ĐẠI HỌC THÁI NGUYÊN

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

****

**BÁO CÁO BÀI TẬP LỚN MÔN**

**Xử lý ngôn ngữ tự nhiên**

# TÊN ĐỀ TÀI: Tìm hiểu về bài toán dịch máy (Machine Translate)

**Nhóm số 5**

**Sinh viên thực hiện:** Khuất Đình Lễ

Trần Trung Kiên

Bùi Quang Huy

# Thái Nguyên: 07/03/2022

**NỘI DUNG CHÍNH CỦA BÁO CÁO**

* 1. **Giới thiệu bài toán**

Nhu cầu của các dịch thuật đối với con người hiện nay :

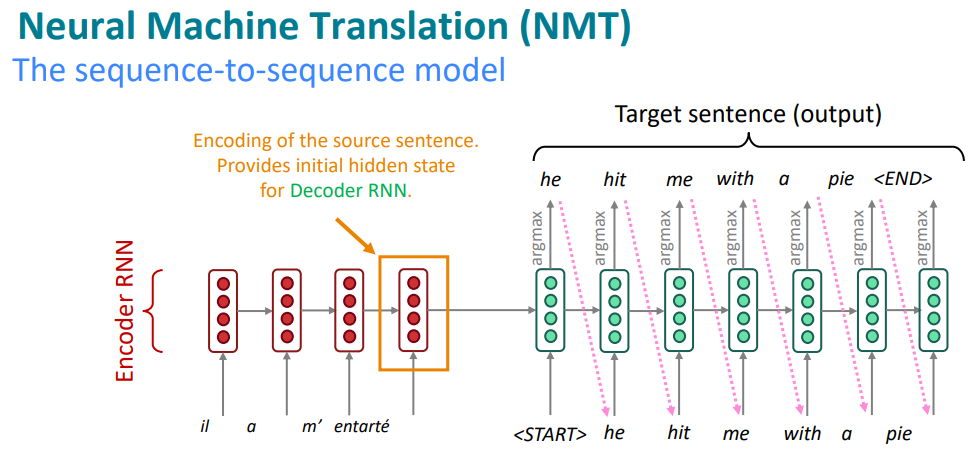
* Có nhiều ngôn ngữ mà con người không thể học hết tất cả.
* Nhu cầu dịch nhanh và chính xác đối với mỗi ngôn ngữ.
* Không phải ai cũng có điều kiện học ngoại ngữ.
* Sự ra đời của các máy dịch ngôn ngữ (ứng dụng của dịch).

# Neural Machine Translation

* **Khái niệm về Neural Machine Translation :**

Neural Machine Translation(NMT) là một cách để cho máy tính thực hiện dịch ngôn ngữ với một end-to-end neural network.

Kiến trúc neural network này được gọi là sequence-to-sequence model(aka seq2seq).

Dưới đây là hình ảnh minh họa:

* **Seq2Seq :**

Seq2Seq là một mô hình được đề xuất từ năm 2014 cho việc dịch tiếng Anh-tiếng Pháp. Seq2Seq cũng là một Conditional Language Model. NMT-Seq2Seq tính toán *P*(*y*∣*x*) một cách trực tiếp:

P(y|x) = *P*(*y*1​∣*x*)*P*(*y*2​∣*y*1​,*x*)*P*(*y*3​∣*y*1​,*y*2​,*y*3​,*x*)...*P*(*yT*​∣*y*1​...*yT*−1​,*x*)

với T là số bước thời gian.

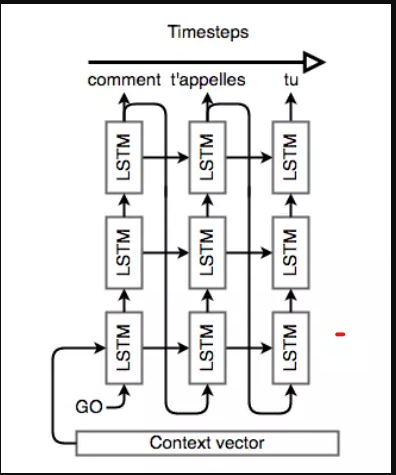
Ở một cấp độ cao hơn, một mô hình Seq2Seq là một mô hình end-to-end với 2 mạng RNN(hoặc các biến thể của RNN):

* Encoder: Bộ mã hóa lấy câu nguồn làm đầu vào của mô hình và encode nó thành một context-vector có kích thước cố định.
* Decoder: Bộ giải mã sử dụng context-vector của encoder làm khởi tạo cho hidden-state đầu tiên.

#### **Kiến trúc Seq2Seq – Encoder :**

Nhiệm vụ của Encoder là đọc câu nguồn cho mô hình Seq2Seq và tạo context-vector C*C* có kích thước cố định. Để làm được điều này, Encoder sử dụng một mạng RNN(thường là LSTM) để đọc từng token tại một bước thời gian t*t*. Hidden-state cuối cùng sẽ trở thành C*C*. Tuy nhiên, rất khó khăn để nén một chuỗi có độ dài tùy ý vào một context-vector có kích thước cố định duy nhất(đặc biệt với các nhiệm vụ khó như dịch thuật). Vì vậy encoder sử dụng multi-layer.

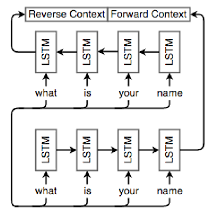
Encoder cũng làm một việc tương đối kỳ lạ đó là xử lý chuỗi theo trình tự ngược lại. Nghe vô lý những lại hết sức hợp lý . Việc xử lý như vậy giúp cho Decoder thực hiện một cách tốt hơn khi có lượng thông tin ở phía đầu chuỗi nguồn nhiều hơn, giúp cho việc dịch được tốt hơn ở những bước đầu, hay hiểu nôm na là: "Đầu có xuôi thì đuôi mới lọt". Sau đây là hình ảnh minh họa về encoder sử dụng LSTM:



##### **Bidirectional RNNs**

Để giúp model nắm bắt tốt sự phụ thuộc trong câu nguồn, ta sử dụng Bidirectional RNNs.

Minh họa cho việc encoder sử dụng Bidirectional RNN:



#### **Kiến trúc Seq2Seq-Decoder**

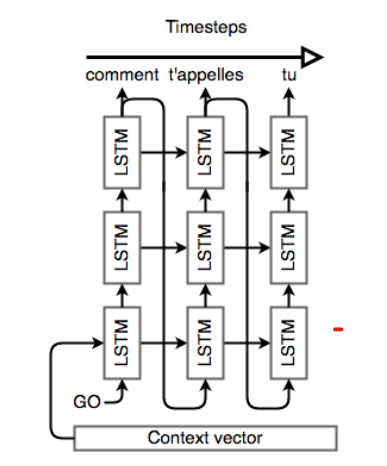
Decoder cũng là một mạng LSTM nhưng sử dụng phức tạp hơn encoder một chút. Về cơ bản ta sử dụng nó để tạo ra các từ ở mỗi bước thời gian. Để làm được điều đó, ta sử dụng multi-layer và khởi tạo hidden-state của layer đầu tiên bằng việc sử dụng context-vector của encoder.

Sau khi Decoder đã được thiết lập với context của encoder Sau đó ta sẽ đặt <EOS><*EOS*> token vào cuối cùng của input(thêm một bước thời gian của encoder) hoặc ta có thể đặt <SOS><*SOS*> vào đầu của output. Sau đó ta sẽ chạy qua các layer của LSTM theo sau cùng là softmax để tạo đầu ra. Ta lặp lại công việc trên qua các bước thời gian.

Khi ta có trình tự đầu ra, bây giờ sẽ đến công việc tính loss. Ở đây ta sử dụng Cross-Entropy Loss.

Một lưu ý là ta cả encoder và decoder đều được training tại cùng một thời điểm, việc này giúp cả hai cùng học được một cách biểu diễn context-vector.

Sau đây là hình ảnh minh họa cho decoder:



Lưu ý không có sự liên quan giữa độ dài của chuỗi nguồn và chuỗi đích. Tuy nhiên Seq2Seq thường kém hiệu quả với các chuỗi nguồn dài do các giới hạn thực tế của LSTM.

##### **Các cách chọn đầu ra cho Seq2Seq-NMT**

Giả sử bây giờ ta đã có được đầu ra softmax của các hidden-state. Công việc của chúng ta bây giờ là phải chọn chuỗi nguồn sao cho hợp lý vì không phải cứ lấy xác suất cao nhất của softmax của tất cả các bước thời gian là tốt. Tính toán đầu ra cuối cùng này thực chất đến từ SMT. Xem xét một mô hình tính toán P(y|x)*P*(*y*∣*x*) của câu đích y và câu nguồn x. Ta muốn tìm argmax\_yP(y|x)*argmaxy*​*P*(*y*∣*x*)

Vì không gian tìm kiếm có thể rất lớn, ta phải thu nhỏ kích thước của nó. Dưới đây là một số cách tiếp cận:

* Exhaustive search: Đây là ý tưởng đơn giản nhất. Chúng tôi tính toán xác xuất của mọi trường hợp có thể xảy ra rồi chọn lấy cái có xác suất cao nhất. Tuy nhiên cách này bất khả thi vì không gian tìm kiếm tăng theo cấp số nhân của input-size.
* Ancestral sampling: Ta thực hiện tính toán xác suất có điều kiện ở bước thời gian T chỉ phụ thuộc vào một số bước thời gian ở đằng trước:

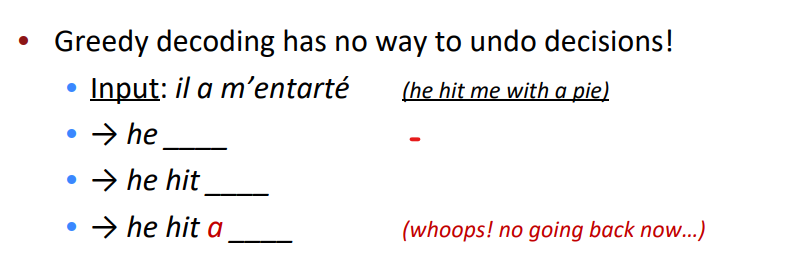
y\_t \sim P(y\_t|y\_1,..,y\_n)*yt*​∼*P*(*yt*​∣*y*1​,..,*yn*​)

Về mặt lý thuyết, cách này hiệu quả và chính xác về mặt góc nhìn. Tuy nhiên, trên thực tế cách này có hiệu suất thấp và phương sai khá cao.

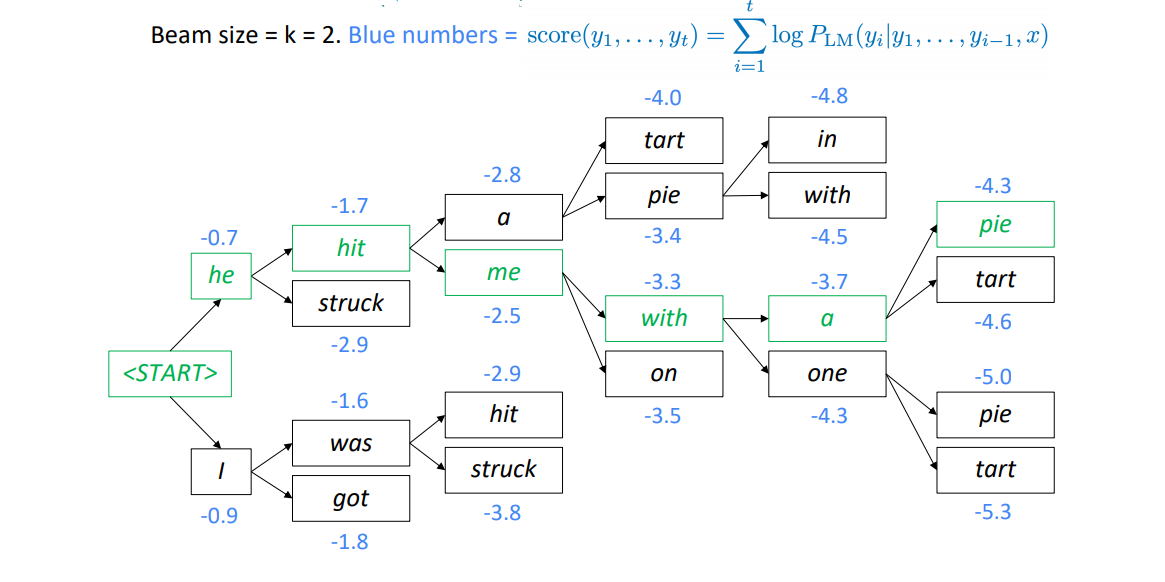
* Greedy Search : Tại mỗi bước thời gian ta chọn xác suất cao nhất có thể xảy ra. Nói cách khác:

y\_t = argmax\_{y\_t}P(y\_t|y\_1,...,y\_{t-1})*yt*​=*argmaxyt*​​*P*(*yt*​∣*y*1​,...,*yt*−1​)

Cách làm này hiệu quả và tương đối tự nhiên. Tuy nhiên, nó không thể khám phá hết tất cả các trường hợp có thể xảy ra trong không gian xác suất của câu, và nếu nó mắc lỗi ở một bước thời gian thì hậu quả là tương đối lơn. Xem ví dụ sau:



Beam Search: Đây là lựa chọn phổ biến và hiệu quả nhất cho đến thời điểm hiện tai. Ý tưởng của beam search là sẽ chọn lấy K*K* xác suất lớn nhất của một softmax trong một bước thời gian và chỉ xem K*K* sự lựa chọn cho một bước thời gian, cụ thể được giải thích trong hình sau:



Beam search không đảm bảo tìm được trường hợp tối ưu, nhưng hiệu quả hơn khá nhiều so với greedy search và exhaustive search.

##### **Ví dụ về Seq2Seq-NMT:**

Giả sử ta muốn dịch một câu tiếng Anh: "What is your name" sang một câu tiếng Pháp: "Quel est ton nom". Đầu tiên ta sẽ bắt đầu với 4 one-hot vectors cho đầu vào. Sau đó ta sẽ sử dụng embedding từ cho 4 one-hot vectors này. Tiếp theo ta đưa chúng vào multi-layer LSTM với việc đọc câu theo thứ tự ngược lại và mã hóa chúng thành một context-vector(Encoder). Context-vector này được sử dụng để khởi tạo hidden-state của layer đầu tiên của một multi-layer LSTM khác. Sau đó thực hiện softmax trên đầu ra của hidden-layer cuối cùng ta được từ đầu tiên của chuỗi đích. Kể từ bước thời gian tiếp theo, ta có 2 sự lựa chọn, hoặc dùng đầu ra của bước thời gian trước làm đầu vào, hoặc dùng từ của câu nguồn của tệp dữ liệu để tiến hành làm đầu vào(teacher forcing). Sau đó tiếp tục tính softmax của hidden-layer cuối cùng. Trong quá trình backprop, weight của LSTM-encoder được cập nhật để giúp cho việc biểu diễn không gian vector từ tốt hơn, trong khi đó weight của LSTM-decoder được cập nhật để tạo ra các câu chính xác về mặt ngữ pháp liên quan đến context-vector của encoder.

### **Ưu điểm của NMT**

* Cho hiệu suất tốt hơn các mô hình Machine Translation khác.
* Sử dụng context hiệu quả hơn.
* Sử dụng các cụm từ tương đồng tốt hơn.
* Cài đặt chỉ bằng một end-to-end neural network
* Cần ít nỗ lực human-engineering hơn SMT

### **Nhược điểm của NMT**

* Khó có thể hiểu được sâu bên trong một mô hình NMT đang thực hiện những gì nên rất khó để debug.
* NMT rất khó để kiểm soát.

**3.Attention**

Ý tưởng của Attention: thay vì chỉ sử dụng 1 context vector duy nhất (last state), decoder có thể đọc thông tin từ tất cả state trong encode cho mỗi lần predict. Hãy quan sát 2 câu sau:

* **This is a book**
* **Đó là một quển sách**

Có thể thấy sự tương ứng giữa các từ ngữ giữa 2 câu: "this"-"đó" , "book"-"quyển sách". Như vậy việc dịch ra "quyển sách" sẽ phụ thuộc vào "book" hơn là "this", "is" hay "a".

Ý tưởng cốt lõi của Attention:

* Thay vì sử dụng 1 context vector duy nhất (last state) khi dịch tất cả các từ, ta sử dụng mỗi context vector riêng biệt khi predict ra từng từ.
* Mỗi context vector được tổng hợp có trọng số từ tất cả các state trong encode.

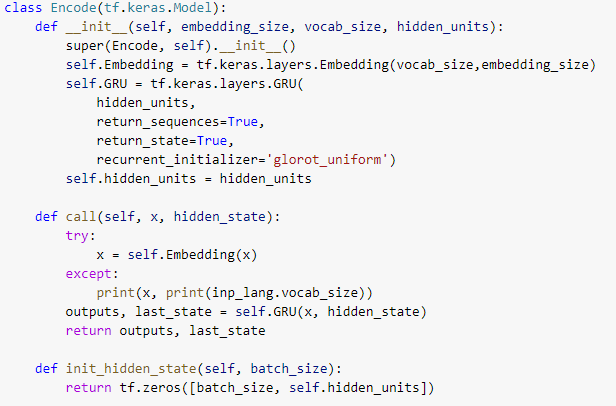
### **Chi tiết thuật toán**

# 

**Encoder :**

Encode 1 câu độ dài n*n* bằng LSTM hoặc GRU, ta thu được n*n* encode\_states: H = [h\_1, h\_2, h\_3 ... h\_n][*h*1​,*h*2​,*h*3​...*hn*​]

Nhiệm vụ của Encoder là đọc câu nguồn cho mô hình Seq2Seq và tạo context-vector C*C* có kích thước cố định. Để làm được điều này, Encoder sử dụng một mạng RNN(thường là LSTM) để đọc từng token tại một bước thời gian t*t*. Hidden-state cuối cùng sẽ trở thành C*C*. Tuy nhiên, rất khó khăn để nén một chuỗi có độ dài tùy ý vào một context-vector có kích thước cố định duy nhất(đặc biệt với các nhiệm vụ khó như dịch thuật). Vì vậy encoder sử dụng multi-layer.



**Decoder** :

Tại decode, để predict ra word\_j*wordj*​, ta cần tính context\_vector CV\_j*CVj*​ tương ứng. Attention dùng khái niệm query, key, value:

* keys=[k\_1, k\_2 ,.. k\_n]*keys*=[*k*1​,*k*2​,..*kn*​], values=[v\_1,v\_2 ... v\_n]*values*=[*v*1​,*v*2​...*vn*​]. Thông thường, keys*keys* = values*values* = [h\_1, h\_2 ... h\_n][*h*1​,*h*2​...*hn*​] (là encode\_states)
* queries = [q\_1, q\_2 ... q\_m]*queries*=[*q*1​,*q*2​...*qm*​] = [h\_n, d\_1, d2, ... d\_{m-1}][*hn*​,*d*1​,*d*2,...*dm*−1​], với m là độ dài chuỗi được sinh ra. Đặc biệt q\_1 = h\_n*q*1​=*hn*​ , còn d\_1 \to d\_{m-1}*d*1​→*dm*−1​ là các decode\_states
* Context vector (CV\_j*CVj*​) là tổng hợp có trọng số các phần tử thuộc values. Trong công thức dưới đây, w\_i^j*wij*​ = f(q\_j, k\_i)*f*(*qj*​,*ki*​) là hàm tính trọng số. Mình sẽ nói chi tiết ở phần 3.

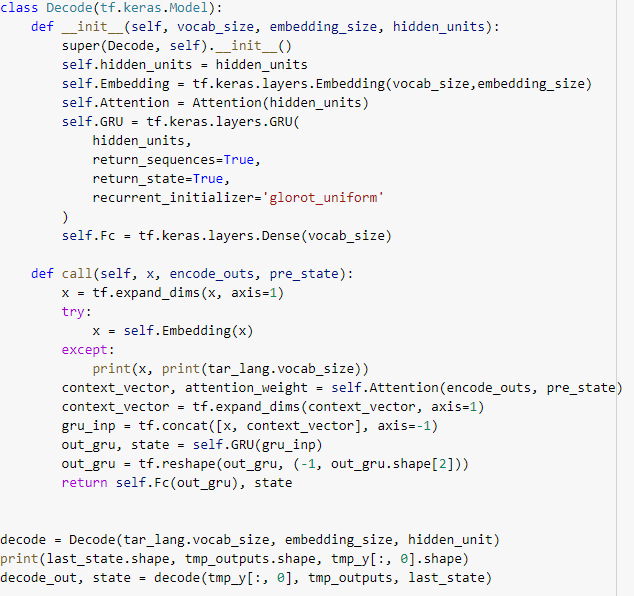
*CVj*​=Σ*i*=0*n*​(*wij*​∗*vi*​)=Σ*i*=0*n*​[*f*(*qj*​,*ki*​)∗*vi*​]

Kết hợp *CVj*​ với từ liền trước output\_{j-1}*outputj*−1​ (tại step 1, output\_1*output*1​ = "<START>"), thu được concat\_vector\_j = CV\_j*CVj*​ + output\_{j-1}*outputj*−1​.

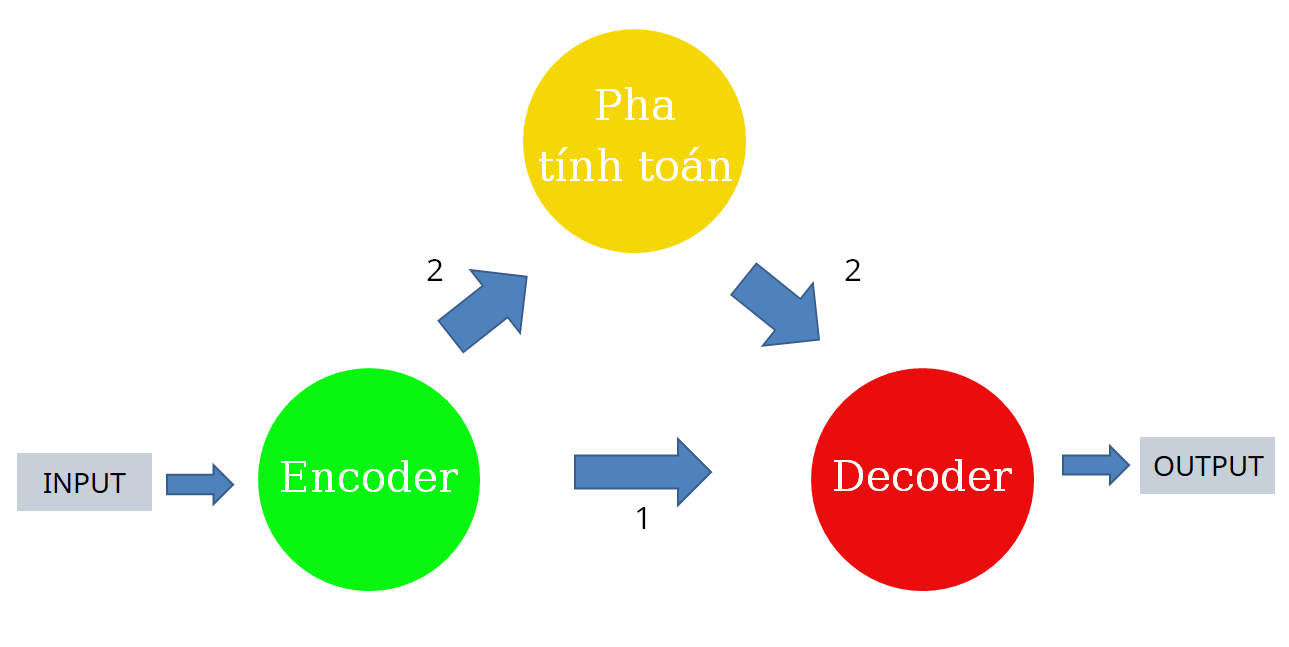
Đưa concat\_vector vào GRU/LSTM, thu được output = "Rahul" và d\_i*di*​. Quá trình được lặp lại tương tự cho từng lần predict các từ kế tiếp (cho đến khi sinh ra kí tự <END>).

Decoder cũng là một mạng LSTM nhưng sử dụng phức tạp hơn encoder một chút. Về cơ bản ta sử dụng nó để tạo ra các từ ở mỗi bước thời gian. Để làm được điều đó, ta sử dụng multi-layer và khởi tạo hidden-state của layer đầu tiên bằng việc sử dụng context-vector của encoder.

Sau khi Decoder đã được thiết lập với context của encoder Sau đó ta sẽ đặt <EOS><*EOS*> token vào cuối cùng của input(thêm một bước thời gian của encoder) hoặc ta có thể đặt <SOS><*SOS*> vào đầu của output. Sau đó ta sẽ chạy qua các layer của LSTM theo sau cùng là softmax để tạo đầu ra. Ta lặp lại công việc trên qua các bước thời gian.



**Pha tính toán** : Pha tính toán xác suất xảy ra của một sự kiện. Nói một cách khái quát, hàm softmax sẽ tính khả năng xuất hiện của một class trong tổng số tất cả các class có thể xuất hiện. Sau đó, xác suất này sẽ được sử dụng để xác định class mục tiêu cho các input.

****

Cách thức hoạt động

* **Cách hoạt động :**

Bước 1 : Từ encoder sang decoder (1) , sẽ dịch word by word (W2W) và cho ra 1 hàm softmax.

Bước 2: Nếu phần dịch chỉ có 1 câu ( hoặc từ ) thì kết thúc nếu không sẽ so sánh với các softmax trước đó và dữ liệu của file train để cho ra kết quả đúng nhất (2).

Bước 3 : Dựa vào kết quả đúng nhất, sau đó dịch theo kết quả đó từ dữ liệu file train.Phần dịch sẽ dừng khi phần dự đoán đến phần có kí tự <END> trong bộ từ vựng .

1. **Dataset :**

Dataset: gồm 2 file train.en.txt và train.vi.txt gồm 100.000 câu song ngữ Anh-Việt

* **Ưu và nhược điểm :**
* **Ưu điểm**
* Cho hiệu suất tốt hơn các mô hình Machine Learning Translation khác.
* Sử dụng context hiệu quả hơn.
* Sử dụng các cụm từ tương đồng tốt nhất.
* **Nhược điểm :**
* Khó có thể hiểu được sâu bên trong mô hình NMT đang thực hiện những gì nên rất khó debug
* **Đánh giá :**

Như vậy, nhóm mình đã giới thiệu một cách tổng quan về NMT. Hiện nay rất nhiều mô hình NMT đã được sinh ra dựa trên Attention, Transformer, BERT, v.v. Nhưng để hiểu được những mô hình NMT mới này, ta chắc chắn phải nắm được gốc dễ của NMT.

**Tài liệu tham khảo và cách trích dẫn**

# [1] Tổng quan về Neural Machine Translation:

# <https://viblo.asia/p/tong-quan-ve-neural-machine-translation-E375zrMd5GW?fbclid=IwAR1eYY5EW4c_alr1bJlB6CgJu94_yle-KJiGHh84EqRm0AY8H_-klT3RQvA>

# [2] Machine Translate với Attention trong Deep Learning:

<https://viblo.asia/p/machine-translate-voi-attention-trong-deep-learning-Az45bY7zlxY?fbclid=IwAR1rOxICE7D_eqI-KHttncgcR789KPhHtxp3MbKa97T8-ThG_Mt1mCKXrhM>

[3] Machine Translation, Sequence-to-Sequence and Attention:

<https://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture07-nmt.pdf>

[4] Kênh youtube stanfordonline: <https://www.youtube.com/watch?v=wzfWHP6SXxY&list=PLoROMvodv4rOSH4v6133s9LFPRHjEmbmJ&index=8>

[5] Source code tham khảo :

[pet-translate/translate.ipynb at master · trungthanhnguyen0502/pet-translate (github.com)](https://github.com/trungthanhnguyen0502/pet-translate/blob/master/translate.ipynb)