TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**
**KHOA CÔNG NGHỆ THÔNG TIN**


**PHẠM ĐỨC MINH HIẾU – 52100796**
**TRẦN NGUYÊN LINH – 52000569**
**NGUYỄN THÀNH QUÍ – 52100995**

# BÁO CÁO CUỐI KỲ
# NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**
**KHOA CÔNG NGHỆ THÔNG TIN**


**PHẠM ĐỨC MINH HIẾU – 52100796**
**TRẦN NGUYÊN LINH – 52000569**
**NGUYỄN THÀNH QUÍ – 52100995**

# BÁO CÁO CUỐI KỲ
# NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

Người hướng dẫn
**PSG.TS. Lê Anh Cường**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

# LỜI CẢM ƠN

Lời đầu tiên chúng em xin được phép cảm ơn quý thầy cô khoa Công nghệ Thông Tin trường Đại học Tôn Đức Thắng đã tạo mọi điều kiện cho chúng em được trau dồi kiến thức về chuyên môn cũng như khả năng tư duy và sáng tạo trong lĩnh vực Khoa học máy tính với môn Nhập môn xử lý ngôn ngữ tự nhiên.

Chúng em cũng xin được gửi lời cảm ơn đến PGS.TS. Lê Anh Cường, thầy đã nhiệt tình giảng dạy, trang bị đầy đủ kiến thức để em có thể hoàn thành dự án cuối kỳ này.

Cuối cùng, do hạn chế về mặt kiến thức, kính mong thầy cô có thể bỏ qua những sai sót nhỏ và chỉ ra được những lỗi sai của chúng em trong bài báo cáo này để những bài báo cáo sau của chúng em được hoàn thiện hơn.

Một lần nữa chúng em xin chân thành cảm ơn thầy và toàn thể quý thầy cô khoa Công Nghệ Thông Tin trường Đại học Tôn Đức Thắng.

*TP. Hồ Chí Minh, ngày 6 tháng 12 năm 2024*

*Tác giả*

*Phạm Đức Minh Hiếu*

*Trần Nguyên Linh*

*Nguyễn Thành Quí*

# CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
# TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của PGS.TS. Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình**. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 6 tháng 12 năm 2024*

*Tác giả*

*Phạm Đức Minh Hiếu*

*Trần Nguyên Linh*

*Nguyễn Thành Quí*

# TÓM TẮT

Bài báo cáo này trình bày về những nội dung sau:

Câu 1:

a) Trình bày cơ chế Attention trong các mô hình Sequence – to – Sequence và phân tích sự khác nhau về sự khác nhau về Atterntion giữa các kiến trúc bao gồm:

- Mô hình dựa trên LSTM
- Mô hình Transformer
- Mô hình GPT (Generative Pretrain Transformer)

b) Xây dựng mô hình LSTM – to – LSTM có attention và không có attention cho bài toán Dịch máy từ ngôn ngữ tiếng Việt sang tiếng Anh.

Câu 2:

a) Chọn 1 bài toán cụ thể thuộc loại trích chọn thông tin (Information Extraction), chọn bài toán có ý nghĩa trong thực tế. Và ưu tiên chọn loại bài toán mới.

b) Xây dựng mô hình Transformer (có thể sử dụng Pretrained model) để huấn luyện và đánh giá bài toán trên và đánh giá trên bộ Test với các độ đo Precison, Recall, F1-score.

# MỤC LỤC

# DANH MỤC HÌNH VẼ

# DANH MỤC BẢNG BIỂU

# DANH MỤC CÁC CHỮ VIẾT TẮT

| | |
|---|---|
| NER | Named entity recognition |
| AI | Artificial Intelligence |
| BERT | Bidirectional Encoder Representations from Transformers |
| GPT | Generative Pre-Trained Transformers |
| LSTM | Long Short-Term Memory |
| RNN | Recurrent Neural Network |
| IE | Information Extraction |

# CHƯƠNG 1. CẤU TRÚC CÁC ATTENTION

## 1.1 Giới thiệu mô hình Seq2Seq

### *1.1.1 Khái niệm và thành phần*

Seq2Seq (Sequence-to-Sequence) là một kiến trúc mạng nơ-ron được sử dụng rộng rãi trong các bài toán xử lý ngôn ngữ tự nhiên (NLP) như dịch máy, tóm tắt văn bản, và sinh văn bản. Ý tưởng chính của mô hình Seq2Seq là chuyển đổi một chuỗi đầu vào có độ dài bất kỳ thành một chuỗi đầu ra có độ dài bất kỳ.

Các thành phần chính:

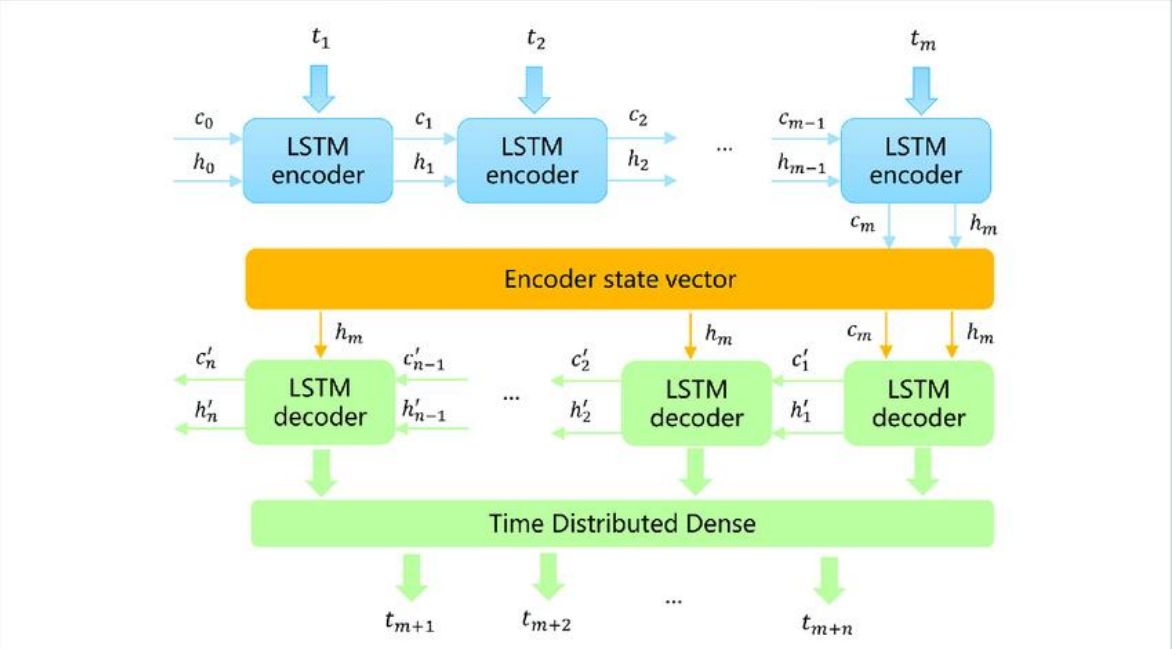
**Encoder**: Là một mạng RNN, LSTM hoặc GRU. Nhiệm vụ của Encoder là đọc chuỗi đầu vào và mã hóa thông tin này thành một vector ngữ cảnh cố định (context vector), đại diện cho toàn bộ chuỗi đầu vào.

Các bước thực hiện:

- Với mỗi từ đầu vào, trạng thái ẩn của RNN được cập nhật.
- Sau khi xử lý toàn bộ chuỗi, trạng thái ẩn cuối cùng được xem là vector ngữ cảnh.

**Decoder**: Là một mạng RNN, LSTM, hoặc GRU tương tự Encoder. Sử dụng vector ngữ cảnh từ Encoder làm đầu vào ban đầu, sau đó sinh ra chuỗi đầu ra (target sequence) từng bước một.

Tại mỗi bước, Decoder dự đoán một từ dựa trên trạng thái hiện tại và từ đã sinh ra ở bước trước.




Hình 1.1 Kiến trúc mô hình Seq2Seq

### *1.1.2 Cách hoạt động*

Với mỗi chuỗi đầu vào $X = (x_1, x_2, \dots x_n)$, thành phần Encoder sẽ tính trạng thái ẩn $h_t$ và trạng thái cell $c_t$ tại mỗi bước t:

$$h_t, c_t = LSTM(x_t, h_{t-1}, c_{t-1})$$

Trong đó:

- $x_t$: Đầu vào của chuỗi tại mỗi bước t
- $h_{t-1}, c_{t-1}$: Trạng thái ẩn và trạng thái cell bước trước đó
- $h_t, c_t$: Trạng thái ẩn và trạng thái cell tại bước t

Trạng thái cuối cùng $h_T, c_T$ sẽ đại diện cho toàn bộ ngữ nghĩa của toàn bộ chuỗi.

Với Decoder, lớp này sẽ sử dụng $h_T, c_T$ từ Encoder làm trạng thái khởi tạo, và $y_0$ luôn là <sos>, $s_0 = h_T, c_0 = c_T$.

Chuỗi đầu ra $Y = (y_1, y_2, \dots y_T)$, sẽ được tạo ra bởi lớp Decoder thông qua từng bước sau:

Tính trạng thái ẩn tại bước t

$$s_t, c_t = LSTM(y_{t-1}, s_{t-1}, c_{t-1})$$

- $s_{t-1}, c_{t-1}$: Trạng thái ẩn và trạng thái cell bước trước đó
- $y_{t-1}$: Từ đầu ra ở từng bước trước

Sau đó dự đoán đầu ra $y_t$:

$$y_t = argmax(softmax(W_0 s_t + b))$$

Và $y_t$ sẽ được đưa vào bước tiếp theo để dự đoán từ tiếp theo $y_{t+1}$.

Tuy nhiên cách thức hoạt động này sẽ tồn đọng một số hạn chế sau:

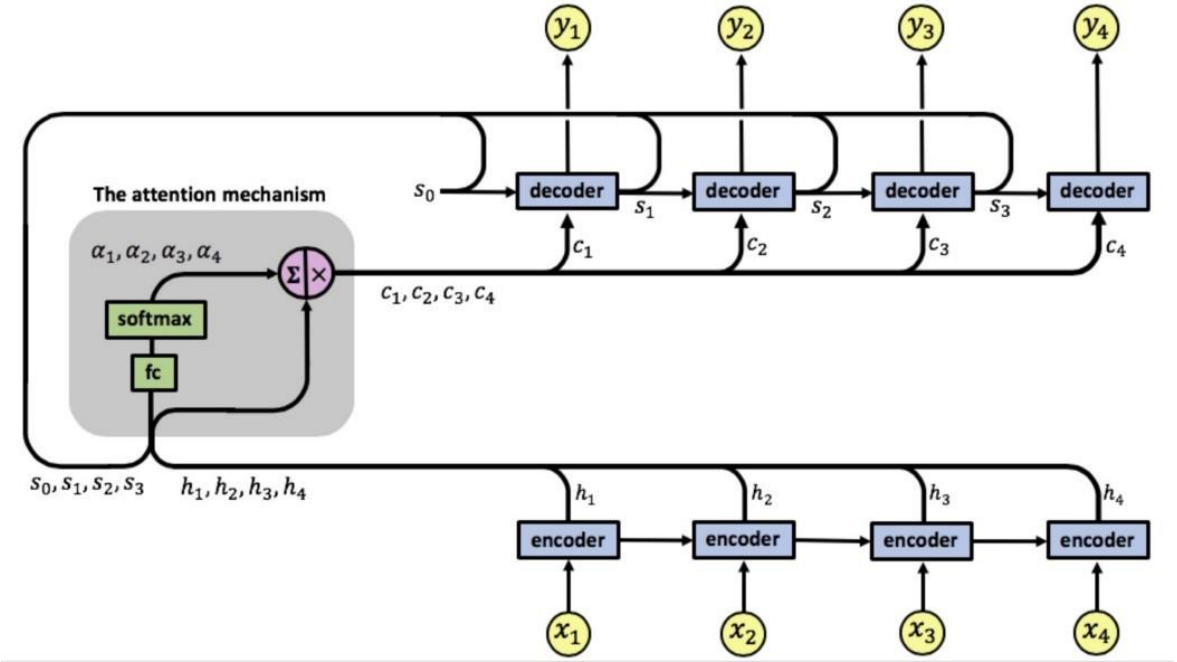
- **Thông tin bị nén quá mức**: Toàn bộ chuỗi đầu vào được mã hóa thành một vector duy nhất $h_T$, dẫn đến mất thông tin, đặc biệt với các chuỗi dài hoặc phức tạp.
- **Thiếu khả năng tập trung:** Mô hình không có tập trung vào các từ quan trọng của chuỗi đầu vào khi sinh ra các từ trong chuỗi đầu ra.
- **Hiệu quả giảm dần với chuỗi dài**: Khi chuỗi đầu vào dài, trạng thái cuối không thể lưu giữ đủ thông tin, dẫn đến việc mô hình không thể sinh ra đầu ra chính xác.

Chính vì vậy mà attention được ra đời để khắc phục đi những hạn chế đó.

## 1.2 Mô hình Seq2Seq dựa trên LSTM với Attention

### *1.2.1 Khái niệm*

Attention trong Seq2Seq là cơ chế giúp mô hình tập trung vào các phần quan trọng của chuỗi đầu vào khi sinh chuỗi đầu ra. Thay vì sử dụng một vector cố định từ Encoder (trạng thái cuối $h_T, c_T$), Attention cho phép mô hình truy cập linh hoạt vào toàn bộ các trạng thái ẩn $h_1, h_2, \dots h_T$ của Encoder.




Hình 1.2 Cơ chế attention trong mô hình Seq2Seq

Đối với LSTM, Attention được giới thiệu lần đầu tiên trong bài báo **"Neural Machine Translation by Jointly Learning to Align and Translate" (Bahdanau et al., 2014)**.

Mục tiêu chính:

- Giải quyết hạn chế của Seq2Seq với LSTM khi thông tin bị nén quá mức trong trạng thái cuối $h_T$.
- Cải thiện hiệu suất dịch máy và các tác vụ xử lý ngôn ngữ tự nhiên (NLP) khác, đặc biệt với chuỗi dài.

### *1.2.2 Cách hoạt động*

Giống với Seq2Seq LSTM, phần Encoder vẫn nhận chuỗi đầu vào $X = (x_1, x_2, \dots x_n)$, thành phần Encoder sẽ tính trạng thái ẩn $h_t$ và trạng thái cell $c_t$ tại mỗi bước t:

$$h_t, c_t = LSTM(x_t, h_{t-1}, c_{t-1})$$

Trong đó:

- $x_t$: Đầu vào của chuỗi tại mỗi bước t
- $h_{t-1}, c_{t-1}$: Trạng thái ẩn và trạng thái cell bước trước đó

- $h_t, c_t$: Trạng thái ẩn và trạng thái cell tại bước t

Đầu ra của Encoder sẽ là dãy các trạng thái ẩn $H = (h_1, h_2, \dots h_T)$.

Sự khác biệt so với mô hình cũ là dùng cơ chế Attention vào lớp Decoder. Khi sinh ra một từ đầu ra $y_t$, Decoder không chỉ dựa vào trạng thái ẩn $s_{t-1}$ mà còn kết hợp thông tin từ toàn bộ trạng thái ẩn H của Encoder theo các bước sau:

Bước 1: Tính score của Attention

Trọng số Attention $e_{t,i}$ xác định mức độ liên quan giữa trạng thái ẩn $s_{t-1}$của Decoder và trạng thái ẩn $h_i$ của Encoder theo công thức:

$$e_{t,i} = score(s_{t-1}, h_i)$$

Một số hàm score được sử dụng:

- Dot – product: $s_{t-1}^T h_i$
- General: $s_{t-1}^T W h_i$ với W là ma trận trọng số học được. (**Luong Attention**)
- Concatenation: $v^T \tanh(W_s s_{t-1} + W_h h_i)$ với $v, W_s, W_h$ là các tham số học được. (**Bahdanau Attention**)

Bước 2: Tính softmax cho $e_{t,i}$ để chuẩn hóa thành $a_{t,i}$ thành phân phối xác suất có tổng bằng 1:

$$a_{t,i} = \frac{e^{e_{t,i}}}{\sum_{j=1}^{T} e^{e_{j,i}}}$$

Bước 3: Tính vector ngữ cảnh

Vector ngữ cảnh gói gọn thông tin từ các phần quan trọng của chuỗi đầu vào:

$$c_t = \sum_{i=1}^{T} a_{t,i} h_i$$

Bước 4: Sinh từ đầu ra bằng cách kết hợp vector ngữ cảnh $c_t$ và trạng thái ẩn $s_{t-1}$ để dự đoán từ đầu ra $y_t$:

$$s_t, c_t = LSTM([y_{t-1}, c_t], s_{t-1}, c_{t-1})$$

Dự đoán từ đầu ra bằng softmax:

$$\hat{y}_t = softmax(W_y s_t + b)$$

Cứ liên tục thực hiện đến khi gặp ký tự <eos> coi như xử lý xong chuỗi đầu ra. Khi đó ta thu được chuỗi đầu ra.

Ưu điểm của Attention:

- **Tập trung vào phần quan trọng**: Attention giúp mô hình chọn lọc thông tin từ các phần quan trọng của chuỗi đầu vào.
- **Cải thiện hiệu suất với chuỗi dài**: Không còn phụ thuộc hoàn toàn vào trạng thái cuối của Encoder.
- **Dịch máy chính xác hơn**: Mỗi từ đầu ra có thể liên kết trực tiếp với các từ tương ứng trong chuỗi đầu vào.

Nhược điểm:

- **Tính toán phức tạp:** Phải tính toán $e_{t,i}$ và $a_{t,i}$ cho mỗi cặp từ đầu vào – đầu ra.
- **Khả năng mở rộng:** Với chuỗi rất dài, việc tính Attention có thể chậm và tốn bộ nhớ.

Attention này là tiền đề cho nhiều mô hình cải tiến sau này như Transformer và GPT, khi đó Attention sẽ được áp dụng hiệu quả hơn.

## 1.3 Transformer

### *1.3.1 Lịch sử ra đời*

Transformer được giới thiệu trong bài báo "Attention is all you need" của Vaswani et al. vào năm 2017. Là một bước đột phá lớn trong lĩnh vực xử lý ngôn ngữ tự nhiên và các bài toán chuỗi nói chung.

Transformer ra đời khắc phục nhược điểm của các mô hình Seq2Seq truyền thống như RNN, LSTM, GRU như:

- Khó khăn trong xử lý chuỗi dài do hiện tượng vanishing gradient.
- Tốc độ tính toán chậm do xử lý tuần tự từ RNN.

Mặc dù đã có cơ chế Attention cho các mô hình này nhưng nó vẫn chưa đáp ứng được những yêu cầu chung do thời gian tính toán lâu vì xử lý tuần tự và xử lý chuỗi quá dài khó khăn.

### *1.3.2 Khái niệm*

Cũng dựa trên kiến trúc Seq2Seq, Transformer có hai thành phần chính:

- Encoder: chuyển đổi chuỗi đầu vào thành một vector ngữ cảnh.
- Decoder: Dùng vector ngữ cảnh để tạo chuỗi đầu ra.

Tuy nhiên khác biệt của Transformer là cả hai thành phần Encoder và Decoder bao gồm:

- Multi-head attentioon: học mối quan hệ giữa các phần tử trong chuỗi.
- Feedforward network: tăng tính phi tuyến và khả năng biểu diễn
- Add & Norm: ổn định và cải thiện quá trình học thông qua ressidual connections và normalization.

Kiến trúc được thể hiện như hình sau:

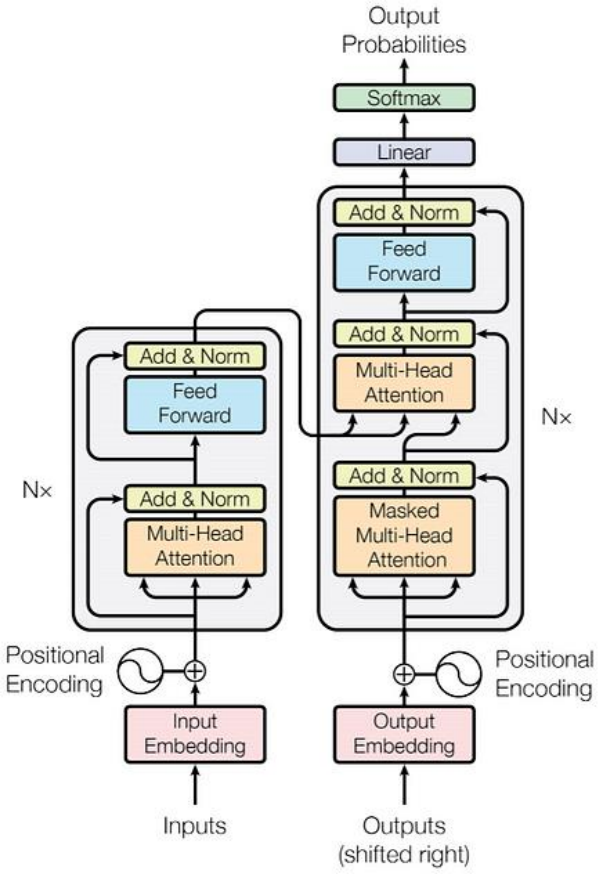
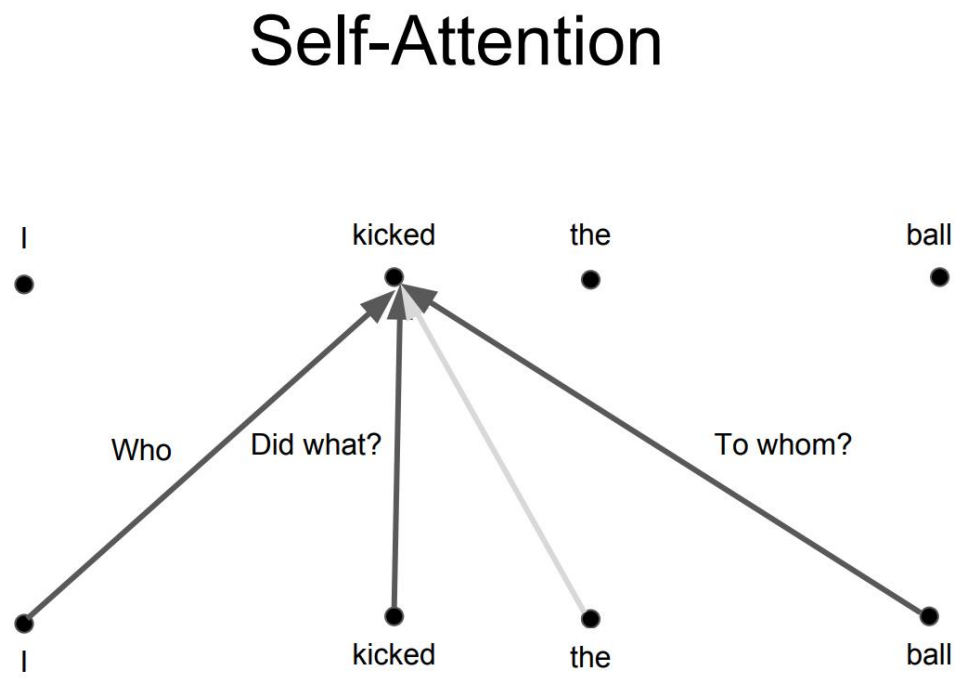



Figure 1: The Transformer - model architecture.

Hình 1.3 Kiến trúc mô hình Transformer

### *1.3.3 Self – attention*

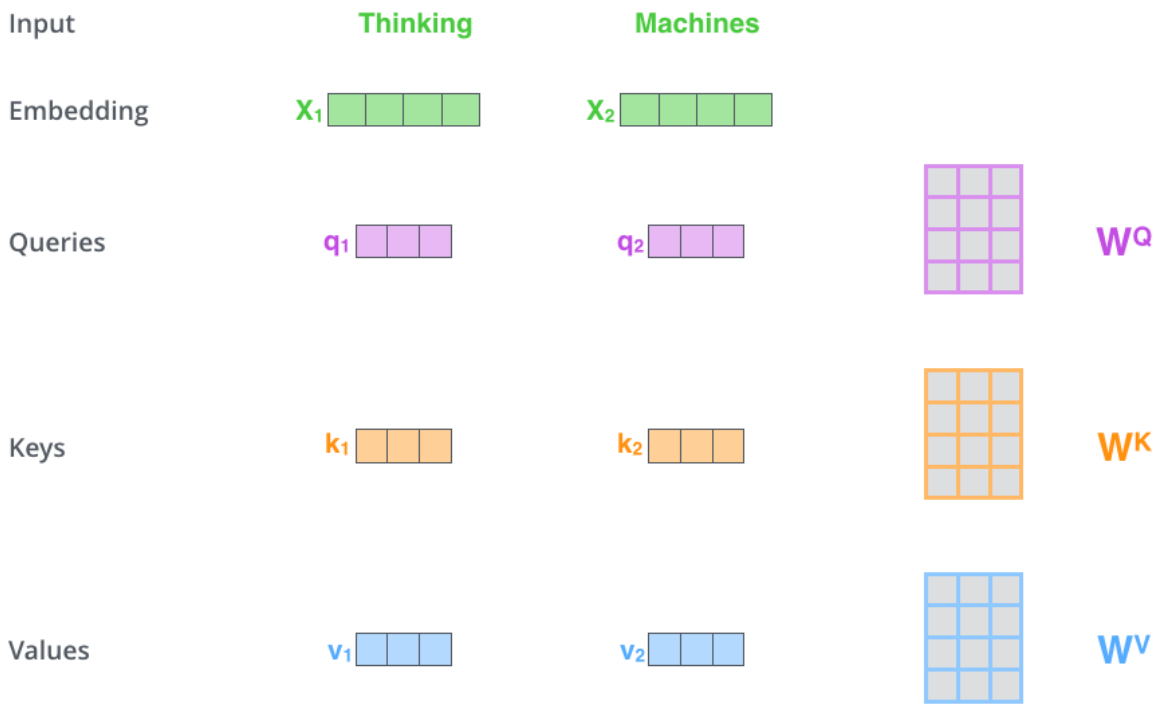
Self-Attention là cơ chế cho phép mỗi từ trong chuỗi đầu vào tương tác với tất cả các từ còn lại và quyết định phần thông tin nào cần tập trung. Nó được thực hiện bằng cách tính một vector biểu diễn mới cho từng từ, dựa trên mối quan hệ của từ đó với các từ khác trong chuỗi. Ví dụ như từ “kicked” trong câu “I kicked the ball” (tôi đã đá quả bóng) liên quan như thế nào đến các từ khác? Rõ ràng nó liên quan mật thiết đến từ “I” (chủ ngữ), “kicked” là chính nó lên sẽ luôn “liên quan mạnh” và “ball” (vị ngữ).




Hình 1.4 Ví dụ self - attention

Cơ chế này trong mô hình Transformer dựa trên 3 vector chính:

- Query (Q): biểu diễn thông tin từ một phần tử trong chuỗi
- Key (K): dùng để so sánh với query
- Value (V): dữ liệu được trọng số hóa để tính toán đầu ra




Hình 1.5 Các thành phần trong self - attention

Các vector này được tính toán bằng cách nhân embedding của từ với các ma trận trọng số đã học:

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

Trong đó:

$X$: là ma trận embedding của chuỗi đầu vào $(n \times d_{model})$ với n là số từ và $d_{model}$ là kích thước vector embedding.

$W^Q, W^K, W^V$: là các ma trận trọng số đã học có kích thước $d_{model} \times d_k$

Bước 1: Tính điểm tương đồng giữa Q và K:

$$Score(Q, K) = QK^T$$

Mỗi phần tử trong $QK^T$ biểu diễn mức độ tương đồng giữa từ truy vấn $q_i$ và từ khóa $k_j$.

Bước 2: Chuẩn hóa bằng softmax:

$$Attention\ weights = softmax\left(\frac{Score(Q, K)}{\sqrt{d_k}}\right)$$

$\sqrt{d_k}$: hệ số chuẩn hóa để tránh các giá trị lớn làm softmax bão hòa

Kết quả là ma trận trọng số attention kích thước $n \times n$.

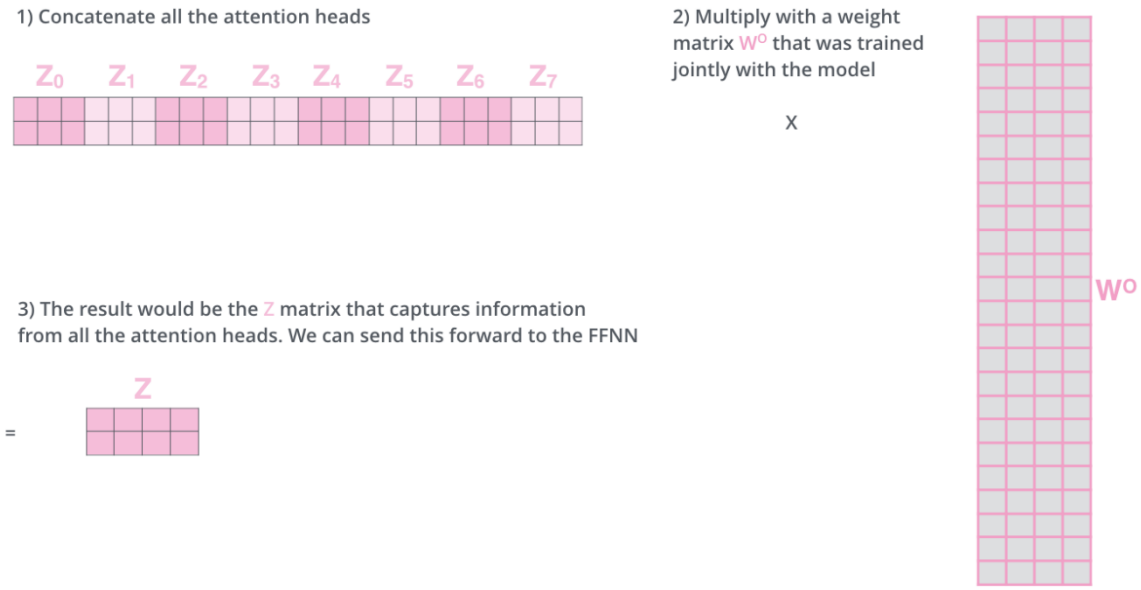
Bước 3: Tính toán giá trị đầu ra:

$$Output = Attention\ weights \times V$$

Kích thước đầu ra sẽ là $n \times d_k$.

### *1.3.4 Multi – head attention*

Multi – head Attention mở rộng Self-Attention bằng cách chia các không gian biểu diễn thành nhiều "head". Điều này giúp mô hình học được nhiều loại mối quan hệ khác nhau giữa các từ trong các không gian con.




Hình 1.6 Multi – head Attention

Cách tính như sau:

Bước 1: Chia thành các head – thay vì thực hiện self – attention trên toàn bộ không gian $d_{model}$, chúng ta chia Q, K, V thành h head (thường là 8 hoặc 16).

$$Q_i = XW_i^Q, K_i = XW_i^K, V_i = XW_i^V$$

Trong đó, $W_i^Q, W_i^K, W_i^V$ là các ma trận trọng số cho từng head.

Bước 2: Thực hiện self – attention cho từng head, tính giống self attention

$$head_i = softmax\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$

Kết quả của mỗi head là một ma trận $n \times d_k$

Bước 3: concat các head lại: nối các kết quả từ h head

$$concat(head_1, head_2, \dots head_h)$$

Kích thước kết quả là $n \times d_{model}$

Bước 4: Chiếu lại không gian ban đầu bằng cách áp dụng phép biến đổi tuyến tính:

$$MultiHead(Q, K, V) = concat(head_1, head_2, \dots head_h)W^O$$

Lợi ích của multi – head attention so với việc tính self – attention cho toàn bộ:

- Học được các mối quan hệ khác nhau giữa các từ trong các không gian con khác nhau.
- Tăng khả năng biểu diễn của mô hình mà không tăng quá nhiều độ phức tạp tính toán.

### *1.3.5 Positional Encoding:*

Transformer không có cơ chế tuần tự như RNN hay LSTM, vì vậy nó không biết vị trí tương đối của các từ trong chuỗi. Tuy nhiên, trong ngôn ngữ, vị trí của từ đóng vai trò rất quan trọng để hiểu ngữ nghĩa. Ví dụ:

- Câu "Con chó chạy" khác với "Chạy con chó".
- Trong dịch máy, từ ở các vị trí khác nhau có các mối liên hệ khác nhau, ví dụ như chủ ngữ và động từ thường nằm ở gần nhau.

Do đó, cần một cách để mô hình biết được vị trí của các từ trong chuỗi. Positional Encoding được thêm vào các vector embedding đầu vào để cung cấp thông tin về vị trí của từ trong chuỗi.

Đặc điểm của PE đó là:

- **Biểu diễn thứ tự**: Giúp mô hình hiểu mối quan hệ vị trí giữa các từ.
- **Tính liên tục**: Cung cấp mối quan hệ tuyến tính giữa các vị trí, phù hợp cho các chuỗi dài.
- **Tương thích với học sâu**: Kích thước của PE được giữ giống với embedding (dmodel), để có thể cộng trực tiếp vào vector từ.

Cách tính PE sử dụng sin và cosin để mã hóa vị trí như sau:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

Sau khi tính toán, PE này sẽ được thêm vào embedding để thêm thông tin thứ tự cho chúng:

$$Input_{encoded} = Embedding(x) + PE$$

### *1.3.6 Điểm mạnh của Transformer*

Loại bỏ tính tuần tự trong xử lý:

- Sử dụng cơ chế Attention, đặc biệt là Self-Attention, để xử lý toàn bộ chuỗi cùng lúc.
- Không cần xử lý tuần tự, cho phép mô hình tính toán toàn bộ chuỗi đồng thời, giúp tăng tốc độ huấn luyện và suy luận đáng kể.
- Nhờ đó mà mô hình Transformer có khả năng song song hóa cao, tận dụng tối đa GPU/TPU.

Xử lí phụ thuộc dài hạn tốt hơn:

- Sử dụng Self – Attention, cho phép mỗi từ trong chuỗi có thể "chú ý" đến mọi từ khác trong chuỗi, bất kể khoảng cách.
- Thời gian và chi phí để học các phụ thuộc dài hạn không tăng theo độ dài chuỗi (phụ thuộc tuyến tính).

Độ phức tạp tính toán thấp hơn:

- Với cơ chế Self – attention, độ phức tạp mỗi bước là $O(n^2 \times d)$ nhưng toàn bộ chuỗi có thể được xử lý đồng thời nhờ tính song song hóa.
- Với các chuỗi dài, hiệu quả tính toán của Transformer vượt trội.

Khả năng mở rộng:

- Dễ dàng mở rộng quy mô, cả về chiều dài chuỗi đầu vào và kích thước mô hình.

- Các mô hình lớn như BERT, GPT-3, và T5 đều dựa trên Transformer, với hàng tỷ tham số, minh chứng cho khả năng mở rộng này.

Tính linh hoạt và hiệu quả:

- Thiết kế đơn giản và hiệu quả, phù hợp với nhiều loại bài toán Seq2Seq.
- Các mô hình dựa trên Transformer (như BERT, GPT, và T5) đã được chứng minh là hiệu quả trong nhiều tác vụ: dịch máy, tóm tắt, trả lời câu hỏi, phân loại văn bản, v.v.

Tích hợp thông tin thứ tự nhờ vào PE:

- Sử dụng Positional Encoding để thêm thông tin thứ tự vào embedding.
- Điều này không chỉ giữ được thứ tự mà còn duy trì khả năng song song hóa.

## 1.4 Attention trong mô hình GPT

### *1.4.1 Khái niệm mô hình GPT*

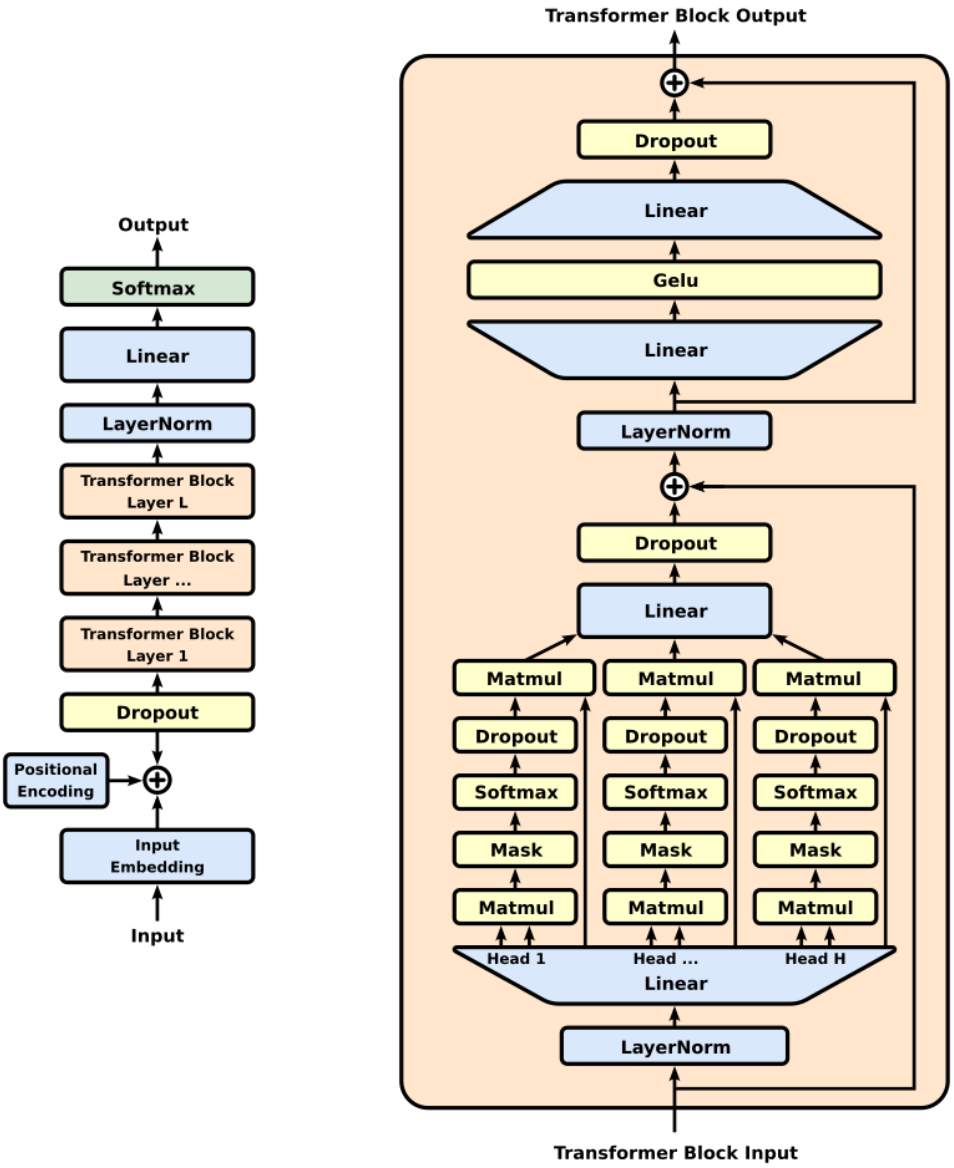
GPT (Generative Pre-trained Transformer) là một mô hình ngôn ngữ tự hồi quy, được xây dựng dựa trên kiến trúc Transformer Decoder. Mô hình này được thiết kế để dự đoán từ tiếp theo trong một chuỗi, từ đó thực hiện các tác vụ xử lý ngôn ngữ tự nhiên (NLP) như sinh văn bản, trả lời câu hỏi, dịch ngôn ngữ, và tóm tắt tài liệu.

Đặc điểm nổi bật của GPT bao gồm:

- **Tự hồi quy (Autoregressive):** Dự đoán từ tiếp theo dựa trên các từ trước đó, giúp tạo ra các văn bản một cách tự nhiên, việc giới hạn chú ý chỉ đến các từ trước đó là rất quan trọng.
- **Pre-training và Fine-tuning**: Huấn luyện trước trên dữ liệu lớn, sau đó tinh chỉnh để phù hợp với các bài toán cụ thể.
- **Học không giám sát**: GPT tận dụng dữ liệu văn bản không gán nhãn, giúp tiết kiệm công sức gắn nhãn dữ liệu.

### *1.4.2 Cách hoạt động*

GPT hoạt động dựa trên kiến trúc Transformer Decoder với quy trình xử lý gồm ba giai đoạn chính: tiền xử lý đầu vào, xử lý qua các lớp Transformer, và dự đoán đầu ra. Mỗi giai đoạn đảm nhận một vai trò cụ thể, được tối ưu hóa để khai thác ngữ cảnh và tạo ra các dự đoán chính xác.




Hình 1.7 Kiến trúc của GPT

Các bước thực hiện:

- **Tiền xử lý đầu vào**: Tokenizer, Embedding và Positional Encoding.
- **Xử lý qua các lớp Transformer Decoder**: GPT sử dụng một chuỗi N lớp Transformer Decoder, mỗi lớp thực hiện các bước chính: Masked

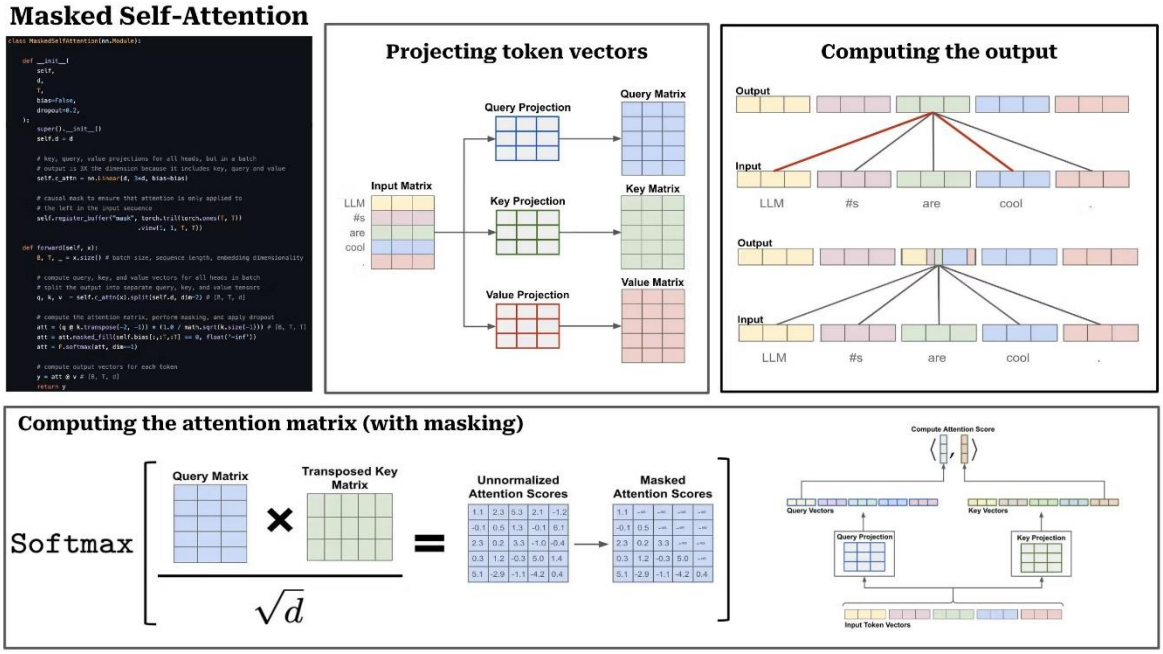
Self – Attention, để học được nhiều ngữ cảnh khác nhau, GPT sử dụng nhiều head Attention (Multi – head Attention). Các đầu này hoạt động song song, mỗi đầu làm việc trên một không gian đặc trưng khác nhau và Feedforward Neural Network.

- **Sinh chuỗi**: Sau khi xử lý qua các lớp Transformer, đầu ra cuối cùng là vector $h_i$ của mỗi token. Vector này được chuyển qua lớp Softmax để tính xác suất của mỗi từ trong từ điển. Sau đó sinh từ tiếp theo bằng cách chọn từ có xác suất cao nhất làm đầu ra hoặc áp dụng các chiến lược sinh khác như Beam Search hoặc Sampling để đa dạng hóa. Khi một từ được sinh, nó được thêm vào chuỗi và quá trình lặp lại cho đến khi đạt giới hạn độ dài hoặc gặp ký tự kết thúc.

Do GPT sử dụng Masked Self – Attention và nó có sự khác biệt so với Transfomer nên nó có sự khác biệt đáng kể so với Transformer gốc về mục tiêu và cách triển khai.

### *1.4.3 Masked Self – Attention*

Masked Self – Attention là thành phần cốt lõi giúp GPT học cách chú ý đến các từ trước đó trong chuỗi. Nó được xây dựng dựa trên Self – Attention từ Transformer. Nó đảm bảo rằng mỗi từ trong chuỗi chỉ được phép chú ý đến các từ đứng trước nó hoặc chính nó, nhờ vào việc áp dụng một ma trận mặt nạ (mask matrix). Điều này phù hợp với tính chất tự hồi quy của GPT, vì việc dự đoán từ tiếp theo không thể sử dụng thông tin từ các từ tương lai.




Hình 1.8 Kiến trúc Masked self - attention

Masked Self-Attention được xây dựng dựa trên cơ chế Attention tổng quát, vốn tính toán mức độ quan trọng của mỗi từ trong chuỗi đối với các từ khác. Cơ chế này hoạt động trên ba ma trận:

- Query (Q): Biểu diễn từ hiện tại cần chú ý.
- Key (K): Biểu diễn các từ trong chuỗi được so sánh với Query.
- Value (V): Biểu diễn thông tin mà mô hình sẽ trích xuất sau khi chú ý.

Các ma trận $Q, K, V$ được ánh xạ từ đầu vào thông qua các ma trận trọng số $W^Q, W^K, W^V$.

Cách hoạt động như sau:

**Bước 1**: Tính điểm tương đồng giữa Q và K:

$$Score(Q, K) = \frac{QK^T}{\sqrt{d_k}}$$

Mỗi phần tử trong $QK^T$ biểu diễn mức độ tương đồng giữa từ truy vấn $q_i$ và từ khóa $k_j$.

**Bước 2**: Áp dụng masked: Mask được sử dụng để che đi các vị trí tương ứng với từ tương lai trong chuỗi. Mask là một ma trận $M$ có kích thước $n \times n$.

- $M_{ij} = 0$ nếu từ $j \leq i$ (từ hiện tại hoặc trước đó)
- $M_{ij} = -\infty$ nếu từ $j > i$ (từ tương tai)

  Vậy lúc này Score sẽ được masked lại như sau:

$$Masked\ Scores = \frac{QK^T}{\sqrt{d_k}} + M$$

**Bước 3**: Chuẩn hóa bằng softmax:

$$Attention\ weights_{ij} = softmax\big(Masked\ Score_{ij}\big)$$

$\sqrt{d_k}$: hệ số chuẩn hóa để tránh các giá trị lớn làm softmax bão hòa

Kết quả là ma trận trọng số attention kích thước $n \times n$.

**Bước 4**: Tính toán giá trị đầu ra:

$$Output = Attention\ weights \times V$$

Đầu ra $Output \in R^{n \times d^k}$ chứa thông tin tổng hợp từ các từ trước đó, theo mức độ quan trọng được xác định bởi Attention Weights.

### *1.4.4 Đặc điểm:*

Hiệu suất cao nhờ Pre-training và Fine-tuning:

- Pre-training trên dữ liệu lớn: GPT được huấn luyện trước (pre-trained) trên lượng lớn dữ liệu từ internet, giúp mô hình học được các mẫu ngôn ngữ tự nhiên và mối quan hệ giữa các từ, cụm từ, và câu.
- Fine-tuning theo từng nhiệm vụ cụ thể: Sau khi pre-training, GPT có thể được tinh chỉnh (fine – tuned) trên các tập dữ liệu nhỏ hơn để thích nghi với các tác vụ cụ thể như trả lời câu hỏi, dịch máy, hoặc tóm tắt văn bản.

Sử dụng kiến trúc Transformer tiên tiến:

- Self – Attention và Masked Self – Attention: GPT sử dụng cơ chế Masked Self – Attention trong Transformer Decoder để nắm bắt các mối quan hệ ngữ cảnh phức tạp trong văn bản. Điều này cho phép mô hình:
  - Hiểu mối quan hệ giữa các từ ở khoảng cách xa.

- Đảm bảo tính nhân quả khi dự đoán từ tiếp theo.

- Tính toán song song: So với các mô hình tuần tự như RNN, GPT có thể xử lý toàn bộ chuỗi văn bản cùng lúc, giúp tăng hiệu suất tính toán.

Tính tự hồi quy (Autoregressive) mạnh mẽ:

- Cách dự đoán từng bước: GPT dự đoán từng từ trong chuỗi một cách tuần tự, sử dụng các từ trước đó làm ngữ cảnh.
- Khả năng sinh văn bản: Với cách hoạt động này, GPT có thể tạo ra các đoạn văn bản mạch lạc và tự nhiên, phù hợp với ngữ cảnh đầu vào.

Khả năng zero-shot, one-shot, và few-shot learning:

- Zero-shot learning: GPT có thể thực hiện các tác vụ mà không cần huấn luyện thêm, chỉ cần cung cấp ngữ cảnh trong lời nhắc (prompt). Ví dụ: "Tóm tắt đoạn văn này:".
- One-shot learning: Mô hình học từ một ví dụ duy nhất, sau đó áp dụng ngay lập tức.
- Few-shot learning: GPT chỉ cần một vài ví dụ để thực hiện tốt các nhiệm vụ phức tạp.

Tính linh hoạt và khả năng áp dụng rộng rãi

GPT có thể được áp dụng cho nhiều tác vụ NLP khác nhau nhờ kiến trúc linh hoạt của nó:

- Sinh văn bản: Viết nội dung, hoàn thiện câu, tạo đoạn văn.
- Trả lời câu hỏi: Đưa ra câu trả lời mạch lạc từ ngữ cảnh.
- Dịch ngôn ngữ: Dịch từ ngôn ngữ này sang ngôn ngữ khác.
- Tóm tắt văn bản: Rút gọn thông tin quan trọng từ văn bản dài.

Tận dụng mô hình lớn (Large-scale Models):

- Kiến trúc lớn hơn, nhiều tham số hơn: GPT được thiết kế với số lượng lớn tham số (hàng tỷ) để học được nhiều mối quan hệ ngữ nghĩa phức tạp hơn.

- Hiệu ứng tăng cường từ mô hình lớn: Các phiên bản lớn hơn của GPT (như GPT-3.5, GPT-4) có thể xử lý ngữ cảnh dài hơn và thực hiện các tác vụ khó hơn với hiệu suất cao.

Tính mở rộng nhờ huấn luyện không giám sát:

- Huấn luyện trên dữ liệu không có nhãn: GPT được huấn luyện trên khối lượng lớn văn bản tự nhiên, không cần dữ liệu có nhãn. Điều này giúp giảm chi phí gắn nhãn dữ liệu và tăng khả năng mở rộng.

## 1.5 So sánh các attention của ba mô hình

### *1.5.1 Seq2Seq với Attention (LSTM-based)*

Cơ chế Attention:

- Seq2Seq với Attention tính trọng số giữa trạng thái hiện tại của Decoder và các trạng thái ẩn của Encoder.
- Sử dụng các hàm tính điểm như Dot-Product, General, hoặc Bahdanau (Additive) để xác định trọng số.

Ưu điểm:

- Tăng khả năng xử lý chuỗi dài bằng cách tập trung vào các phần quan trọng thay vì dựa vào một vector ngữ cảnh duy nhất.
- Cải thiện chất lượng đầu ra trong dịch máy hoặc sinh văn bản.

Nhược điểm: Tính toán phức tạp, không tối ưu cho chuỗi dài vì phụ thuộc vào trạng thái của từng bước.

### *1.5.2 Transformer với Self-Attention*

Cơ chế Attention:

- Self - Attention trong Transformer cho phép mỗi từ trong chuỗi đầu vào tương tác với tất cả các từ khác để quyết định trọng số.
- Sử dụng Query, Key, và Value để tính toán mối quan hệ giữa các từ trong chuỗi.

Ưu điểm:

- Xử lý toàn bộ chuỗi đồng thời, tăng tốc độ tính toán so với Seq2Seq.
- Học tốt các phụ thuộc dài hạn trong chuỗi nhờ Self-Attention.
- Multi-head Attention cải thiện khả năng biểu diễn bằng cách học các loại mối quan hệ khác nhau trong dữ liệu.

Nhược điểm: Tốn tài nguyên tính toán và bộ nhớ cho các chuỗi rất dài do ma trận Attention có kích thước

### *1.5.3 GPT với Masked Self-Attention*

Cơ chế Attention:

- GPT sử dụng Masked Self-Attention, trong đó mỗi từ chỉ được phép chú ý đến các từ trước đó trong chuỗi.
- Được xây dựng dựa trên Decoder của Transformer, tập trung vào tính tự hồi quy.

Ưu điểm:

- Phù hợp cho các tác vụ sinh văn bản nhờ tính tự hồi quy.
- Tận dụng dữ liệu lớn trong quá trình tiền huấn luyện để học ngữ cảnh hiệu quả.

Nhược điểm: Không thể tận dụng thông tin từ tương lai, hạn chế trong các bài toán yêu cầu ngữ cảnh toàn chuỗi.

Vậy ta có bảng so sánh sau:

| **Đặc điểm** | **Seq2Seq LSTM Attention** | **Transformer (Self-Attention)** | **GPT (Masked Self-Attention)** |
|---|---|---|---|
| **Cấu trúc Attention** | Dựa vào Encoder-Decoder | Toàn bộ chuỗi với Self-Attention | Masked Self-Attention (tự hồi quy) |
| **Ưu điểm chính** | Tập trung vào các phần quan trọng | Song song hóa, học phụ thuộc dài hạn | Tối ưu cho sinh văn bản |

| | | | |
|---|---|---|---|
| **Nhược điểm** | Phức tạp, không hiệu quả với chuỗi dài | Tốn tài nguyên với chuỗi rất dài | Không sử dụng thông tin từ tương lai |
| **Ứng dụng phổ biến** | Dịch máy, tóm tắt văn bản | Dịch máy, phân loại, trích chọn thông tin | Sinh văn bản, trả lời câu hỏi |

## 1.6 Ứng dụng mô hình LSTM2LSTM cho bài toán dịch máy từ tiếng Việt sang tiếng Anh

Cả 2 mô hình đều được xây dựng bằng PyTorch – thư viện học máy phổ biến hiện nay cho các mô hình Seq2Seq. Sử dụng GPU T4 để huấn luyện mô hình.

Dữ liệu được lấy từ liên kết: https://github.com/ccr-cheng/English-to-Vietnamese-NMT-Model/tree/master/en_vi

Đầu tiên chúng ta tiền xử lý dữ liệu đưa vào lớp Lang: Dùng để ánh xạ từ sang chỉ số (word2index) và ngược lại (index2word). Thống kê số lần xuất hiện của mỗi từ (word2count). Dùng hàm normalizeString để chuẩn hóa văn bản và loại bỏ ký tự không mong muốn. Dùng hàm readLangs để đọc dữ liệu file và chuẩn hóa sau đó sử dụng hàm prepareData để kết hợp các hàm trên xây dựng bộ dữ liệu huấn luyện.

```
SOS_token = 0
EOS_token = 1

class Lang:
    def __init__(self, name):
        self.name = name
        self.word2index = {}
        self.word2count = {}
        self.index2word = {0: "SOS", 1: "EOS"}
        self.n_words = 2  # Count SOS and EOS

    def addSentence(self, sentence):
        for word in sentence.split(' '):
            self.addWord(word)

    def addWord(self, word):
        if word not in self.word2index:
            self.word2index[word] = self.n_words
            self.word2count[word] = 1
            self.index2word[self.n_words] = word
            self.n_words += 1
        else:
            self.word2count[word] += 1
```

Hình 1.9 Lớp Lang

```python
def unicodeToAscii(s):
    return unicodedata.normalize('NFKC', s)  # Chuyển về dạng chuẩn mà không xóa dấu

# Không loại bỏ dấu tiếng Việt và giữ lại số
def normalizeString(s):
    s = s.lower().strip()  # Chuyển về chữ thường và loại bỏ khoảng trắng dư thừa

    # Thay thế các thực thể HTML phổ biến
    s = s.replace("&apos;", "'")
    s = s.replace("&quot;", "\"")

    # Giữ lại chữ cái, số, dấu câu và ký tự tiếng Việt
    s = re.sub(r"[^a-zA-Z0-9áàảãạăắằẳẵặâấầẩẫậđéèẻẽẹêếềểễệíìỉĩịóòỏõọôốồổỗộơớờởỡợúùủũụưứừửữựýỳỷỹỵ.!?,']+", " ", s)

    return s.strip()
```

Hình 1.10 Hàm normalizeString

```python
def readLangs(src_file, tgt_file, lang1, lang2, reverse=False):
    print("Reading lines...")

    # Đọc file nguồn và đích
    with open(src_file, encoding='utf-8') as f_src, open(tgt_file, encoding='utf-8') as f_tgt:
        src_lines = f_src.read().strip().split('\n')
        tgt_lines = f_tgt.read().strip().split('\n')

    # Kiểm tra số lượng dòng phải khớp
    assert len(src_lines) == len(tgt_lines), "Số dòng trong file nguồn và đích không khớp!"
    print(len(src_lines))
    print(len(tgt_lines))
    # Tạo danh sách cặp câu
    pairs = [[normalizeString(src), normalizeString(tgt)] for src, tgt in zip(src_lines, tgt_lines)]

    # Đảo ngược nếu cần thiết
    if reverse:
        pairs = [list(reversed(p)) for p in pairs]
        input_lang = Lang(lang2)
        output_lang = Lang(lang1)
    else:
        input_lang = Lang(lang1)
        output_lang = Lang(lang2)

    return input_lang, output_lang, pairs
```

Hình 1.11 Hàm readLangs

```python
def prepareData(lang1, lang2, reverse=False):
    input_lang, output_lang, pairs = readLangs(f'{prefix_file}/train.src', f'{prefix_file}/train.tgt', lang1, lang2, reverse)
    print("Read %s sentence pairs" % len(pairs))
    pairs = filterPairs(pairs)
    print("Trimmed to %s sentence pairs" % len(pairs))
    print("Counting words...")
    for pair in pairs:
        input_lang.addSentence(pair[0])
        output_lang.addSentence(pair[1])
    print("Counted words:")
    print(input_lang.name, input_lang.n_words)
    print(output_lang.name, output_lang.n_words)
    return input_lang, output_lang, pairs
```

Hình 1.12 Hàm prepareData

Tiếp đến là xây dựng mô hình LSTM-to-LSTM:

Lớp Encoder (EncoderRNN):

- Dùng nn.Embedding để ánh xạ từ thành vector.
- Sử dụng LSTM để mã hóa chuỗi đầu vào thành trạng thái ẩn.

Decoder (DecoderRNN):

- Tạo dự đoán từ từng bước bằng trạng thái ẩn của Encoder.
- Hỗ trợ hai chế độ:
  - Teacher Forcing: Dùng từ mục tiêu làm đầu vào.
  - Tự hồi quy: Dùng từ dự đoán trước làm đầu vào kế tiếp.

Dùng cơ chế Bahdanau Attention và đưa vào lớp AttnDecoder:

- Tính toán trọng số attention dựa trên trạng thái hiện tại của Decoder và trạng thái ẩn của Encoder thông qua hàm tanh.
- Kết hợp thông tin từ trạng thái ẩn và vector ngữ cảnh (context vector).

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size, dropout_p=0.1):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, input):
        embedded = self.dropout(self.embedding(input))
        output, hidden = self.lstm(embedded)
        return output, hidden
```

Hình 1.13 Lớp Encoder

```python
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None):
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, device=device).fill_(SOS_token)
        decoder_hidden = encoder_hidden
        decoder_outputs = []

        for i in range(MAX_LENGTH):
            decoder_output, decoder_hidden  = self.forward_step(decoder_input, decoder_hidden)
            decoder_outputs.append(decoder_output)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teacher forcing
            else:
                # Without teacher forcing: use its own predictions as the next input
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from history as input

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        return decoder_outputs, decoder_hidden, None # We return `None` for consistency in the training loop

    def forward_step(self, input, hidden):
        output = self.embedding(input)
        output = F.relu(output)
        output, hidden = self.lstm(output, hidden)
        output = self.out(output)
        return output, hidden
```

Hình 1.14 Lớp Decoder

```python
class BahdanauAttention(nn.Module):
    def __init__(self, hidden_size):
        super(BahdanauAttention, self).__init__()
        self.Wa = nn.Linear(hidden_size, hidden_size)
        self.Ua = nn.Linear(hidden_size, hidden_size)
        self.Va = nn.Linear(hidden_size, 1)

    def forward(self, query, keys):
        scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))
        scores = scores.squeeze(2).unsqueeze(1)

        weights = F.softmax(scores, dim=-1)
        context = torch.bmm(weights, keys)

        return context, weights
```

Hình 1.15 Lớp Attention

```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1):
        super(AttnDecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.attention = BahdanauAttention(hidden_size)
        self.lstm = nn.LSTM(2 * hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None):
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, device=device).fill_(SOS_token)
        decoder_hidden = encoder_hidden
        decoder_outputs = []
        attentions = []

        for i in range(MAX_LENGTH):
            decoder_output, decoder_hidden, attn_weights = self.forward_step(
                decoder_input, decoder_hidden, encoder_outputs
            )
            decoder_outputs.append(decoder_output)
            attentions.append(attn_weights)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teacher forcing
            else:
                # Without teacher forcing: use its own predictions as the next input
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from history as input

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        attentions = torch.cat(attentions, dim=1)

        return decoder_outputs, decoder_hidden, attentions
```

Hình 1.16 Lớp AttnDecoder

Kế đến tiến hành tạo các mô hình LSTM-to-LSTM không có attention và có attention:

```
hidden_size = 128
batch_size = 32

input_lang, output_lang, train_dataloader = get_dataloader(batch_size)

encoder = EncoderRNN(input_lang.n_words, hidden_size).to(device)
decoder = DecoderRNN(hidden_size, output_lang.n_words).to(device)

train(train_dataloader, encoder, decoder, 80, print_every=1, plot_every=5)
```

Hình 1.17 Mô hình không có attention

```
> giảm một nửa . giảm một nửa .
= halved . halved .
< halved . halved . <EOS>

> vậy nên tôi thường đưa ra một câu đố bất ngờ cho rufus .
= so i actually threw a pop quiz here onto rufus .
< so i just threw some of your ethical in the story . <EOS>

> thời điểm chụp là sau cuộc cách mạng hồi giáo năm 1979 .
= it was after the islamic revolution of 1979 .
< it was the last popular version of the next year olds . <EOS>

> ngôi nhà đầu tiên được xây dựng trong dãy nhà này là nhà số một .
= the first house ever built on a block is house number one .
< the first house is built in the house and terrify the first house of the big house . <EOS>

> tuy nhiên , phần lớn chúng ta không suy nghĩ sâu xa về việc ngủ .
= and yet , for most of us , we don 't give sleep a second thought .
< but mostly we have to think about it , but not as part of the web . <EOS>

> và đó cũng giống như là vắc xin cúm toàn cầu , cùng một thứ ?
= and that 's the same with universal flu vaccine , the same kind of thing ?
< and it 's all about cars like the flu , is the same situation , polio ? <EOS>

> và khi đó , tôi đang làm việc trên đường phố .
= now at the time , i was working down the street .
< so , as i am working on the road to flavor . <EOS>

> cảm ơn .
= thank you .
< thank you . <EOS>

> chúng tôi bắt tay thiết kế 1 toà nhà cùng nhau .
= and you can see , we started making a building together .
< we started working together for a company . <EOS>

> đây là những gì mà intel đang làm để cố gắng tăng thêm nhiều nhân trên chip
= this is what intel does to keep adding more cores onto the chip .
< this is what intel does to find housing with a much better product . <EOS>
```

Hình 1.18 Test trên các câu với mô hình không có attention

Xây dựng mô hình có attention:

```
hidden_size = 128
batch_size = 32

input_lang, output_lang, train_dataloader = get_dataloader(batch_size)

encoder_attention = EncoderRNN(input_lang.n_words, hidden_size).to(device)
decoder_attention = AttnDecoderRNN(hidden_size, output_lang.n_words).to(device)

train(train_dataloader, encoder_attention, decoder_attention, 80, print_every=5, plot_every=5)
```

```
Reading lines...
133317
133317
Read 133317 sentence pairs
Trimmed to 64755 sentence pairs
Counting words...
Counted words:
vie 11750
eng 23896
7m 35s (- 113m 45s) (5 6%) 2.1631
15m 1s (- 105m 10s) (10 12%) 1.4951
22m 28s (- 97m 23s) (15 18%) 1.2081
29m 55s (- 89m 46s) (20 25%) 1.0475
37m 21s (- 82m 11s) (25 31%) 0.9453
44m 47s (- 74m 39s) (30 37%) 0.8735
52m 16s (- 67m 12s) (35 43%) 0.8190
59m 51s (- 59m 51s) (40 50%) 0.7757
67m 19s (- 52m 21s) (45 56%) 0.7411
74m 46s (- 44m 51s) (50 62%) 0.7117
82m 17s (- 37m 24s) (55 68%) 0.6874
89m 46s (- 29m 55s) (60 75%) 0.6657
97m 14s (- 22m 26s) (65 81%) 0.6471
104m 42s (- 14m 57s) (70 87%) 0.6306
112m 13s (- 7m 28s) (75 93%) 0.6154
119m 42s (- 0m 0s) (80 100%) 0.6025
```

Hình 1.19 Mô hình có attention

```
> tất cả đều nhận được 10 đô la nếu họ đồng ý tham dự .
= they all get 10 if they agree to show up .
< all of the 10 dollars if they were aware of the needs to pray . <EOS>

> đây là 1 công ten nơ chở hàng . xây dựng và hiệu quả
= this is a shipping container . built and works .
< this is a shipping container . built and adapt . <EOS>

> cảm ơn đã mời chúng tôi đến đây ngày hôm nay .
= thank you for inviting us here today .
< thank you for inviting us here today . <EOS>

> điều đó đang xảy ra ngay tại đây .
= and this is happening here .
< it 's happening in here . <EOS>

> bạn có khoảng 30.000 gen
= you have about 30,000 genes .
< you have about 30,000 genes . <EOS>

> xin cám ơn .
= thank you .
< thank you . <EOS>

> và với mỗi trạm giá 1 triệu đô , tổng cộng sẽ tốn 12 tỷ đô .
= and at a million dollars each , that would be about 12 billion dollars .
< and every seven million dollar stereos are 12 billion dollars . <EOS>

> và ở 50 thành phố khác khắp thế giới , người ta cũng tham gia .
= and also in 50 other cities around the world , people participated .
< and in 50 cities that 's the world that people are participating . <EOS>

> bạn sẽ ở đây chứ ?
= are you going to be there ?
< now you 're here ? <EOS>
```

Hình 1.20 Thử nghiệm trên mô hình có attention

**BLEU (Bilingual Evaluation Understudy) score** là một chỉ số dùng để đánh giá chất lượng của các hệ thống dịch máy tự động bằng cách so sánh kết quả dịch của hệ thống với các câu dịch chuẩn (reference translations).

BLEU đo độ tương đồng giữa câu được dịch (candidate) và các câu tham chiếu (references) dựa trên các yếu tố:

Sự trùng khớp của n-gram:

- BLEU kiểm tra tần suất các n-gram (cụm từ liên tiếp gồm 1, 2, 3,... từ) trong câu được dịch xuất hiện trong câu tham chiếu.
- Tính toán precision cho mỗi loại n-gram (unigram, bigram, trigram,...).

Tính ngắn gọn (Brevity Penalty - BP): Nếu câu được dịch ngắn hơn câu tham chiếu một cách đáng kể, BLEU sẽ giảm điểm để tránh trường hợp hệ thống tạo ra câu ngắn không đầy đủ thông tin.

Vậy BLEU Score của hai mô hình lần lượt là:

Với mô hình không có attention:

```
test_pairs = prepareTestData(f'{prefix_file}/test.src', f'{prefix_file}/test.tgt', input_lang, output_lang)
# evaluateOnTestSet(encoder, decoder, test_pairs, input_lang, output_lang)
evaluateWithBLEU(encoder, decoder, test_pairs, input_lang, output_lang)
```

```
Average BLEU score: 0.0176
```

Hình 1.21 BLEU Score của mô hình không có attention

Với mô hình có attention:

```
test_pairs = prepareTestData(f'{prefix_file}/test.src', f'{prefix_file}/test.tgt', input_lang, output_lang)
# evaluateOnTestSet(encoder, decoder, test_pairs, input_lang, output_lang)
evaluateWithBLEU(encoder_attention, decoder_attention, test_pairs, input_lang, output_lang)
```

```
Average BLEU score: 0.1095
```

Hình 1.22 BLEU Score của mô hình có attention

Từ đó chúng ta có thể thấy rằng mô hình có attention hoạt động tốt hơn rất nhiều so với mô hình không sử dụng attention mặc dù thời gian huấn luyện sẽ lâu hơn và tiêu tốn tài nguyên hơn nhưng kết quả đem lại là khá tốt.

# CHƯƠNG 2. INFORMATION EXTRACTION

Information Extraction (IE) là một lĩnh vực trong xử lý ngôn ngữ tự nhiên (NLP) nhằm trích xuất thông tin có cấu trúc từ dữ liệu không có cấu trúc hoặc bán cấu trúc, chẳng hạn như văn bản, email hoặc bài viết trực tuyến.

## 2.1 Mục tiêu chính

- Nhận dạng thực thể (Named Entity Recognition - NER): Xác định và phân loại các thực thể trong văn bản (ví dụ: tên người, địa điểm, tổ chức).
- Trích xuất quan hệ (Relation Extraction): Xác định mối quan hệ giữa các thực thể (ví dụ: mối quan hệ giữa người và tổ chức).
- Trích xuất sự kiện (Event Extraction): Xác định các sự kiện và các thành phần liên quan (ví dụ: sự kiện mua bán, gồm người mua, người bán, sản phẩm, thời gian).
- Trích xuất thuộc tính (Attribute Extraction): Tìm kiếm các thuộc tính cụ thể của một thực thể (ví dụ: nghề nghiệp, địa chỉ).

## 2.2 Ứng dụng

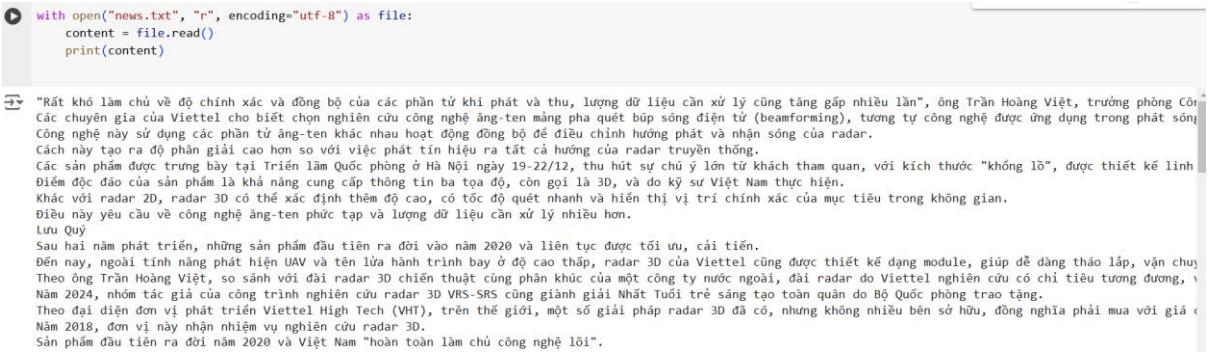
Information Extraction (IE) có rất nhiều ứng dụng thực tế trong các lĩnh vực khác nhau, giúp tự động hóa quy trình phân tích và xử lý thông tin. Trong các hệ thống hỏi đáp, IE được sử dụng để trích xuất thông tin cụ thể từ văn bản để trả lời các câu hỏi của người dùng một cách nhanh chóng và chính xác. Ngoài ra, IE đóng vai trò quan trọng trong việc phân tích văn bản, chẳng hạn như tóm tắt nội dung, phân loại chủ đề hoặc phân tích cảm xúc, từ đó cung cấp cái nhìn sâu sắc về ý nghĩa và cảm nhận ẩn chứa trong văn bản. Trong lĩnh vực tài chính, IE hỗ trợ trích xuất các dữ liệu quan trọng từ báo cáo tài chính, hợp đồng, hoặc tin tức thị trường, giúp các doanh nghiệp đưa ra quyết định nhanh chóng và hiệu quả hơn. Công nghệ này cũng tăng cường khả năng tìm kiếm thông qua việc gán thông tin có cấu trúc cho nội dung, giúp người dùng dễ dàng tìm thấy những gì họ cần. Đặc biệt, trong lĩnh vực pháp lý, IE giúp tự động trích xuất các điều khoản quan trọng từ các tài liệu pháp lý dài dòng, giúp tiết kiệm thời gian và giảm thiểu sai sót do con người.

## 2.3 Các kỹ thuật

Có nhiều kỹ thuật được áp dụng trong Information Extraction, từ những phương pháp đơn giản dựa trên quy tắc cho đến những mô hình học sâu tiên tiến. Các hệ thống dựa trên luật (Rule-based) sử dụng các quy tắc viết tay, được thiết kế bởi các chuyên gia, để nhận diện và trích xuất thông tin từ văn bản. Tuy nhiên, phương pháp này có tính linh hoạt thấp và phụ thuộc nhiều vào ngữ cảnh. Ngược lại, các mô hình học máy (Machine Learning) lại tận dụng dữ liệu huấn luyện để học và nhận dạng mẫu, từ đó trích xuất thông tin hiệu quả hơn trong nhiều ngữ cảnh khác nhau. Trong những năm gần đây, các kỹ thuật học sâu (Deep Learning) như RNN, LSTM hay Transformer ngày càng được ưa chuộng nhờ khả năng xử lý ngôn ngữ tự nhiên mạnh mẽ và độ chính xác cao. Ngoài ra, các hệ thống lai (Hybrid Systems), kết hợp giữa phương pháp dựa trên luật và học máy, mang lại hiệu quả tối ưu bằng cách tận dụng điểm mạnh của cả hai phương pháp.

## 2.4 Phương pháp xây dựng dữ liệu

Sử dụng Crawl data để lấy các đoạn văn trên vnexpress



```
with open("news.txt", "r", encoding="utf-8") as file:
    content = file.read()
    print(content)
```

```
"Rất khó làm chủ về độ chính xác và đồng bộ của các phần tử khi phát và thu, lượng dữ liệu cần xử lý cũng tăng gấp nhiều lần", ông Trần Hoàng Việt, trưởng phòng Cô
Các chuyên gia của Viettel cho biết chọn nghiên cứu công nghệ ăng-ten mảng pha quét búp sóng điện tử (beamforming), tương tự công nghệ được ứng dụng trong phát sóng
Công nghệ này sử dụng các phần tử ăng-ten khác nhau hoạt động đồng bộ để điều chỉnh hướng phát và nhận sóng của radar.
Cách này tạo ra độ phân giải cao hơn so với việc phát tín hiệu ra tất cả hướng của radar truyền thống.
Các sản phẩm được trưng bày tại Triển lãm Quốc phòng ở Hà Nội ngày 19-22/12, thu hút sự chú ý lớn từ khách tham quan, với kích thước "khổng lồ", được thiết kế linh
Điểm độc đáo của sản phẩm là khả năng cung cấp thông tin ba tọa độ, còn gọi là 3D, và do kỹ sư Việt Nam thực hiện.
Khác với radar 2D, radar 3D có thể xác định thêm độ cao, có tốc độ quét nhanh và hiển thị vị trí chính xác của mục tiêu trong không gian.
Điều này yêu cầu về công nghệ ăng-ten phức tạp và lượng dữ liệu cần xử lý nhiều hơn.
Lưu Quý
Sau hai năm phát triển, những sản phẩm đầu tiên ra đời vào năm 2020 và liên tục được tối ưu, cải tiến.
Đến nay, ngoài tính năng phát hiện UAV và tên lửa hành trình bay ở độ cao thấp, radar 3D của Viettel cũng được thiết kế dạng module, giúp dễ dàng tháo lắp, vận chuy
Theo ông Trần Hoàng Việt, so sánh với đài radar 3D chiến thuật cùng phân khúc của một công ty nước ngoài, đài radar do Viettel nghiên cứu có chỉ tiêu tương đương,
Năm 2024, nhóm tác giả của công trình nghiên cứu radar 3D VRS-SRS cũng giành giải Nhất Tuổi trẻ sáng tạo toàn quân do Bộ Quốc phòng trao tặng.
Theo đại diện đơn vị phát triển Viettel High Tech (VHT), trên thế giới, một số giải pháp radar 3D đã có, nhưng không nhiều bên sở hữu, đồng nghĩa phải mua với giá
Năm 2018, đơn vị này nhận nhiệm vụ nghiên cứu radar 3D.
Sản phẩm đầu tiên ra đời năm 2020 và Việt Nam "hoàn toàn làm chủ công nghệ lõi".
```

Hình 2.1 Crawl data trên vnexpress

Sau đó gắn nhãn trực tiếp từng từ trong câu bằng Llama 3 8B for labelling data:

```
pipeline = transformers.pipeline(
    "text-generation",
    model=model_id,
    model_kwargs={"torch_dtype": torch.bfloat16},
    device_map="auto",
)
```

Hình 2.2 Dùng pipeline của Llama 3 8B for labelling data

```
def process_sentence(sentence, pipeline):
    """Trích xuất thực thể từ một câu và định dạng JSON."""
    messages = [
        {"role": "system", "content":
         "Bạn là một mô hình AI chuyên trích xuất thực thể. Trả về kết quả theo định dạng JSON
với cấu trúc:\n"
         "{\n"
         "  \"sentence\": \"<câu đầu vào giữ nguyên cấu trúc>\",\n"
         "  \"entities\": {\n"
         "    {\"<nhãn>\": [\"<thực thể>\"]},\n"
         "    ...\n"
         "  }\n"
         "}\n"
         "Chỉ sử dụng các nhãn sau: \"technology\", \"Datetime\", \"Organization\", \"Person
\". Không thêm bất kỳ văn bản nào ngoài JSON."},
        {"role": "user", "content": sentence}
    ]
```

Hình 2.3 Sử dụng Model tự định nghĩa để gắn nhãn theo ý muốn

{'sentence': 'Rất khó làm chủ về độ chính xác và đồng bộ của các phần tử khi phát và thu, ông Trần Hoàng Việt, trưởng phòng Công nghệ ăng-ten mảng pha chủ động của VHT, cho biết.', 'entities': {'Person': ['Trần Hoàng Việt'], 'Organization': ['VHT']}}
{'sentence': 'Các chuyên gia của Viettel cho biết chọn nghiên cứu công nghệ ăng-ten mảng pha quét búp sóng điện tử (beamforming), tương tự công nghệ được ứng dụng trong phát sóng 5G.', 'entities': {'Organization': ['Viettel'], 'Technology': ['beamforming', '5G']}}
{'sentence': 'Công nghệ này sử dụng các phần tử ăng-ten khác nhau hoạt động đồng bộ để điều chỉnh hướng phát và nhận sóng của radar.', 'entities': {'technology': ['phần tử ăng-ten', 'radar']}}
{'sentence': 'Cách này tạo ra độ phân giải cao hơn so với việc phát tín hiệu ra tất cả hướng của radar truyền thống.', 'entities': {'technology': ['radar']}}
{'sentence': 'Các sản phẩm được trưng bày tại Triển lãm Quốc phòng ở Hà Nội ngày 19-22/12, thu hút sự chú ý lớn từ khách tham quan, với kích thước khổng lồ, được thiết kế linh hoạt khi đặt cố định dưới đất hoặc gắn trên xe chuyên dụng.', 'entities': {'Organization': ['Triển lãm Quốc phòng'], 'Datetime': ['19-22/12'], 'Location': ['Hà Nội']}}
{'sentence': 'Điểm độc đáo của sản phẩm là khả năng cung cấp thông tin ba tọa độ, còn gọi là 3D, và do kỹ sư Việt Nam thực hiện.', 'entities': {'Person': ['kỹ sư Việt Nam'], 'technology': ['3D']}}
{'sentence': 'Khác với radar 2D, radar 3D có thể xác định thêm độ cao, có tốc độ quét nhanh và hiển thị vị trí chính xác của mục tiêu trong không gian.', 'entities': {'technology': ['radar 2D', 'radar 3D'], 'Organization': []}}

Hình 2.4 Kết quả sau khi xử lí nhãn

```
print(training_data[:512])
```

```
Rất O
khó O
làm O
chủ O
về O
độ O
chính O
xác O
và O
đồng O
bộ O
của O
các O
phần O
tử O
khi O
phát O
và O
thu O
```

Hình 2.5 Kết quả test thử cho 1 câu ngẫu nhiên

Tách dữ liệu kiểm tra (test)

Trích xuất 10% dữ liệu để test

```
import random

def split_data(data, test_size=0.1, seed=42):
    random.seed(seed)
    random.shuffle(data)
    split_idx = int(len(data) * (1 - test_size))
    train_data = data[:split_idx]
    test_data = data[split_idx:]
    return train_data, test_data

train_data, test_data = split_data(data, test_size=0.1)

print(f"Số câu trong tập training: {len(train_data)}")
print(f"Số câu trong tập test: {len(test_data)}")
```

```
Số câu trong tập training: 630
Số câu trong tập test: 70
```

Hình 2.6 Tách dữ liệu để train và test

## 2.5 Huấn luyện và đánh giá bài toán

### *2.5.1 Chuẩn bị dữ liệu*

In [18]:

```python
# Đọc dữ liệu từ file
def load_data(file_path):
    sentences, labels = [], []
    with open(file_path, "r", encoding="utf-8") as f:
        sentence, label = [], []
        for line in f:
            line = line.strip()
            if not line:
                if sentence:
                    sentences.append(" ".join(sentence))
                    labels.append(label)
                    sentence, label = [], []
            else:
                word, tag = line.split()
                sentence.append(word)
                label.append(tag)
        if sentence:
            sentences.append(" ".join(sentence))
            labels.append(label)
    return sentences, labels

file_path = "/kaggle/input/phobertdata/training_data.txt"
sentences, labels = load_data(file_path)
print(f"Loaded {len(sentences)} sentences.")
```

```
Loaded 700 sentences.
```

Hình 2.7 Đọc dữ liệu

```python
# Mapping nhãn
label_map = {
    "O": 0,
    "B-Person": 1, "I-Person": 2,
    "B-Organization": 3, "I-Organization": 4,
    "B-Technology": 5, "I-Technology": 6,
    "B-Datetime": 7, "I-Datetime": 8,
}
label_map_rev = {v: k for k, v in label_map.items()}
```

Hình 2.8 Mapping nhãn

### *2.5.2 Sử dụng mô hình PhoBERT*

```
model = AutoModelForTokenClassification.from_pretrained(model_name, num_labels=len(label_ma
p)).to(device)
optimizer = AdamW(model.parameters(), lr=5e-5)
num_training_steps = len(train_loader) * 20
lr_scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0, num_training_s
teps=num_training_steps)
```

Hình 2.9 Ứng dụng mô hình PhoBERT

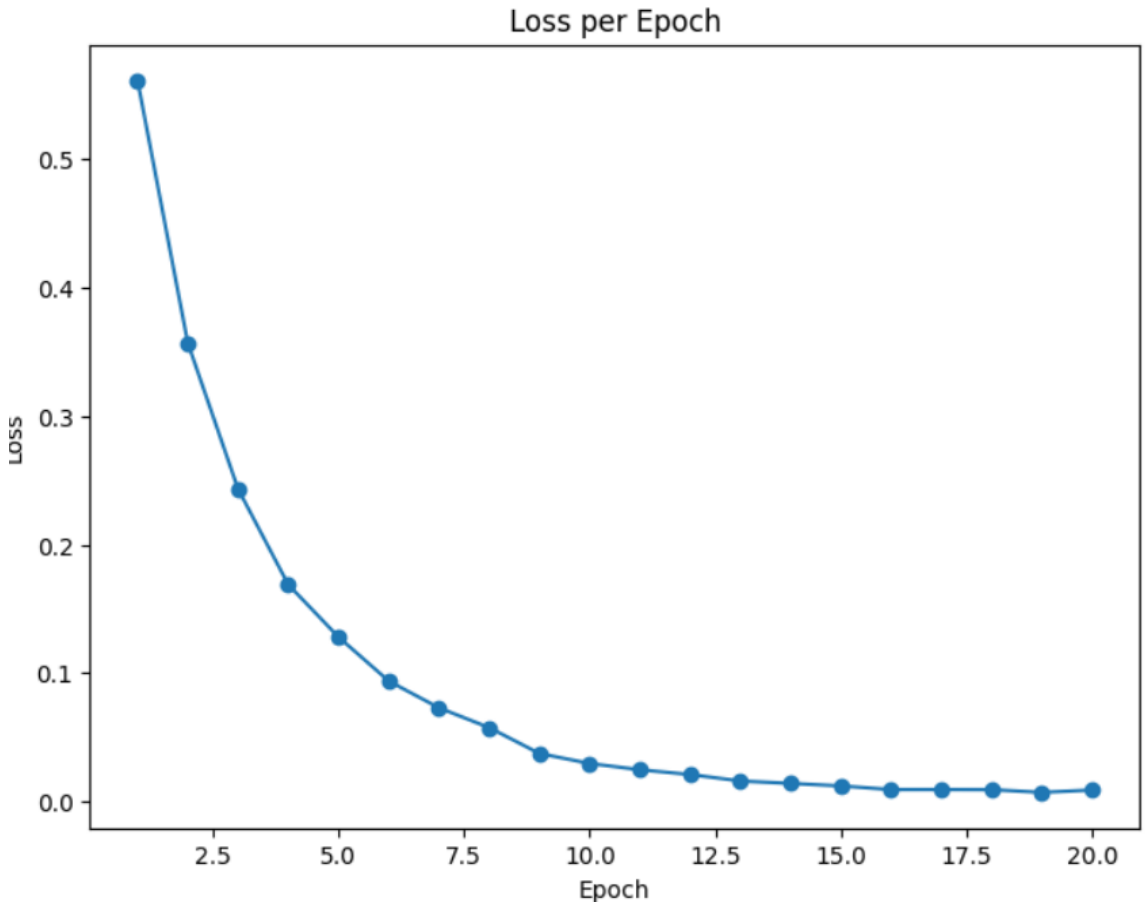
```
Epoch 1/20

/tmp/ipykernel_23/658509426.py:51: UserWarning: To copy construct from a tensor, it is rec
ommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_gr
ad_(True), rather than torch.tensor(sourceTensor).
  "labels": torch.tensor(self.labels[idx]),

Loss: 0.5615
Epoch 2/20
Loss: 0.3571
Epoch 3/20
Loss: 0.2434
Epoch 4/20
Loss: 0.1688
Epoch 5/20
Loss: 0.1280
Epoch 6/20
Loss: 0.0937
Epoch 7/20
Loss: 0.0729
```

Hình 2.10 Kết quả train

### *2.5.3 Test trên câu mẫu*




Hình 2.11 Biểu đồ thể hiện sự thay đổi loss qua các Epoch

### *2.5.4 Đánh giá trên bộ Test*

```
def evaluate_model(data_loader, model):
    model.eval()
    y_true, y_pred = [], []
    with torch.no_grad():
        for batch in data_loader:
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["labels"].to(device)

            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask,
            )
            logits = outputs.logits
            predictions = torch.argmax(logits, dim=-1)

            for label, prediction, mask in zip(labels, predictions, attention_mask):
                active_labels = label[mask == 1].tolist()
                active_predictions = prediction[mask == 1].tolist()
                y_true.extend(active_labels)
                y_pred.extend(active_predictions)
    print("Classification Report:")
    print(classification_report(y_true, y_pred, labels=list(label_map.values()), target_names=lis
t(label_map.keys())))
```

Hình 2.12 Đánh giá trên bộ Test

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| O | 0.90 | 1.00 | 0.95 | 19417 |
| B-Person | 1.00 | 0.99 | 1.00 | 223 |
| I-Person | 0.64 | 1.00 | 0.78 | 200 |
| B-Organization | 0.99 | 1.00 | 0.99 | 573 |
| I-Organization | 0.63 | 1.00 | 0.77 | 438 |
| B-Technology | 0.99 | 1.00 | 0.99 | 373 |
| I-Technology | 0.57 | 1.00 | 0.73 | 312 |
| B-Datetime | 0.97 | 1.00 | 0.99 | 68 |
| I-Datetime | 0.79 | 1.00 | 0.88 | 33 |
| micro avg | 0.89 | 1.00 | 0.94 | 21637 |
| macro avg | 0.83 | 1.00 | 0.90 | 21637 |
| weighted avg | 0.90 | 1.00 | 0.94 | 21637 |

Hình 2.13 Classification Report

```
test_sentence = "Rất khó làm chủ về độ chính xác và đồng bộ của các phần tử khi phát và thu, lượn
g dữ liệu cần xử lý cũng tăng gấp nhiều lần, ông Trần Hoàng Việt, trưởng phòng Công nghệ ăng-ten
mảng pha chủ động của VHT, cho biết."
results = extract_entities(test_sentence, model, tokenizer, label_map_rev)

print("Kết quả nhận dạng thực thể:")
for entity_type, entities in results.items():
    print(f"{entity_type.lower()}: {', '.join(entities)}")
```

```
Kết quả nhận dạng thực thể:
person: Trần Hoàng Việt@@
organization: VHT@@
```

Hình 2.14 Test 1 số câu

```
test_sentence = "Trong đó, Lumiere có khả năng tạo video được cho là tốt hơn Sora của OpenAI tron
g giai đoạn thử nghiệm cũng như phiên bản Gemini mới với khả năng lý luận."
results = extract_entities(test_sentence, model, tokenizer, label_map_rev)

print("Kết quả nhận dạng thực thể:")
for entity_type, entities in results.items():
    print(f"{entity_type.lower()}: {', '.join(entities)}")
```

```
Kết quả nhận dạng thực thể:
technology: Lumiere, Sora, Gemini
```

Hình 2.15 Test 1 số câu

# TÀI LIỆU THAM KHẢO

Tiếng Việt

[1]. D2L AI Vietnam. (n.d.). Kiến trúc Transformer. Truy cập 12/12/2024, từ https://d2l.aivivn.com/chapter_attention-mechanisms/transformer_vn.html

Tiếng Anh

[2]. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. Retrieved December 12, 2024, from https://arxiv.org/abs/1706.03762

[3]. Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. Retrieved December 12, 2024, from https://arxiv.org/abs/1409.0473

[4]. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pretraining. Retrieved December 12, 2024, from https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf