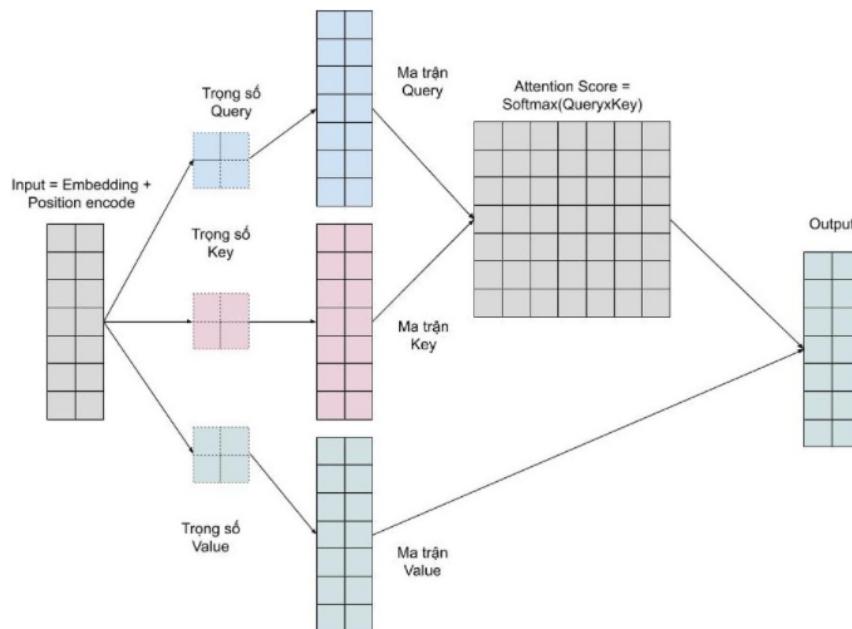
Đầu tiên, với mỗi từ chúng ta cần tạo ra 3 vector: query, key, value vector bằng cách nhân ma trận biểu diễn các từ đầu vào với ma trận học tương ứng.

- **Query vector** là vector dùng để chứa thông tin của từ được tìm kiếm, so sánh. Giống như là câu query của google search.
- **Key vector** là 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** là vector biểu diễn nội dung, ý nghĩa của các từ. Chúng ta có thể tưởng 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.



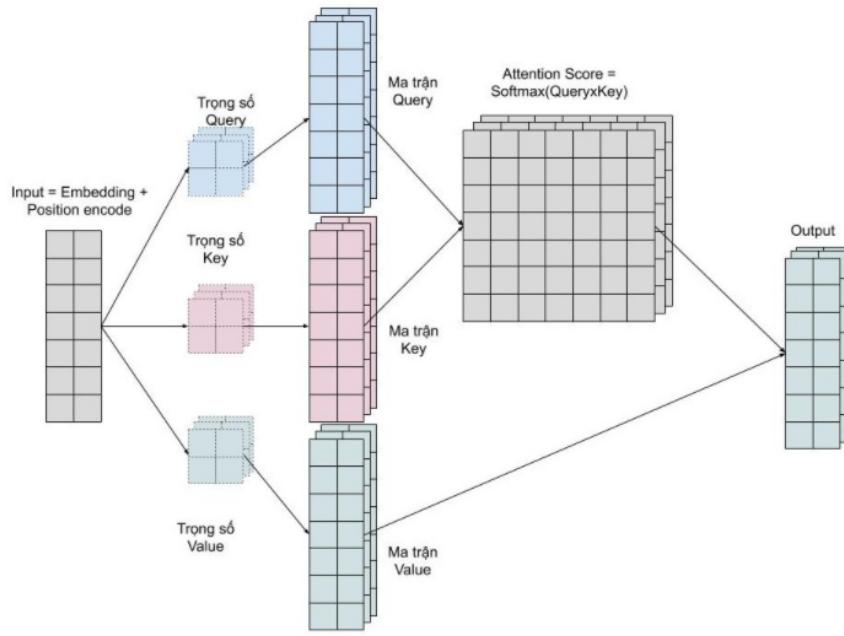
Cụ thể hơn, quá trình tính toán attention vector có thể được tóm tắt làm 3 bước như sau:

- **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 giữ 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.



11.5.2 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.

```

1 def attention(q, k, v, mask=None, dropout=None):
2     """
3     q: batch_size x head x seq_length x d_model
4     k: batch_size x head x seq_length x d_model
5     v: batch_size x head x seq_length x d_model
6     mask: batch_size x 1 x 1 x seq_length
7     output: batch_size x head x seq_length x d_model
8     """
9
10
11     #Attention score is evaluated by q * k
12     d_k = q.size(-1)
13     scores = torch.matmul(q, k.transpose(-2, -1))/math.sqrt(d_k)
14
15     if mask is not None:
16         mask = mask.unsqueeze(1)
17         scores = scores.masked_fill(mask==0, -1e9)

```

```

17     #Normalize score by softmax function
18     scores = F.softmax(scores, dim=-1)
19
20     if dropout is not None:
21         scores = dropout(scores)
22
23     output = torch.matmul(scores, v)
24     return output, scores
25
26 #Attention(torch.rand(32, 8, 30, 512), torch.rand(32, 8, 30, 512), torch.rand(32, 8, 30, 512)).shape

```

```

1 class MultiHeadAttention(nn.Module):
2     def __init__(self, heads, d_model, dropout=0.1):
3         super().__init__()
4         assert d_model % heads == 0
5
6         self.d_model = d_model
7         self.d_k = d_model//heads
8         self.h = heads
9         self.attn = None
10
11     #Generate 3 weight matrices (q_linear, k_linear, v_linear)
12     self.q_linear = nn.Linear(d_model, d_model)
13     self.k_linear = nn.Linear(d_model, d_model)
14     self.v_linear = nn.Linear(d_model, d_model)
15
16     self.dropout = nn.Dropout(dropout)
17     self.out = nn.Linear(d_model, d_model)
18
19     def forward(self, q, k, v, mask=None):
20         """
21             q: batch_size x seq_length x d_model
22             k: batch_size x seq_length x d_model
23             v: batch_size x seq_length x d_model
24             mask: batch_size x 1 x seq_length
25             output: batch_size x seq_length x d_model
26         """
27         bs = q.size(0)
28         #Multiply the weight matrix (q_linear, k_linear, v_linear) by the input data (q, k, v)
29         #In the encode step, please note that q, k, v is just 1
30         q = self.q_linear(q).view(bs, -1, self.h, self.d_k)
31         k = self.k_linear(k).view(bs, -1, self.h, self.d_k)
32         v = self.v_linear(v).view(bs, -1, self.h, self.d_k)
33
34         q = q.transpose(1, 2)
35         k = k.transpose(1, 2)
36         v = v.transpose(1, 2)
37
38         #Evaluate attention score
39         scores, self.attn = attention(q, k, v, mask, self.dropout)
40
41         concat = scores.transpose(1, 2).contiguous().view(bs, -1, self.d_model)
42
43         output = self.out(concat)

```

```

44     return output
45
46 # MultiHeadAttention(8, 512)(torch.rand(32, 30, 512), torch.rand(32, 30, 512), torch.rand(32, 30,
512)).shape

```

11.6 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. Hai kỹ thuật giúp cho mô hình huấn luyện nhanh hơn và trách mắng mát thông tin trong quá trình huấn luyện mô hình, ví dụ như là thông tin của vị trí các từ được mã hóa.

```

1 class Norm(nn.Module):
2     def __init__(self, d_model, eps = 1e-6):
3         super().__init__()
4
5         self.size = d_model
6
7         #Create two learnable parameters to calibrate normalisation
8         self.alpha = nn.Parameter(torch.ones(self.size))
9         self.bias = nn.Parameter(torch.zeros(self.size))
10
11    self.eps = eps
12
13    def forward(self, x):
14        norm = self.alpha * (x - x.mean(dim=-1, keepdim=True)) \
15        / (x.std(dim=-1, keepdim=True) + self.eps) + self.bias
16        return norm

```

```

1 class FeedForward(nn.Module):
2     """ In our architecture there is a linear layer
3     """
4
5     def __init__(self, d_model, d_ff=2048, dropout = 0.1):
6         super().__init__()
7
8         #We set d_ff as a default to 2048
9         self.linear_1 = nn.Linear(d_model, d_ff)
10        self.dropout = nn.Dropout(dropout)
11        self.linear_2 = nn.Linear(d_ff, d_model)
12
13    def forward(self, x):
14        x = self.dropout(F.relu(self.linear_1(x)))
15        x = self.linear_2(x)
16        return x

```

```

1 class EncoderLayer(nn.Module):
2     def __init__(self, d_model, heads, dropout=0.1):
3         super().__init__()
4         self.norm_1 = Norm(d_model)
5         self.norm_2 = Norm(d_model)
6         self.attn = MultiHeadAttention(heads, d_model, dropout=dropout)
7         self.ff = FeedForward(d_model, dropout=dropout)
8         self.dropout_1 = nn.Dropout(dropout)

```

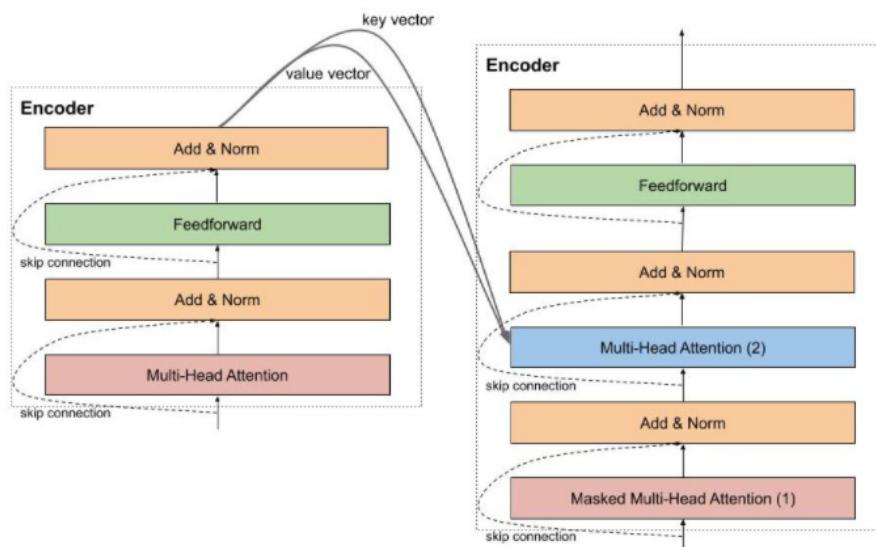
```

9     self.dropout_2 = nn.Dropout(dropout)
10
11    def forward(self, x, mask):
12        """
13            x: batch_size x seq_length x d_model
14            mask: batch_size x 1 x seq_length
15            output: batch_size x seq_length x d_model
16        """
17
18
19        x2 = self.norm_1(x)
20        #Evaluate the attention value
21        #Note q, k, v are the same
22        x = x + self.dropout_1(self.attn(x2,x2,x2,mask))
23        x2 = self.norm_2(x)
24        x = x + self.dropout_2(self.ff(x2))
25
26    return x
27
# EncoderLayer(512, 8)(torch.rand(32, 30, 512), torch.rand(32 , 1, 30)).shape

```

11.7 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.



11.7.1 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 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, lớp 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.

```
1 class DecoderLayer(nn.Module):
2     def __init__(self, d_model, heads, dropout=0.1):
3         super().__init__()
4         self.norm_1 = Norm(d_model)
5         self.norm_2 = Norm(d_model)
6         self.norm_3 = Norm(d_model)
7
8         self.dropout_1 = nn.Dropout(dropout)
9         self.dropout_2 = nn.Dropout(dropout)
10        self.dropout_3 = nn.Dropout(dropout)
11
12        self.attn_1 = MultiHeadAttention(heads, d_model, dropout=dropout)
13        self.attn_2 = MultiHeadAttention(heads, d_model, dropout=dropout)
14        self.ff = FeedForward(d_model, dropout=dropout)
15
16    def forward(self, x, e_outputs, src_mask, trg_mask):
17        """
18            x: batch_size x seq_length x d_model
19            e_outputs: batch_size x seq_length x d_model
20            src_mask: batch_size x 1 x seq_length
21            trg_mask: batch_size x 1 x seq_length
22        """
23        x2 = self.norm_1(x)
24        #Multi head attention 1st, notice the words in target
25        x = x + self.dropout_1(self.attn_1(x2, x2, x2, trg_mask))
26        x2 = self.norm_2(x)
27        #Masked multihead attention 2nd with k, v is the output value of the encoder model
28        x = x + self.dropout_2(self.attn_2(x2, e_outputs, e_outputs, src_mask))
29        x2 = self.norm_3(x)
30        x = x + self.dropout_3(self.ff(x2))
31
32    return x
33
34 # DecoderLayer(512, 8)(torch.rand(32, 30, 512), torch.rand(32, 30, 512), torch.rand(32, 1, 30),
35 #                     torch.rand(32, 1, 30)).shape
```

11.8 Cài đặt Encoder

Bao gồm N encoder layers.

```
1 import copy
2
3 def get_clones(module, N):
4     return nn.ModuleList([copy.deepcopy(module) for i in range(N)])
5
6 class Encoder(nn.Module):
7     """An Encoder has many encoder layers!!!
8     """
9     def __init__(self, vocab_size, d_model, N, heads, dropout):
10        super().__init__()
```

```

11     self.N = N
12     self.embed = Embedder(vocab_size, d_model)
13     self.pe = PositionalEncoder(d_model, dropout=dropout)
14     self.layers = get_clones(EncoderLayer(d_model, heads, dropout), N)
15     self.norm = Norm(d_model)
16
17     def forward(self, src, mask):
18         """
19             src: batch_size x seq_length
20             mask: batch_size x 1 x seq_length
21             output: batch_size x seq_length x d_model
22         """
23
24         x = self.embed(src)
25         x = self.pe(x)
26         for i in range(self.N):
27             x = self.layers[i](x, mask)
28
29     return self.norm(x)
# Encoder(232, 512, 6, 8, 0.1)(torch.LongTensor(32, 30).random_(0, 10), torch.rand(32, 1, 30)).shape

```

11.9 Cài đặt Decoder

Bao gồm N decoder layers.

```

1 class Decoder(nn.Module):
2     """A Decoder also has many decoder layers!!!
3     """
4
5     def __init__(self, vocab_size, d_model, N, heads, dropout):
6         super().__init__()
7         self.N = N
8         self.embed = Embedder(vocab_size, d_model)
9         self.pe = PositionalEncoder(d_model, dropout=dropout)
10        self.layers = get_clones(DecoderLayer(d_model, heads, dropout), N)
11        self.norm = Norm(d_model)
12
13    def forward(self, trg, e_outputs, src_mask, trg_mask):
14        """
15            trg: batch_size x seq_length
16            e_outputs: batch_size x seq_length x d_model
17            src_mask: batch_size x 1 x seq_length
18            trg_mask: batch_size x 1 x seq_length
19            output: batch_size x seq_length x d_model
20        """
21
22        x = self.embed(trg)
23        x = self.pe(x)
24        for i in range(self.N):
25            x = self.layers[i](x, e_outputs, src_mask, trg_mask)
26
27    return self.norm(x)
# Decoder(232, 512, 6, 8, 0.1)(torch.LongTensor(32, 30).random_(0, 10), torch.rand(32, 30, 512),
28      torch.rand(32, 1, 30), torch.rand(32, 1, 30)).shape

```

11.10 Cài đặt Transformer

Bao gồm encoder và decoder.

```
1 class Transformer(nn.Module):
2     """ Finally put them together to get a complete transformer model.
3     """
4     def __init__(self, src_vocab, trg_vocab, d_model, N, heads, dropout):
5         super().__init__()
6         self.encoder = Encoder(src_vocab, d_model, N, heads, dropout)
7         self.decoder = Decoder(trg_vocab, d_model, N, heads, dropout)
8         self.out = nn.Linear(d_model, trg_vocab)
9     def forward(self, src, trg, src_mask, trg_mask):
10        """
11        src: batch_size x seq_length
12        trg: batch_size x seq_length
13        src_mask: batch_size x 1 x seq_length
14        trg_mask batch_size x 1 x seq_length
15        output: batch_size x seq_length x vocab_size
16        """
17        e_outputs = self.encoder(src, src_mask)
18
19        d_output = self.decoder(trg, e_outputs, src_mask, trg_mask)
20        output = self.out(d_output)
21
22        return output
23
24 # Transformer(232, 232, 512, 6, 8, 0.1)(torch.LongTensor(32, 30).random_(0, 10), torch.LongTensor(32,
25     30).random_(0, 10),torch.rand(32, 1, 30),torch.rand(32, 1, 30)).shape
```

Chúng ta sử dụng torchtext để load dữ liệu, giúp giảm thời gian và hiệu quả.

```
1 from torchtext import data
2
3 class MyIterator(data.Iterator):
4     def create_batches(self):
5         if self.train:
6             def pool(d, random_shuffler):
7                 for p in data.batch(d, self.batch_size * 100):
8                     p_batch = data.batch(
9                         sorted(p, key=self.sort_key),
10                         self.batch_size, self.batch_size_fn)
11                     for b in random_shuffler(list(p_batch)):
12                         yield b
13             self.batches = pool(self.data(), self.random_shuffler)
14
15         else:
16             self.batches = []
17             for b in data.batch(self.data(), self.batch_size,
18                                 self.batch_size_fn):
19                 self.batches.append(sorted(b, key=self.sort_key))
20
21 global max_src_in_batch, max_tgt_in_batch
22
23 def batch_size_fn(new, count, sofar):
24     "Keep augmenting batch and calculate total number of tokens + padding."
```

```

25 global max_src_in_batch, max_tgt_in_batch
26 if count == 1:
27     max_src_in_batch = 0
28     max_tgt_in_batch = 0
29 max_src_in_batch = max(max_src_in_batch, len(new.src))
30 max_tgt_in_batch = max(max_tgt_in_batch, len(new.trg) + 2)
31 src_elements = count * max_src_in_batch
32 tgt_elements = count * max_tgt_in_batch
33 return max(src_elements, tgt_elements)


---


1 def nopeak_mask(size, device):
2     """Masking is used in the decoder so that when predicting during training the model does not see
3         future words.
4     """
5     np_mask = np.triu(np.ones((1, size, size)),
6                       k=1).astype('uint8')
7     np_mask = Variable(torch.from_numpy(np_mask) == 0)
8     np_mask = np_mask.to(device)
9
10    return np_mask
11
12 def create_masks(src, trg, src_pad, trg_pad, device):
13     """ Create a mask for the encoder, so that the model does not ignore the information of the PAD
14         characters we added
15     """
16     src_mask = (src != src_pad).unsqueeze(-2)
17
18     if trg is not None:
19         trg_mask = (trg != trg_pad).unsqueeze(-2)
20         size = trg.size(1) # get seq_len for matrix
21         np_mask = nopeak_mask(size, device)
22         if trg.is_cuda:
23             np_mask.cuda()
24         trg_mask = trg_mask & np_mask
25
26     else:
27         trg_mask = None
28     return src_mask, trg_mask


---


1 from nltk.corpus import wordnet
2 import re
3
4 def get_synonym(word, SRC):
5     syns = wordnet.synsets(word)
6     for s in syns:
7         for l in s.lemmas():
8             if SRC.vocab.stoi[l.name()] != 0:
9                 return SRC.vocab.stoi[l.name()]
10
11    return 0
12
13 def multiple_replace(dict, text):
14     # Create a regular expression from the dictionary keys

```

```

15     regex = re.compile("(%) % ".join(map(re.escape, dict.keys())))
16
17     # For each match, look-up corresponding value in dictionary
18     return regex.sub(lambda mo: dict[mo.string[mo.start():mo.end()]], text)


---


1 def init_vars(src, model, SRC, TRG, device, k, max_len):
2     """ Calculate the required matrices during translation after the model is finished learning.
3     """
4     init_tok = TRG.vocab.stoi['<sos>']
5     src_mask = (src != SRC.vocab.stoi['<pad>']).unsqueeze(-2)
6
7     #Encoder output available
8     e_output = model.encoder(src, src_mask)
9
10    outputs = torch.LongTensor([[init_tok]])
11
12    outputs = outputs.to(device)
13
14    trg_mask = nopeak_mask(1, device)
15    #Guess the first character
16    out = model.out(model.decoder(outputs,
17                      e_output, src_mask, trg_mask))
18    out = F.softmax(out, dim=-1)
19
20    probs, ix = out[:, -1].data.topk(k)
21    log_scores = torch.Tensor([math.log(prob) for prob in probs.data[0]]).unsqueeze(0)
22
23    outputs = torch.zeros(k, max_len).long()
24    outputs = outputs.to(device)
25    outputs[:, 0] = init_tok
26    outputs[:, 1] = ix[0]
27
28    e_outputs = torch.zeros(k, e_output.size(-2), e_output.size(-1))
29
30    e_outputs = e_outputs.to(device)
31    e_outputs[:, :] = e_output[0]
32
33    return outputs, e_outputs, log_scores
34
35 def k_best_outputs(outputs, out, log_scores, i, k):
36
37    probs, ix = out[:, -1].data.topk(k)
38    log_probs = torch.Tensor([math.log(p) for p in probs.data.view(-1)]).view(k, -1) +
39        log_scores.transpose(0,1)
40    k_probs, k_ix = log_probs.view(-1).topk(k)
41
42    row = k_ix // k
43    col = k_ix % k
44
45    outputs[:, :i] = outputs[row, :i]
46    outputs[:, i] = ix[row, col]
47
48    log_scores = k_probs.unsqueeze(0)

```

```

49     return outputs, log_scores
50
51 def beam_search(src, model, SRC, TRG, device, k, max_len):
52
53     outputs, e_outputs, log_scores = init_vars(src, model, SRC, TRG, device, k, max_len)
54     eos_tok = TRG.vocab.stoi['<eos>']
55     src_mask = (src != SRC.vocab.stoi['<pad>']).unsqueeze(-2)
56     ind = None
57     for i in range(2, max_len):
58
59         trg_mask = nopeak_mask(i, device)
60
61         out = model.out(model.decoder(outputs[:, :i],
62                               e_outputs, src_mask, trg_mask))
63
64         out = F.softmax(out, dim=-1)
65
66         outputs, log_scores = k_best_outputs(outputs, out, log_scores, i, k)
67
68         ones = (outputs==eos_tok).nonzero() #Occurrences of end symbols for all input sentences.
69         sentence_lengths = torch.zeros(len(outputs), dtype=torch.long).cuda()
70         for vec in ones:
71             i = vec[0]
72             if sentence_lengths[i]==0: # First end symbol has not been found yet
73                 sentence_lengths[i] = vec[1] # Position of first end symbol
74
75         num_finished_sentences = len([s for s in sentence_lengths if s > 0])
76
77         if num_finished_sentences == k:
78             alpha = 0.7
79             div = 1/(sentence_lengths.type_as(log_scores)**alpha)
80             _, ind = torch.max(log_scores * div, 1)
81             ind = ind.data[0]
82             break
83
84         if ind is None:
85
86             length = (outputs[0]==eos_tok).nonzero()[0] if len((outputs[0]==eos_tok).nonzero()) > 0 else -1
87             return ''.join([TRG.vocab.itos[tok] for tok in outputs[0][1:length]])
88
89         else:
90             length = (outputs[ind]==eos_tok).nonzero()[0]
91             return ''.join([TRG.vocab.itos[tok] for tok in outputs[ind][1:length]])

```

```

1 def translate_sentence(sentence, model, SRC, TRG, device, k, max_len):
2     """Translate a sentence using beamsearch
3     """
4     model.eval()
5     indexed = []
6     sentence = SRC.preprocess(sentence)
7
8     for tok in sentence:
9         if SRC.vocab.stoi[tok] != SRC.vocab.stoi['<eos>']:
10            indexed.append(SRC.vocab.stoi[tok])

```

```

11     else:
12         indexed.append(get_synonym(tok, SRC))
13
14 sentence = Variable(torch.LongTensor([indexed]))
15
16 sentence = sentence.to(device)
17
18 sentence = beam_search(sentence, model, SRC, TRG, device, k, max_len)
19
20 return multiple_replace{'?': '?', '!': '!', '.': '.', '\n': '\n', ',': ','}, sentence

```

```

import spacy
import re

class tokenize(object):

    def __init__(self, lang):
        self.nlp = spacy.load(lang)

    def tokenizer(self, sentence):
        sentence = re.sub(
            r"[*\""\n\\..\\+\\-\\=\\((\\)*:\\[\\]\\|'!;]", " ", str(sentence))
        sentence = re.sub(r"[ ]+", " ", sentence)
        sentence = re.sub(r"\!+", "!", sentence)
        sentence = re.sub(r"\,+",".",sentence)
        sentence = re.sub(r"\?+","?",sentence)
        sentence = sentence.lower()
        return [tok.text for tok in self.nlp.tokenizer(sentence) if tok.text != " "]

```

11.11 Data Loader

Sử dụng torchtext để load dữ liệu nhanh chóng.

```

1 import os
2 import dill as pickle
3 import pandas as pd
4
5 def read_data(src_file, trg_file):
6     src_data = open(src_file).read().strip().split('\n')
7
8     trg_data = open(trg_file).read().strip().split('\n')
9
10    return src_data, trg_data
11
12 def create_fields(src_lang, trg_lang):
13
14     print("loading spacy tokenizers...")
15

```

```

16     t_src = tokenize(src_lang)
17     t_trg = tokenize(trg_lang)
18
19     TRG = data.Field(lower=True, tokenize=t_trg.tokenizer, init_token='<sos>', eos_token='<eos>')
20     SRC = data.Field(lower=True, tokenize=t_src.tokenizer)
21
22     return SRC, TRG
23
24 def create_dataset(src_data, trg_data, max_strlen, batchsize, device, SRC, TRG, istrain=True):
25
26     print("creating dataset and iterator... ")
27
28     raw_data = {'src' : [line for line in src_data], 'trg': [line for line in trg_data]}
29     df = pd.DataFrame(raw_data, columns=["src", "trg"])
30
31     mask = (df['src'].str.count(' ') < max_strlen) & (df['trg'].str.count(' ') < max_strlen)
32     df = df.loc[mask]
33
34     df.to_csv("translate_transformer_temp.csv", index=False)
35
36     data_fields = [('src', SRC), ('trg', TRG)]
37     train = data.TabularDataset('./translate_transformer_temp.csv', format='csv', fields=data_fields)
38
39     train_iter = MyIterator(train, batch_size=batchsize, device=device,
40                            repeat=False, sort_key=lambda x: (len(x.src), len(x.trg)),
41                            batch_size_fn=batch_size_fn, train=istrain, shuffle=True)
42
43     os.remove('translate_transformer_temp.csv')
44
45     if istrain:
46         SRC.build_vocab(train)
47         TRG.build_vocab(train)
48
49     return train_iter

```

```

1 def step(model, optimizer, batch, criterion):
2     """
3     One time update model
4     """
5     model.train()
6
7     src = batch.src.transpose(0,1).cuda()
8     trg = batch.trg.transpose(0,1).cuda()
9     trg_input = trg[:, :-1]
10    src_mask, trg_mask = create_masks(src, trg_input, src_pad, trg_pad, opt['device'])
11    preds = model(src, trg_input, src_mask, trg_mask)
12
13    ys = trg[:, 1:].contiguous().view(-1)
14
15    optimizer.zero_grad()
16    loss = criterion(preds.view(-1, preds.size(-1)), ys)
17    loss.backward()
18    optimizer.step_and_update_lr()
19

```

```

20     loss = loss.item()
21
22     return loss


---

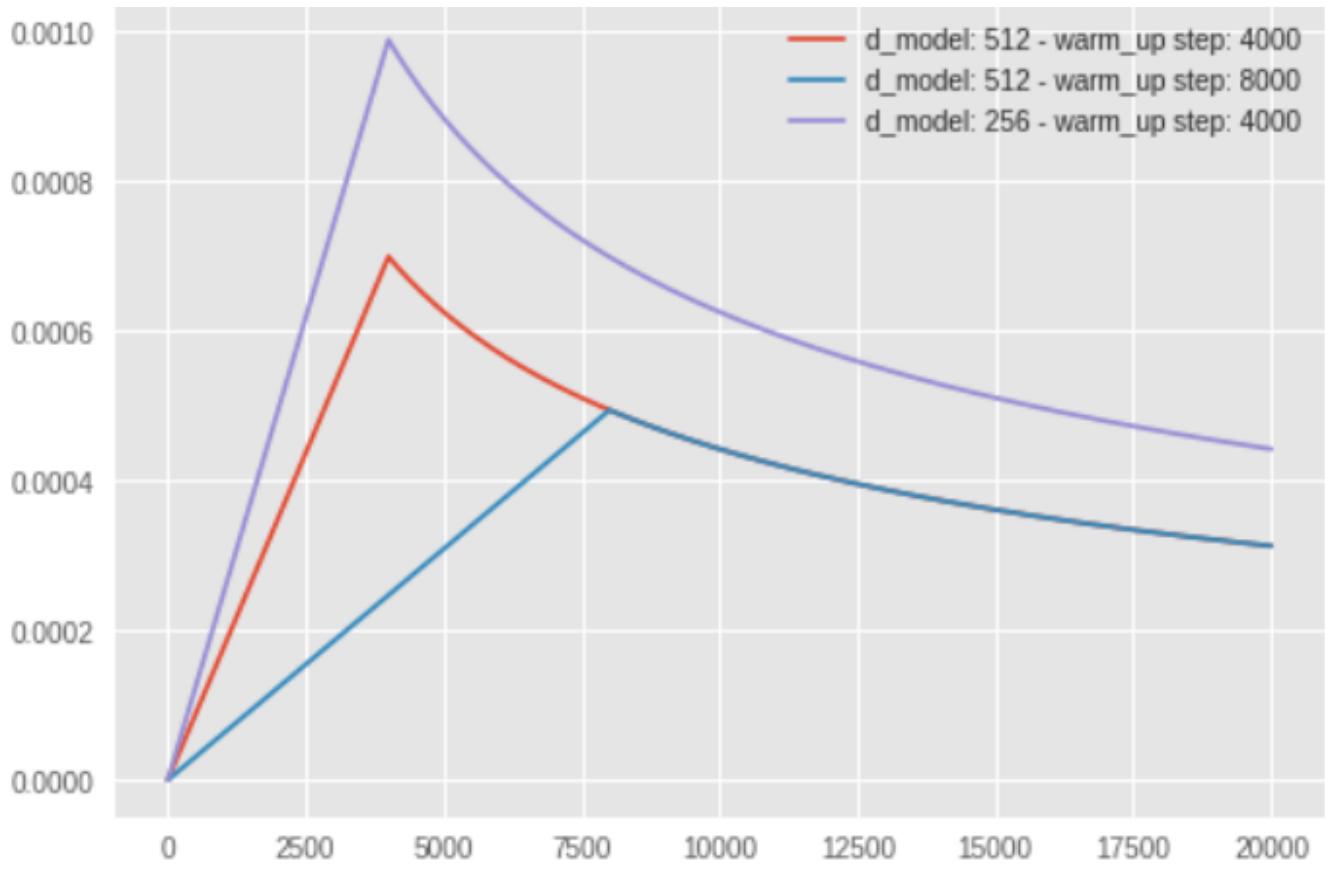

1 def validate(model, valid_iter, criterion):
2     """Calculating loss on validation set
3     """
4     model.eval()
5
6     with torch.no_grad():
7         total_loss = []
8         for batch in valid_iter:
9             src = batch.src.transpose(0,1).cuda()
10            trg = batch.trg.transpose(0,1).cuda()
11            trg_input = trg[:, :-1]
12            src_mask, trg_mask = create_masks(src, trg_input, src_pad, trg_pad, opt['device'])
13            preds = model(src, trg_input, src_mask, trg_mask)
14
15            ys = trg[:, 1:].contiguous().view(-1)
16
17            loss = criterion(preds.view(-1, preds.size(-1)), ys)
18
19            loss = loss.item()
20
21            total_loss.append(loss)
22
23    avg_loss = np.mean(total_loss)
24
25    return avg_loss

```

11.12 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:

$$\text{lr_rate} = d_{\text{d_model}}^{-0.5} * \min(\text{step_num}^{-0.5}, \text{step_num} * \text{warmup_steps}^{-1.5}) \quad (23)$$



```

1 class ScheduledOptim():
2     '''A simple wrapper class for learning rate scheduling'''
3
4     def __init__(self, optimizer, init_lr, d_model, n_warmup_steps):
5         self._optimizer = optimizer
6         self.init_lr = init_lr
7         self.d_model = d_model
8         self.n_warmup_steps = n_warmup_steps
9         self.n_steps = 0
10
11
12     def step_and_update_lr(self):
13         "Step with the inner optimizer"
14         self._update_learning_rate()
15         self._optimizer.step()
16
17
18     def zero_grad(self):
19         "Zero out the gradients with the inner optimizer"
20         self._optimizer.zero_grad()
21
22
23     def _get_lr_scale(self):
24         d_model = self.d_model

```

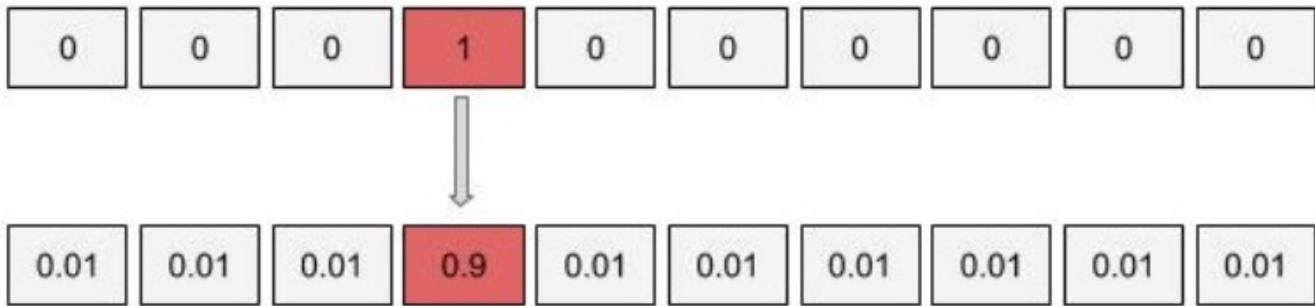
```

25     n_steps, n_warmup_steps = self.n_steps, self.n_warmup_steps
26     return (d_model ** -0.5) * min(n_steps ** (-0.5), n_steps * n_warmup_steps ** (-1.5))
27
28     def state_dict(self):
29         optimizer_state_dict = {
30             'init_lr':self.init_lr,
31             'd_model':self.d_model,
32             'n_warmup_steps':self.n_warmup_steps,
33             'n_steps':self.n_steps,
34             '_optimizer':self._optimizer.state_dict(),
35         }
36
37         return optimizer_state_dict
38
39     def load_state_dict(self, state_dict):
40         self.init_lr = state_dict['init_lr']
41         self.d_model = state_dict['d_model']
42         self.n_warmup_steps = state_dict['n_warmup_steps']
43         self.n_steps = state_dict['n_steps']
44
45         self._optimizer.load_state_dict(state_dict['_optimizer'])
46
47     def _update_learning_rate(self):
48         ''' Learning rate scheduling per step '''
49
50         self.n_steps += 1
51         lr = self.init_lr * self._get_lr_scale()
52
53         for param_group in self._optimizer.param_groups:
54             param_group['lr'] = lr

```

11.13 Label Smoothing

Với mô hình nhiều tham số của transformer, thì việc 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 bổ một tí xác suất vào các trường hợp còn lại.



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

```
1 class LabelSmoothingLoss(nn.Module):
```

```

2     def __init__(self, classes, padding_idx, smoothing=0.0, dim=-1):
3         super(LabelSmoothingLoss, self).__init__()
4         self.confidence = 1.0 - smoothing
5         self.smoothing = smoothing
6         self.cls = classes
7         self.dim = dim
8         self.padding_idx = padding_idx
9
10    def forward(self, pred, target):
11        pred = pred.log_softmax(dim=self.dim)
12        with torch.no_grad():
13            # true_dist = pred.data.clone()
14            true_dist = torch.zeros_like(pred)
15            true_dist.fill_(self.smoothing / (self.cls - 2))
16            true_dist.scatter_(1, target.data.unsqueeze(1), self.confidence)
17            true_dist[:, self.padding_idx] = 0
18            mask = torch.nonzero(target.data == self.padding_idx, as_tuple=False)
19            if mask.dim() > 0:
20                true_dist.index_fill_(0, mask.squeeze(), 0.0)
21
22        return torch.mean(torch.sum(-true_dist * pred, dim=self.dim))

```

```

1 from torchtext.data.metrics import bleu_score
2
3 def bleu(valid_src_data, valid_trg_data, model, SRC, TRG, device, k, max_strlen):
4     pred_sents = []
5     for sentence in valid_src_data:
6         pred_trg = translate_sentence(sentence, model, SRC, TRG, device, k, max_strlen)
7         pred_sents.append(pred_trg)
8
9     pred_sents = [TRG.preprocess(sent) for sent in pred_sents]
10    trg_sents = [[sent.split()] for sent in valid_trg_data]
11
12    return bleu_score(pred_sents, trg_sents)

```

```

1 opt = {
2     'train_src_data': './data/train.en',
3     'train_trg_data': './data/train.vi',
4     'valid_src_data': './data/tst2013.en',
5     'valid_trg_data': './data/tst2013.vi',
6     'src_lang': 'en',
7     'trg_lang': 'en', #'vi_spacy_model',
8     'max_strlen': 160,
9     'batchsize': 1500,
10    'device': 'cuda',
11    'd_model': 512,
12    'n_layers': 6,
13    'heads': 8,
14    'dropout': 0.1,
15    'lr': 0.0001,
16    'epochs': 30,
17    'printevery': 200,
18    'k': 5,
}

```

```
1 os.makedirs('./data/', exist_ok=True)
2 ! gdown --id 1Fuo_ALIFK1UvOPbK5rUA50fAS2wKn_95
```

Downloading...

From: https://drive.google.com/uc?id=1Fuo_ALIFK1UvOPbK5rUA50fAS2wKn_95
To: /content/en_vi.zip
10.1MB [00:00, 27.7MB/s]

```
1 ! unzip -o en_vi.zip
```

Archive: en_vi.zip

inflating: data/tst2013.en
inflating: data/tst2012.vi
inflating: data/train.en
inflating: data/tst2013.vi
inflating: data/train.vi
inflating: data/tst2012.en

```
1 train_src_data, train_trg_data = read_data(opt['train_src_data'], opt['train_trg_data'])
2 valid_src_data, valid_trg_data = read_data(opt['valid_src_data'], opt['valid_trg_data'])
3
4 SRC, TRG = create_fields(opt['src_lang'], opt['trg_lang'])
5 train_iter = create_dataset(train_src_data, train_trg_data, opt['max_strlen'], opt['batchsize'],
   opt['device'], SRC, TRG, istrain=True)
6 valid_iter = create_dataset(valid_src_data, valid_trg_data, opt['max_strlen'], opt['batchsize'],
   opt['device'], SRC, TRG, istrain=False)
```

loading spacy tokenizers...

creating dataset and iterator...

creating dataset and iterator...

```
1 src_pad = SRC.vocab.stoi['<pad>']
2 trg_pad = TRG.vocab.stoi['<pad>']
```

```
1 model = Transformer(len(SRC.vocab), len(TRG.vocab), opt['d_model'], opt['n_layers'], opt['heads'],
   opt['dropout'])
2
3 for p in model.parameters():
4     if p.dim() > 1:
5         nn.init.xavier_uniform_(p)
6
7 model = model.to(opt['device'])
```

```
1 optimizer = ScheduledOptim(
2     torch.optim.Adam(model.parameters(), betas=(0.9, 0.98), eps=1e-09),
```

```
3     0.2, opt['d_model'], 4000)
4
5 criterion = LabelSmoothingLoss(len(TRG.vocab), padding_idx=trg_pad, smoothing=0.1)


---


1 import time
2
3 for epoch in range(opt['epochs']):
4     total_loss = 0
5
6     for i, batch in enumerate(train_iter):
7         s = time.time()
8         loss = step(model, optimizer, batch, criterion)
9
10    total_loss += loss
11
12    if (i + 1) % opt['printevery'] == 0:
13        avg_loss = total_loss/opt['printevery']
14        print('epoch: {:03d} - iter: {:05d} - train loss: {:.4f} - time: {:.4f}'.format(epoch, i,
15            avg_loss, time.time()- s))
16        total_loss = 0
17
18    s = time.time()
19    valid_loss = validate(model, valid_iter, criterion)
20    bleuscore = bleu(valid_src_data[:500], valid_trg_data[:500], model, SRC, TRG, opt['device'],
21        opt['k'], opt['max_strlen'])
22    print('epoch: {:03d} - iter: {:05d} - valid loss: {:.4f} - bleu score: {:.4f} - time:
23        {:.4f}'.format(epoch, i, valid_loss, bleuscore, time.time() - s))
```

epoch: 000 - iter: 00199 - train loss: 9.2811 - time: 0.1356
epoch: 000 - iter: 00399 - train loss: 8.3767 - 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: 6.3143 - time: 0.1322
epoch: 000 - iter: 01199 - train loss: 6.1971 - time: 0.1359
epoch: 000 - iter: 01399 - train loss: 6.0398 - time: 0.1345
epoch: 000 - iter: 01599 - train loss: 5.8988 - time: 0.1341
epoch: 000 - iter: 01799 - train loss: 5.7358 - time: 0.1329
epoch: 000 - iter: 01999 - train loss: 5.6722 - time: 0.1329
epoch: 000 - iter: 02199 - train loss: 5.5183 - time: 0.1367
epoch: 000 - iter: 02399 - train loss: 5.3774 - time: 0.1333
epoch: 000 - iter: 02497 - valid loss: 3.9765 - bleu score: 0.0114 - time: 125.6958
epoch: 001 - iter: 00199 - train loss: 5.1674 - time: 0.1123
epoch: 001 - iter: 00399 - train loss: 5.1040 - time: 0.1263
epoch: 001 - iter: 00599 - train loss: 5.0452 - time: 0.1275
epoch: 001 - iter: 00799 - train loss: 4.9208 - time: 0.1284
epoch: 001 - iter: 00999 - train loss: 4.8937 - time: 0.1328
epoch: 001 - iter: 01199 - train loss: 4.7339 - time: 0.1111
epoch: 001 - iter: 01399 - train loss: 4.6557 - time: 0.1318
epoch: 001 - iter: 01599 - train loss: 4.6376 - time: 0.1239
epoch: 001 - iter: 01799 - train loss: 4.5240 - time: 0.1126
epoch: 001 - iter: 01999 - train loss: 4.4562 - time: 0.1289
epoch: 001 - iter: 02199 - train loss: 4.3455 - time: 0.1285
epoch: 001 - iter: 02399 - train loss: 4.2996 - time: 0.1293
epoch: 001 - iter: 02497 - valid loss: 3.1694 - bleu score: 0.1072 - time: 151.4589
epoch: 002 - iter: 00199 - train loss: 4.1609 - time: 0.1289
epoch: 002 - iter: 00399 - train loss: 4.1451 - time: 0.1294
epoch: 002 - iter: 00599 - train loss: 4.0512 - time: 0.1327
epoch: 002 - iter: 00799 - train loss: 4.0498 - time: 0.1295
epoch: 002 - iter: 00999 - train loss: 4.0009 - time: 0.1270
epoch: 002 - iter: 01199 - train loss: 3.9476 - time: 0.1072
epoch: 002 - iter: 01399 - train loss: 3.9692 - time: 0.1329
epoch: 002 - iter: 01599 - train loss: 3.8881 - time: 0.1314
epoch: 002 - iter: 01799 - train loss: 3.8662 - time: 0.1100
epoch: 002 - iter: 01999 - train loss: 3.8042 - time: 0.1300
epoch: 002 - iter: 02199 - train loss: 3.8014 - time: 0.1299
epoch: 002 - iter: 02399 - train loss: 3.7214 - time: 0.1257
epoch: 002 - iter: 02497 - valid loss: 2.7862 - bleu score: 0.1741 - time: 163.0977
epoch: 003 - iter: 00199 - train loss: 3.5903 - time: 0.1264
epoch: 003 - iter: 00399 - train loss: 3.6542 - time: 0.1370
epoch: 003 - iter: 00599 - train loss: 3.6340 - time: 0.1280
epoch: 003 - iter: 00799 - train loss: 3.5999 - time: 0.1286
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: 3.5828 - time: 0.1296
epoch: 003 - iter: 01599 - train loss: 3.5253 - time: 0.1230
epoch: 003 - iter: 01799 - train loss: 3.5224 - time: 0.1263
epoch: 003 - iter: 01999 - train loss: 3.5052 - time: 0.1346
epoch: 003 - iter: 02199 - train loss: 3.4956 - time: 0.1257
epoch: 003 - iter: 02399 - train loss: 3.5021 - time: 0.1297
epoch: 003 - iter: 02497 - valid loss: 2.6176 - bleu score: 0.2073 - time: 172.6203
epoch: 004 - iter: 00199 - train loss: 3.3623 - time: 0.1337
epoch: 004 - iter: 00399 - train loss: 3.3587 - time: 0.1293
epoch: 004 - iter: 00599 - train loss: 3.3721 - time: 0.1255
epoch: 004 - iter: 00799 - train loss: 3.3160 - time: 0.1272
epoch: 004 - iter: 00999 - train loss: 3.3913 - time: 0.1115
epoch: 004 - iter: 01199 - train loss: 3.3439 - time: 0.1363

1 bleu(valid_src_data, valid_trg_data, model, SRC, TRG, opt['device'], opt['k'], opt['max_strlen'])

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
```

Kết quả trả về:

👤 'gia đình tôi không nghèo, và bản thân tôi, tôi chưa bao giờ trải qua nẠn đói.'

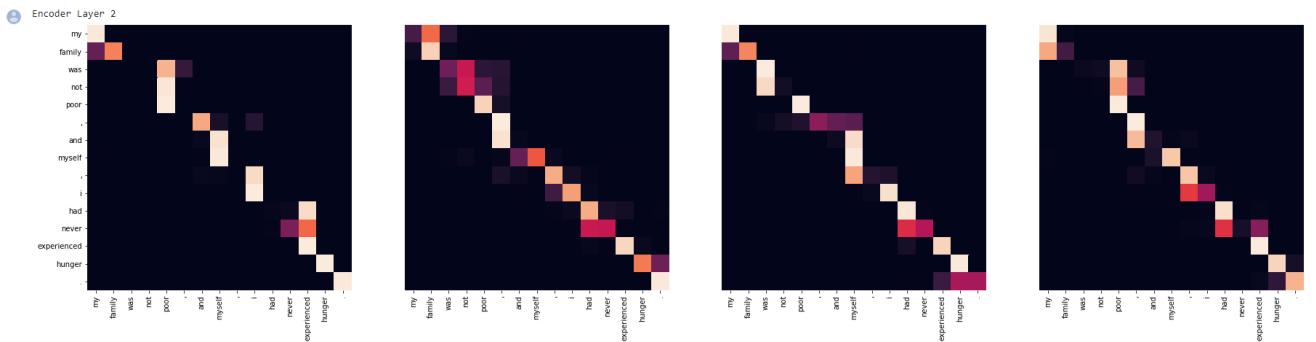
11.14 Visualize

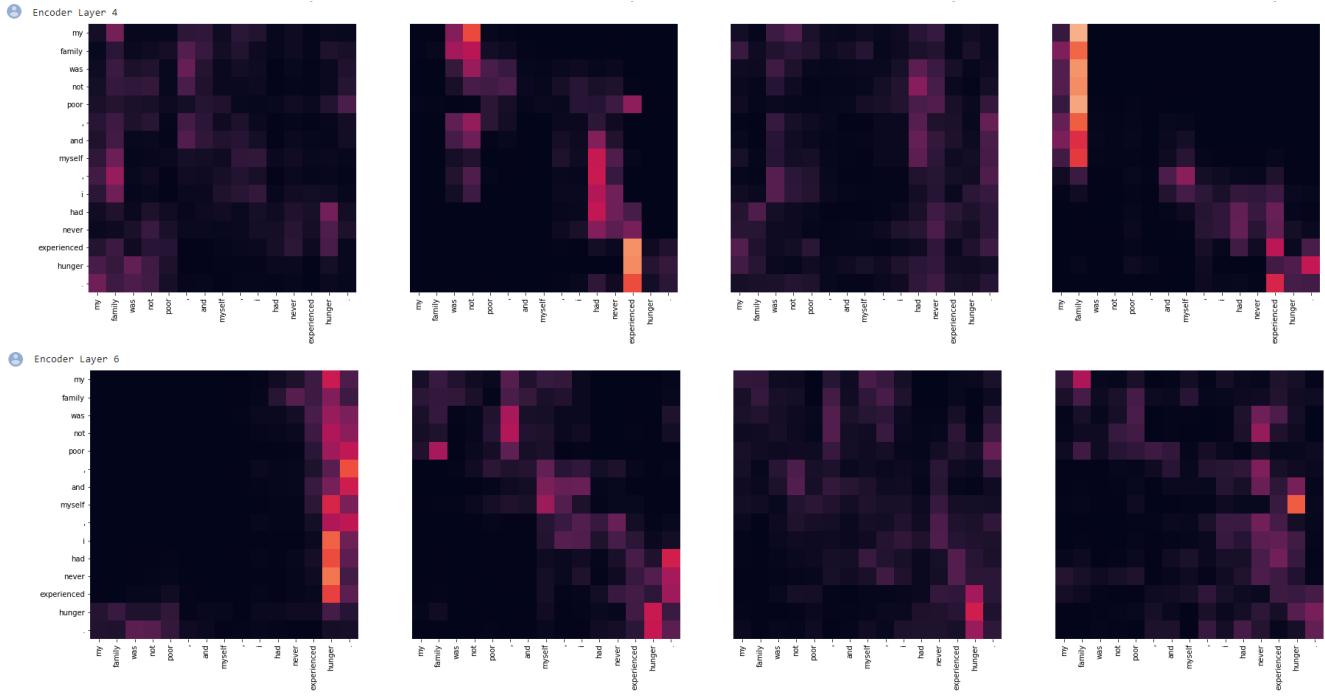
```
1 # ! gdown --id 1Ty1bGrd0sCwEqXhs0ViCUaNKa3lFwmPH
2 # model.load_state_dict(torch.load('./transformer.pth'))
3 import seaborn
4 import matplotlib.pyplot as plt
5
6 def draw(data, x, y, ax):
7     seaborn.heatmap(data,
8                     xticklabels=x, square=True, yticklabels=y, vmin=0.0, vmax=1.0,
9                     cbar=False, ax=ax, annot=False)
```

11.14.1 Visualize Encoder

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.

```
1 sent = SRC.preprocess(sentence)
2
3 for layer in range(1, 6, 2):
4     fig, axs = plt.subplots(1, 4, figsize=(30, 15))
5     print("Encoder Layer", layer+1)
6     for h in range(4):
7         draw(model.encoder.layers[layer].attn.attn[0, h].data.cpu(),
8              sent, sent if h == 0 else [], ax=axs[h])
9 plt.show()
```





11.14.2 Visualize Decoder

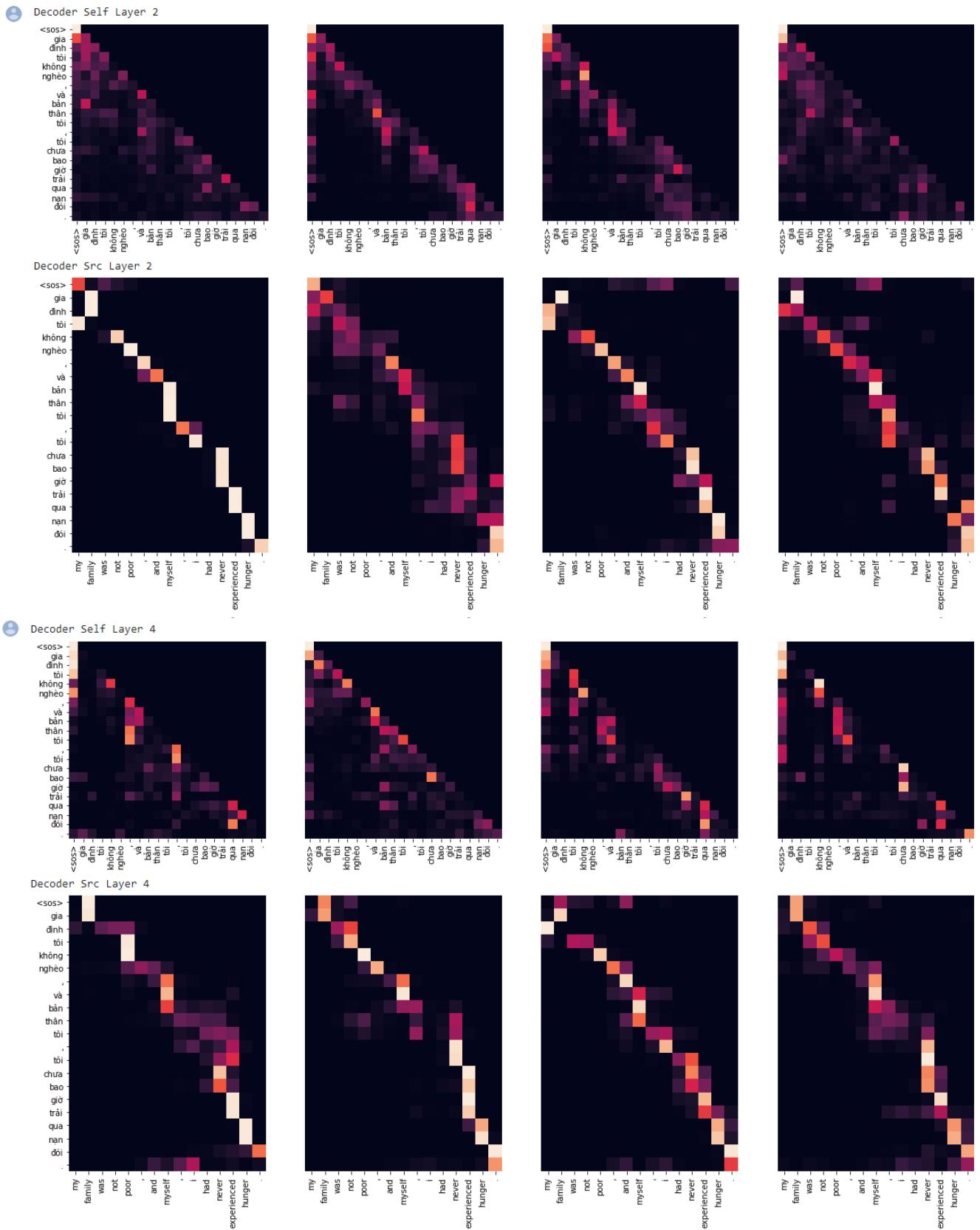
Ở decoder, các bạn có 2 loại visualization

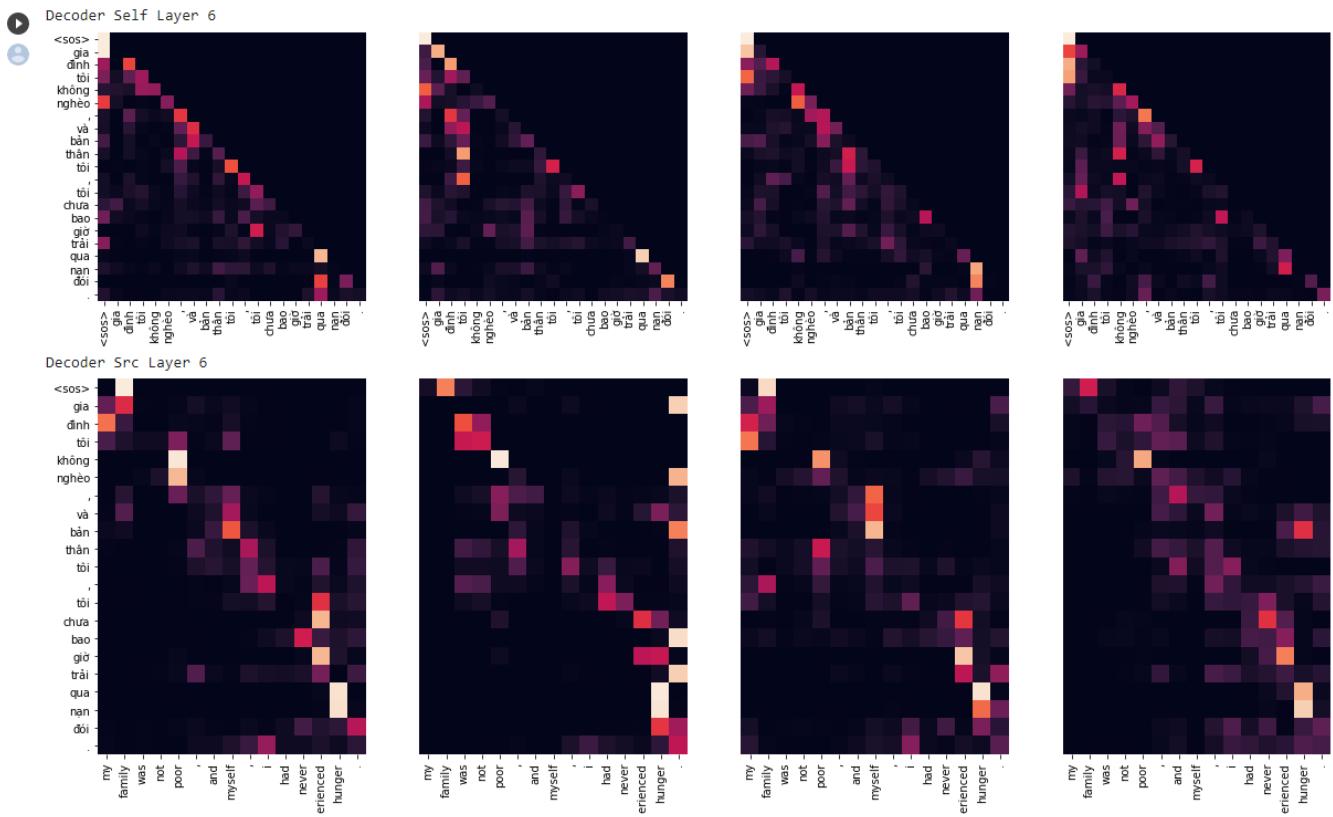
- **Self Attention:** giá trị attention khi mô hình decoder mã hóa câu đích lúc dịch.
- **Src Attention:** giá trị attention khi mô hình decoder sử dụng câu src.

```

1  trg_sent = ['<sos>'] + TRG.preprocess(trans_sent)
2
3  for layer in range(1, 6, 2):
4      fig, axs = plt.subplots(1,4, figsize=(20, 10))
5      print("Decoder Self Layer", layer+1)
6      for h in range(4):
7          draw(model.decoder.layers[layer].attn_1.attn[0, h].data[:len(trg_sent), :len(trg_sent)].cpu(),
8               trg_sent, trg_sent if h ==0 else [], ax=axs[h])
9      plt.show()
10     print("Decoder Src Layer", layer+1)
11     fig, axs = plt.subplots(1,4, figsize=(20, 10))
12     for h in range(4):
13         draw(model.decoder.layers[layer].attn_2.attn[0, h].data[:len(trg_sent), :len(sent)].cpu(),
14              sent, trg_sent if h ==0 else [], ax=axs[h])
15     plt.show()

```





References

- [1] Artetxe, M. and Schwenk, H. (2019). Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. TACL 7, 597–610.
- [2] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, et al. 2018. Achieving human parity on automatic Chinese to English news translation.
- [3] Vishrav Chaudhary, Yuqing Tang, Francisco Guzman, Holger Schwenk, and Philipp Koehn. 2019. Lowresource corpus filtering using multilingual sentence embeddings. In Proceedings of the Fourth Conference on Machine Translation (Volume 3: Shared Task Papers, Day 2), pages 263–268, Florence, Italy. Association for Computational Linguistics.
- [4] Mandy Guo, Qinlan Shen, Yinfei Yang, Heming Ge, Daniel Cer, Gustavo Hernandez Abrego, Keith Stevens, Noah Constant, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Effective parallel corpus mining using bilingual sentence embeddings. In Proceedings of the Third Conference on Machine Translation: Research Papers, pages 165–176, Belgium, Brussels. Association for Computational Linguistics.
- [5] Thompson, B. and Koehn, P. (2019). Vecalign: Improved sentence alignment in linear time and space. EMNLP.
- [6] Miller, G. A. and Beebe-Center, J. G. (1958). Some psychological methods for evaluating the quality of translations. Mechanical Translation 3, 73–80.
- [7] Post, M. (2018). A call for clarity in reporting BLEU scores. WMT 2018.
- [8] Callison-Burch, C., Osborne, M., and Koehn, P. (2006). Re-evaluating the role of BLEU in machine translation research. EACL.
- [9] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation.