Hệ thống dịch multilingual

# 1. Tổng quan

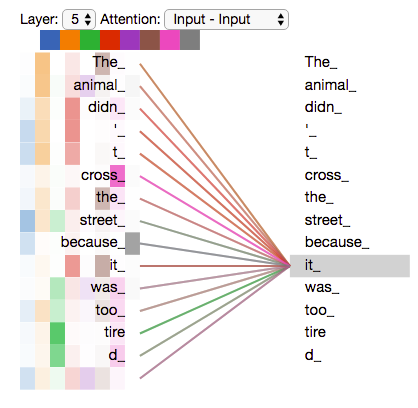
Các hệ thống dịch hiện tại thường được tổ chức dưới dạng 1-1, tức ngôn ngữ nguồn - ngôn ngữ đích. Điều này đồng nghĩa với việc các hệ thống dịch đa hướng sẽ cần nhiều model cùng chạy, dẫn đến tốn kém và lãng phí bộ nhớ.

Ngoài ra, các hệ thống NMT đang có kết quả cao với dữ liệu vừa và lớn (>100k song ngữ), dẫn đến việc các ngôn ngữ hiếm và ít sử dụng không có đủ dữ liệu để tạo ra các model có kết quả cao.

Phương án giải quyết: Sử dụng các mô hình multilingual để có thể dùng chung cho nhiều bộ dữ liệu khác nhau, đạt được hiệu năng cao trên dữ liệu ít.

# 2. Cấu trúc hệ thống

Về mặt bản chất, các hệ thống multilingual sẽ không khác nhiều với mô hình hiện tại như Seq2Seq, Attention, Transformer. Mục tiêu chính trong hiện tại sẽ là cấu trúc Transformer.

Cấu trúc trên sử dụng bước Self-Attention làm phương án chính để truy cập thông tin đã có cho cả Encoder và Decoder, thay vì đi tuần tự qua các timestep như các phương pháp Seq2Seq thông thường. Ngoài lợi thế này, Self-Attention kết hợp với cấu trúc Multi-Head cho phép một từ có thể truy cập vào và lấy thông tin từ nhiều vị trí khác nhau mà không làm tăng lượng tính toán cần thiết cho hệ thống.

Để hệ thống có thể adapt được cho bài toán multilingual, chúng ta sẽ sử dụng các cấu trúc thường thấy cho những bài toán này như dùng chung embedding, pretrain decoder, backtranslation, etc.

# 2.1 Mô hình

Như sẽ thấy ở hình bên, chúng ta sẽ có các lớp thực hiện tùy theo module tương ứng:

- **Embedding:** Vector [vocab\_size, embedding\_size], khởi tạo ngẫu nhiên, từ id của từ → vector cỡ embedding\_size tương ứng. Có lựa chọn load từ bên ngoài vào.

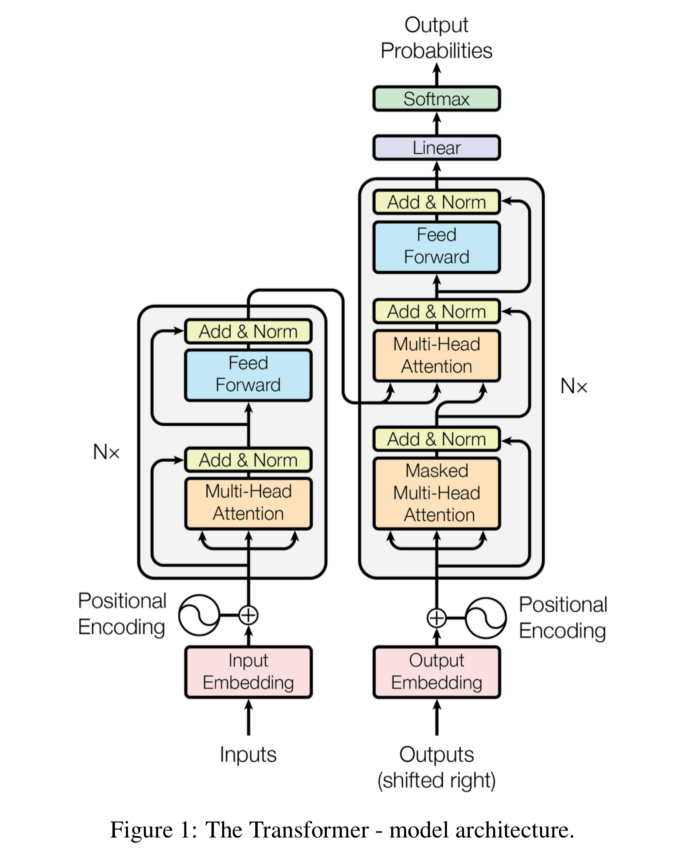
- **Feedforward:** Một Dense Layer với cấu trúc và activation tùy chọn, nhưng kích cỡ input và output phải giống nhau [num\_units → num\_units] để hỗ trợ cho Add&Norm.

- **Add&Norm:** Cộng hai vector lại với nhau và normalize theo L2. [num\_units → num\_units]

- **Linear:** Tương tự là Dense Layer, nhưng kích cỡ output phải là vocab\_size của ngôn ngữ đích [num\_units → vocab\_size].

- **Softmax:** Đã có, chuyển probability trên toàn vocab về sum=1.

- **Dropout:** Không có trong hình, thường sẽ kết hợp với các layer Add&Norm để thực hiện dropout cho các vector. [num\_units → num\_units]

- **Positional Encoding:** Layer [num\_units → num\_units], cộng vector vị trí vào trong câu.

- **Multi-Head Attention:** Layer atttention dạng multi-head. Trên hình có 2 layer cùng tên này, nhưng thực hiện khác nhau:

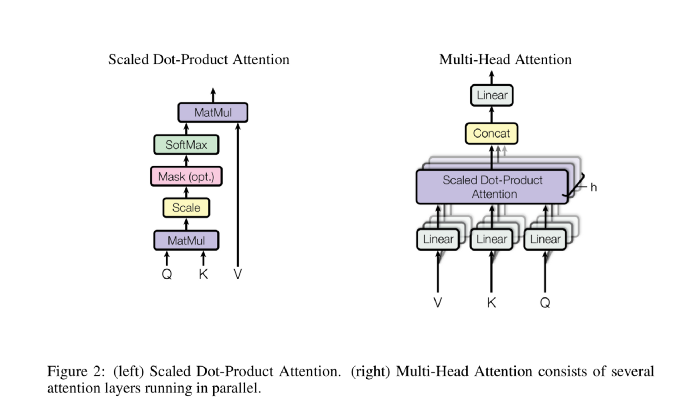
+ Layer thuộc về Encoder (bên trái) tự lấy attention trên chiều của bản thân (self-attention, có thể nhân ma trận đơn vị để tránh attention chính nó). Giá trị attention sinh ra sẽ là [num\_heads, src\_len, src\_len]

+ Layer thuộc về Decoder (bên phải) sẽ lấy attention trên đầu ra của encoder tùy chọn. Giá trị attention sinh ra sẽ là [num\_heads, tgt\_len, src\_len]

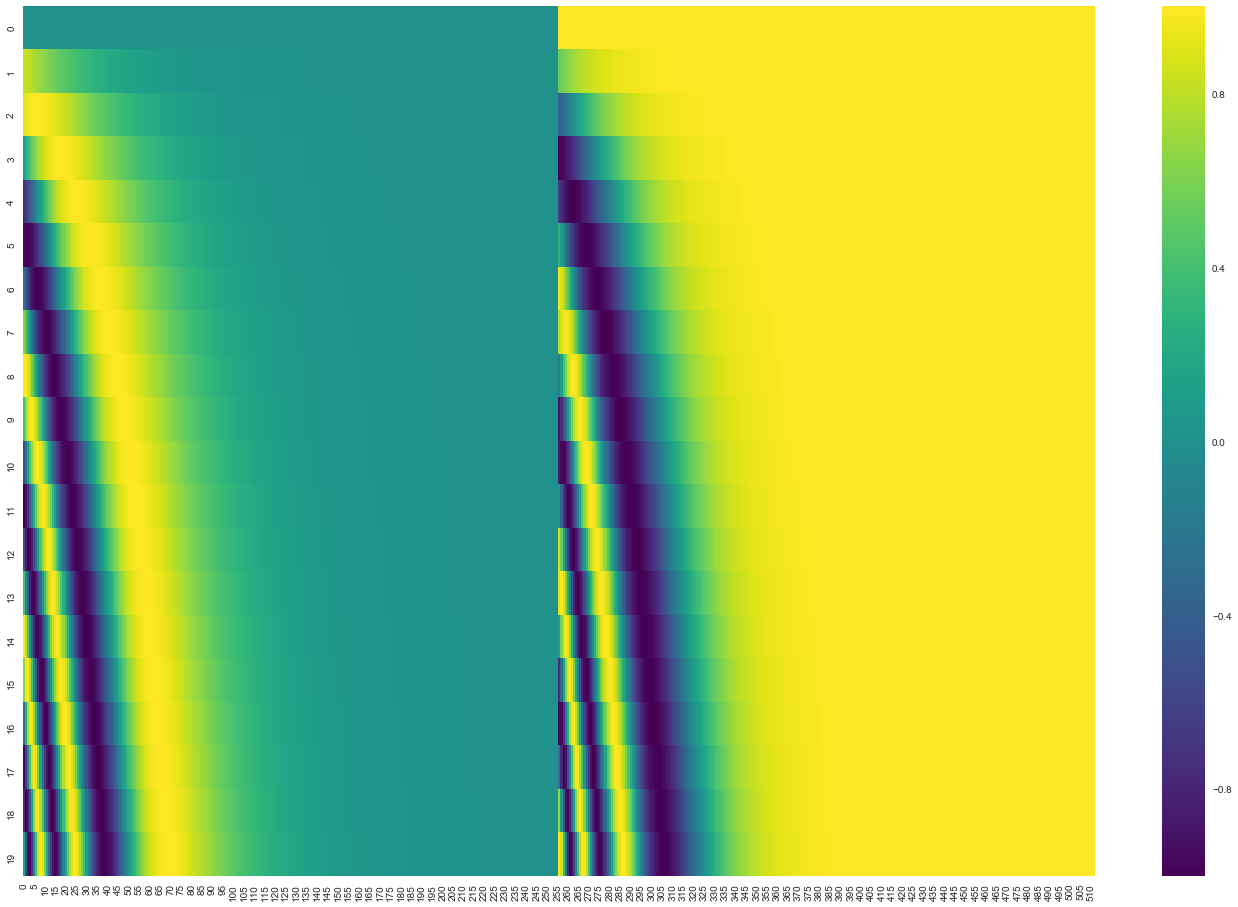
- **Masked Multi-Head Attention:** Về bản chất, layer này chính là layer self-attention ở trên, nhưng do ở bước Decoder nên có phức tạp: Trong training, chúng ta phải generate một mask tam giác dưới để đảm bảo mỗi từ chỉ có thể attention vào các thông tin trước nó; Trong infer, attention này sẽ thực hiện với các timestep trước như các encoder NMT thông thường. Giá trị attention sinh ra sẽ là [num\_heads, tgt\_len, tgt\_len]

Các lớp con này sẽ tổng hợp lại thành lớp Self-Attention lớn hơn cho cả Encoder và Decoder; cho phép chồng lên nhau với số lượng tùy ý.

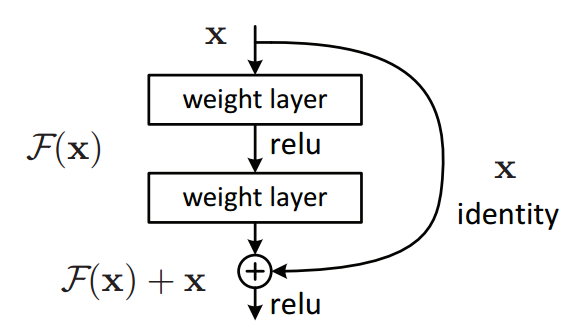
# 2.2 Multi-Head Attention

Cấu trúc Multi-Head là cấu trúc mở rộng của Attention, cho phép attention vào nhiều khu vực hơn và giảm lượng tính toán. Với cấu trúc attention gốc, chúng ta có các giá trị Q, K, V (query, key, value), với query là giá trị cần tìm kiếm attention; key và value là giá trị để tính attention (Q º K) và giá trị để tạo dot product kết quả. Multi-Head về mặt bản chất là tách Q, K, V theo chiều cuối ra thành num\_heads phần, tính toán attention và dot product mỗi phần độc lập với nhau, và gắn các phần đó lại bằng hàm concat.

# 2.3 Positional Encoding

Trong quá trình chạy Encoding/Decoding, phương án để một từ có thể nhận thông tin của các từ trước nó hoặc một lượng từ gốc là qua quá trình attention. Điều này dẫn đến việc thông tin giữa các từ với nhau không chứa vị trí của từ đó trong câu, làm ảnh hưởng đến chất lượng của bản dịch (đặc biệt là ở chiều encoder, khi không có phương pháp Positional Encoding đầu vào sẽ tương đương bag-of-word thay vì một câu như bình thường). Để xử lý vấn đề này, các embedding sẽ được cộng vào một vector Positional có giá trị liên quan đến vị trí của mỗi từ trong câu.

# 2.4 Residual Connection (Add&Norm)

Để đạt được kết quả như mong muốn, thông tin của mô hình dịch sẽ phải trải qua rất nhiều tầng biến đổi, có thể dẫn đến hiện tượng gradient vanishing hoặc exploding. Các cấu trúc NMT gốc dùng LSTM cũng chỉ có thể xử lý được hiện tượng này theo độ dài timestep; do đó với các cấu trúc sâu như Transformer, vẫn sẽ phải thực hiện kết nối giữa các tầng để đảm bảo gradient luôn được truyền xuống.

# 3. Cấu trúc chương trình

Hệ thống dự tính sẽ được xây dựng theo các class có thể kế thừa được với nhau, để đảm bảo ổn định trong quá trình nâng cấp và theo cấu trúc chung của pytorch. Các lớp chính sẽ là:

- Main: lớp chạy gốc của python, dự tính sẽ sử dụng argparse để tạo chương trình ở command line và chạy những option không ảnh hưởng đến mô hình (debug level, visualization, etc.); có nhiệm vụ load/save/run cho model cho các tác vụ khác nhau

- Config: không hẳn là một lớp, nhưng sẽ có cấu trúc dạng nested dict để lưu giữ các thông tin quan trọng của mô hình như hyperparameters, layer size, etc.; mỗi lần chạy sẽ luôn lấy thông tin từ config để xây dựng lại mô hình. Dự định sẽ lưu lại dạng yml/json và luôn giữ lại một bản trong quá trình chạy.

- Model: Mô hình của hệ thống. Sẽ tuân thủ các interface quan trọng với class Module của pytorch gồm train/eval/run và đọc các hyperparameter cần thiết từ Config khi tạo mô hình.

- Module: Gồm các thành phần có nhiệm vụ riêng biệt không cản trở lẫn nhau, lần lượt sẽ dùng để xử lý dữ liệu (Loader), mã hóa (Encoder) và giải mã (Decoder)

- Layers: Gồm các cấu trúc lớp ẩn khác nhau để cấu trúc nên hệ thống mạng neuron, tương đương với các lớp Layers đã có như Dense, Dropout, Normalization, etc.

### Tổng quan luồng hoạt động của hệ thống

Hệ thống sẽ chạy ở cấp bậc câu. Khi xây dựng mô hình, chúng ta sẽ thực hiện train dùng các bộ dữ liệu có sẵn và evaluate bằng các metric thông dụng (e.g BLEU). Format của các input này sẽ được xác định theo cấu trúc của mô hình. Khi dựng mô hình để sử dụng, chúng ta sẽ deploy mô hình qua Serving của Torch, với input là một batch các câu có độ dài khác nhau và output là batch có thể chứa 1 hoặc nhiều bản dịch của các câu đầu vào.

Inference/Web flow

Model

Training Data

Optimizer (.train())

Evaluation (.eval())

Input sentences

Correct output sentences

User Input API (Sentences)

Output API (Sentences)

Tokenizer (Loader)

Encoder

Decoder

Detokenizer (Loader)

Training/Evaluation flow

# Graph mô tả quá trình hoạt động chung của các module bên trong model:

Source sentence (String)

Loader

Tokenizer [.tokenize()] → [src\_len] of str

Source Vocabulary [.lookup()] → [src\_len] of int

Embedder [.embed()] → [src\_len, emb\_size] of float

Embedded Input Vector [src\_len, emb\_size]

Encoder (n layer)

Encode 1 [.encode()] → [src\_len, num\_units] of float

Encode n [.encode()] → [src\_len, num\_units] of float

...

Config:

- Encoder layers: n

- Decoder layer: m

- Number of hidden units: num\_units

- Size of embedding: emb\_size

- Size of target vocabulary: vocab\_size

Encoded Attention Data [n, src\_len, num\_units]

Decoder (m layer)

Decode 1 [.decode()] → [tgt\_len, num\_units] of float

...

Decode m [.decode()] → [tgt\_len, num\_units] of float

Projection(Linear+Softmax) [.project()] → [tgt\_len, vocab\_size] of float

Per-word Probability [tgt\_len, vocab\_size]

Loader

Target Vocabulary [.reverse\_lookup()] → [tgt\_len] of string

Tokenizer [.detokenize()] → string

Target sentence (String)

# Các abstract class dự kiến của hệ thống:

**Loader**:

- **tokenize** (string → string[]): thực hiện tokenize để cắt câu về một mảng các từ.

- **lookup** (string → int): Dựa vào bảng vocab để tìm index của từ.

- **embed** (int → float[emb\_size]): lấy embedding của các từ dưới dạng index và add thêm các positional encoding tương ứng.

- **reverse\_lookup** (int → string): Tìm ngược các từ dạng string từ index được tạo ra.

- **detokenize** (string[] → string): Nối các từ lại thành một câu hoàn chỉnh.

**Encoder**:

- **encode** ([src\_len, emb\_size|num\_units] → [n, src\_len, num\_units]): Thực hiện quá trình một lớp encode xử lý thông tin như đã miêu tả ở trên (self-attention, feedforward, add&norm, dropout)

- **\_\_call\_\_**(): chạy toàn bộ module gồm tất cả các lớp. Override khi phát triển thêm (pretrained encoder, etc.)

**Decoder**:

- **decode** ([tgt\_len, num\_units] → [tgt\_len, num\_units]): Thực hiện quá trình decoder xử lý thông tin. Tùy thuộc vào mode đang chạy mà decode sẽ thực hiện khác nhau:

+ Nếu đang trong chế độ train, decode sẽ chạy cùng lúc cho [tgt\_len, num\_units] đồng thời cho câu đích chuẩn với mask các từ ở sau để mô phỏng quá tình inference.

+ Nếu trong chế độ infer, decoder sẽ chạy cho từng bước của timestep và lấy kết quả projection từ bước trước làm input cho bước sau.

- **project** ([num\_units] → [vocab\_size]): ở bước encode cuối cùng, chúng ta tạo một probability distribution trên toàn bộ vocab đích, và chọn câu tốt nhất dựa theo kết quả. Thường là một layer nn.Linear với activation softmax

- **\_\_call\_\_**(): chạy toàn bộ module gồm tất cả các lớp. Override khi phát triển thêm (beam search, pretrained decoder, etc.)

**Model**:

- **train**: thực hiện chạy training. Dữ liệu sẽ được ghi vào config tương ứng, lưu model theo epoch/steps/etc.

- **eval**: thực hiện chạy evaluation. Sử dụng dữ liệu đã có hoặc thông tin nhận được từ ngoài.

- **debug**: thực hiện chạy theo một yêu cầu nhất định của dev, cho phép sửa đổi/ktra chương trình mà không ảnh hưởng đến các hàm của Model đang có.

- **save**(directory): Lưu model đang có theo quy chuẩn của Torch.

- **load**(directory): Khởi tạo lại model đã có trên đường dẫn.

### Các dependency của hệ thống:

- Python 3. (3.6?)

- Pytorch (version TBD), Numpy(, Torchvision, Sklearn?)

- NLTK/pyvi (cho tokenizer cấp câu/từ), yaml/json/pickle/dill (để lưu trữ dữ liệu dưới dạng tương ứng)