ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

DATA MINING

Báo cáo Bài tập lớn - Học kì 242

BASIC TRANSLATE ENG-VN USING TRANSFORMER

Giáo viên hướng dẫn: Bùi Tiến Đức
Nhóm:
Sinh viên: Nguyễn Tất Đạt - 2210697
Nguyễn Minh Đạt - 2210693

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 4 - 2025

Họ và tên	MSSV	Công việc	Phần trăm hoàn thành
Nguyễn Tất Đạt	2210697	Tìm hiểu, xây dựng model	100%
Nguyễn Minh Đạt	2210693	Tìm hiểu, xây dựng model	100%

Contents

1 Giới thiệu bài toán **3**
1.1 Tổng quan bối cảnh hiện nay . . . 3
1.2 Mục tiêu . . . 3
1.3 Phương pháp sử dụng . . . 3
1.4 Phạm vi bài làm . . . 3

2 Tham khảo kết quả các Model lớn **4**
2.1 ChatGPT 4o . . . 4
2.2 Google Translate . . . 4

3 Dữ liệu và xử lí dữ liệu **6**
3.1 Dữ liệu sử dụng . . . 6
3.2 Tiền xử lí dữ liệu và xây dựng các thành phần cơ bản . . . 7

4 Transformer **9**
4.1 Sơ lược về transformer . . . 9
4.2 Kí hiệu . . . 10
4.3 Embedding . . . 10
4.4 Position Encoding . . . 11
4.5 Encoder . . . 12
4.5.1 Cơ chế Attention . . . 12
4.5.2 Multi Head Attention . . . 14
4.5.3 Vector ngữ cảnh (context vector) . . . 16
4.5.4 Add & Norm . . . 16
4.5.5 Feed Forward . . . 17
4.5.6 Tổng hợp khối Encoder . . . 17
4.6 Decoder . . . 18
4.6.1 Vector mục tiêu (target vector) . . . 18
4.6.2 Mask Multi Head Attention . . . 19
4.6.3 Multi Head Attention trong Decoder . . . 19
4.6.4 Tổng hợp khối Decoder . . . 19
4.7 Điểm mạnh và yếu . . . 21

5 Các thông số lựa chọn cho bài toán dịch **21**

6 Kết quả **22**

7 So sánh **26**

8 Hướng tiếp cận mở rộng trong tương lai **27**

1 Giới thiệu bài toán

1.1 Tổng quan bối cảnh hiện nay

Với sự phát triển mạnh mẽ của công nghệ thông tin và trí tuệ nhân tạo, nhu cầu giao tiếp và trao đổi thông tin xuyên ngôn ngữ ngày càng tăng cao. Các mô hình dịch máy đã trở nên phổ biến, đặc biệt là trong việc hỗ trợ người dùng hiểu được ngôn ngữ nước ngoài nhanh chóng và hiệu quả. Tuy nhiên, việc dịch thuật tiếng Anh sang tiếng Việt với các mô hình cũ vẫn gặp nhiều thách thức do sự khác biệt lớn về ngữ pháp, cú pháp và ngữ nghĩa giữa hai ngôn ngữ. Hạn chế từ các mô hình cũ không thể hiểu được tổng quát ý nghĩa của câu nói, đặc biệt là những câu hoặc những đoạn có độ dài lớn.

1.2 Mục tiêu

Mục tiêu của bài tập này là xây dựng và huấn luyện một mô hình dịch thuật mới với khả năng dịch mạnh mẽ từ tiếng Anh sang tiếng Việt hơn. Nhằm cải thiện độ chính xác và sự trôi chảy của các bản dịch, giúp người dùng Việt Nam dễ dàng tiếp cận và sử dụng các thông tin quốc tế.

1.3 Phương pháp sử dụng

Để thực hiện được điều đó, nhóm tận dụng những ưu điểm của kiến trúc Transformers với cơ chế Self-Attention để có thể thực nghiệm lên bài toán này.

1.4 Phạm vi bài làm

Nhóm tập trung vào các tiêu chí:

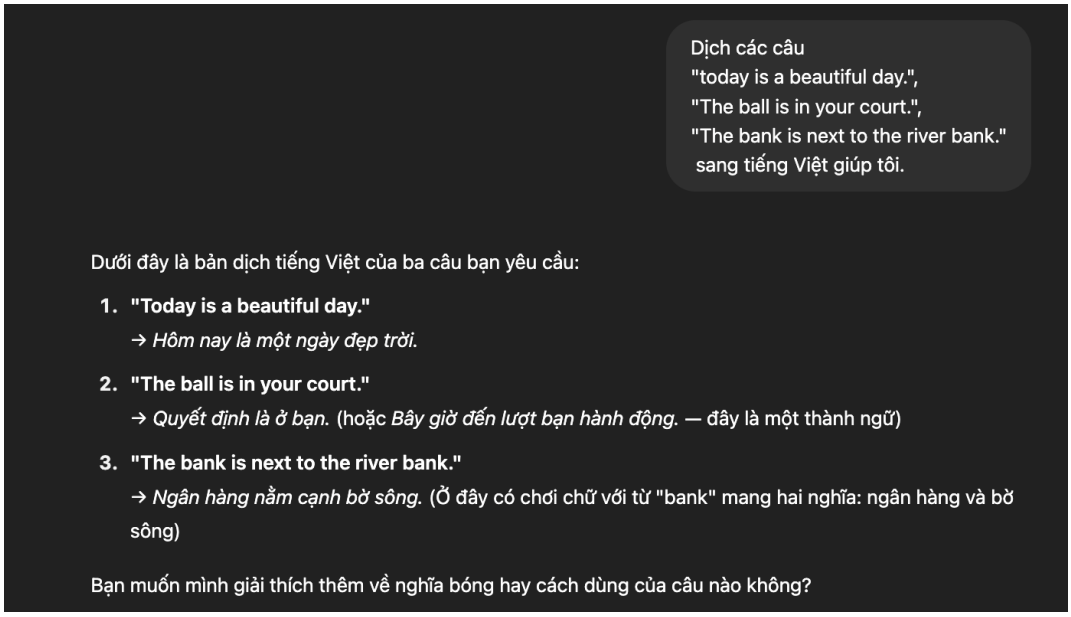
1. Áp dụng, giải thích được ưu điểm của kiến trúc Transformers.
2. Dựa vào các ưu điểm đó giải thích được vì sao kiến trúc này phù hợp với bài toán dịch máy.
3. Sử dụng bộ Dataset với các câu đơn riêng biệt (không sử dụng đoạn văn).
4. Hiện thực lại được kiến trúc Transformers (code from scratch).

2 Tham khảo kết quả các Model lớn

Nhóm tiến hành thực hiện thực hiện dịch thử với các model lớn với 1 câu bình thường, 1 câu thành ngữ và 1 câu có nghĩa chơi chữ:

1. "Today is a beautiful day." Đây là câu bình thường mang nghĩa "Hôm nay là một ngày đẹp trời."
2. "The ball is in your court." Đây là câu thành ngữ mang nghĩa "Tùy bạn."
3. "The bank is next to the river bank." Đây là một câu chơi chữ với từ bank mang 2 ý nghĩa khác nhau mang nghĩa "Ngân hàng nằm bên cạnh bờ sông."

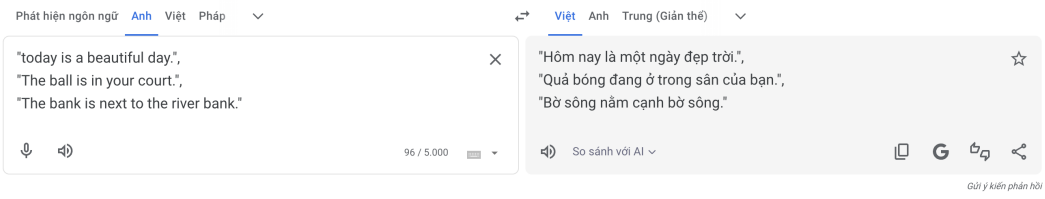
2.1 ChatGPT 4o



Hình 1: Tham khảo kết quả dịch của ChatGPT 4o

Nhận xét: ChatGPT 4o dịch khá tự nhiên, chú ý tới ngữ cảnh, thành ngữ ("The ball is in your court") và hiểu nghĩa đa tầng của từ ("bank"). Ngoài ra, ChatGPT 4o còn cung cấp thêm chú thích giải nghĩa, giúp người đọc dễ hiểu hơn.

2.2 Google Translate



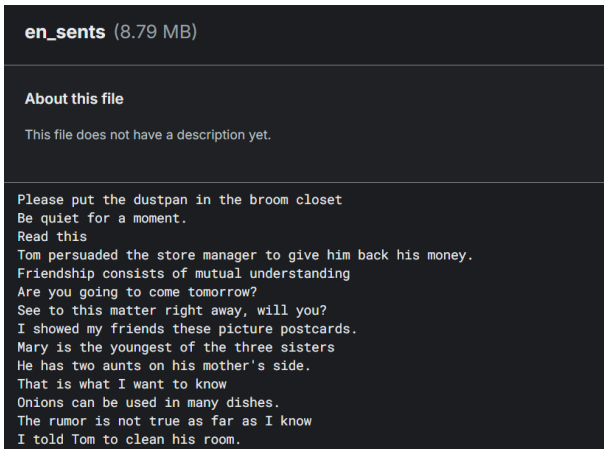
Hình 2: Tham khảo kết quả dịch của Google Translate

Nhận xét: Google Translate dịch khá sát nghĩa đen. Tuy nhiên, đối với các câu mang tính thành ngữ hay chơi chữ (ví dụ "The ball is in your court" hay "bank" trong ngữ cảnh), hệ thống này không nắm được nghĩa bóng, dẫn tới bản dịch gây khó hiểu cho người đọc tiếng Việt.

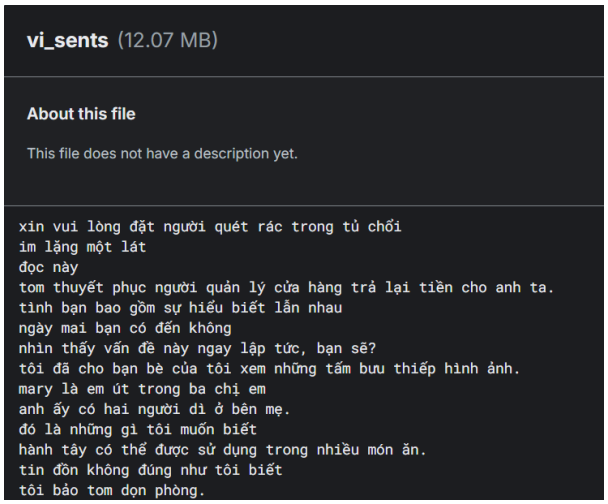
3 Dữ liệu và xử lí dữ liệu

3.1 Dữ liệu sử dụng

- Để có thể huấn luyện một mô hình dịch máy nhóm sử dụng bộ dữ liệu **English-Vietnamese Translation** được lấy từ Kaggle. Vì để làm đơn giản hơn bài toán nhóm chỉ sủ dụng bộ dữ liệu có các cặp câu đơn (không dùng cả một đoạn văn) của tiếng anh và tiếng việt.
- Bộ dữ liệu này bao gồm:
  1. Các câu nguồn (tiếng anh) được lưu từng câu ứng với mỗi dòng trong file en_sents.txt.
  2. Các câu đích (tiếng việt) được lưu từng câu ứng với mỗi dòng trong file vi_sents.txt.
  3. Ứng với mỗi dòng tương ứng sẽ là các cặp câu nguồn đích đi đôi với nhau.



Hình 3: Dữ liệu các câu tiếng anh



Hình 4: Dữ liệu các câu tiếng việt

- **Link bộ dữ liệu:** https://www.kaggle.com/datasets/hungnm/englishvietnamese-translation/data

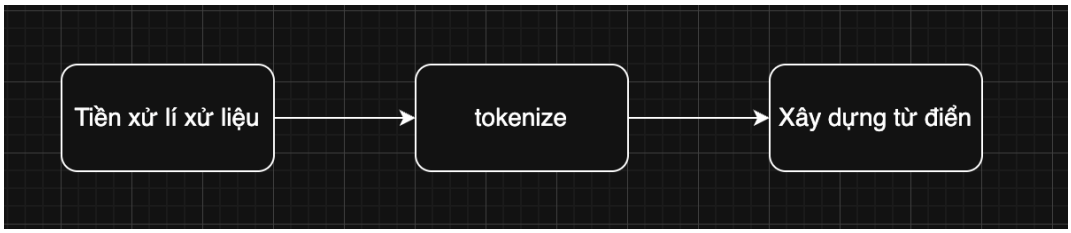
3.2 Tiền xử lí dữ liệu và xây dựng các thành phần cơ bản

Trong bài toán dịch máy sử dụng mô hình Transformer, ở phần tiền xử lí dữ liệu nhóm bỏ qua các bước:

- Xóa dấu câu (Remove punctuations).
- Xóa từ dừng (Remove stopwords).
- Stemming và Lemmatization.

Bởi vì:

- Các dấu câu, từ dừng và biến thể từ mang ý nghĩa ngữ pháp, ngữ nghĩa quan trọng, cần giữ nguyên để đảm bảo chất lượng dịch thuật.
- Mô hình Transformer với cơ chế Self-Attention có khả năng học ngữ cảnh mạnh mẽ mà không cần đơn giản hóa văn bản.
- Xóa hoặc rút gọn từ có thể làm mất thông tin cần thiết cho việc hiểu và dịch chính xác câu.



Hình 5: Pipeline hiện thực

- **Tiền xử lý:** Các câu nguồn và câu đích được chuyển về dạng chữ thường (lowercase). Ngoài ra với thư viện Underthesea, các câu tiếng Việt sai chính tả với đặt vị trí dấu câu sai cũng có thể được sửa lại cho đúng.
- **Tokenize:** Với câu tiếng Anh nhóm sử dụng thư viện Spacy, còn tiếng Việt sử dụng Underthesea (Một bộ công cụ xử lí ngôn ngữ tự nhiên mã nguồn mở dành cho tiếng Việt).

```
    def tokenize_vi(text: str) -> List[List[str]]:
        # Step 1: Sentence Tokenization
        sentences = sent_tokenize(text)

        # Step 2: Text Normalization (assuming it's just lowercasing here)
        sentences = [text_normalize(sentence) for sentence in sentences]

        # Step 3: Word Tokenization
        tokenized_sentences = [word_tokenize(sentence) for sentence in sentences]

        # Flatten the list
        tokenized_sentences = [word for sentence in tokenized_sentences for word in sentence]

        # Lowercase all tokens
        tokenized_sentences = [word.lower() for word in tokenized_sentences]

        return tokenized_sentences

```

```
    # Define the tokenizer for English
    en_tokenizer = get_tokenizer('spacy', language='en_core_web_md')

    # Define the tokenizer for Vietnamese
    vi_tokenizer = get_tokenizer(tokenize_vi)
```

- **Xây dựng từ điển (Vocabulary):**
  - Sử dụng hàm `build_vocab_from_iterator` để tạo từ điển cho tiếng Anh và tiếng Việt.
  - Thêm vào từ điển các token đặc biệt: `<unk>`, `<pad>`, `<sos>`, `<eos>`.
  - Thiết lập chỉ số mặc định cho từ không có trong từ điển là `<unk>`.

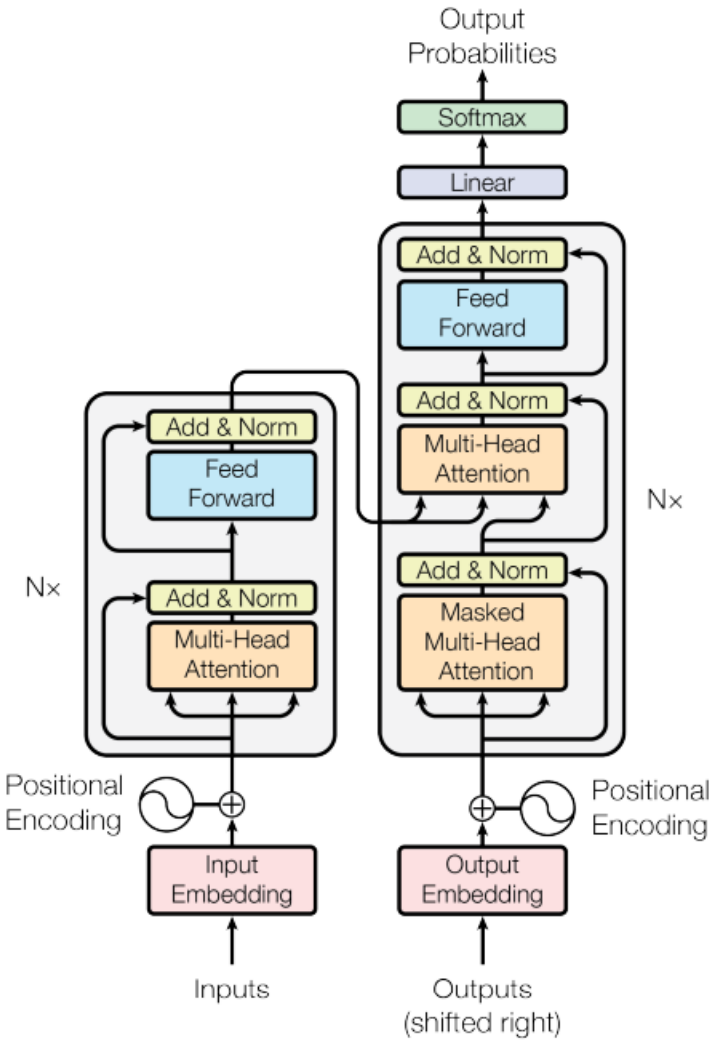
```
    # Build  src vocabularies
    self.src_vocab = build_vocab_from_iterator(
        self.df['english'].apply(lambda x: self.src_tokenizer(x.lower())),
        specials=['<unk>', '<pad>', '<sos>', '<eos>']
    )
    self.src_vocab.set_default_index(self.src_vocab['<unk>'])

    # Build  tgt vocabularies
    self.tgt_vocab = build_vocab_from_iterator(
        self.df['vietnamese'].apply(lambda x: self.tgt_tokenizer(x.lower())),
        specials=['<unk>', '<pad>', '<sos>', '<eos>']
    )
    self.tgt_vocab.set_default_index(self.tgt_vocab['<unk>'])
```

4 Transformer

4.1 Sơ lược về transformer

Transformers là một kiến trúc mạng nơ-ron tiên tiến được giới thiệu lần đầu tiên vào năm 2017 bởi Vaswani và cộng sự, đánh dấu bước ngoặt lớn trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP). Khác với các mô hình truyền thống dựa trên mạng hồi quy (RNN) hay mạng tích chập (CNN), Transformers nổi bật với cơ chế tự chú ý (self-attention), cho phép mô hình nắm bắt tốt hơn mối quan hệ giữa các từ ngữ trong một câu, không phụ thuộc vào khoảng cách hay vị trí cụ thể. Điều này giúp Transformer xử lý hiệu quả các nhiệm vụ NLP phức tạp như dịch máy, tóm tắt văn bản, phân loại câu, và hỏi đáp tự động. Nhờ khả năng huấn luyện song song hiệu quả và hiệu suất vượt trội, Transformers đã trở thành nền tảng cho hàng loạt mô hình nổi bật như GPT, BERT, và nhiều biến thể tiên tiến khác hiện đang được sử dụng rộng rãi trong các ứng dụng thực tiễn.



Hình 6: Kiến trúc Transformers

Trong đó khối bên trái còn được gọi là Encoder và khối bên phải còn được gọi là Decoder.

4.2 Kí hiệu

Ở bước tiền sử lý mọi câu trong bộ dữ liệu english và vietnamese đều đã được padding hoặc truncate về 1 độ dài, nên ta có các kí hiệu đồng bộ sau:

1. **S:** độ dài của câu đầu vào thuộc khối Encoder.
2. **S':** độ dài câu đầu vào thuộc khối Decoder.
3. **D:** Độ dài vector embedding thuộc khối Encoder.
4. **D':** Độ dài vector embedding thuộc khối Decoder.

4.3 Embedding

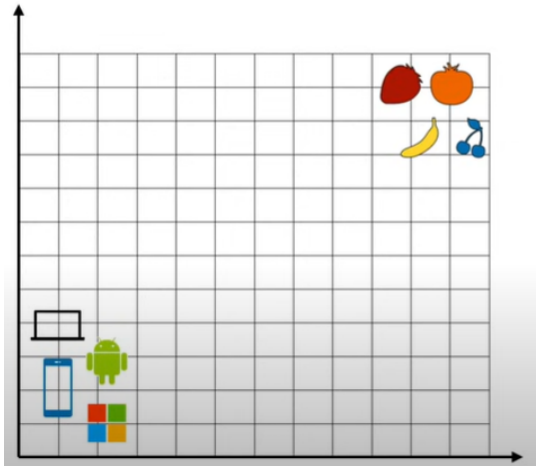
- Như mọi mô hình Deeplearning trong bài toán NLP trước đó, sau khi xây dựng từ điển và áp dụng one-hot encoding cho từng từ để giúp máy tính có thể làm việc với dữ liệu dạng chữ thì các từ đó đều phải được embedding (vector hóa) để có thể tham gia vào quá trình training.

  Ví dụ: "I am good" $\xrightarrow{\text{one-hot encoding}}$ $[9, 10, 18]$ $\xrightarrow{\text{Embedding}}$ $\begin{bmatrix} 1.2 & 2.4 & 3.5 \\ 3.2 & 0.9 & 7.3 \\ 9.1 & 6.2 & 5.5 \end{bmatrix}$

- Ứng với mỗi hàng sẽ là 1 vector có số chiều **D** đại diện cho 1 từ mà nó đang biểu diễn (trong ví dụ trên vector [1.2 2.4 3.5] đại diện cho số 9 trong từ điển xây dựng được tức là từ "I").
- Như vậy ứng với mỗi câu đầu vào có chiều là **S** sau khi embedding ta sẽ có một ma trận có kích thước **SxD**, với ý nghĩa mỗi hàng sẽ là 1 vector đại diện cho một từ trong câu.

  Vậy tại sao cần phải Embedding?

- Embedding bản chất là tạo ra một chiều không gian vector. Trong không gian vector này các từ vựng được đại diện bởi 1 vector có tọa độ nào đó. Và bản chất train mô hình chính là tạo ra được một không gian vector mà ở đó các từ có liên hệ mật thiết với nhau sẽ được biểu diễn bằng các vector gần tương đồng nhau.



Hình 7: Minh họa tạo ra một không gian vector cho các từ

- Nếu không embedding (tức từ vựng không được biểu diễn dưới dạng vector) mô hình sẽ gần như không có không gian hoặc mục tiêu để học từ đó không thể train được mô hình.

```
nn.Embedding(vocab_size, d_model)
```

4.4 Position Encoding

- Một trong những cải tiến quan trọng của Transformer so với các model cũ như RNN, LSTM,... chính là Transformer có thể sử lý song song dữ liệu thay vì phải đưa lần lượt từng token vào model. Dựa vào cải tiến đó, thời gian sử lý một câu với cùng một số lượng token thì transformer đang thể hiện với thời gian thực thi nhanh hơn đồng thời xử lý các câu dài tốt hơn cải thiện luôn cả vấn đề gradient vashing từ các model cũ.
- Nhưng ở các model cũ, với việc xử lý lần lượt từng token nên model có thể hiểu được thứ tự của token đó trong câu. Transformer với việc cải tiến trên thì model không thể hiểu được thứ tự vì các token đã được đưa vào cùng một lúc.

  Ví dụ: Câu "Tôi đang ăn thịt" vì transformer đã đưa một lượt các token vào model nên lúc này có thể model đang hiểu thành "Thịt đang ăn tôi".
- Ở ví dụ trên có thể thấy việc mô hình hiểu nhầm thứ tự từ trong câu gây ra một lỗi sai trầm trọng. Nên cần phải có một biện pháp để có thể giúp cho Transformer có thể hiểu được thứ tự từ.
- Giải pháp đơn giản rằng ta sẽ tạo ra một vector mang vị trí sau đó cộng lại với từng vector hàng. Ở đây có các cách sau để tạo ra được vector mang vị trí tương ứng:
  - Cách 1: Cộng với các vector lần lượt mang giá trị tương ứng với vị trí của từ.

    VD: $\begin{bmatrix} 1.2 & 2.4 & 3.5 \\ 3.2 & 0.9 & 7.3 \\ 9.1 & 6.2 & 5.5 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}$

    => Cách này cũng có thể giúp model hiểu được thứ tự của từ nhưng với các câu dài con số cộng lại sẽ rất lớn từ đó có thể gây ra tình trạng lệch trọng số ở các token (sau khi embedding các giá trị của vector có giá trị 0->1 nhưng sau khi positional encoding thì các token ở cuối lại có giá trị có thể lên đến 256). Đây là một cách chưa hoàn chỉnh.
  - Cách 2: Để có thể khắc phục tình trạng lệch như vậy, ta chỉ cần cộng số nhỏ hơn.

    VD: $\begin{bmatrix} 1.2 & 2.4 & 3.5 \\ 3.2 & 0.9 & 7.3 \\ 9.1 & 6.2 & 5.5 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 \end{bmatrix}$

    => Cách này có khả năng cải thiện được tình trạng ở cách 1 nhưng vì vẫn cộng các chiều của vector cùng 1 con số nên xảy ra tình trạng nghèo nàn thông tin giữa các chiều
  - Cách 3: Tận dụng tính liên tục và tuần hoàn của hàm sin, cos Transformer sử dụng 2 hàm này để mã hóa vị trí cho các từ.

$$PE_{(pos,\,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$
$$PE_{(pos,\,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

Trong đó:

1. pos: vị trí của token trong câu.
2. i: Chiều thứ i của vector.
3. d_{model} : Chiều của vector (D).

=> Vì cải thiện được cả cách 1 và cách 2 và giữa được điểm mạnh của 2 cách trên nên đây là cách tốt để có thể giúp cho model hiểu được vị trí của từ trong câu mà không mất quá nhiều chi phí tính toán.

```python
class PositionalEncoding(nn.Module):
    def __init__(self, d_model: int, max_len: int = 5000):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2).float()
            * -(torch.log(torch.tensor(10000.0)) / d_model)
        )
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer("pe", pe)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = x + self.pe[:, : x.size(1)]
        return x
```

4.5 Encoder

Sau khi câu đầu vào (src_input) đã được đưa vào chiều không gian cần thiết (embedding) và đánh dấu vị trí token (positional encoding), vấn đề tiếp theo là thiết kế ra được một cơ chế có thể giúp model hiểu được mối quan hệ, ngữ cảnh giữa các từ với nhau. Vậy tại sao phải giúp model hiểu được ngữ cảnh của các từ với nhau mà không dịch theo ý nghĩa từng từ một?

Ví dụ: "The bank is next to the river bank" - Chữ "bank" đầu tiên khác nghĩa với chữ "bank" ở vị trí cuối câu vì chữ "bank" ở vị trí cuối là một cụm danh từ, bị ảnh hưởng bởi chữ "river" trước nó.

4.5.1 Cơ chế Attention

- Dựa vào ví dụ trên, nhận thấy tầm quan trọng của việc giúp model hiểu được ngữ cảnh tổng quát, ngữ cảnh của các từ đứng liền kề nhau ta cần phát triển một chỉ số có thể đánh giá được sự tương

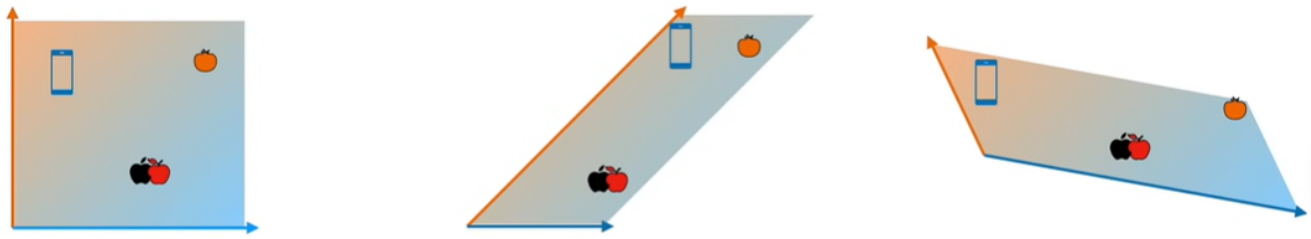
đồng của các từ vựng. Thêm vào đó các từ vựng đã được mô hình hóa thành vector ở bước trước đó vậy **bản chất đánh giá sự tương đồng của từ vựng chính là đánh giá sự tương đồng của các vector**. Bám sát vào lý luận đó ta lựa chọn một trong những chỉ số đánh giá tốt nhất giữa 2 vector chính là **tích vô hướng**. Và cơ bản nhất là tích vô hướng trên một không gian chính tắt. Tích vô hướng có thể so sánh 2 vector dựa trên góc lệch và độ dài của 2 vector đó. Nếu 2 vector quá khác nhau góc sẽ lớn làm cho giá trị vô hướng nhỏ, nếu 2 vector đó gần tương đồng nhau nhưng vị trí xa nhau hoặc sắc thái nghĩa khác nhau thì độ dài có thể khác nhau.

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{D} a_i b_i = [a_1, a_2, \ldots, a_D] \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_D \end{bmatrix} = \|\vec{a}\| \, \|\vec{b}\| \cos\theta$$

- Và vì cơ chế "Attention" - tự chú ý, nên các từ trong câu đều cần phải được đánh giá với nhau và các vector từ này đã được mô hình thành ma trận có kích thước **SxD** đã giới thiệu từ trước nên nó trở thành tích giữa 2 ma trận với nhau.

$$AA^\top = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1D} \\ a_{21} & a_{22} & \cdots & a_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ a_{S1} & a_{S2} & \cdots & a_{SD} \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{S1} \\ a_{12} & a_{22} & \cdots & a_{S2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1D} & a_{2D} & \cdots & a_{SD} \end{bmatrix}$$

- Với ý tưởng là vậy nhưng khi huấn luyện mô hình gặp nhiều khó khăn để mô hình có thể phân biệt được chính xác sự tương đồng với nhau vì giới hạn của không gian chính tắt. Lúc này ta cần chuyển cơ sở của các vector về một không gian khác mà có thể giúp phân biệt tốt hơn giữa các từ giống nhau nhưng khác biệt hoàn toàn nhau về nghĩa.



Hình 8: Tầm quan trọng của không gian cơ sở

Ví dụ: Có thể thấy được ở hình trên khi cơ sở là hình 3 sẽ có thể phân biệt được từ "apple" là điện thoại và quả táo tốt hơn rất nhiều so với hình 1 và hình 2.

- Dựa vào các lý luận đó ta tạo ra bộ ba các ma trận chuyển cơ sở **Q, K, V** lần lượt là Query, Key, Value với mục đích **Q** và **K** sẽ đảm nhận nhiệm vụ tạo ra một không gian cơ sở phù hợp khi ta truy vấn một từ nào đó nó sẽ phân biệt được tốt nhất từ đó đang mang ngữ cảnh gì và Value sẽ là một

không gian cơ sở để gọi ngược lại giá trị của từ đó bằng tiếng anh. Nên các vector trong ma trận ban đầu lần lượt sẽ được đi qua các cơ sở đó để vào một chiều không gian mới, khi tạo được một chiều không gian đủ tốt tức có thể tạo ra một ngữ cảnh tổng quát tốt mà ở đó các từ có thể dịch dựa vào ngữ cảnh của cả câu mà không bị nhầm lẫn nghĩa nên các vector lúc này còn được gọi là **vector ngữ cảnh (context vector)**.

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

$$\text{với } X \in \mathbb{R}^{S\times D}, \quad W^Q, W^K, W^V \in \mathbb{R}^{D\times D}, \quad Q, K, V \in \mathbb{R}^{S\times D}$$

- Lúc này vì đã sinh ra các không gian phù hợp và tiếp nối ý tưởng để so sánh sự tương đồng của các từ trong câu trước đó, vector **K** đóng vai trò như một khóa đại diện cho từ, và **Q** sẽ truy vấn đến các vector **K** của các từ trong câu. Theo đó, 2 từ liên quan đến nhau sẽ có "Score" lớn và ngược lại. Tiếp đó không mất tính tổng quát nhưng phù hợp với bộ nhớ máy tính ta sẽ chia cho $\sqrt{D}$ để scale lại giá trị cho các vector.

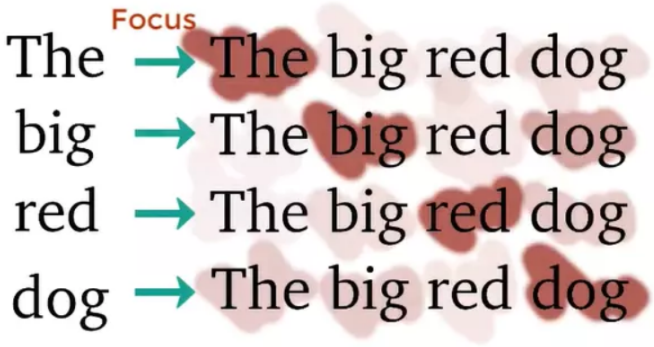
$$Score = \frac{QK^T}{\sqrt{D}}$$

- Cuối cùng để có thể tạo ra phân bố xác suất sự liên quan của các token với nhau kết quả score được đi qua một lớp **Softmax** và dựa vào phân bố xác suất đó và **V** có thể dự đoán được từ kế. Cả cơ chế ánh xạ vào một không gian mới để tạo ra một phân bố xác xuất liên quan, tương đồng của các từ này còn được gọi là cơ chế "Attention".

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{D}}\right)V$$

4.5.2 Multi Head Attention

- Nhưng vấn đề của cơ chế Attention đơn giản là nó sẽ chú ý vào nó nhiều nhất (có chỉ số score lớn nhất) vì đơn giản tích vô hướng 2 vector y chang nhau sẽ cho ra kết quả gần như cao nhất. Nhưng để model hiểu được ngữ cảnh tốt nhất thì token đó phải tập trung so với các token khác nhiều hơn.



Hình 9: Self-Attention tự chú ý vào chính nó nhiều nhất

- Giải pháp ở đây chính là ta sẽ chia vector thành nhiều head với mỗi head đại diện cho chiều tương ứng của vector đó và cho Attention để tìm điểm tương đồng của các head mang chiều tương ứng

với nhau với hi vọng có thể làm model chú ý thêm nhiều thông tin khác ở trong câu. Gọi mỗi head có số chiều là **H (H<D)**, vậy ta sẽ có D/H + D%H số lượng head (head cuối sẽ có chiều D%H).

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1D} \\ a_{21} & a_{22} & \cdots & a_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ a_{S1} & a_{S2} & \cdots & a_{SD} \end{bmatrix} \xrightarrow{\text{multi-head với } H=2} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{S1} & a_{S2} \end{bmatrix} \begin{bmatrix} a_{13} & a_{14} \\ a_{23} & a_{24} \\ \vdots & \vdots \\ a_{S3} & a_{S4} \end{bmatrix} \ldots \begin{bmatrix} a_{1D} \\ a_{2D} \\ \vdots \\ a_{SD} \end{bmatrix}$$

- Các head này tiếp tục được tính sự tương đồng dựa trên các chiều mà nó mang.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{S1} & a_{S2} \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{S1} \\ a_{12} & a_{22} & \cdots & a_{S2} \end{bmatrix}$$

- Ta có mỗi head có chiều **SxH** và có D/H + D%H số lượng head, vậy khi nhân các head tương ứng với nhau ta sẽ có D/H +D%H head có chiều **SxS**. Cuối cùng khi gộp (concate) các kết quả lại ta sẽ có một ma trận có chiều **SxD** tương tự khi xài cơ chế self-attention thông thường. Cơ chế này được mô hình hóa tổng quát bằng công thức:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

```python
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model: int, n_heads: int):
        super(MultiHeadAttention, self).__init__()
        assert d_model % n_heads == 0, "d_model must be divisible by n_heads"

        self.d_model = d_model
        self.num_heads = n_heads
        self.head_dim = d_model // n_heads

        self.W_q = nn.Linear(d_model, d_model)
        self.W_k = nn.Linear(d_model, d_model)
        self.W_v = nn.Linear(d_model, d_model)
        self.W_o = nn.Linear(d_model, d_model)

    def scaled_dot_product_attention(self, Q, K, V, mask=None):
        attn_scores = torch.matmul(Q, K.transpose(-2, -1)) / math.sqrt(self.head_dim)

        if mask is not None:
            attn_scores = attn_scores.masked_fill(mask == 0, -1e9)

        attn_probs = torch.softmax(attn_scores, dim=-1)
```

```
        output = torch.matmul(attn_probs, V)
        return output, attn_probs

    def split_heads(self, x: torch.Tensor) -> torch.Tensor:
        batch_size, seq_length, d_model = x.size()
        return x.view(batch_size, seq_length, self.num_heads, self.head_dim).transpose(
            1, 2
        )

    def combine_heads(self, x: torch.Tensor) -> torch.Tensor:
        batch_size, _, seq_length, head_dim = x.size()
        return x.transpose(1, 2).contiguous().view(batch_size, seq_length, self.d_model)

    def forward(
        self,
        query: torch.Tensor,
        key: torch.Tensor,
        value: torch.Tensor,
        mask: torch.Tensor = None,
    ) -> torch.Tensor:
        Q = self.W_q(query)
        K = self.W_k(key)
        V = self.W_v(value)

        Q = self.split_heads(Q)
        K = self.split_heads(K)
        V = self.split_heads(V)

        attn_output, _ = self.scaled_dot_product_attention(Q, K, V, mask)
        output = self.combine_heads(attn_output)
        output = self.W_o(output)
        return output
```

4.5.3 Vector ngữ cảnh (context vector)

Sau khi các ma trận mang các vector đại diện token đi qua cơ chế Multi-head Attention, sẽ sinh ra được một ma trận mang ngữ cảnh của cả câu, ngữ cảnh tương đồng của các từ đứng kế nhau. Từ đó ma trận mang các vector đó còn được gọi là **vector ngữ cảnh (context vector)**.

4.5.4 Add & Norm

Tiếp đó ma trận sẽ được chuẩn hóa lại và sử dụng kĩ thuật residual block với ma trận input ban đầu để có thể biểu diễn thông tin tốt nhất có thể.

$$X' = \text{LayerNorm}(X + \text{MultiHead}(Q, K, V))$$

```
class AddNorm(nn.Module):
    def __init__(self, d_model: int):
        super(AddNorm, self).__init__()
        self.norm = nn.LayerNorm(d_model)
```

```
    def forward(self, x: torch.Tensor, sublayer: torch.Tensor) -> torch.Tensor:
        return self.norm(x + sublayer)
```

4.5.5 Feed Forward

Cuối cùng các vector này được đẩy qua lớp Fully Connected Layer với hàm kích hoạt ReLU.

$$\text{FFN}(x) = \max(0,\, xW_1 + b_1)W_2 + b_2$$

```
class FeedForwardNetwork(nn.Module):
    def __init__(self, d_model: int, dropout: float = 0.1, forward_expansion: int = 4):
        super(FeedForwardNetwork, self).__init__()
        hidden_dim = d_model * forward_expansion
        self.fc_1 = nn.Linear(d_model, hidden_dim)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout)
        self.fc_2 = nn.Linear(hidden_dim, d_model)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.fc_1(x)
        x = self.relu(x)

        x = self.dropout(x)
        x = self.fc_2(x)
        return x
```

4.5.6 Tổng hợp khối Encoder

```
class EncoderLayer(nn.Module):
    def __init__(
        self,
        d_model: int,
        heads: int,
        dropout: float = 0.1,
        forward_expansion: int = 4,
    ):
        super(EncoderLayer, self).__init__()
        self.attention = MultiHeadAttention(d_model, heads)

        self.norm1 = AddNorm(d_model)
        self.norm2 = AddNorm(d_model)

        self.ffn = FeedForwardNetwork(d_model, dropout, forward_expansion)

        self.dropout = nn.Dropout(dropout)

    def forward(self, x: torch.Tensor, mask: torch.Tensor) -> torch.Tensor:
```

```
        attention_output = self.attention(query=x, key=x, value=x, mask=mask)

        x = self.norm1(x, self.dropout(attention_output))
        ffn_output = self.ffn(x)

        output = self.norm2(x, self.dropout(ffn_output))
        return output
class Encoder(nn.Module):
    def __init__(
        self,
        vocab_size: int,
        d_model: int,
        num_layers: int,
        heads: int,
        dropout: float,
        forward_expansion: int,
        max_len: int,
    ):
        super(Encoder, self).__init__()
        self.d_model = d_model
        self.embedding = nn.Embedding(vocab_size, d_model)
        self.positional_encoding = PositionalEncoding(d_model, max_len)

        self.layers = nn.ModuleList(
            [
                EncoderLayer(d_model, heads, dropout, forward_expansion)
                for _ in range(num_layers)
            ]
        )

        self.dropout = nn.Dropout(dropout)

    def forward(self, x: torch.Tensor, mask: torch.Tensor) -> torch.Tensor:
        embedding_output = self.embedding(x)
        positional_output = self.positional_encoding(embedding_output)
        out = self.dropout(positional_output)

        for layer in self.layers:
            out = layer(out, mask)

        return out
```

4.6 Decoder

Các layer trong decoder hầu hết đều hoạt động giống encoder, chỉ có một số thay đổi như sau:

4.6.1 Vector mục tiêu (target vector)

Ở khối Decoder input sẽ là vector kết quả mà mình muốn hướng đến (ở bài toán dịch tiếng anh-tiếng việt thì vector mục tiêu sẽ là vector đại diện cho câu tiếng việt tương ứng với inut của khối encoder). Câu tiếng việt tương ứng này cũng được đi qua lớp embedding và positional encoding.

4.6.2 Mask Multi Head Attention

Công việc của Decoder là giải mã thông tin từ Encoder và sinh ra từng từ tiếng việt dựa trên những từ trước đó. Vậy nên, nếu ta sử dụng Multi-head attention trên cả câu như ở Encoder, Decoder sẽ "thấy" luôn từ tiếp theo mà nó cần dịch. Để ngăn điều đó, khi Decoder dịch đến từ thứ i, phần sau của câu tiếng Anh sẽ bị che lại (masked) và Decoder chỉ được phép "nhìn" thấy phần nó đã dịch trước đó.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + \text{mask}\right)V$$

$$\text{mask} = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Khi cộng với các số vô cùng bé và đi qua softmax những xác suất tại đó sẽ có giá trị là 0 (token tại vị trí i,j tương ứng không có quan hệ gì với nhau-không dự đoán được). Tương tự attention encoder ma trận cuối thu được vẫn có kích thước **S'xD'**

4.6.3 Multi Head Attention trong Decoder

Với bộ ba **Q, K, V** ma trận Q sẽ được lấy tiếp tục từ output của khối mask multi head Attention nhưng K và V sẽ được lấy từ output của khối Encoder vì lúc này các đầu vào sẽ đóng vai trò như sau:

1. Q: Decoder cố gắng sinh ra một từ mới nên mỗi vị trí Q là câu hỏi "Tôi nên tập trung vào phần nào của Encoder?" -> nên Q phải đến từ decoder.
2. K: đóng vai trò như một key nên nó phải cụ thể tới "Tôi có thể trả lời loại câu hỏi nào"
3. V: đóng vai trò như đáp án nên nó sẽ trả lời "Nếu bạn chọn tôi đây là thông tin bạn nhận được" -> K,V chỉ là những người trả lời nên cần phải có tri thức từ trước nên phải đến từ encoder.

Như vậy ta sẽ có [S'xD'] x [DxS] x [SxD], kết luận được chiều embedding của tiếng anh và tiếng việt phải cùng chiều (D=D') và kết quả cuối ta sẽ thu được ma trận **[S'xD]**. Ma trận này là ma trận ngữ cảnh tương ứng của tiếng anh được ánh xạ qua tiếng việt. Cuối cùng từng vector trong ma trận sẽ qua một lớp Linear cuối cùng để có size về với vocab size của tiếng việt và tiếng hành dự đoán từ có xác xuất cao nhất trong vocab size.

4.6.4 Tổng hợp khối Decoder

```
class DecoderLayer(nn.Module):
    def __init__(
        self, d_model: int, heads: int, dropout: float = 0.1, forward_expansion: int = 4
    ):
        super(DecoderLayer, self).__init__()

        self.self_attention = MultiHeadAttention(d_model, heads)
        self.norm1 = AddNorm(d_model)

        self.encoder_decoder_attention = MultiHeadAttention(d_model, heads)
```

```
        self.norm2 = AddNorm(d_model)

        self.ffn = FeedForwardNetwork(d_model, dropout, forward_expansion)
        self.norm3 = AddNorm(d_model)

        self.dropout = nn.Dropout(dropout)  # Dropout layer applied after sub-layers

    def forward(
        self,
        x: torch.Tensor,
        enc_out: torch.Tensor,
        src_mask: torch.Tensor,
        trg_mask: torch.Tensor,
    ):
        self_attn_output = self.self_attention(query=x, key=x, value=x, mask=trg_mask)
        x_norm1 = self.norm1(x, self.dropout(self_attn_output))

        enc_dec_attn_output = self.encoder_decoder_attention(
            query=x_norm1, key=enc_out, value=enc_out, mask=src_mask
        )
        x_norm2 = self.norm2(
            x_norm1, self.dropout(enc_dec_attn_output)
        )

        ffn_output = self.ffn(x_norm2)
        output = self.norm3(
            x_norm2, self.dropout(ffn_output)
        )
        return output
class Decoder(nn.Module):
    def __init__(
        self,
        vocab_size: int,
        d_model: int,
        num_layers: int,
        heads: int,
        dropout: float,
        forward_expansion: int,
        max_len: int,
    ):
        super(Decoder, self).__init__()
        self.d_model = d_model
        self.embedding = nn.Embedding(vocab_size, d_model)
        self.positional_encoding = PositionalEncoding(d_model, max_len)

        self.layers = nn.ModuleList(
            [
                DecoderLayer(d_model, heads, dropout, forward_expansion)
                for _ in range(num_layers)
            ]
        )

        self.dropout = nn.Dropout(dropout)
```

```
    def forward(
        self,
        trg: torch.Tensor,
        enc_out: torch.Tensor,
        src_mask: torch.Tensor,
        trg_mask: torch.Tensor,
    ) -> torch.Tensor:
        batch_size, trg_len = trg.shape

        embedding_output = self.embedding(trg)
        out = self.positional_encoding(embedding_output)
        out = self.dropout(out)

        for layer in self.layers:
            out = layer(x=out, enc_out=enc_out, src_mask=src_mask, trg_mask=trg_mask)

        return out
```

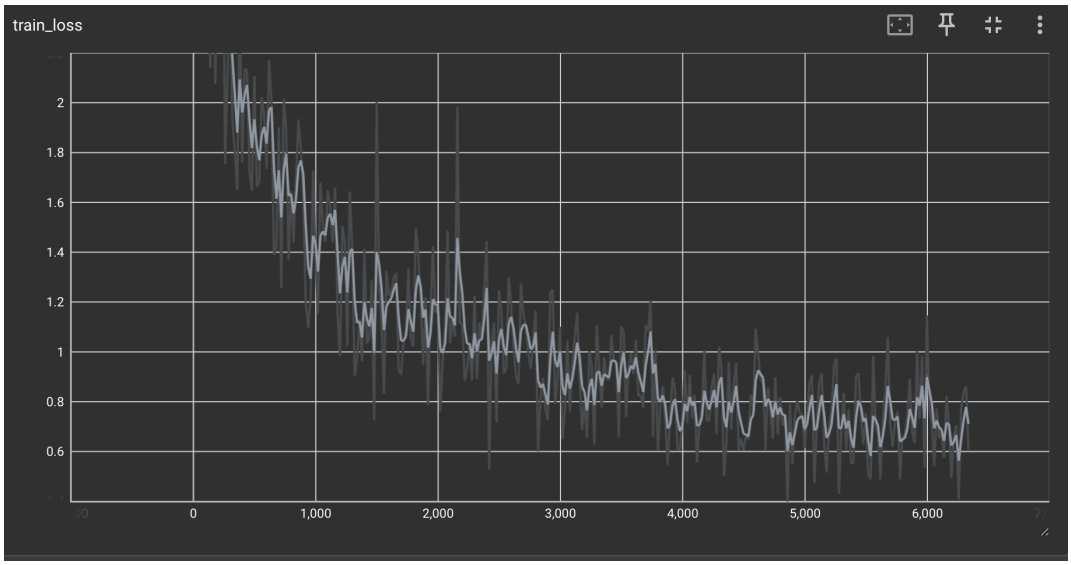
4.7 Điểm mạnh và yếu

Khía cạnh	Ưu điểm	Nhược điểm
Ngữ cảnh	Xử lý phụ thuộc dài hiệu quả	Thiếu kiến thức ngữ pháp ngầm định
Cấu trúc	Học ánh xạ cú pháp giữa hai ngôn ngữ	Dịch máy móc nếu thiếu dữ liệu tốt
Hiệu năng	Huấn luyện và suy diễn nhanh, song song	Cần GPU/bộ nhớ lớn
Đầu ra	Dịch mượt mà, tự nhiên	Có thể sai nghĩa nếu ngữ cảnh mơ hồ

5 Các thông số lựa chọn cho bài toán dịch

1. **Batch size**: 32
2. **Learning rate**: 0.0005
3. **Optimizer**: Adam
4. **Scheduler**: ReduceLROnPlateau
5. **Số epoch huấn luyện**: 20
6. **Chiều dài tối đa của chuỗi**: không cố định (sử dụng padding động với `pad_sequence`)
7. **Embedding dimension**: 512
8. **Số tầng encoder/decoder**: 6
9. **Số head trong multi-head attention**: 8

6 Kết quả



Hình 10: Biểu đồ train loss

Quan sát biểu đồ, ta nhận thấy:

- Giá trị loss giảm đều từ khoảng 2.2 xuống còn khoảng 0.6 sau quá trình huấn luyện, cho thấy mô hình đã học được thông tin hữu ích từ dữ liệu.
- Trong giai đoạn đầu, loss giảm nhanh chóng, sau đó tiến độ giảm chậm lại và xuất hiện dao động nhỏ, nguyên nhân có thể do kích thước batch nhỏ hoặc dữ liệu huấn luyện có tính đa dạng cao.
- Tổng thể, mô hình cho thấy sự hội tụ tốt trong tập train.

Nhóm chọn ra 10 câu ngẫu nhiên trong bộ validation để dịch và đánh giá BLEU score

```
--- Translations ---
Source:    what you need to do next is fill out this application form .
Target:    những gì bạn cần làm tiếp theo là điền vào mẫu đơn này .
Predicted: những gì bạn cần phải làm tiếp theo là điền vào mẫu đơn này .
------------------------------------------------------------
Source:    it will cost 500 dollars to fly to paris .
Target:    nó sẽ có giá 500 đô la để bay đến paris .
Predicted: nó sẽ tốn 500 đô la để bay đến paris .
------------------------------------------------------------
Source:    we 're not the only ones here from boston
Target:    chúng tôi không phải là những người duy nhất ở đây từ boston
Predicted: chúng tôi không phải là người duy nhất ở đây từ boston
------------------------------------------------------------
Source:    she lives far from there .
Target:    cô ấy sống xa đó
Predicted: cô ấy sống xa đây .
------------------------------------------------------------
Source:    you 're the most beautiful girl i 've ever seen
Target:    bạn là cô gái đẹp nhất tôi từng thấy
Predicted: bạn là cô gái xinh đẹp nhất tôi từng thấy
------------------------------------------------------------
Source:    nobody invited me to the party
Target:    không ai mời tôi đến bữa tiệc
Predicted: không ai mời tôi đến bữa tiệc
...
Source:    let 's stay home and watch tv .
Target:    hãy ở nhà và xem tv .
Predicted: hãy ở nhà và xem tivi
```

Hình 11: Dịch 10 câu random trong tập validation

```python
bleu = BLEUScore()

references = []
hypotheses = []

for src, tgt, pred in translations:
    references.append([tgt])
    hypotheses.append(pred)

bleu_score = bleu(hypotheses, references)

print(f"\nBLEU score for 10 translations (torchmetrics): {bleu_score:.4f}")
```

```
BLEU score for 10 translations (torchmetrics): 0.6552
```

Hình 12: Bleu score cho 10 câu ngẫu nhiên trên

Nhận xét:

Các kết quả dịch cho thấy mô hình đã đạt được khả năng dịch tự nhiên và chính xác ở mức khá. Khi so sánh giữa câu nguồn (Source), câu mục tiêu mong muốn (Target) và câu mô hình dự đoán (Predicted), có thể nhận thấy:

- **Độ chính xác ngữ nghĩa:** Các câu dịch Predicted giữ được phần lớn ý nghĩa chính xác của câu gốc. Trong nhiều trường hợp, câu dịch dù khác từ ngữ nhỏ nhưng vẫn đúng về nội dung (ví dụ: "tốn" so với "có giá", "xinh đẹp" so với "đẹp").
- **Độ tự nhiên:** Một số câu dịch sử dụng cách diễn đạt tự nhiên hơn trong tiếng Việt hiện đại, ví dụ thay vì "TV" thì dùng "tivi" hoặc thay "cô gái đẹp nhất" bằng "cô gái xinh đẹp nhất", thể hiện mô hình học được sự linh hoạt trong ngôn ngữ.
- **Một số lỗi nhỏ:** Vẫn có một vài trường hợp mô hình dịch chưa hoàn toàn chính xác, ví dụ như sai lệch nhẹ về ngữ cảnh không gian (dịch "xa đó" thành "xa đây"). Tuy nhiên, những lỗi này nhìn chung không làm thay đổi nghiêm trọng ý nghĩa tổng thể của câu.

Mô hình đạt được **BLEU score = 0.6552** trên 10 câu thử nghiệm. Đây là một kết quả tốt đối với một hệ thống dịch máy cơ bản, cho thấy mô hình đã học được sự tương ứng giữa hai ngôn ngữ ở mức độ ổn định.

Nhóm thử dịch một vài câu mẫu

```python
sentence = "Hello, how are you?"
src, pred = translate_one_sentence(model, data_module, sentence)
print(f"Source:    {src}")
print(f"Predicted: {pred}")
sentence = "The cat is sleeping on mat"
src, pred = translate_one_sentence(model, data_module, sentence)
print(f"Source:    {src}")
print(f"Predicted: {pred}")
sentence = "you are so beautiful"
src, pred = translate_one_sentence(model, data_module, sentence)
print(f"Source:    {src}")
print(f"Predicted: {pred}")
sentence = "I am a student at Hanoi University of Science and Technology"
src, pred = translate_one_sentence(model, data_module, sentence)
print(f"Source:    {src}")
print(f"Predicted: {pred}")
sentence = "I love you"
src, pred = translate_one_sentence(model, data_module, sentence)
print(f"Source:    {src}")
print(f"Predicted: {pred}")
```

```
Source:    Hello, how are you?
Predicted: xin chào , bạn thế nào ?
Source:    The cat is sleeping on mat
Predicted: con mèo đang ngủ trên tấm thảm
Source:    you are so beautiful
Predicted: bạn thật đẹp
Source:    I am a student at Hanoi University of Science and Technology
Predicted: tôi là học sinh tại trường đại học và công nghệ
Source:    I love you
Predicted: tôi yêu bạn
```

Hình 13: Kết quả

Kết quả dịch cho thấy mô hình có khả năng chuyển ngữ cơ bản khá tốt. Các câu dịch đều giữ được ý nghĩa tổng thể so với câu gốc, mặc dù vẫn còn một số hạn chế nhỏ:

- Các câu đơn giản như "Hello, how are you?" hay "I love you" được dịch chính xác và tự nhiên.
- Những câu dài hơn, phức tạp hơn như "I am a student at Hanoi University of Science and Technology" vẫn dịch được ý chính, tuy nhiên thông tin chi tiết (ví dụ: tên trường) chưa được dịch trọn vẹn hoặc chưa chính xác hoàn toàn.
- Các câu miêu tả như "The cat is sleeping on mat" được dịch khá trôi chảy, đúng ngữ pháp tiếng Việt.

Tổng thể, mô hình đạt được mức độ dịch ổn định với những câu ngắn và phổ biến. Để cải thiện chất lượng dịch đối với các câu dài hoặc có từ ngữ chuyên ngành, mô hình cần được huấn luyện thêm với tập dữ liệu lớn hơn và đa dạng hơn.

7 So sánh

Dựa trên các kết quả dịch thu được, nhóm tiến hành so sánh giữa mô hình Transformer huấn luyện nội bộ với các hệ thống dịch tự động lớn như ChatGPT 4o và Google Translate.

- **Về độ chính xác ngữ nghĩa:**
  - Mô hình tự huấn luyện đã thể hiện khả năng dịch khá tốt, đặc biệt ở các câu đơn giản và phổ biến. Ý nghĩa tổng thể của câu được giữ lại, mặc dù một số câu có sự khác biệt nhỏ trong cách diễn đạt.
  - ChatGPT 4o dịch rất chính xác, hiểu đúng cả thành ngữ và các câu chơi chữ phức tạp. Ngoài ra, còn bổ sung giải thích ngữ nghĩa, điều này hỗ trợ người dùng hiểu sâu hơn.
  - Google Translate có xu hướng dịch sát nghĩa đen. Ở các câu đơn giản, kết quả khá tốt, nhưng khi gặp thành ngữ hoặc chơi chữ, bản dịch trở nên máy móc và thiếu tự nhiên.
- **Về độ tự nhiên trong ngôn ngữ:**
  - Các câu dịch của mô hình nội bộ nhìn chung dễ hiểu và tự nhiên đối với các câu ngắn, nhưng với các câu phức tạp hoặc ngôn ngữ đời thường thì còn hạn chế.
  - ChatGPT 4o thể hiện độ tự nhiên vượt trội, lựa chọn từ ngữ linh hoạt và phù hợp với bối cảnh giao tiếp tiếng Việt thực tế.
  - Google Translate đôi khi dịch thiếu sự tự nhiên, từ ngữ còn mang nặng ảnh hưởng cấu trúc tiếng Anh.
- **Về khả năng xử lý ngữ cảnh phức tạp:**
  - Mô hình tự huấn luyện gặp khó khăn trong các câu chơi chữ hoặc thành ngữ, chủ yếu dịch sát nghĩa từng từ.
  - ChatGPT 4o xử lý tốt thành ngữ và câu đa nghĩa, cho thấy khả năng hiểu ngữ cảnh sâu.
  - Google Translate hầu như không nhận ra sự khác biệt ngữ nghĩa trong trường hợp từ có nhiều nghĩa (ví dụ từ "bank"), dẫn tới dịch thiếu chính xác.

Tóm lại, mô hình Transformer nội bộ có thể đáp ứng các bài toán dịch cơ bản với độ chính xác khá. Tuy nhiên, để đạt chất lượng dịch tương đương với các hệ thống thương mại lớn như ChatGPT 4o, cần cải thiện thêm khả năng hiểu ngữ cảnh, thành ngữ và ngôn ngữ tự nhiên qua việc huấn luyện trên tập dữ liệu lớn hơn và đa dạng hơn.

8 Hướng tiếp cận mở rộng trong tương lai

AI Agents đang trở thành một hướng nghiên cứu và ứng dụng quan trọng trong lĩnh vực trí tuệ nhân tạo, đặc biệt là trong các bài toán liên quan đến ngôn ngữ tự nhiên như dịch thuật. Trong tương lai, nhóm định hướng mở rộng hệ thống dịch hiện tại theo các hướng sau:

- **Tích hợp AI Agents:** Thay vì chỉ dựa trên mô hình dịch đơn lẻ, hệ thống có thể được tích hợp thêm các AI Agents có khả năng đọc hiểu văn bản, nắm bắt ngữ cảnh, và đưa ra dịch thuật theo ngữ nghĩa mở rộng. Điều này sẽ giúp xử lý tốt hơn các câu có tính đa nghĩa, thành ngữ, hoặc câu đòi hỏi suy luận.
- **Fine-tune mô hình với dữ liệu chuyên biệt:** Bằng cách huấn luyện bổ sung trên các tập dữ liệu ngữ cảnh phức tạp, thành ngữ, câu chơi chữ hoặc các ngữ liệu tiếng Việt đời thường, mô hình có thể cải thiện đáng kể khả năng dịch tự nhiên và chính xác.
- **Kết hợp với hệ thống đánh giá tự động:** Phát triển thêm các thành phần AI tự động đánh giá chất lượng bản dịch (ví dụ sử dụng các chỉ số như BLEU, ROUGE, METEOR, hoặc đánh giá dựa trên human feedback) để cải tiến mô hình liên tục theo thời gian thực.
- **Ứng dụng Reinforcement Learning từ Feedback:** Áp dụng kỹ thuật Reinforcement Learning with Human Feedback (RLHF) để tinh chỉnh mô hình dịch dựa trên phản hồi thực tế của người dùng, giúp mô hình thích nghi tốt hơn với nhu cầu và phong cách ngôn ngữ khác nhau.
- **Mở rộng sang đa ngôn ngữ:** Phát triển mô hình dịch không chỉ từ tiếng Anh sang tiếng Việt mà còn hỗ trợ thêm nhiều ngôn ngữ khác, xây dựng hệ thống dịch đa ngôn ngữ theo hướng zero-shot translation.
- **Tăng cường tính tương tác người dùng:** Xây dựng giao diện tương tác cho phép người dùng chọn lựa phong cách dịch (ví dụ: trang trọng, đời thường, kỹ thuật) nhằm cá nhân hóa trải nghiệm dịch thuật.

Với các hướng tiếp cận này, nhóm kỳ vọng sẽ xây dựng được một hệ thống dịch thuật thông minh hơn, linh hoạt hơn và thân thiện hơn với người sử dụng trong các ứng dụng thực tế.

References

[1] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, 2002.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martín Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8026–8037, 2019.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[4] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, and Joe Brew. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, pages 38–45, 2020.