BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

TRẦN QUỐC TUẤN **19133064**

NGUYỄN LÂM SƠN **19133050**

Đề tài:

TÌM HIỂU KIẾN TRÚC TRANSFORMER VÀ ỨNG DỤNG CHO BÀI TOÁN TRẢ LỜI CÂU HỎI

KHÓA LUẬN TỐT NGHIỆP

GIÁO VIÊN HƯỚNG DẪN

Ths. QUÁCH ĐÌNH HOÀNG

KHÓA 2019 - 2023

LỜI CAM ĐOAN

Đồ án được chúng tôi thực hiện với mục đích học tập và nghiên cứu để nâng cao kiến thức và trình độ chuyên môn nên chúng tôi đã làm luận văn này một cách nghiêm túc và hoàn toàn trung thực.

Trong luận văn, chúng tôi chỉ sử dụng một số tài liệu tham khảo của một số tác giả. Chúng tôi đã chú thích và nêu ra trong phần tài liệu tham khảo ở cuối phần luận văn.

Chúng tôi xin cam đoan và chịu trách nhiệm về nội dung và sự trung thực trong luận văn tốt nghiệp của mình.

LỜI CẢM ƠN

Chúng tôi xin gửi lời cảm ơn đến các quý thầy cô khoa công nghệ thông tin, Trường đại học Sư phạm kỹ thuật thành phố Hồ Chí Minh (HCMC University of Technology and Education) đã tạo cơ hội cho chúng tôi được học tập, rèn luyện và tích lũy kiến thức, kỹ năng để thực hiện khóa luận.

Đặc biệt, chúng tôi xin được gửi lời cảm ơn đến giảng viên hướng dẫn thầy Quách Đình Hoàng đã tận tình chỉ dẫn, theo dõi và đưa ra những lời khuyên bổ ích giúp chúng tôi giải quyết được các vấn đề gặp phải trong quá trình nghiên cứu và hoàn thành đề tài một cách tốt nhất.

Tuy nhiên, vì kiến thức của bản thân còn hạn chế và thiếu kinh nghiệm thực tiễn nên nội dung khóa luận khó tránh khỏi có vài thiếu sót, chúng tôi rất mong nhận được sự góp ý, chỉ bảo thêm của quý thầy cô.

Chúng tôi xin chân thành cảm ơn!

ĐH SƯ PHẠM KỸ THUẬT TP.HCM
KHOA CNTT
****

XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh Phúc
****

PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Họ và tên Sinh viên 1 : Trần Quốc Tuấn MSSV 1: 19133064
Họ và tên Sinh viên 2 : Nguyên Lâm Sơn MSSV 2: 19133050
Ngành: Kỹ thuật dữ liệu
Tên đề tài: Tìm hiểu kiến trúc Transformer và ứng dụng cho bài toán trả lời câu hỏi

Họ và tên Giáo viên hướng dẫn: Quách Đình Hoàng

NHẬN XÉT

1. Về nội dung đề tài & khối lượng thực hiện:

2. Ưu điểm:

3. Khuyết điểm

4. Đề nghị cho bảo vệ hay không ?
5. Đánh giá loại :
6. Điểm :

Tp. *Hồ Chí Minh, ngày tháng năm 2023*
Giáo viên hướng dẫn
(Ký & ghi rõ họ tên)

ĐH SƯ PHẠM KỸ THUẬT TP.HCM
KHOA CNTT
****

XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh Phúc
****

PHIẾU NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

Họ và tên Sinh viên 1 : Trần Quốc Tuấn MSSV 1: 19133064
Họ và tên Sinh viên 2 : Nguyên Lâm Sơn MSSV 2: 19133050
Ngành: Kỹ thuật dữ liệu
Tên đề tài: Tìm hiểu kiến trúc Transformer và ứng dụng cho bài toán trả lời câu hỏi

Họ và tên Giáo viên phản biện: Nguyễn Thiên Bảo

NHẬN XÉT

1. Về nội dung đề tài & khối lượng thực hiện:

2. Ưu điểm:

3. Khuyết điểm

4. Đề nghị cho bảo vệ hay không ?
5. Đánh giá loại :
6. Điểm :

Tp. *Hồ Chí Minh, ngày tháng năm 2023*
Giáo viên phản biện
(Ký & ghi rõ họ tên)

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC TỪ VIẾT TẮT

Ký hiệu	Từ gốc
NLP	**N**atural **L**anguage **P**rocessing
RNN	**R**ecurrent **N**eural **N**etwork
LSTM	**L**ong **S**hort-**T**erm Memory
GRU	**G**ated **R**ecurrent **U**nit
BERT	**B**idirectional **Encoder R**epresentations from **Transformer**
GPT	**G**enerative **P**re-trained **Transformer**
LSA	**L**atent **S**emantic **A**nalysis
AI	**A**rtificial **I**ntelligence
seq2seq	**s**equence **t**o **s**equence
CV	**C**omputer **V**ersion

LỜI MỞ ĐẦU

Trước kia, khi kiến trúc Transformer chưa có thì việc xử lý các ngôn ngữ tự nhiên và dịch máy được thực hiện theo từ hay cụm từ. Tức là, nó dựa vào hàng triệu từ hay cụm từ đã được mã hóa để đối chiếu, so sánh và chọn cụm từ sát nhất bằng phương pháp thống kê để đưa vào kết quả. Hiện nay, việc xử lý các ngôn ngữ tự nhiên và dịch máy có thể thực hiện cả câu, rồi dùng ngữ cảnh để tiến hành dự đoán xem từ đó có trong ngữ cảnh không từ đó đưa ra kết quả chính xác nhất.

Trước khi deep learning được sử dụng rộng rãi, việc xử lý các ngôn tự nhiên đều dựa vào các mô hình RNN, LSTM, GRU. Các mô hình này được dùng trong mô hình hóa ngôn ngữ và dịch máy do khắc phục được những hạn chế của vấn đề phụ thuộc xa (long-term dependency) trong mạng nơron truyền thống. Tuy nhiên, trong một số bài toán việc cải thiện cũng không đánh kể, ở giai đoạn phát triển tiếp theo, kỹ thuật Attention, được giới thiệu lần đầu năm 2014 bởi nhóm nghiên cứu của GS Joshua Bengio, đã được sử dụng và mang lại hiệu quả cao hơn. Cách tiếp cận Sequence-To-Sequence with Attention là một trong những mô hình đầu tiên áp dụng kỹ thuật Attention kết hợp với LSTM. các mô hình được đào tạo trước (pre-trained models) như BERT và GPT.

Hiện nay, kiến trúc Transformer được ứng dụng trong rất nhiều lĩnh vực như chatbot (mà ChatGPT là một ví dụ rất nổi bật gần đây), dịch ngôn ngữ (translation), hoặc trả lời câu hỏi (question answering).

Trong phạm vi khóa luận, chúng tôi tập trung tìm hiểu kiến trúc Transformer và ứng dụng nó vào bài toán trả lời câu hỏi (question answering).

PHẦN 1: MỞ ĐẦU

CHƯƠNG 1: TỔNG QUAN

1.1. Đặt vấn đề

Nhân loại ngày càng phát triển, mọi công việc của chúng ta đều đòi hỏi tốc độ và độ chính xác ngày càng cao. Cùng với sự phát triển đó, việc tìm được đáp án chính xác và nhanh chóng cho mỗi câu hỏi là một nhu cầu quan trọng và cấp thiết. Đây cũng chính là những mục tiêu của bài toán trả lời câu hỏi (Question Answering - QA). Trong lĩnh vực xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP), nó là một bài toán quan trọng.

Trước kia, các phương pháp trả lời câu hỏi dựa trên các kỹ thuật tìm kiếm theo từ khóa hoặc hoặc các mô hình xử lý ngôn ngữ truyền thống như N-gram. Tuy nhiên, những phương pháp này không đủ chính xác khi xử lý những câu hỏi phức tạp hoặc yêu cầu các kiến thức đặc biệt.

Trong những năm gần đây, các công ty về lĩnh vực AI đã phát triển ra các mô hình hiện đại hơn cho bài toán trả lời câu hỏi. Trong đó, có rất nhiều mô hình dựa trên kiến trúc Transformer đã được phát triển và trở thành công cụ hữu ích. Kiến trúc Transformer giúp các mô hình có khả năng xử lý dữ liệu dạng chuỗi với độ dài khác nhau và hiểu được ngữ cảnh của các từ trong câu dựa vào cơ chế Attention. Những ứng dụng của kiến trúc Transformer có thể xây dựng các mô hình học sâu cho bài toán trả lời câu hỏi với độ chính xác cao hơn và khả năng xử lý các câu hỏi phức tạp tốt hơn.

1.2. Mục tiêu nghiên cứu

- Về lý thuyết, chúng tôi muốn tìm hiểu các thành phần chi tiết của kiến trúc Transformer và cách hoạt động của nó. Kiến trúc này gồm hai thành phần chính là Encoder và Decoder.
- Về ứng dụng, chúng tôi sẽ áp dụng các mô hình dựa trên kiến trúc Transformer kết hợp với kỹ thuật Fine-Tuning cho bài toán trả lời câu hỏi dùng tập dữ liệu tiếng Việt để xây dựng một ứng dụng trả lời câu hỏi bằng tiếng Việt.

1.3. Phạm vi nghiên cứu

- Kiến trúc Transformer và các khái niệm liên quan.
- Tập dữ liệu Tiếng Việt liên quan đến bài toán trả lời câu hỏi mà chúng tôi có thể thu thập được.

1.4.Tầm quan trọng của đồ án

Đây là một kiến trúc hiện đại và quan trọng trong để giải quyết bài toán trả lời câu hỏi tự động, đặc biệt là trong lĩnh vực xử lý ngôn ngữ tự nhiên. Để có thể sử dụng các mô hình dựa trên kiến trúc Transformer hiệu quả, chúng ta cần phải hiểu rõ được các thành phần của nó và phải tinh chỉnh (fine-tune) mô hình trên tập dữ liệu mà ta làm

việc. Ngoài ra, ở nước ta có rất ít trường nghiên cứu bài toán trả lời câu hỏi áp dụng cho bộ từ ngữ tiếng Việt. Đây cũng là một động lực để chúng tôi thực hiện nghiên cứu này. Việc sử dụng kiến trúc Transformer và phương pháp fine-tuning giúp tăng độ chính xác của các mô hình trả lời câu hỏi, từ đó giúp giải quyết các vấn đề thực tiễn như tăng hiệu quả và độ chính xác của các hệ thống trả lời câu hỏi tự động trong các ứng dụng thương mại điện tử và cải thiện trải nghiệm người dùng khi tương tác với các hệ thống AI.

1.5. Phương pháp nghiên cứu

Chúng tôi tiến hành nghiên cứu dựa bao gồm hai thành phần chính:

- Nghiên cứu dựa trên lý thuyết: chúng tôi sẽ tiến hành nghiên cứu các tài liệu liên quan về Transformer và các ứng dụng của nó trong bài toán trả lời câu hỏi (bao gồm các tài liệu từ các bài báo khoa học, các tài liệu tham khảo từ các ứng dụng thực tế, và các khóa học trực tuyến về chủ đề này). Kết quả của quá trình này là một tài liệu lý thuyết về kiến trúc Transformer và các ứng dụng của nó.
- Thực nghiệm: Sau khi chúng tôi đã xây dựng được cơ sở lý thuyết, chúng tôi sẽ tiến hành thực nghiệm trên một tập dữ liệu để đánh giá hiệu quả của Transformer trong bài toán trả lời câu hỏi. Kết quả của mô hình sẽ được đánh giá bằng các tiêu chí đánh giá thông qua các tiêu chí trong bài toán trả lời câu hỏi là độ chính xác của mô hình.

PHẦN 2: NỘI DUNG

CHƯƠNG 2: KIẾN TRÚC TRANSFORMER

2.1. Transformer

2.1.1. Giới thiệu

Trước khi có mô hình Transformer, hầu hết các hệ thống NLP tiên tiến đều dựa vào RNN, như LSTM và gate recurrent units (GRUs).

Sự ra đời của cơ chế Attention vào năm 2017 đã tạo ra một kiến trúc mới dành cho các bài toán NLP mà không cần có sự xuất hiện của mạng nơ-ron hồi tiếp (RNN, LSTM,…) hay là mạng nơ-ron tích chập (CNN) đó là Transformer.

2.1.2. Transformer là gì?

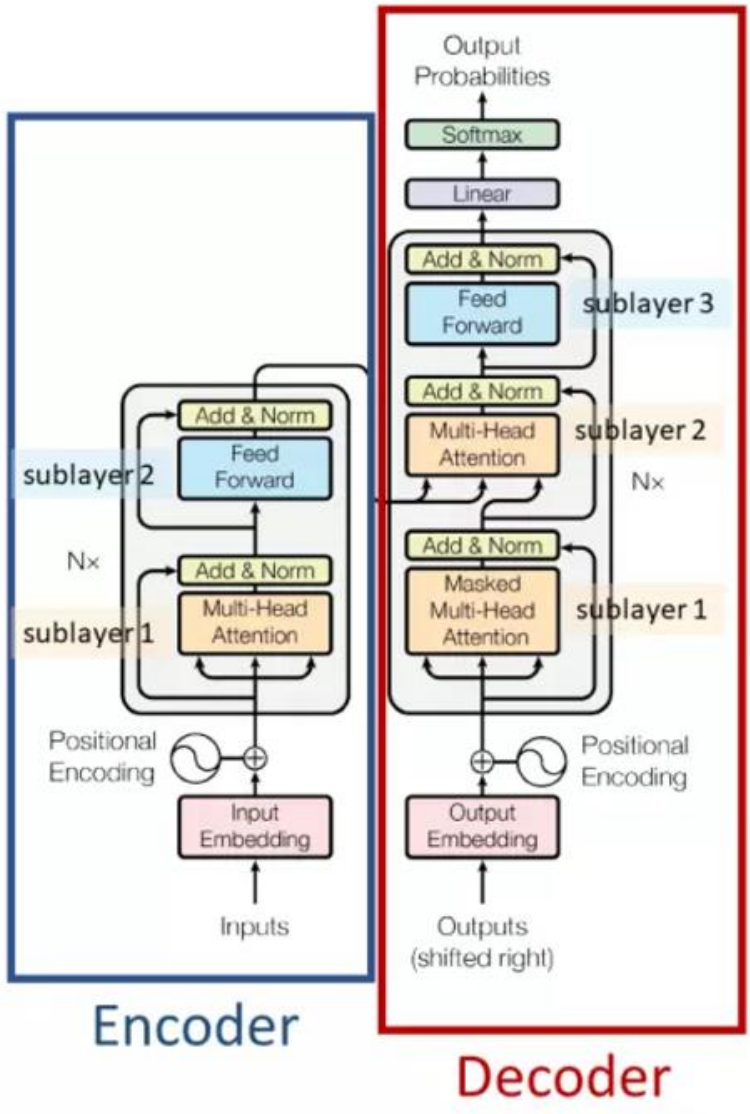
Transformer là một mô hình Deep Learning được giới thiệu vào năm 2017 thường được dùng chủ yếu ở các lĩnh vực xử lý các ngôn ngữ tự nhiên (NLP) và thị giác máy tính (CV). Ngoài ra, mô hình còn hỗ trợ lĩnh vực âm thanh nó có thể giúp nhận dạng giọng nói và phân loại từng loại âm thanh hoặc ở lĩnh vực đa phương thức Transformer còn hỗ trợ trả lời câu hỏi trên bảng, nhận dạng ký tự quang học, trích xuất thông tin từ tài liệu được quét và phân loại video và trả lời câu hỏi bằng hình ảnh.

Transformer cũng xử lý dữ liệu tuần tự giống như mạng nơron hồi quy (Recurrent Neural Network - RNN). Tuy nhiên khác với RNN, nó không yêu cầu dữ liệu tuần tự phải được xử lý một cách thứ tự.

Ví dụ: Chúng ta có một câu ngôn ngữ tự nhiên thay vì như trước đây ta phải xử lý đầu câu rồi mới đến cuối câu. Nhưng Transformer không yêu cầu điều đó nó cả thể xử lý cả câu văn nên Transformer cho phép thực hiện nhiều phép tính song song điều này giúp giảm thời gian huấn luyện mô hình một cách đáng kể.

2.1.3. Kiến trúc Transformer

Transformer gồm hai thành phần chính là Encoder và Decoder như trong hình 1, cả hai thành phần này đều có cấu tạo tương tự nhau gồm nhiều Layers, mỗi Layer được thiết kế với nhiều Sub-Layer. Sub-Layer đầu tiên là Multi-Head Attention Layer, Sub-Layer tiếp theo là Feed Forward Neral Network Layer.



Hình 1: Mô hình kiến trúc Transformer [2]

2.2. Các thành phần

2.2.1. Input Embedding

Tất cả các thuật toán học sâu đều yêu cầu đầu vào là dữ liệu dạng số. Tuy nhiên, trong ngôn ngữ tự nhiên, các từ không phải là các giá trị số. Do đó, chúng ta cần phải chuyển đổi các từ sang dạng số để có thể đưa chúng vào mô hình học sâu.

Một trong những cách để chuyển đổi từ sang số là sử dụng một lớp nhúng từ (Word Embedding Layer). Lớp này hoạt động như một bảng tra cứu để lấy một biểu diễn vectơ đã học của mỗi từ. Các mạng nơ-ron học thông qua các số nên mỗi từ sẽ được ánh xạ sang một vector với các giá trị liên tục để biểu diễn từ đó.

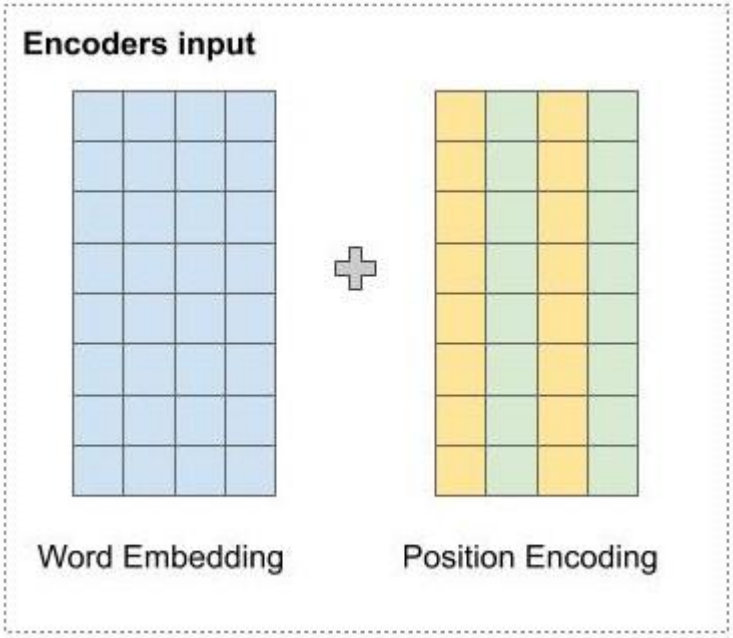
Ví dụ, các từ "học sinh" và "đi học" có thể được ánh xạ sang các vector khác nhau nhưng vẫn giữ nguyên các mối quan hệ giữa chúng. Điều này cho phép mô hình học được các đặc trưng của từ và sử dụng chúng để thực hiện các tác vụ như phân loại văn

bản hay dịch máy. Các con số sẽ dùng để ánh xạ các từ tới một vector có các giá trị liên tục để biểu thị từ đó.

2.2.2. Positional Encoding

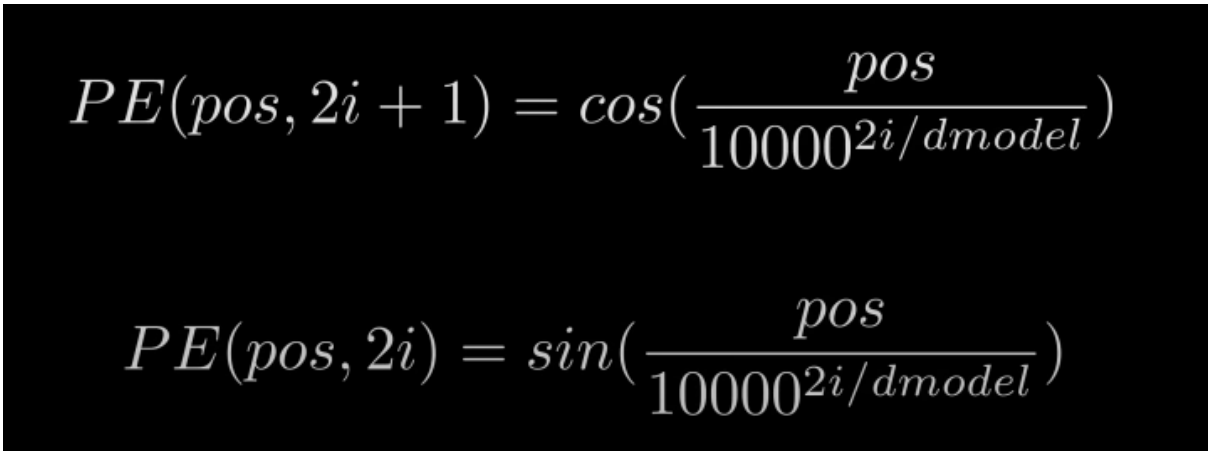
Trước khi đi vào Encoder, thì chúng tôi xin giới thiệu một cơ chế rất thú vị là Positional Encoding mang thông tin về vị trí các từ để truyền vào bên trong kiến trúc Transformer. Trước tiên, các từ được biểu diễn bằng một vector sử dụng một ma trận Word Embedding có số dòng bằng kích thước của tập từ vựng. Tiếp theo các từ trong câu được tìm kiếm trong ma trận này, và được nối với nhau thành các dòng của một ma trận hai chiều chứa ngữ nghĩa của từng từ riêng biệt. Nhưng do Transformer xử lý các từ một cách song song do đó chỉ sử dụng Word Embedding thì mô hình không thể nào biết được vị trí các từ. Như vậy mô hình cần có một cơ chế để đưa thông tin về về vị trí các từ vào trong vector đầu vào. Đó là lúc Positional Encoding xuất hiện và giải quyết vấn đề này.

Vị trí của các từ được mã hóa bằng một ma trận có kích thước bằng với Word Embedding và sau đó được công trực tiếp vào Word Embedding giống phép cộng hai ma trận trong hình 2:



Hình 2: Encoder Input [11]

Positional Encoding sẽ biến đổi vị trí lẻ bằng cách dùng hàm Cos và vị trí chẵn sẽ dùng hàm Sin để tính giá trị tại đó, công thức trong hình 3:



Hình 3: Công thức tính giá trị Sin & Cos [10]

Trong đó:

pos: vị trí của một từ trong câu đầu vào.

dmodel: số chiều của không gian Embedding đầu ra.

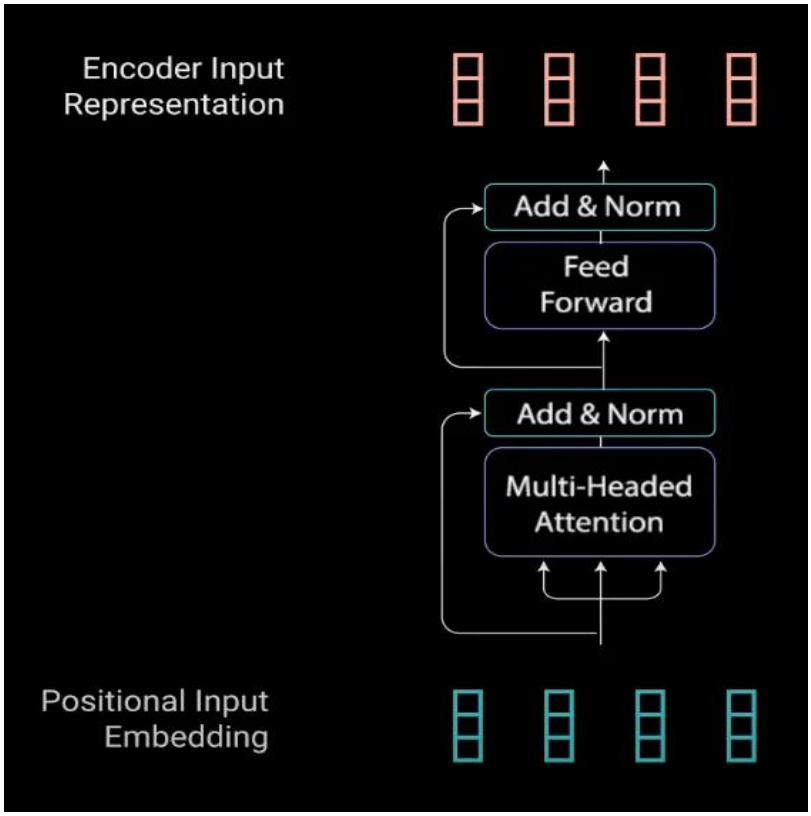
PE(pos, j): hàm vị trí để ánh xạ một vị trí pos trong câu đầu vào để lập chỉ mục (pos, j) của vị trí của ma trận.

i: được sử dụng để ánh xạ tới các chỉ số cột $0 \leq i \leq dmodel/2$ với một giá trị i được ánh xạ cho cả hàm Sin và Cos.

Theo cách này thì mỗi chiều của Positional Encoding sẽ tương ứng với một hình Sin có bước sóng khác nhau ở các chiều khác nhau từ 2π đến 10000.2π.

2.2.3. Encoder

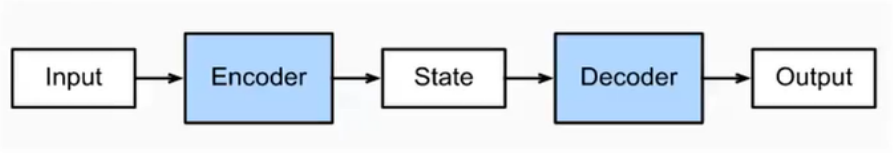
Encoder là ánh xạ tất cả các chuỗi đầu vào thành một biểu diễn liên tục chứa thông tin đã học cho toàn bộ chuỗi đó. Mỗi Encoder sẽ gồm hai thành phần chính là Multi-Headed Attention và Feed Forward Netword, ngoài ra còn có cả Skip Connection và Normalization để trực quan hơn chúng tôi đã dùng hình 4 để biểu diễn cấu tạo của Encoder.

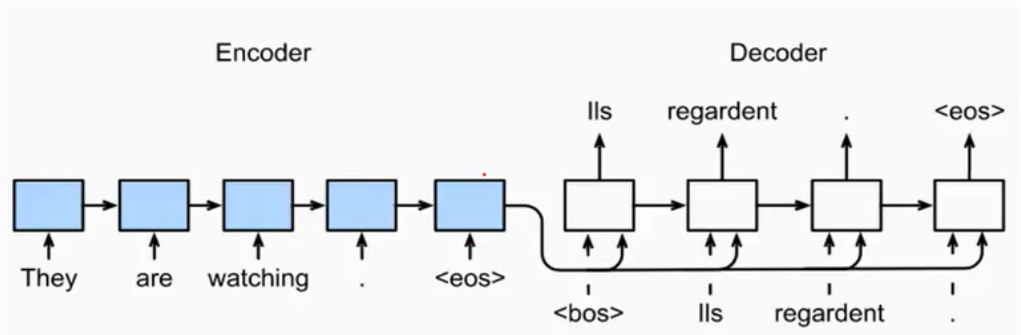


Hình 4: Các thành phần của Encoder trong một module [15].

2.2.4. Decoder

a) Decoder là gì ?





Hình 5: Quá trình xử một câu đối với Encoder và Decoder [21]

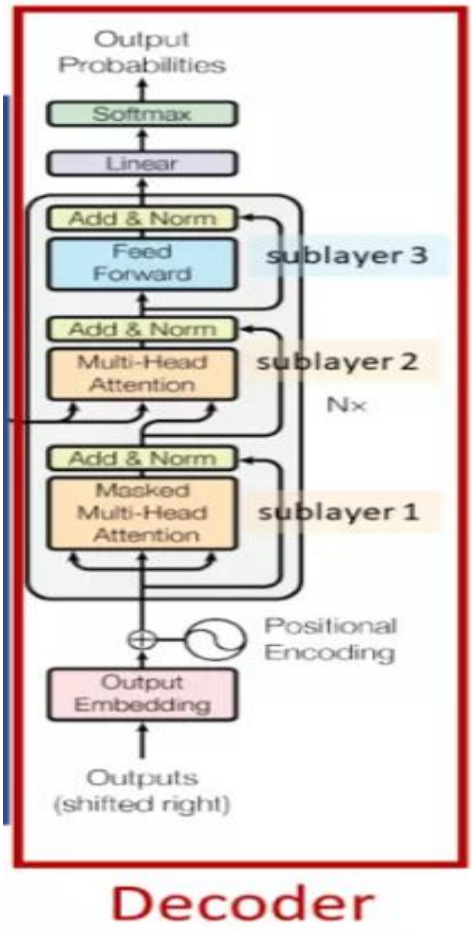
Decoder chính là đầu ra của Encoder giống như ở hình 5. Phrase này nhằm mục đích tìm ra phân phối xác suất từ các features learning ở Encoder từ đó xác định đâu là nhãn đầu ra. Kết quả đầu ra của Decoder tùy thuộc vào đầu vào của Encoder nó có thể là một nhãn đối với mô hình phân loại hoặc một chuỗi theo thứ tự thời gian đối với mô hình Sequence To Sequence (Seq2Seq).

Ví dụ: Trong bài toán chuyển từ một câu tiếng Anh sang tiếng Pháp thì công việc của Decoder là giải mã (translate) thông tin từ Encoder và sinh ra từng từ tiếng Pháp dựa trên những từ trước đó.

b) Quá trình Decoder

Ở hình 6, mỗi Decoder sẽ gồm 3 thành phần chính:

- Self-Attention
- Multi-Head Attention
- Feed-forward Neural Network



Hình 6: Mô hình kiến trúc Transformer [2].

Thành phần Self-Attention ở Decoder giống như ở Encoder nó cho phép mỗi từ trong câu đầu ra chú ý đến các từ khác có trong câu đó và tính toán trọng số của các từ đó để tính toán vector biểu diễn từ đó. Nhiệm vụ của Self-Attention trong Decoder giống với Encoder là giúp cho mô hình học được mối quan hệ giữa các từ trong câu đầu ra giúp đưa ra các từ đúng thứ tự và phù hợp ngữ cảnh.

Thành phần Multi-Head Attention cũng giống như ở Encoder do vậy chúng tôi sẽ trình bày chi tiết ở phần sau.

Sau khi đã hoàn thành xong việc tính toán trọng số của các từ trong các câu đầu vào, vector biểu diễn của từ tiếp theo trong câu đầu ra sẽ được tính toán bằng cách tính

trung bình các trọng số các vector biểu diễn các từ trong câu đầu vào và vector biểu diễn các từ đang dự đoán trước. Khi đã tính trung bình các trọng số các vector xong các từ này sẽ được đưa vào Feed Forward để tiếp tục xử lý.

Feed Forward ở Decoder có nhiệm vụ giống như ở Encoder là xử lý các vector biểu diễn các từ và trích xuất các đặc trưng quan trọng.

Ở Decoder, Transformer có thể học được cách dự đoán từ tiếp theo trong câu đầu ra và dựa trên các đặc trưng hay ngữ cảnh của câu đầu vào và các từ đã xử lý trong câu đầu ra. Một khi mô hình đã học được các trọng số và đặc trưng ngữ cảnh quan trọng của các từ trong các câu, nó sẽ có khả năng dự đoán các từ tiếp theo trong câu đầu ra dựa trên ngữ cảnh và thông tin từ câu đầu vào.

Sau khi các từ trong câu đầu ra được dự đoán, mô hình Transformer sẽ sử dụng Decoder để tạo ra các vector biểu diễn cho toàn bộ câu đầu ra. Vector biểu diễn này được sử dụng để tính toán xác suất cho toàn bộ câu đầu ra và đưa ra dự đoán cuối cùng.

Tóm lại nhiệm vụ của Decoder trong Transformer là tạo ra các chuỗi vector biểu diễn các từ trong câu đầu ra dựa trên vector biểu diễn của câu đầu vào và thông tin từ các lớp Encoder trước đó. Các từ được dự đoán dựa trên các đặc trưng quan trọng của các câu đầu vào và ngữ cảnh trong câu đầu ra đã được dự đoán trước đó. Decoder giúp mô hình học được các mối quan hệ giữa các từ trong câu đầu ra và đưa ra dự đoán chính xác trong câu đầu ra.

2.2.5. Thành phần Attention

Trong một vài năm gần đây, khi cơ chế Attention ra đời đã thu hút rất nhiều sự quan tâm, cơ chế mô tả một nhóm mới gồm các lớp trong mạng nơ-ron đặc biệt là trong các tác vụ liên quan đến chuỗi dữ liệu. Cơ chế Attention, mô tả một trung bình có trọng số của các phần tử (trong chuỗi) với các trọng số được tính động dựa trên một truy vấn đầu vào và các khóa của các phần tử. Cơ chế này có mục tiêu là lấy trung bình của các đặc trưng của nhiều phần tử. Tuy nhiên, thay vì gán trọng số như nhau cho mỗi phần tử, cơ chế muốn gán trọng số dựa trên giá trị thực tế của chúng. Nói cách khác đi, cơ chế sẽ dùng các thông tin đầu vào để quyết định xem nên "chú ý" nhiều đến các phần tử nào hơn so với các phần tử khác. Đối với một cơ chế Attention, thường bao gồm ba phần:

- Truy vấn (Query): Truy vấn là một vector đặc trưng mô tả điều chúng ta đang tìm kiếm trong chuỗi, tức là điều chúng ta có thể muốn chú ý đến.
- Khóa (Key): Đối với mỗi phần tử đầu vào, chúng ta có một khóa tương ứng là một vector đặc trưng. Vector đặc trưng này mô tả đại khái những gì phần tử cung cấp hoặc khi nào nó có thể quan trọng. Các khóa nên được thiết kế sao cho chúng ta có thể xác định các phần tử mà chúng ta muốn chú ý dựa trên truy vấn.
- Giá trị (Value): Đối với mỗi phần tử đầu vào, chúng ta cũng có một vector giá trị. Vector đặc trưng này là cái mà chúng ta muốn lấy trung bình.

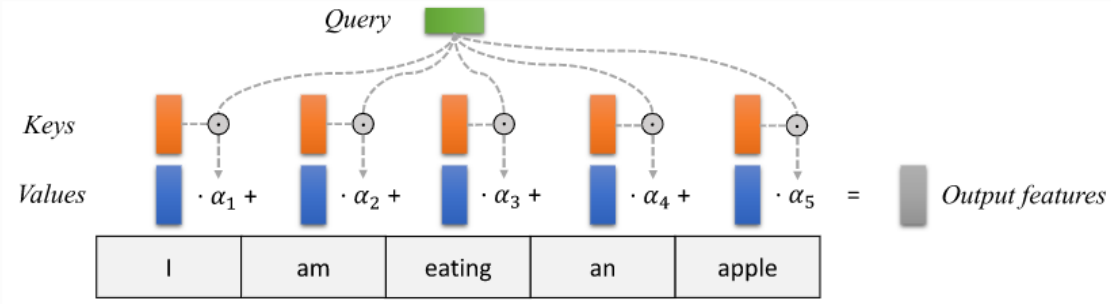
Các trọng số của trung bình được tính bằng hàm Softmax, trên tất cả đầu ra của hàm điểm. Do đó, sẽ cần gán các vector giá trị có trọng số cao hơn cho các vector khóa tương

ứng mà tương đồng nhất với truy vấn. Nếu sử dụng mô tả bằng Pseudo-Math thì có thể dùng công thức trong hình số 7:

$$\alpha_i = \frac{\exp\left(f_{attn}\left(\text{key}_i, \text{query}\right)\right)}{\sum_j \exp\left(f_{attn}\left(\text{key}_j, \text{query}\right)\right)}, \quad \text{out} = \sum_i \alpha_i \cdot \text{value}_i$$

Hình 7: Công thức tính Pseudo-Math [4]

Cơ chế Attention được trực quan hóa trên một câu như hình 8:



Hình 8: Self-Attention *[7]*

Mỗi từ thì sẽ có một vector Key và một vector Query, với Query sẽ được sử dụng để so sánh trọng số của tất cả các Key bằng một hàm (gọi là Scaled Dot Product). Softmax thì không dễ để trực quan, cuối cùng thì các vector Value sẽ được tính trung bình bằng cách sử dụng trọng số của Attention. Hầu hết các cơ chế Attention sẽ khác nhau ở phần điều kiện là chúng sẽ Query những gì, cách mà các vector Key và Value sẽ được xác định và hàm dùng để tính trọng số. Trong kiến trúc Transformer thì có áp dụng Attention và được gọi là Self-Attention, trong đó mỗi phần tử trong chuỗi cung cấp một khóa (Key), giá trị (Value), và truy vấn (Query). Đối với mỗi phần tử, sẽ thực hiện một lớp chú ý (Attention Layer) trong đó, dựa trên truy vấn của nó, kiểm tra sự tương đồng giữa các khóa của tất cả các phần tử trong chuỗi và trả về một vector giá trị khác nhau và được trung bình hóa cho mỗi phần tử. Bây giờ chúng tôi sẽ đi vào một chút chi tiết hơn bằng cách xem xét cơ chế Self-Attention trong trường hợp của Transformer, được gọi là Scaled Dot Product Attention.

2.2.6. Thành phần Scaled Dot Product Attention

Mục đích là có một cơ chế Attention với bất kỳ phần tử trong câu có thể chú ý đến các phần tử khác trong khi đó vẫn hiệu quả để tính toán. Dot Product Attention lấy đầu vào là một tập các:

- Queries: $Q \in R^{T\ X\ d_k}$
- Keys: $K \in R^{T\ X\ d_k}$
- Values: $V \in R^{T\ X\ d_v}$

Trong đó:

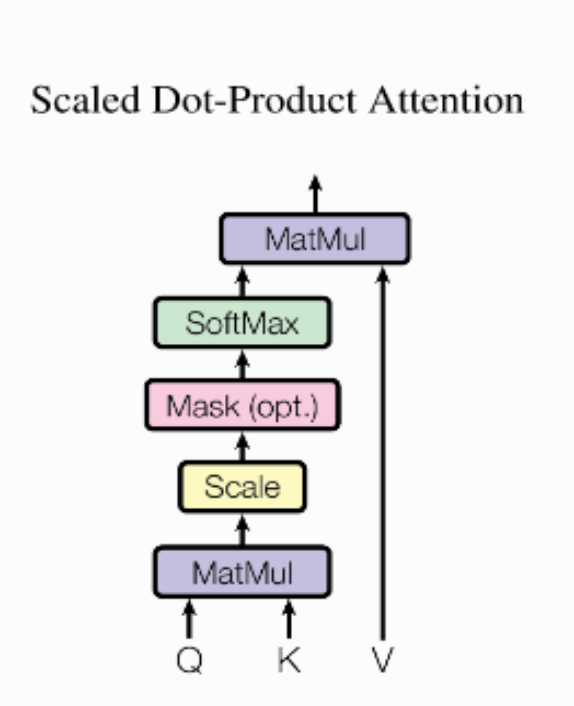
- T là độ dài của câu.
- d_k và d_v : là chiều ẩn cho vector Queries, Keys, Values tương ứng.

Để đơn giản thì sẽ bỏ qua số chiều của batch. Giá trị Attention ở phần tử thứ i tới j nó dựa trên sự giống nhau của Query Q_i và Key K_j. Sử dụng tích vô hướng (Dot Product) làm đo lường độ tương đồng, trong toán học, chúng ta tính toán Scaled Dot Product Attention như hình 9:

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

Hình 9: Công thức tính giá trị Attention [4]

Phép nhân ma trận QK^T thể hiện tích vô hướng cho mỗi cặp Key và Query kết quả thu về là một ma trận T x T chiều, mỗi hàng đại diện cho các giá trị Logits Attention cho một phần tử i cụ thể tới tất cả các phần tử khác trong câu. Trên những giá trị Logits này, chúng ta áp dụng hàm Softmax và nhân với vector giá trị để thu được trung bình có trọng số (các trọng số được xác định bởi Attention). Một góc nhìn khác về cơ chế Attention này đưa ra đồ thị tính toán được mô phỏng ở hình 10 dưới đây.



Hình 10: Scaled Dot-Product Attention [17]

2.2.7. Thành phần Multi-Head Attention

Cơ chế Attention dựa trên Scaled Dot Product cho phép mạng nơ-ron chú ý đến một chuỗi dữ liệu. Tuy nhiên, thường có nhiều khía cạnh khác nhau mà một phần tử trong chuỗi muốn quan tâm đến, và việc lấy trung bình có trọng số

duy nhất không phải là một lựa chọn tốt. Đây là lý do chúng ta mở rộng cơ chế thành Multi-Head Attention, tức là nhiều bộ ba Query-Key-Value khác nhau trên cùng các đặc trưng. Về chi tiết cho trước ma trận truy vấn (Query), ma trận khóa (Key), và ma trận giá trị (Value), chúng ta biến đổi chúng thành các ma trận con truy vấn (Sub-Query), ma trận con khóa (Sub-Key), và ma trận con giá trị (Sub-Value), mà chúng ta đưa qua quá trình Scaled Dot Product Attention độc lập với nhau. Sau đó, chúng ta nối các Heads lại và kết hợp chúng bằng một ma trận trọng số cuối cùng. Có thể biểu diễn phép toán này như 11 bên dưới:

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

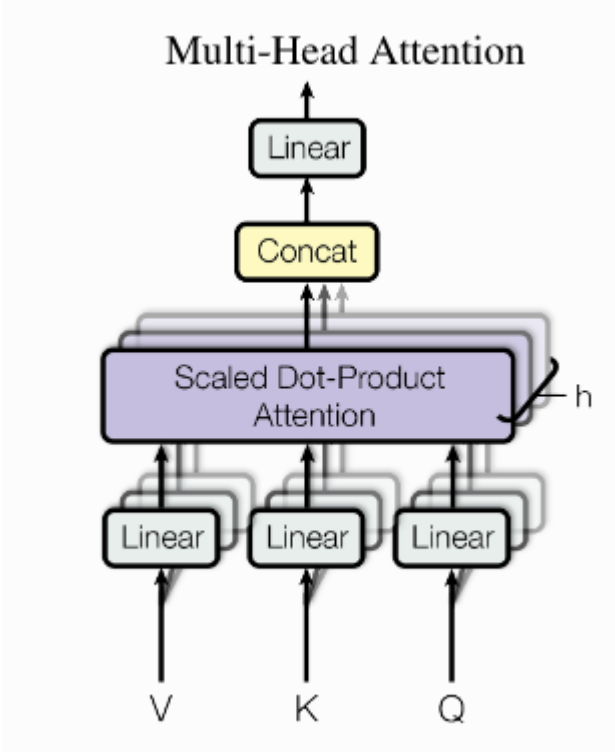
Hình 11: Công thức tính Multi-head Attention [4].

Tầng Multi-Head Attention với các tham số có thể học như biểu thức ở hình 12:

$$W_{1...h}^Q \in \mathbb{R}^{D\times d_k}, W_{1...h}^K \in \mathbb{R}^{D\times d_k}, W_{1...h}^V \in \mathbb{R}^{D\times d_v}, \text{and } W^O \in \mathbb{R}^{h\cdot d_v \times d_{out}}$$

Hình 12: Tầng Multi-Head Attention [*4*]

Trong đó D là số chiều của đầu vào, để quan sát mô hình một cách rõ ràng có thể xem qua hình ảnh một cách trực quan.



Hình 13: Mô hình Multi-Head Attention [17]

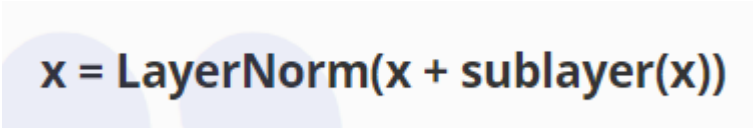
Bằng cách nào chúng ta áp dụng một lớp Multi-Head Attention trong mạng nơ-ron khi chúng ta không có một vector Query, Key và Value tùy ý làm đầu vào? Chúng ta hãy nhìn vào đồ thị tính toán ở trên, một cách triển khai đơn

giản nhưng hiệu quả là đặt bản đồ đặc trưng hiện tại trong một mạng nơ-ron. Như hình 13 ở trên, một cách triển khai đơn giản nhưng hiệu quả là đặt bản đồ đặc trưng hiện tại trong một mạng nơ-ron.

2.2.8. Thành phần Add và Normalize

Add và Normalize (cộng và chuẩn hóa) là một kỹ thuật được sử dụng trong kiến trúc Transformer để giúp mô hinh học được các thông tin có tính khả diễn giải cao và đạt hiệu quả cao hơn trong các tác vụ xử lý ngôn ngữ tự nhiên. Kỹ thuật này chỉ được áp dụng sau khi mô hình đã tính toán đầu ra tại mỗi lớp Attention hoặc Feedforward. Nó bao gồm hai bước.

- Add (cộng): Đầu ra của mô hình được cộng với đầu vào ban đầu của lớp đó. Cụ thể với mỗi vector đầu ra tại mỗi lớp Attention hoặc Feedforward, nó sẽ được cộng với đầu vào ban đầu của lớp đó để tạo ra một vector mới. Công thức tính như hình 14:

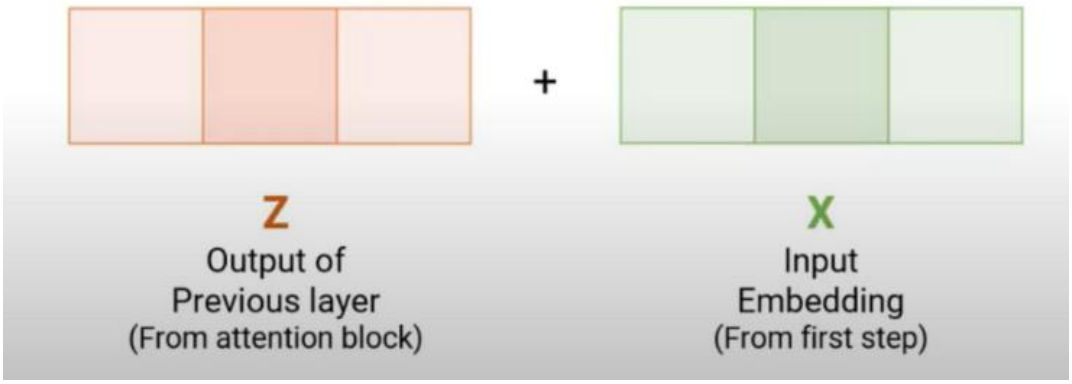


Hình 14: Công thức tính Add [3]

Trong đó:

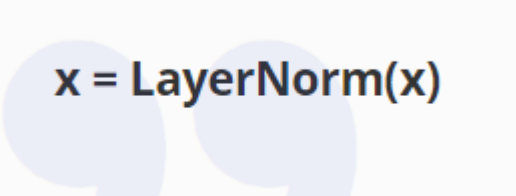
- x là đầu vào ban đầu của lớp đó.
- Sublayer(x) là đầu ra tại mỗi lớp Attention hoặc feedforward.

Hình 15 bên dưới sẽ trình bày trức quan hơn về công thức ở hình 14:



Hình 15: Ví dụ về công thức ở hình 14

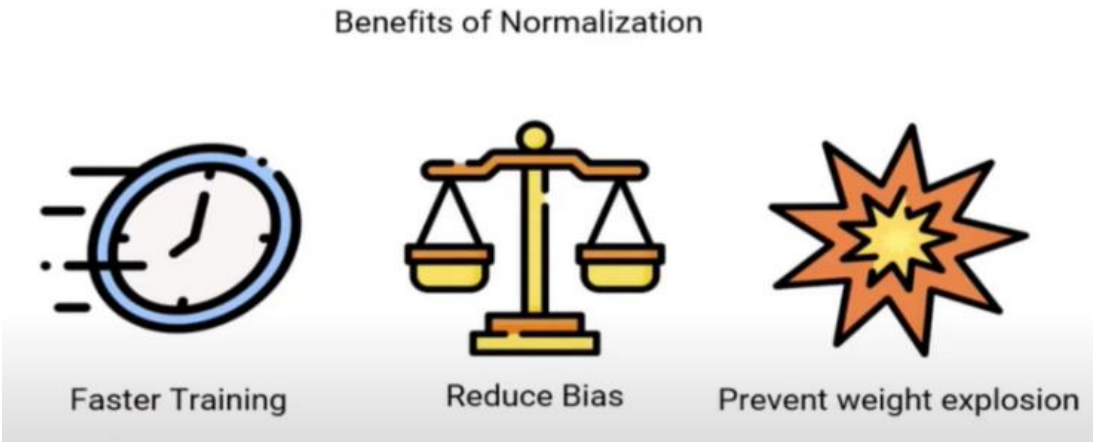
- Normalize (chuẩn hóa): sau khi thực hiện phép cộng, vector mới được chuẩn hóa bằng cách sử dụng lớp chuẩn hóa (Layer Normalization) và có công thức tính như sau:.



Hình 16: Công thức Normalize [3]

Việc chuẩn hóa như vậy giúp cho trên mỗi chiều vector đầu vào, đảm bảo rằng các giá trị trong vector đầu ra có phân bố chuẩn với trung bình bằng không và độ lệch chuẩn bằng một.

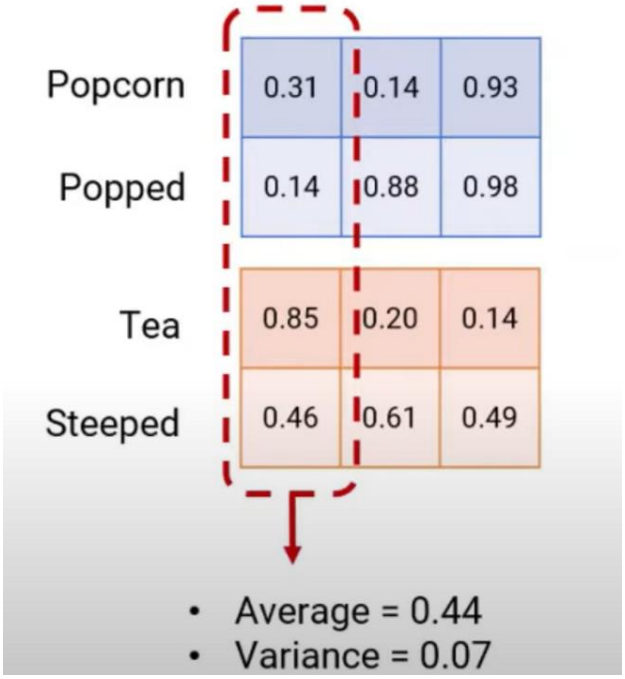
⇒ Khi kết hợp Add và Normalize (cộng và chuẩn hóa) giúp mô hình Transformer học được các thông tin liên quan đến vị trí và thứ tự các từ trong câu. Hiệu quả của việc chuẩn hóa giúp cho quá trình huấn luyện diễn ra nhanh hơn, tốn ít tài nguyên hơn, ngăn chặn weight explosion xảy ra trong quá trình huấn luyện. Đó là lợi ích của Add và Normalize chúng tôi sẽ biểu diễn ngắn gọn lại như hình 17.



Hình 17: Hiệu quả của Add và Norm [18]

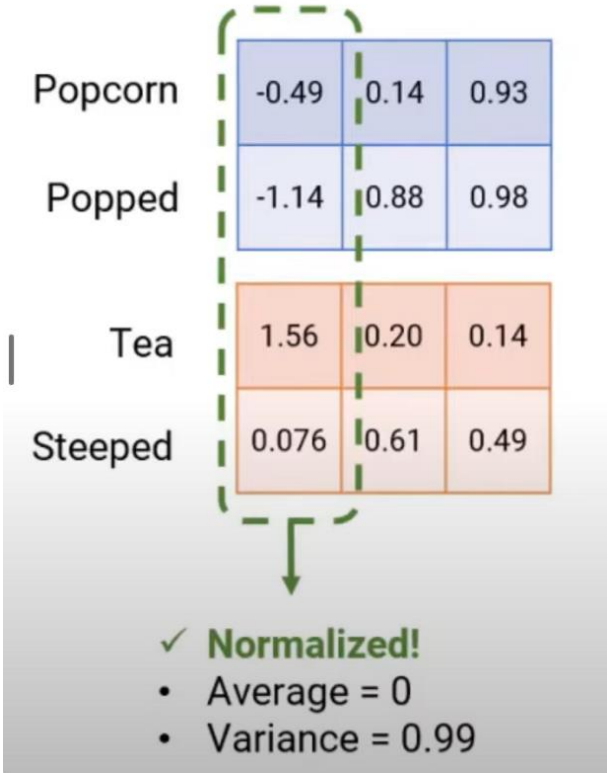
- Với phương pháp Batch Norm thì sẽ làm cho data có trung bình bằng 0 và variance khoảng 1, trong đó các giá trị đầu ra của một batch dữ liệu được chuẩn hóa theo trung bình và độ lệch chuẩn của batch đó.

  Ví dụ:
  Hình 18 chuỗi đầu vào khi chưa được chuẩn hóa:



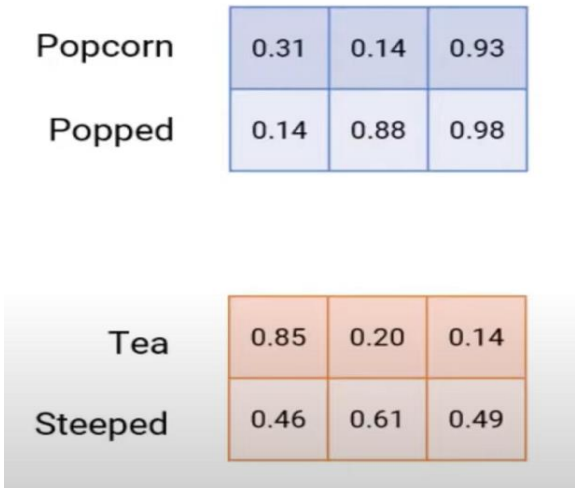
Hình 18: Trước khi chuẩn hóa [18]

Sau khi chuẩn hóa chúng tôi có được trọng số của chuỗi như hình 19:



Hình 19: Sau khi chuẩn hóa [18]

- Ở Lay Norm chúng ta sẽ lấy trung bình và variance từ tất cả các feature của một câu đơn. Ví dụ chúng tôi sẽ lấy trung bình trọng số của các chuỗi như hình 20:



Hình 20: Giá trị trung bình Lay Norm [18]

2.2.9. Thành phần Feed Forward

Feed Forward hay còn gọi là Position-Wise Feed-Forward Network (mạng chuyển tiếp theo vị trí) đây là một thành phần quan trọng khác trong khối Transformer. Thành phần này chấp nhận đầu vào với ba kiểu kích thước là kích thước batch, độ dài chuỗi, kích thước đặc trưng. Feed Forward bao gồm hai tầng dày đặc áp dụng trên chiều cuối cùng của đầu vào.

Cơ chế Feed Forward được sử dụng để tối ưu hóa dữ liệu. Cơ chế này cung cấp một cấu trúc linearly nhất định để mô hình học các quan hệ giữa các phần tử trong dữ liệu liên tục. Chính vì cách hoạt động này nên việc áp dụng nó tương đương được Two Convolutional Layers 1x1.two convolutional layers 1x1.

Chính vì vậy mà lớp Fully-Connected Feed Forward và Self-Attention được chọn để sử dụng trong Transformer mà không sử dụng các mạng như CNN(Convolutional Neural Network) hoặc RNN (Recurrent Neural Network) truyền thống.

2.2.10. Masked Multi-Head Attention

Masked Multi-Head Attention tất nhiên là Multi-Head Attention mà chúng ta đã nói đến ở trên, ở đây thành phần này sẽ giúp cho các từ trong tương lai không được sử dụng trong quá trình dự đoán các từ tiếp theo. Cụ thể nó sẽ được sử dụng để tạo ra các trọng số Attention cho từng từ trong câu đang được giải mã. Những trọng số này dùng để quyết định mức độ quan trọng của các từ trong quá trình dự đoán từ tiếp theo. Đầu vào của Masked Multi-Head Attention là vector Query (Q), vector Key (K) và vector Value (V). Tất cả ba vector này được tính từ bước Decoder trước đó trong quá trình dự đoán từ tiếp theo.

Thành phần này có quá trình gồm các bước sau:

1. Tạo Mask:

Trước khi Attention tình toán, thì mô hình sẽ tạo ra một ma trận vuông mask có kích thước bằng chiều dài của chuỗi đầu vào [L, L] để che giấu các từ

trong tương lai. Ví dụ, nếu ta đang dự đoán từ thứ i, thì các từ từ vị trí i+1 đến L sẽ được che giấu bằng cách gán giá trị không cho các phần tử tương ứng của ma trận mask.

2. Tính Attention Scores:

Sau khi mô hình có được ma trận mask, tiếp theo nó Attention Scores bằng cách thực hiện phép nhân giữa các vector Query (Q) và vector Key (K) theo công thức sau trong hình 21:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^\mathbf{T})}{\sqrt{d_k}}\mathbf{V}$$

Hình 21: Công thức tính Attention Scores [4]

Trong đó, Q là ma trận Query có kích thước (L, d_model), K là ma trận Key có kích thước (seq_L, d_model), và d_k là kích thước của mỗi vector Query/Key. Bước chia tỉ lệ cho d_k giúp kiểm soát độ lớn của Attention scores.

3. Áp dụng Mask:

Sau khi tính Attention scores, ta áp dụng ma trận mask bằng cách nhân phần tử-wise giữa Attention scores và ma trận mask điều này có tác dụng gán giá trị không cho các vị trí tương ứng với các từ trong tương lai, từ đó đảm bảo rằng các từ này không được sử dụng trong quá trình dự đoán từ tiếp theo.

4. Tính Weighted Sum:

Cuối cùng, ta tính trọng số weighted sum của vector Value (V) bằng cách nhân Attention scores với ma trận Value và cộng tổng các giá trị. Kết quả của việc tính toán này là chúng ta sẽ có được một vector weighted sum, chứa thông tin quan trọng từ các từ quan sát được trong quá trình tính toán Attention.

Ví dụ:

Giả sử ta đang giải mã câu "Hôm nay chúng tôi đi chợ" và đang dự đoán từ tiếp theo là "chợ".

CHƯƠNG 3: MÔ HÌNH BERT

3.1. Giới thiệu về mô hình BERT

BERT là viết tắt của cụm từ Bidirectional Encoder Representation from Transformer có nghĩa là mô hình biểu diễn bộ mã hóa từ theo hai chiều ứng dụng kỹ thuật Transformer. Đây là một kỹ thuật học máy dựa trên các Transformer được dùng cho việc huấn luyện trước xử lý ngôn ngữ tự nhiên (NLP) được Google công bố vào đầu tháng 11 năm 2018 do Jacob Devlin và cộng sự từ Google đã tạo ra. BERT thường xử lý các bài toán như xử lý ngôn ngữ tự nhiên, trả lời câu hỏi, suy luận ngôn ngữ,... mà không thay đổi quá nhiều so với kiến trúc cũ. BERT được thiết kế để huấn luyện trước các biểu diễn từ (huấn luyện lại với Word Embedding). Điểm đặc biệt ở BERT đó là nó có thể điều hòa cân bằng bối cảnh theo cả hai chiều trái và phải do đó việc huấn luyện lại mô hình BERT với thêm một tầng đầu ra sẽ giúp tạo ra mô hình hiện đại với các nhiệm vụ như: suy luận ngôn ngữ, trả lời câu hỏi, mà không phải sửa đổi đáng kể đến kiến trúc của mô hình. Mô hình đạt được kết quả tiên tiến nhất trên 11 việc về xử lý ngôn ngữ tự nhiên, bao gồm cả việc đẩy điểm GLUE lên 80,5% (7,7 điểm cải thiện tuyệt đối), Độ chính xác của MultiNLI đến 86,7% (4,6% tuyệt đối cải thiện), trả lời câu hỏi SQuAD v1.1 Mô hình đạt kết quả trên tập F1 lên 93,2 và kết quả với tập test trên bộ dữ liệu SQuAD v2.0 F1 lên 83,1.

Quá trình pre-train BERT được huấn luyện với các liệu không dán nhãn, còn đối với quá trình fine-tuning, mô hình BERT đầu tiên được khởi tạo với tất cả các tham số trong quá trình pre-train và các tham số này được điều chỉnh với với dữ liệu được dán nhãn từ các Downstream Tasks. Mỗi Downstream Task có các mô hình tinh chỉnh riêng biệt, mặc dù chúng được khỏi tạo với cùng các tham số trong quá trình pre-train.
Mô hình BERT được chia thành 2 loại dựa theo kích thước:

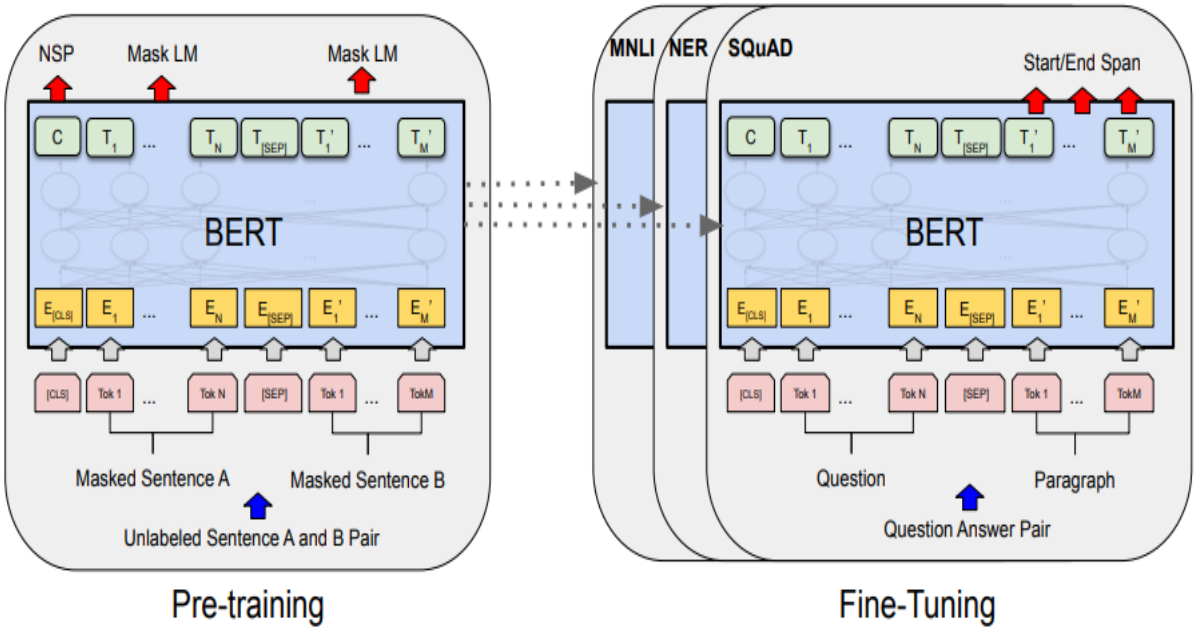
- $BERT_{BASE}$ (L=12, H=768, A=12): tổng tham số 110 triệu
- $BERT_{LARGE}$ (L= 24, H=102, A=16): tổng tham số 340 triệu

Hiện nay có rất nhiều phiên bản khác nhau của BERT đã ra đời, các phiên bản này đều có một điểm chung đó là đều dựa trên việc thay đổi kiến trúc của Transformer tập chung ở ba tham số:

- L: Số lượng các khối Sub Layer của Transformer
- H: kích thước của Embedding vector (hay còn gọi là kích thước ẩn)
- A: Số lượng Head trong tầng Multi-Head, mỗi một Head sẽ chứa nhiều Self-Attention

3.2. Sơ lược về pre-train và fine-tuning BERT

Mô hình BERT được pre-train trên tập Stanford Question Answering Dataset (SQuAD v1.1) với 100 ngàn cặp câu hỏi và đáp. Đầu vào của mô hình sẽ được cho trước một câu hỏi và một đoạn văn trên Wikipedia có chứa câu trả lời, nhiệm vụ của mô hình là phải tìm ra câu trả lời trong đoạn văn.



Hình 22: Overall pre-training and fine –tuning procedures for BERT

Trong hình 22 đối với bài toán hỏi và đáp chúng ta trình bày đầu vào là một câu hỏi (question) và một đoạn văn (passage) với câu hỏi thì với một câu hỏi sẽ sử dụng A embedding và đoạn văn sẽ dùng B embedding chúng tôi chỉ giới thiệu một vector bắt đầu S ∈ R^H và vector kết thúc E ∈ R^H trong quá trình fine-tuning. Xác suất của một từ thứ i đang được tính bằng vị trí bắt đầu của câu trả lời và dựa vào công thức của Dot Product giữa T_i và S theo một Softmax đi qua tất cả các từ trong đoạn văn:

$$P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$$

Hình 23: Công thức tính xác suất ở vị trí thứ i [17]

Công thức ở hình 23 tương tự được sử dụng vị trí cuối của đoạn văn trả lời. Điểm số của một ứng cử viên được trải dài từ vị trí i đến vị trí j được định nghĩa là S.T_i + E.T_i và điểm số cao nhất mà ở đó j ≥ i sẽ được sử dụng để dự đoán. Mục tiêu của quá trình huấn luyện là tính tổng Log-Likelihoods của những dự đoán đúng về vị trí bắt đầu và kết thúc của câu trả lời.

3.3. Mô hình PhoBERT

Mô hình BERT thì vô cùng phổ biến và có ích để cải thiện đáng kể các vấn đề về xử lý ngôn ngữ tự nhiên (NLP), đó chính là sự thành công trong việc huấn luyện lại mô hình BERT tuy nhiên mô hình này lại bị giới hạn với ngôn ngữ là tiếng Anh. Còn những ngôn ngữ khác thì có thể sử dụng lại kiến trúc của BERT để huấn luyện lại. Đối với ngôn ngữ tiếng Việt, thì sẽ có hai mối quan ngại chính như sau:

- Thứ nhất là các dữ liệu trong kho văn bản tiếng Việt trên Wikipedia chỉ được dùng để huấn luyện các mô hình đơn ngữ, và các dữ liệu trên cũng chỉ là các dataset tiếng Việt được dùng để huấn luyện trong các mô hình đa ngôn ngữ ngoại trừ mô hình XML-R điều này ảnh hưởng rất nhiều đến kết quả dự đoán của mô hình bới vì kích thước của dataset tiếng Việt rất nhỏ chỉ khoảng 1 GB dữ liệu không nén, trong khi đó để có được kết quả tốt cần rất nhiều dữ liệu để pre-train mô hình.
- Điểm thứ hai là tất cả các mô hình ngôn ngữ dựa trên BERT đơn ngữ và đa ngôn ngữ đều không nhận thức được sự khác biệt giữa âm tiết tiếng Việt và các token của từ. Chúng tôi đề cập lý do này là bởi vì đến từ thực tế là khoảng trắng cũng được sử dụng để tách các âm tiết tạo thành từ khi viết bằng tiếng Việt.

Ví dụ:

- Chúng tôi có một câu có năm âm tiết: Tôi là một học sinh.
- Trong khi đó tiếng Anh thì sẽ có bốn âm tiết: I am a student.

Nếu như trường hợp mô hình không tách đúng tooken như dạng: Tôi là một học_sinh thì lúc này kết quả của mô hình nó không được chính xác như các tập dataset tiếng Anh vì thế cần có một bước tiền xử lý cho phân đoạn từ (Word Segmentation). Quá trình tiền xử lý này có thể đượct thực hiện bằng Bye-Pair Encoding (BPE).

Để giải quyết mối bận tâm đầu tiên thì có một mô hình là PhoBERT sử dụng 20 GB dữ liệu không nén có kiểu dữ liệu là văn bản để huấn luyện lại mô hình, đây là dataset được kết hợp bởi hai bộ dữ liệu (i) cái đầu tiên là kho ngữ liệu Wikipedia tiếng Việt (gần 1 GB) và (ii) kho dữ liệu thứ hai (gần 19 GB) là được tạo ra bằng cách loại bỏ các bài báo tương tự và trùng lặp khỏi kho văn bản tin tức tiếng Việt 50 GB. Để khắc phục được vấn đề thứ hai, chúng ta sẽ sử dụng RDRSegmenter từ VnCoreNLP và thực hiện phân đoạn từ và câu trên tập dữ liệu trước khi huấn luyện, với gần 145 triệu câu được phân đoạn từ (gần 3 tỷ tokens).

CHƯƠNG 4: XÂY DỰNG MÔ HÌNH ÁP DỤNG KIẾN TRÚC TRANSFORMER CHO BÀI TOÁN TRẢ LỜI CÂU HỎI

4.1. Tập dữ liệu

4.1.1. Tổng quan về tập dữ liệu

Chúng tôi sử dụng tập dataset UIT-ViSQuAD (Vietnamese University Information Technology Squad) là một tập dữ liệu tiếng Việt được sử dụng để huấn luyện mô hình trả lời câu hỏi tự động. Tập dữ liệu này được xây dựng bởi đại học Công nghệ Thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh, Việt Nam. Bộ dữ liệu tiếng Việt UIT-ViQuAD bao gồm hơn 23.000 cặp câu hỏi - câu trả lời do con người tạo ra dựa trên 5.109 đoạn văn của 174 bài viết tiếng Việt từ Wikipedia. Bộ dữ liệu phù hợp để đánh giá các mô hình Machine Reading Comprehension (MRC) trên tiếng Việt.

Các bài viết tiếng Việt từ trang web Wikipedia được chọn bằng cách: sử dụng Project Nayuki's Wikipedia's Internal PageRanks để có được một bộ 5.000 bài viết hàng đầu của Việt Nam, từ đó chọn ngẫu nhiên các bài viết để tạo dữ liệu. Cấu trúc mỗi dữ liệu bao gồm:

- Đoạn văn: mỗi đoạn văn là một đoạn tương ứng với một đoạn văn trong.
- Trong mỗi bài viết hình ảnh, mục lục và bảng được loại trừ, các đoạn văn ngắn hơn 300 ký tự hoặc chứa nhiều ký tự đặc biệt sẽ bị loại bỏ.
- Câu hỏi: mỗi câu hỏi được con người đặt ra dựa trên đoạn văn.
- Câu trả lời: câu trả lời cho câu hỏi là một khoảng thuộc đoạn văn.

Hình 24 là một ví dụ về một phần dữ liệu trong tập UIT-ViSquAD gồm có: đoạn văn (passage), các câu hỏi (question) được đặt ra dựa trên đoạn văn, câu trả lời (answer) được trích dẫn từ đoạn văn.

Passage: Nước biển có độ mặn không đồng đều trên toàn thế giới mặc dù phần lớn có độ mặn nằm trong khoảng từ 3,1% tới 3,8%. Khi sự pha trộn với nước ngọt đổ ra từ các con sông hay gần các sông băng đang tan chảy thì nước biển nhạt hơn một cách đáng kể. Nước biển nhạt nhất có tại vịnh Phần Lan, một phần của biển Baltic.
(**English**: Seawater has uneven salinity throughout the world although most salinity ranges from 3.1% to 3.8%. When the mix with freshwater pouring from rivers or near glaciers is melting, the seawater is significantly lighter. The lightest seawater is found in the Gulf of Finland, a part of the Baltic Sea.)

Question: Độ mặn thấp nhất của nước biển là bao nhiêu? (**English**: What is the lowest salinity of seawater?)
Answer: 3.1% (**English**: 3.1%)

Question: Nước biển ở đâu có hàm lượng muối thấp nhất? (**English**: Where is the lowest salt content?)
Answer: Vịnh Phần Lan. (**English**: Gulf of Finland.)

Hình *24*: Ví dụ cho bài toán trả lời câu *hỏi [16]*

Tuy nhiên tập dữ liệu mà chúng tôi tìm được chỉ có 19240 cặp câu hỏi và trả lời. Tập dữ liệu được cung cấp dưới định dạng JSON, trong đó mỗi đoạn văn bản được chia thành các câu riêng biệt. Mỗi câu hỏi được liên kết với một hoặc nhiều câu trả lời phù hợp. Các câu trả lời này được đánh số thứ tự để phân biệt giữa các câu trả lời khác nhau cho cùng một câu hỏi.

Quá trình tạo ra tập dữ liệu UIT-ViSQuAD gồm các bước:

1. Thu thập dữ liệu: bộ dữ liệu ban đầu được thu thập từ các nguồn trên mạng, bao gồm các tài liệu văn bản và các câu hỏi liên quan đến chúng.
2. Xử lý dữ liệu: bộ dữ liệu được đưa qua các bước xử lý dữ liệu để chuẩn bị cho quá trình huấn luyện mô hình. Các bước xử lý dữ liệu bao gồm chia tài liệu thành các câu và đoạn văn, tách câu hỏi ra khỏi các tài liệu, tiền xử lý các văn bản để loại bỏ các ký tự không cần thiết và các ký tự đặc biệt, tạo các đối tượng câu hỏi và đáp án, và đánh chỉ mục cho các từ trong văn bản.
3. Tạo tập dữ liệu: bộ dữ liệu được tạo ra bằng cách ghép các đối tượng câu hỏi và đáp án vào các đoạn văn tương ứng để tạo ra các cặp câu hỏi và đáp án.
4. Đánh giá và xử lý lỗi: bộ dữ liệu được đánh giá và xử lý lỗi để đảm bảo tính chính xác và độ tin cậy của dữ liệu.

4.1.2. Quá trình xử lý dữ liệu

Tập dữ liệu của chúng tôi gồm 19240 cặp câu hỏi và trả lời. Dữ liệu được tập thành có dạng giống như từ điển là Key và Value.

1. id: mã số định danh cho mỗi câu hỏi, giá trị của nó là một chuỗi.
2. question: câu hỏi của người dùng, có giá trị là một chuỗi.
3. context: đoạn văn bản liên quan đến câu hỏi của người dùng, có giá trị là một chuỗi.
4. text: câu trả lời cho câu hỏi tương ứng với đoạn văn bản , giá trị là một chuỗi.
5. start position: vị trí đầu tiên của câu trả lời trong đoạn văn bản mang giá trị là một số.

Hình 25 bên dưới là đoạn code để kiểm tra dữ liệu đầu vào gồm có đoạn văn, câu hỏi, câu trả lời, vị trí bắt đầu của câu trả lời. Ở kết quả đầu ra, chúng tôi thấy rằng tất cả kết quả đều ra không chứng tỏ dữ liệu mà chúng tôi thu thập được qua quá trình kiểm tra thì không chứa giá trị Null và NA.

```
print(len(context_empty))
print(len(question_empty))
print(len(answer_empty))
print(len(text_empty))
print(len(answer_start_empty))
```

```
0
0
0
0
0
```

Hình 25: Kiểm tra dữ liệu

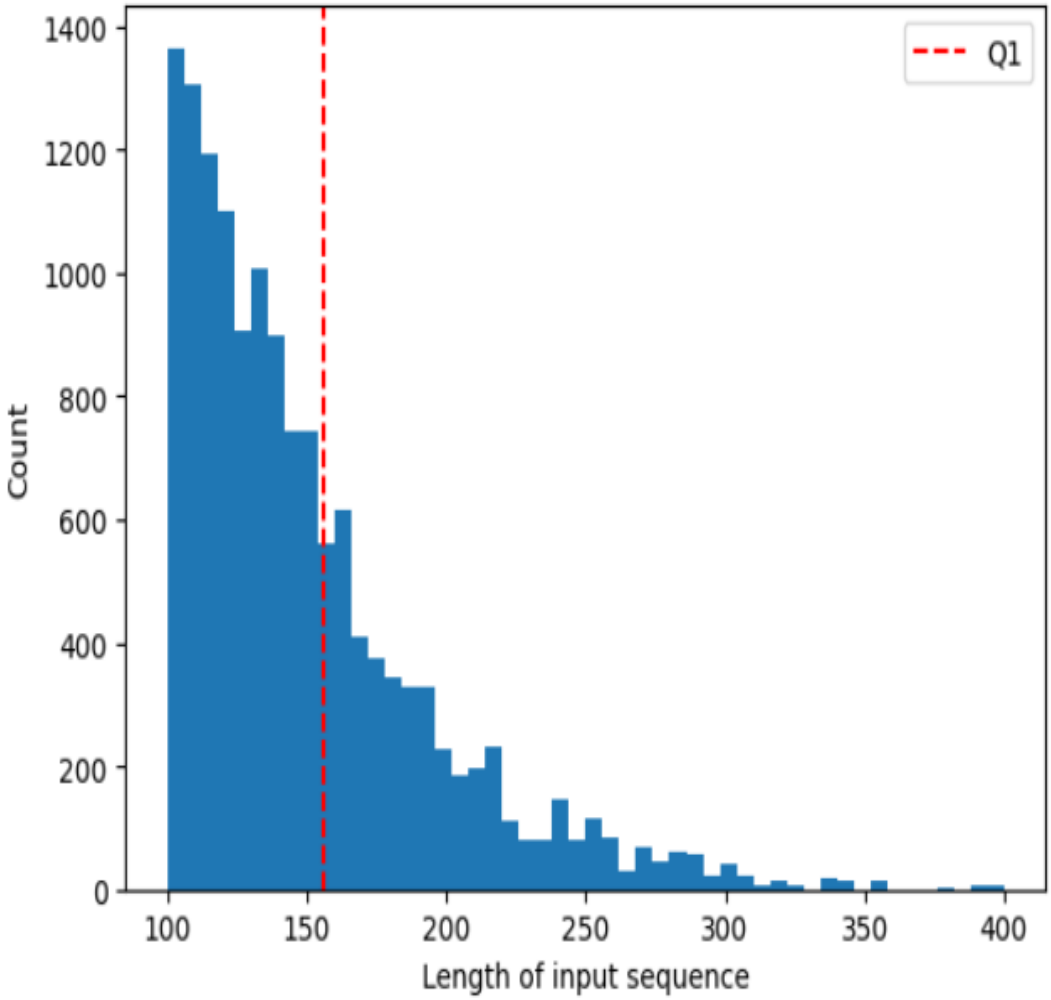
Đầu vào của mô hình không thể nhận vào với dạng chuỗi được nên cần phải xử lý chuỗi thành các token. Chúng tôi sử dụng thử viện VnCoreNLP (Vietnamese Natural Language Processing) là một mô hình được tạo ra với chức năng xử lý ngôn ngữ tự nhiên cơ bản như tách từ, gán nhãn từ loại, phân tích cú pháp và trích xuất thông tin. VnCoreNLP sử dụng các giải thuật máy học tiên tiến như CRF (Conditional Random Field) và SVM (Support Vector Machine) để đạt độ chính xác cao trong việc xử lý ngôn ngữ tự nhiên tiếng Việt. Chúng tôi sử dụng thư viện để tách đoạn văn, câu hỏi, câu trả lời thành các token phù hợp.
Ví dụ về một đoạn token ở hình 26 sau khi được chuyển đổi từ đoạn văn sang:

```
text = "Ông Nguyễn Khắc Chúc  đang làm việc tại Đại học Quốc gia Hà Nội. Bà Lan, vợ ông Chúc, cũng làm việc tại đây."
# ['Ông Nguyễn_Khắc_Chúc đang làm_việc tại Đại_học Quốc_gia Hà_Nội .', 'Bà Lan , vợ ông Chúc , cũng làm_việc tại đây .']
```

Hình 26: Ví dụ token

Sau khi văn bản được tách thành các token tiếp theo cần phải xử lý cắt bớt văn bản do đầu vào của các mô hình liên quan đến kiến trúc Transformer đa số sẽ nhỏ hơn 521 token, và nếu con số phù hợp sẽ giảm tiết kiệm tài nguyên và giảm thời gian huấn luyện của mô hình. Chúng tôi cũng tiến hành trực quan các đoạn văn của toàn bộ dataset để tìm ra con số phù hợp:

```
Q1=  156.0
```



Hình 27: Mô hình trực quan hóa dữ liệu

Từ hình 27 chúng tôi chọn đồ dài phù nhất để cắt các đoạn văn dài hơn là 256 token, chiếm 97% độ dài phổ biến của tập dữ liệu.

Tiếp theo để huấn luyện mô hình thì chúng tôi chia tập dữ liệu thành 3 tập:

- Tập train chiếm 80% với tên “train_vi.json”.
- Tập test chiếm 10% với tên “test_vi.json”.
- Tập validate chiếm 10% với tên “val_vi.json”.

Trong hình 28 là đoạn code dùng để chia dữ liệu từ một tập UIT-ViSquAD chúng tôi chia thành ba tập nhỏ hơn để phục vụ cho quá trình huấn luyện mô hình:

```python
import json

split_ratio = 0.8  # 80% for training, 20% for validation
# Load the entire dataset from the JSON file
with open('/content/UIT-ViQuAD.json', 'r', encoding='utf-8') as f:
    dataset = json.load(f)['data']

# Split the dataset into training and validation sets
split_index = int(len(dataset) * split_ratio)
split10 = int(len(dataset) * 0.1)
train_data = dataset[:split_index]
remain = dataset[split_index:]
val = remain[:split10]
test = remain[split10:]
# Save the training set to a new file
with open('train_vi.json', 'w', encoding='utf-8') as f:
    json.dump({'data': train_data}, f, ensure_ascii=False)

# Save the validation set to a new file
with open('val_vi.json', 'w', encoding='utf-8') as f:
    json.dump({'data': val}, f, ensure_ascii=False)

    # Save the validation set to a new file
with open('test_vi.json', 'w', encoding='utf-8') as f:
    json.dump({'data': test}, f, ensure_ascii=False)
```

Hình *28*: Code chia tập dữ liệu

4.2. Mô hình dùng để huấn luyện cho bài toán trả lời câu hỏi

Chúng tôi sử dụng mô hình PhoBERT để để huấn luyện mô hình vì mô hình này có một số ưu điểm:

- Mô hình hiểu được ngữ cảnh và ý nghĩa của Tiếng Việt so với các mô hình BERT khác được huấn luyện trên các tập dataset bằng tiếng Anh, những mô hình này không đảm bảo rằng chúng có thể hiểu được cấu trúc câu và cấu trúc ngữ pháp của tiếng Việt. Trong khi đó, PhoBERT được huấn luyện với khoảng trên 20 GB dữ liệu tiếng Việt, giúp cho mô hình có thể hiểu được ngữ cảnh và cấu trúc ngữ pháp tốt hơn các mô hình được huấn luyện và tiếng Anh.
- Mô hình tối ưu hóa cho tiếng Việt: PhoBERT được tối ưu hóa cho tiếng Việt, điều này có nghĩa là mô hình có thể hiện và xử lý tốt hơn các đặc điểm của tiếng Việt, chẳng hạn như sự phức tạp của các từ ghép hay các âm đầu, âm cuối của từ.

- Hiệu suất tốt của mô hình tốt: các thử nghiệm cho thấy, so với các mô hình được huấn luyện lại với các tập dữ liệu bằng tiếng Anh, PhoBERT cho kết quả tốt hơn cho các bài toán NLP trong tiếng Việt, chẳng hạn như phân loại văn bản, dịch máy, và cả Question Answering.

Các mô hình PhoBERT sẽ có ba loại giống như ở hình 29: base, large, base-v2 với đầu vào của các mô hình này là 256 token.

Model	#params	Arch.	Max length	Pre-training data
`vinai/phobert-base`	135M	base	256	20GB of Wikipedia and News texts
`vinai/phobert-large`	370M	large	256	20GB of Wikipedia and News texts
`vinai/phobert-base-v2`	135M	base	256	20GB of Wikipedia and News texts + 120GB of texts from OSCAR-2301

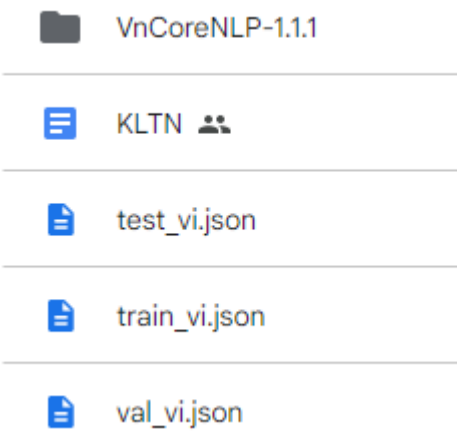
Hình 29: Các loại mô hình PhoBERT [14]

Hiện tại chúng tôi sử dụng PhoBERT-large để thực hiện lại huấn luyện lại cho tập dataset UIT-ViSQuAD.

4.3. Quá trình huấn luyện mô hình

4.3.1. Tổ chức file

Chúng tôi sử dụng Google Drive như bố trí ở hình 30 để lưu trữ file và các nguồn tài nguyên do bảo mật và dễ sử dụng, đặc biệt là việc huấn luyện có thể sử dụng Colab của Google là một công cụ khá mạnh để xử lý các bài toán về Machine Learning.



Hình 30: Tổ chức file

Trong đây có các tệp: train, test, val là các files chứa dữ liệu. Thư mục VnCoreNLP là để chứa thư viện xử lý chuỗi thành các token hỗ trợ cho việc huấn luyện.

Link Google Drive của chúng tôi: Google drive PhoBERT

4.3.2. Các thư viện Python được sử dụng

Có ba thư viện chính mà chúng tôi sử dụng: transformer, torch, vncorenlp và được được thiết lặp sẵn như hình 31 để chạy trong Colab:

```
!pip install transformers
!pip install torch
!pip install vncorenlp
```

Hình *31*: Lệnh tải thư viện trên Cola*b*

- transformer: là một thư viện mã nguồn mở của Hugging Face dùng để huấn luyện và sử dụng các mô hình ngôn ngữ tự nhiên, bao gồm cả mô hình Transformer. Thư viện này cho phép bạn truy cập đến nhiều mô hình Transformer khác nhau và thực hiện các tác vụ xử lý ngôn ngữ tự nhiên như mã hóa, giải mã, phân loại và tạo ra các dự đoán.
- torch: là một thư viện mã nguồn mở cho phép bạn thực hiện tính toán trên đồ thị tính toán. Thư viện này cung cấp một loạt các công cụ hỗ trợ, bao gồm cả các phép toán Tensor và các phép toán trên Tensor, cho phép bạn xây dựng các mô hình và huấn luyện chúng trên GPU.
- vncorenlp: là một thư viện mã nguồn mở để xử lý ngôn ngữ tự nhiên tiếng Việt, cung cấp các công cụ phân tích từ loại, phân tích cú pháp, tách từ, gán nhãn từ và gán nhãn câu. Thư viện này có thể được sử dụng để tiền xử lý dữ liệu tiếng Việt trước khi đưa vào huấn luyện mô hình Machine Learning hoặc Deep Learning.

4.3.3. Các hàm huấn luyện

Chúng tôi tiến hành tải mô hình PhoBERT từ Hugging Face về và huấn luyện lại với PhoBERT-large để tạo ra một mô hình mới như hình 32:

```
phobert = AutoModel.from_pretrained("vinai/phobert-large")
tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-large")
model = RobertaForQuestionAnswering(phobert.config).from_pretrained("vinai/phobert-large")
```

Hình 32: Code tải mô hình từ Hugging Face

Tiếp theo chũng tôi sử dụng VnCoreNLP như hình 33 để phân tích và trích xuất các thông tin cần thiết từ đoạn văn bản đầu vào. Cụ thể, hàm này nhận đầu vào là một đoạn văn bản X_train, sau đó sử dụng VnCoreNLP để phân tích cú pháp và đưa ra các thông tin về từ vựng (Word Segmentation), phân loại từ (Part-Of-Speech Tagging), nhận dạng thực thể (Named Entity Recognition), cấu trúc câu (Dependency Parsing),... Sau

khi xử lý, hàm trả về một chuỗi văn bản đã được xử lý, chuẩn hóa định dạng và loại bỏ các khoảng trắng và kí tự không cần thiết.

```
from vncorenlp import VnCoreNLP
annotator = VnCoreNLP("/content/drive/MyDrive/pho-bert/VnCoreNLP-1.1.1/VnCoreNLP-1.1.1/
def VnCoreNLP1(X_train):
  X_train = annotator.tokenize(X_train)
  p = ''
  for i in range(0, len(X_train)):
    for x in X_train[i]:
      w = ''
      w = w + x + ' '
      p+=w
  p=p.replace(" ,",",")
  p=p.replace(" .",".")
  p=p.replace("( ","(")
  p=p.replace(" )",")")
  p=p.replace(' " ',' "')
  p=p.replace("* ","*")
  p=p.replace(" ;",";")
  p=p.replace(" ?","?")
  p=p.replace(" {","{")
  p=p.replace("{ ","{")
  p=p.replace(" }","}")
  p=p.replace("} ","}")
  p=p.replace(" /","/")
  p=p.replace("/ ","/")
  return p
```

Hình 33: Hàm VnCorreNLP1

Hàm checkAns được sử dụng để kiểm tra và xác định xem văn bản được đề cập trong câu trả lời có trùng khớp với văn bản trong đoạn văn bản hay không như chúng tôi để ở hình 34. Nếu trùng khớp, hàm sẽ trả về câu trả lời và vị trí của nó trong đoạn văn bản. Nếu không trùng khớp, hàm sẽ tìm kiếm xung quanh vị trí của câu trả lời trong đoạn văn bản và trả về câu trả lời gần đúng và vị trí của nó. Hàm này giúp cải thiện chất lượng của câu trả lời khi sử dụng mô hình trả lời câu hỏi.

```
def checkAns(answer, context_text):
  ans=context_text[answer["answer_start"] : answer["answer_start"]+ len(answer["text"])]
  l=ans.replace("_"," ")
  if(l == answer["text"]):
    return ans, answer["answer_start"]
  else:
    firstWord=answer["text"].split(' ')[0]
    lenWord=len(firstWord)
    for i in range(1,8):
      try:
        position_1=answer["answer_start"]-i
        word=context_text[position_1: position_1+ lenWord]
        if(firstWord == word):
          return context_text[position_1: position_1+ len(answer["text"])],position_1
        position_2=answer["answer_start"]+i
        word=context_text[position_2: position_2+ lenWord]
        if(firstWord == word):
          return context_text[position_2: position_2+ len(answer["text"])],position_2
      except:
        return ans, answer["answer_start"]
    return ans, answer["answer_start"]
```

Hình 34: Hàm checkAns

Hàm checkTempAns giống hình 35 dùng để kiểm tra và sửa chữa phần câu trả lời (answer) nếu phần trả lời bị cắt đứt (vì bị tách thành hai phần hoặc bị thiếu từ ở đầu hay cuối câu trả lời) trong quá trình chạy mô hình Question Answering. Cụ thể, hàm sẽ kiểm tra từ đầu câu trả lời, xem có từ nào trùng với từ đầu tiên của câu trả lời không. Nếu có, hàm sẽ trả về câu trả lời được sửa chữa và vị trí của nó trong đoạn văn bản (context). Tương tự, nếu hàm không tìm thấy từ nào trùng khớp, nó sẽ trả về câu trả lời ban đầu. Hàm này được sử dụng trong trường hợp câu trả lời bị cắt đứt để đảm bảo độ chính xác và đầy đủ của câu trả lời trả về cho người dùng.

```
def checkTempAns(answer, context_text):
  firstWord=answer["text"].split(' ')[0]
  lenWord=len(firstWord)
  for i in range(1,8):
    try:
      position_1=answer["answer_start"]-i
      word=context_text[position_1: position_1+ lenWord]
      if(firstWord == word):
        return context_text[position_1: position_1+ len(answer["text"])],position_1
      position_2=answer["answer_start"]+i
      word=context_text[position_2: position_2+ lenWord]
      if(firstWord == word):
        return context_text[position_2: position_2+ len(answer["text"])],position_2
    except:
      return context_text[answer["answer_start"] : answer["answer_start"]+ len(answer["text"])], answer["answer_start"]
  return context_text[answer["answer_start"] : answer["answer_start"]+ len(answer["text"])], answer["answer_start"]
```

Hình 35: Hàm checkTempAns

Đến với lớp VnCoreSquad như hình 36 được viết dựa trên lớp SquadProcessor để tiền xử lý dữ liệu cho bộ dữ liệu SQuAD đối với tiếng Việt. Lớp này có ba thuộc tính, trong đó hai thuộc tính train_file và dev_file được sử dụng để lưu tên các tệp dữ liệu huấn luyện và kiểm tra SQuAD cho tiếng Việt. Phương thức _create_examples được sử dụng để tạo ra các ví dụ cho huấn luyện và kiểm tra, nó nhận đầu vào là các dữ liệu đầu vào (input_data) và kiểu tập dữ liệu (set_type) và trả về một danh sách các ví dụ. Trong quá trình tạo ví dụ, lớp này sử dụng các hàm checkAns và checkTempAns để xử lý văn bản và tìm kiếm vị trí câu trả lời. Sau đó, các ví dụ được tạo ra bằng cách sử dụng lớp SquadExample, trong đó chứa các thông tin như qas_id, question_text, context_text, answer_text, start_position_character, title, is_impossible và answers.

```
from tqdm import tqdm
class VnCoreSquad(SquadProcessor):
  train_file = "train-v1.1.json"
  dev_file = "dev-v1.1.json"
  def _create_examples(self, input_data, set_type):
        is_training = set_type == "train"
        examples = []
        demso=0
        for entry in tqdm(input_data):
            title = entry["title"]
            for paragraph in entry["paragraphs"]:
                context_text = VnCoreNLP1(paragraph["context"])
                for qa in paragraph["qas"]:
                    qas_id = qa["id"]
                    question_text = VnCoreNLP1(qa["question"])
                    start_position_character = None
                    answer_text = None
                    answers = []

                    is_impossible = qa.get("is_impossible", False)
                    if not is_impossible:
                        if is_training:
                            answer = qa["answers"][0]
                            answer_text, start_position_character=checkAns(answer, context_text)
                            qa["answers"][0].update({"text": answer_text})
                            qa["answers"][0].update({"answer_start": start_position_character})
                        else:
                            for k in range(0, len(qa["answers"])):
                              ans, start_position=checkTempAns(qa["answers"][k],context_text)
                              qa["answers"][k].update({"text": ans})
                              qa["answers"][k].update({"answer_start": start_position})

                            answers = qa["answers"]
                    example = SquadExample(
                        qas_id=qas_id,
                        question_text=question_text,
                        context_text=context_text,
                        answer_text=answer_text,
                        start_position_character=start_position_character,
                        title=title,
                        is_impossible=is_impossible,
                        answers=answers,
                    )
                    examples.append(example)
        return examples
processor = VnCoreSquad()
```

Hình *36*: Hàm _createExample

Đến với phương thức tạo ra hai đối tượng train_examples và dev_examples bằng cách gọi hai phương thức của lớp VnCoreSquad đã được định nghĩa trước đó. Cụ thể processor.get_train_examples (", "/content/drive/MyDrive/pho-bert/train_vi.json'): phương thức này trả về một danh sách các đối tượng SquadExample dùng để huấn luyện mô hình. Đối số thứ hai truyền vào đường dẫn tới file dữ liệu huấn luyện. Trong đoạn code này, file dữ liệu huấn luyện là /content/drive/MyDrive/pho-bert/train_vi.json. Còn với processor.get_dev_examples (", '/content/drive/MyDrive/pho-bert/val_vi.json'): phương thức này trả về một danh sách các đối tượng SquadExample dùng để đánh giá

mô hình. Phương thức này có đối đối số thứ hai truyền vào đường dẫn tới file dữ liệu đánh giá. Trong đoạn code này, file dữ liệu đánh giá là /content/drive/MyDrive/pho-bert/val_vi.json.

Việc tạo ra các đối tượng SquadExample từ file dữ liệu đầu vào cụ thể giống cách chúng tôi truyền file ở hình 37 được thực hiện bởi phương thức _create_examples() đã được định nghĩa trước đó trong lớp VnCoreSquad.

```
train_examples = processor.get_train_examples('','/content/drive/MyDrive/pho-bert/train_vi.json')
dev_examples = processor.get_dev_examples('','/content/drive/MyDrive/pho-bert/val_vi.json')
```

Hình 37: Đoạn code truyền dữ liệu vào cho processor

4.3.4. Fine-tuning mô hình PhoBERT

a) Các hàm xử lý đầu vào của mô hình PhoBERT

Hàm dùng để chuyển đổi các ví dụ huấn luyện trong định dạng của đối tượng SquadExample sang các đặc trưng và tạo tập dữ liệu huấn luyện PyTorch train_dataset dùng để huấn luyện mô hình.

```
train_features, train_dataset = squad_convert_examples_to_features(train_examples,
                                                                   tokenizer,
                                                                   max_seq_length = 256,
                                                                   doc_stride = 81,
                                                                   max_query_length = 81,
                                                                   is_training = True,
                                                                   return_dataset = 'pt',
                                                                   threads = 10
                                                                   )
```

Hình 38: Code xử lý đầu vào

Cụ thể trong hình 38 gồm:

- train_examples: tập hợp các ví dụ huấn luyện (instances) được lấy ra từ tệp huấn luyện train-vi.json.
- tokenizer: đối tượng tokenizer dùng để chuyển đổi văn bản thành dãy các số nguyên và thêm các mã thông báo đặc biệt, ví dụ như mã token đặc biệt để phân biệt giữa văn bản đầu vào và văn bản câu trả lời.
- max_seq_length: chiều dài tối đa của một chuỗi đầu vào, bao gồm cả văn bản đầu vào và văn bản câu trả lời.
- doc_stride: khi văn bản đầu vào quá dài, cần cắt nó thành các phần có độ dài là max_seq_length, với doc_stride là bước trượt giữa các phần. Bước này giúp cho mô hình có thể đọc được toàn bộ văn bản đầu vào dài hơn max_seq_length.
- max_query_length: chiều dài tối đa của câu hỏi đầu vào.

- is_training: xác định đang ở chế độ huấn luyện hay không.
- return_dataset: xác định định dạng của đầu ra là PyTorch Dataset hoặc TensorFlow tf.data.Dataset.
- threads: số lượng luồng được sử dụng để chuyển đổi đối tượng SquadExample thành các đặc trưng.

Ở đây thì chúng tôi có tinh chỉnh các tham số như:

- max_seq_length: đây là độ dài tối đa của đoạn văn bản được đưa vào cho mô hình. Nếu văn bản có độ dài lớn hơn giá trị này nó sẽ bị cắt ngắn, giá trị 256 được chọn là một giá trị phổ biến trong các mô hình học sâu và đủ để chứa khoảng hai đến câu của đoạn văn bản tiếng Việt trung bình.
- doc_stride: đây là độ dài cách nhau giữa các đoạn văn bản liên tiếp được xử lý bởi mô hình. Nếu không có độ dài trống này, mô hình sẽ dễ dàng bỏ qua một phần của đoạn văn bản khi chúng nằm ở giữa hai đoạn được xử lý liên tiếp. Giá trị 81 được chọn là một giá trị phổ biến trong các bộ dữ liệu hỏi đáp, đủ để đảm bảo rằng các đoạn văn bản gần nhau sẽ được xử lý bởi mô hình.
- max_query_length: Đây là độ dài tối đa của câu hỏi được đưa vào cho mô hình. Nếu câu hỏi có độ dài lớn hơn giá trị này, nó sẽ bị cắt ngắn. Giá trị 81 được chọn để đảm bảo rằng các câu hỏi có độ dài trung bình hoặc dài sẽ được xử lý đầy đủ bởi mô hình.

Tương tự như trên chúng tôi có hình 39 đối với tập validate:

```
dev_features, dev_dataset = squad_convert_examples_to_features(dev_examples,
                                                               tokenizer,
                                                               max_seq_length = 256,
                                                               doc_stride = 81,
                                                               max_query_length = 81,
                                                               is_training = False,
                                                               return_dataset = 'pt',
                                                               threads = 10
                                                               )
```

Hình 39: Hiệu chỉnh tham số cho tập dev_dataset

Hàm này sẽ tạo ra các đặc trưng (dev_features) và tập dữ liệu (dev_dataset) từ các ví dụ trong tập dữ liệu đó. Tham số max_seq_length, doc_stride và max_query_length được sử dụng để định dạng các đặc trưng. Tham số is_training được đặt là False để chỉ định rằng đây là tập dữ liệu đánh giá và không phải là tập dữ liệu huấn luyện.

b) Hàm huấn luyện mô hình:

```
num_epochs = 4
tb_writer = SummaryWriter()
train_sampler = RandomSampler(train_dataset)
train_dataloader = DataLoader(train_dataset, sampler=train_sampler, batch_size=16)
t_total = len(train_dataloader) // 1 * num_epochs

output_dir = os.path.join('/content/drive/MyDrive/pho-bert/', 'final_model')
no_decay = ["bias", "LayerNorm.weight"]

optimizer_grouped_parameters = [
    {
        "params": [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)],
        "weight_decay": 0,
    },
    {"params": [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)], "weight_decay": 0.0},
]
optimizer = AdamW(optimizer_grouped_parameters, lr=3e-5, eps = 1e-8)
scheduler = get_linear_schedule_with_warmup(
    optimizer, num_warmup_steps=814, num_training_steps=t_total
)

device = torch.device('cuda')

model.to(device)

global_step = 1
epochs_trained = 0
steps_trained_in_current_epoch = 0
tr_loss, logging_loss = 0.0, 0.0

model.zero_grad()
train_iterator = trange(
    epochs_trained, int(num_epochs), desc="Epoch", disable=-1 not in [-1, 0]
)
```

Hình *40*: Code huấn luyện mô hình

```
from functools import partial
tqdm = partial(tqdm, position=0, leave=True)
max_f1 = 0
patient = 0
for _ in train_iterator:
    epoch_iterator = tqdm(train_dataloader, desc="Iteration", disable=False)
    for step, batch in enumerate(epoch_iterator):
        model.train()
        batch = tuple(t.to(device) for t in batch)

        inputs = {
            "input_ids": batch[0],
            "attention_mask": batch[1],
            "start_positions": batch[3],
            "end_positions": batch[4],
        }

        outputs = model(**inputs)
        loss = outputs[0]
        loss.backward()
        tr_loss += loss.item()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
        optimizer.step()
        scheduler.step()
        model.zero_grad()
        global_step += 1

        if global_step % 5000 == 0:
            print(" global_step = %s, average loss = %s", global_step, tr_loss/global_step)
    model.eval()
    model_2 = model
    tokenizer_2 = tokenizer
    results = evaluate(model, tokenizer, dev_dataset, dev_examples, dev_features)
    print("exact: " + str(results['exact']))
    print(results['f1'])
    if results['f1'] >= max_f1:
        model_to_save = model.module if hasattr(model, "module") else model
        model_to_save.save_pretrained(output_dir)
        max_f1 = results['f1']
        patient = 0
    else:
        patient = patient+1
    if patient == 3:
        break

del model
```

Hình 41: Code khởi tạo biến

Trong đoạn mã trên ở hình 40 và hình 41, các biến được khởi tạo như sau:

- num_epochs: số lượng epoch (vòng lặp qua toàn bộ dữ liệu huấn luyện) để huấn luyện mô hình.
- tb_writer: một đối tượng để ghi log lại quá trình huấn luyện và hiển thị thông tin trên TensorBoard.
- train_sampler: một sampler để lấy ngẫu nhiên các mẫu từ tập dữ liệu huấn luyện.
- train_dataloader: DataLoader dùng để tạo ra các batch dữ liệu cho quá trình huấn luyện.

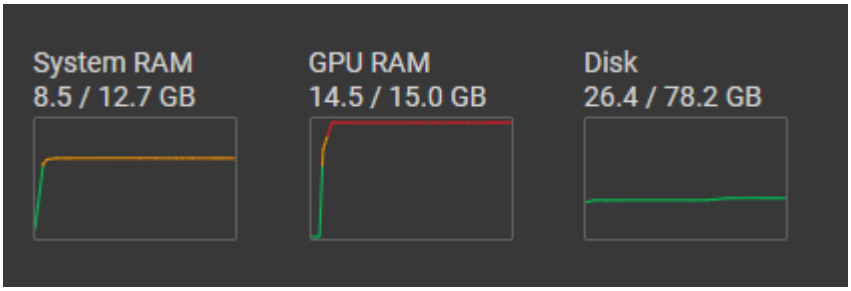
- t_total: tổng số lượng bước huấn luyện trong quá trình huấn luyện.
- output_dir: đường dẫn tới thư mục lưu trữ mô hình được huấn luyện.
- no_decay: danh sách các tên của các tham số mà không áp dụng lệch phân vùng trong quá trình tối ưu.
- optimizer_grouped_parameters: các thông số được sử dụng trong quá trình tối ưu.
- optimizer: một thuật toán tối ưu hóa được sử dụng để tối ưu các tham số của mô hình.
- scheduler: một scheduler được sử dụng để điều chỉnh tốc độ học trong quá trình huấn luyện.
- device: thiết bị được sử dụng để huấn luyện mô hình.
- model.to(device): dùng để đưa mô hình lên thiết bị GPU để tăng tốc độ huấn luyện.
- global_step: số lượng bước huấn luyện hiện tại.
- epochs_trained: số lượng epoch đã được huấn luyện trước đó.
- steps_trained_in_current_epoch: số lượng bước đã huấn luyện trong epoch hiện tại.
- tr_loss: giá trị hàm mất mát trung bình trong quá trình huấn luyện.
- logging_loss: giá trị hàm mất mát dùng để ghi log.
- model.zero_grad(): đặt độ dốc của các tham số về không
- train_iterator: vòng lặp qua các epoch của quá trình huấn luyện.

Với hàm trên các chức năng sẽ được thực hiện bao gồm:

- Hàm thiết lập thông số cho quá trình huấn luyện như số lần lặp lại huấn luyện (num_epochs), tên thư mục lưu trữ kết quả (output_dir), bộ tối ưu hóa (optimizer) và lịch trình giảm dần tốc độ học (scheduler).
- Quá trình truyền dữ liệu huấn luyện vào mô hình để tính toán sự mất mát và lan truyền ngược để cập nhật các trọng số của mô hình.
- Sau mỗi epoch, hàm sẽ đánh giá hiệu suất của mô hình trên tập dữ liệu xác thực và lưu mô hình có hiệu suất tốt nhất vào output_dir.
- Nếu hiệu suất của mô hình không cải thiện trong ba epoch liên tiếp, quá trình huấn luyện sẽ dừng lại.
- Hàm train_iterator được sử dụng để lặp lại quá trình huấn luyện cho số lần được chỉ định trong num_epochs. Đối với mỗi epoch, hàm epoch_iterator được sử dụng để lặp lại các batch dữ liệu trong tập huấn luyện. Với mỗi batch, mô hình được huấn luyện bằng cách tính toán đầu ra và mất mát của mô hình, sau đó lan truyền ngược để cập nhật các trọng số của mô hình.
- Hàm evaluate được sử dụng để đánh giá hiệu suất của mô hình trên tập dữ liệu xác thực sau mỗi epoch. Nếu hiệu suất của mô hình trên tập dữ liệu xác thực tốt hơn hiệu suất tốt nhất đã được lưu trữ trước đó, mô hình mới sẽ được lưu vào output_dir. Nếu không có cải thiện trong ba epoch liên tiếp, quá trình huấn luyện sẽ kết thúc.

Ở đây có một vài thông số sẽ được tinh chỉnh để tăng độ chính xác và tốc độ huấn luyện:

- num_epochs bằng bốn là sẽ lặp qua toàn bộ dữ liệu bốn lần, chúng tôi chọn số này là do thời gian giới hạn của Google Colab và một phần khác là độ chính xác cũng ở các lần chạy sau cũng không có cải thiện nhiều.
- batch_size=16 là kích thước của batch (tập hợp các mẫu dữ liệu) được sử dụng để huấn luyện mô hình trong mỗi lần cập nhật tham số. Cụ thể, trong đoạn code trên, train_dataloader sử dụng batch_size=16 để tạo ra các batch chứa 16 mẫu dữ liệu. Khi huấn luyện mô hình, các batch này sẽ được truyền qua mô hình và tính toán gradient của hàm mất mát để cập nhật tham số của mô hình. Sau khi huấn luyện xong một batch, mô hình sẽ được cập nhật trên một batch mới. Việc sử dụng batch giúp tăng tốc độ huấn luyện và làm cho quá trình huấn luyện có thể thực hiện được trên các GPU với bộ nhớ hạn chế. Với chúng tôi hiện tại chỉ dùng được 16 do sử dụng đến 14.5 GB/15 GB RAM của GPU như hình 42:



Hình 42: Các thông số của hệ thống trong quá trình huấn luyện

lr=3e-5 và eps=1e-8 là các siêu tham số được sử dụng trong thuật toán tối ưu hóa AdamW, được sử dụng để tối ưu hóa các tham số của mô hình.

lr (Learning Rate) là một tham số quan trọng trong thuật toán tối ưu hóa, được sử dụng để điều chỉnh tốc độ học của mô hình. Giá trị 3e-5 được sử dụng trong đoạn mã này là một giá trị phổ biến được sử dụng trong các tác vụ học sâu.

eps (Epsilon) là một giá trị nhỏ được sử dụng để tránh việc chia cho số không hoặc gần bằng số không trong quá trình tối ưu hóa. Giá trị 1e-8 được sử dụng ở đây là một giá trị phổ biến và an toàn để tránh lỗi chia cho số không trong thuật toán AdamW.

4.4. Phương pháp đánh giá mô hình

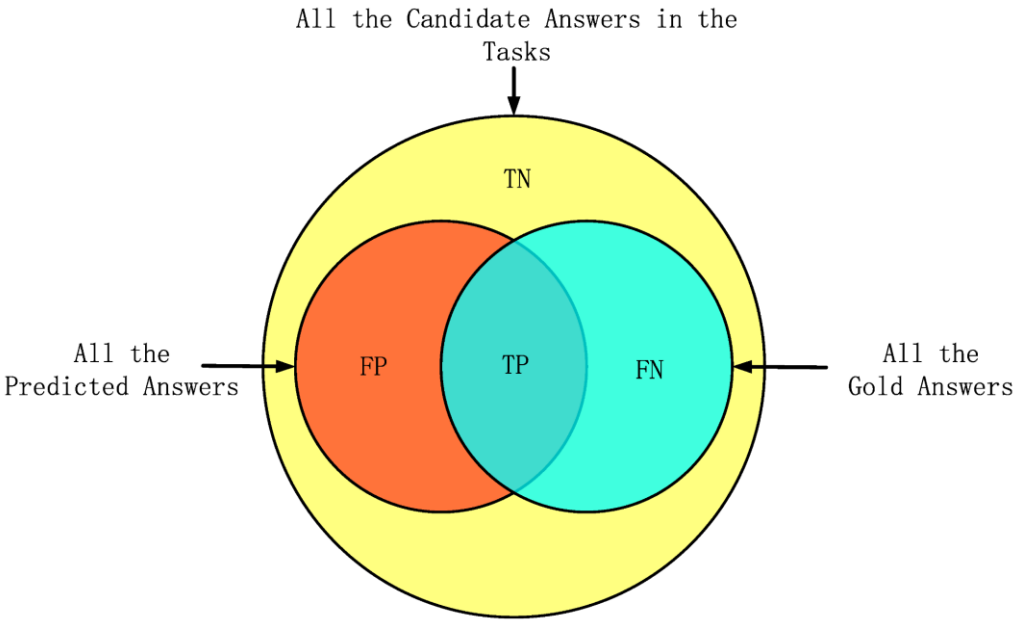
Phương pháp đánh giá hiệu suất của mô hình đối với bài toán Question-Answering chúng tôi sử dụng hai độ đo là Exact Match (EM) và F1-Score (F1) dựa trên các đánh giá trên bộ SQuAD hay chi tiết các độ đo được trình bày cụ thể trong nghiên cứu của Zeng và cộng sự [1].

4.4.1. Độ đo F1-score

Số liệu này đo lường sự chồng chéo giữa dự đoán và câu trả lời đúng. Theo phương pháp đánh giá trong SQuAD, coi câu trả lời dự đoán và câu trả lời đúng là túi token, trong khi bỏ qua tất cả các dấu chấm câu và các từ bài viết như "a" và "an" hoặc

"the". F1-score được tính thông qua hai độ đo khác là Precision và Recall, chi tiết của các độ đo chúng tôi sẽ trình bày trong phần bên dưới:

Precision: Thể hiện phần trăm trùng lặp giữa từ trong câu trả lời thật và câu trả lời dự đoán. Để có được Precision ở cấp độ từ, trước tiên chúng ta cần hiểu ý nghĩa của true positive (TP), false positive (FP), true negative (TN), and false negative (FN) như trong hình 43.



Hình *43*: Phần trăm trùng lặp 3 biến

Hình 43 trên mô tả:

- TP thể hiện các từ giống nhau giữa câu trả lời dự đoán và câu trả lời đúng.
- FP thể hiện các từ không có trong câu trả lời đúng nhưng trong câu trả lời dự đoán.
- FN thể hiện các từ không có trong câu trả lời dự đoán mà là câu trả lời đúng.

Precision ở cấp độ từ cho một câu hỏi được tính như hình 44 bên dưới:

$$\text{Precision} = \frac{\text{Num}(TP)}{\text{Num}(TP) + \text{Num}(FP)}$$

Hình 44: Công thức tính Precision [10].

Trong đó Num (TP) là số lượng từ TP, Num (FP) là số lượng từ FP.

- Recall: Đại diện cho tỷ lệ phần trăm từ trong câu trả lời đúng đã được dự đoán chính xác trong một câu hỏi được thể hiện như hình 45:

$$\text{Recall} = \frac{\text{Num}(TP)}{\text{Num}(TP) + \text{Num}(FN)}$$

Hình 45: Công thức tính Recall [10].

Trong đó Num (TP) là số lượng từ TP, Num (FN) là số lượng từ FN. F1-score được tính như hình 46:

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Hình 46: Công thức tính F1-Score [10]

4.4.2. Độ đo Exact Match (EM)

Nếu câu trả lời đúng cho câu hỏi là một câu hoặc một cụm từ, có thể một số từ trong câu trả lời do hệ thống tạo ra là câu trả lời đúng và các từ khác không phải là câu trả lời đúng. Trong trường hợp này, Exact Match đại diện cho tỷ lệ phần trăm câu hỏi mà câu trả lời do hệ thống tạo ra hoàn toàn khớp với câu trả lời đúng, có nghĩa là mọi từ đều giống nhau. Kết hợp chính xác thường được viết tắt là EM.

Ví dụ: nếu một tác vụ của mô hình có chứa N câu hỏi, mỗi câu hỏi tương ứng với một câu trả lời đúng, câu trả lời có thể là một từ, cụm từ hoặc câu và số câu hỏi mà hệ thống trả lời đúng là M. Exact Match sau đó có thể được tính như hình 47:

$$\text{EM} = \frac{M}{N}$$

Hình 47: Công thức tính Exact Match

4.4.3. Hàm đánh giá mô hình

Hàm này được sử dụng sau khi mô hình chạy xong mỗi epoch như hình 48:

```python
import os
def evaluate(model, tokenizer, dev_dataset, dev_examples, dev_features):
    eval_sampler = SequentialSampler(dev_dataset)
    eval_dataloader = DataLoader(dev_dataset, sampler=eval_sampler, batch_size=40)
    all_results = []
#     start_time = timeit.default_timer()
    for batch in tqdm(eval_dataloader, desc="Evaluating"):
        model.eval()
        batch = tuple(t.to(device) for t in batch)
        with torch.no_grad():
            inputs = {
                "input_ids": batch[0],
                "attention_mask": batch[1],
#                 "token_type_ids": batch[2],
            }
            example_indices = batch[3]
            outputs = model(**inputs)
            output_model = torch.cat((outputs.start_logits, outputs.end_logits), dim=1) # concatenate start_logits and end
        for i, example_index in enumerate(example_indices):
            eval_feature = dev_features[example_index.item()]
            unique_id = int(eval_feature.unique_id)
            output = outputs.start_logits[i], outputs.end_logits[i]
            if isinstance(output, tuple):  # regular logits
                start_logits, end_logits = output
            else:  # top k logits
                start_logits, end_logits, _, _ = output.chunk(4)
            result = SquadResult(unique_id, start_logits.detach().cpu().numpy(), end_logits.detach().cpu().numpy())
            all_results.append(result)
            # all_results.append(RawResult(unique_id=feature.unique_id, start_logits=start_logits.astype(np.float64), end_
```

Hình *48*: Hàm đánh giá mô hình

```python
    predictions = compute_predictions_logits(
            dev_examples,
            dev_features,
            all_results,
            20,
            128,
            False,
          None,
          None,
          None,
            True,
            False,
            0.0,
            tokenizer,
        )
    results = squad_evaluate(dev_examples, predictions)
    return results
```

Hình 49: Hàm evaluate

Hàm evaluate nhận vào năm đối số như hình 49:

- model: mô hình BERT đã được huấn luyện để giải quyết bài toán Extractive Question Answering (QA).
- tokenizer: tokenizer tương ứng với mô hình BERT đó để chuyển đổi câu hỏi và đoạn văn bản thành đầu vào cho mô hình.
- dev_dataset: tập dữ liệu đánh giá được biểu diễn dưới dạng PyTorch Dataset.
- dev_examples: danh sách các câu hỏi và đoạn văn bản tương ứng trong tập dữ liệu đánh giá.
- dev_features: danh sách các đặc trưng được trích xuất từ dev_examples để tạo thành dev_dataset.

Trong hàm này, trước tiên khởi tạo eval_sampler và eval_dataloader để lấy ra các batch dữ liệu từ dev_dataset. Tiếp theo, chúng tôi sẽ tạo một danh sách all_results để chứa kết quả dự đoán của mô hình trên các batch dữ liệu. Sau đó, chúng tôi chạy một vòng lặp for để duyệt qua từng batch dữ liệu trong eval_dataloader. Trong mỗi lần lặp, hãm sẽ đưa đầu vào của batch vào mô hình để lấy ra kết quả dự đoán (output) của mô hình. Từ kết quả đầu ra này, chúng tôi sẽ trích xuất ra các giá trị start_logits và end_logits tương ứng với câu hỏi và đoạn văn bản. Cuối cùng, chúng tôi tạo một đối tượng SquadResult lưu trữ kết quả dự đoán của mô hình và thêm vào danh sách all_results.

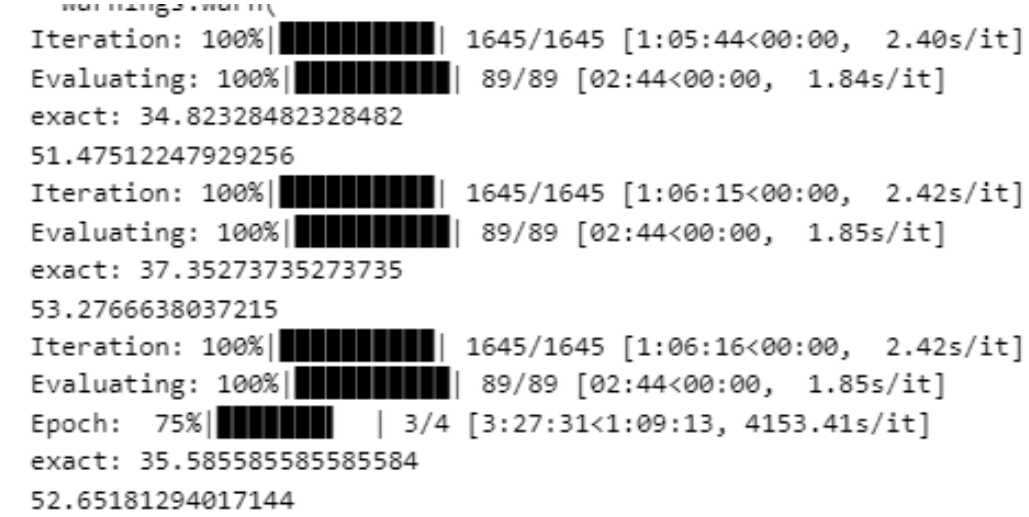
Sau khi đã duyệt qua hết các batch dữ liệu trong eval_dataloader, sử dụng all_results để tính toán các kết quả dự đoán cuối cùng thông qua hàm compute_predictions_logits. Hàm này nhận vào 14 đối số để tính toán kết quả dự đoán cuối cùng và trả về danh sách các kết quả dự đoán cho các câu hỏi trong tập dev_examples.

Cuối cùng, chúng tôi sử dụng kết quả dự đoán và tập dữ liệu đánh giá dev_examples để tính toán độ chính xác bằng cách gọi hàm squad_evaluate. Kết quả cuối cùng của hàm evaluate là một thư viện của Python chứa giá trị Exact match và F1 score.

Kết quả của mô hình:

Đối với tập validate kết quả như hình 50:

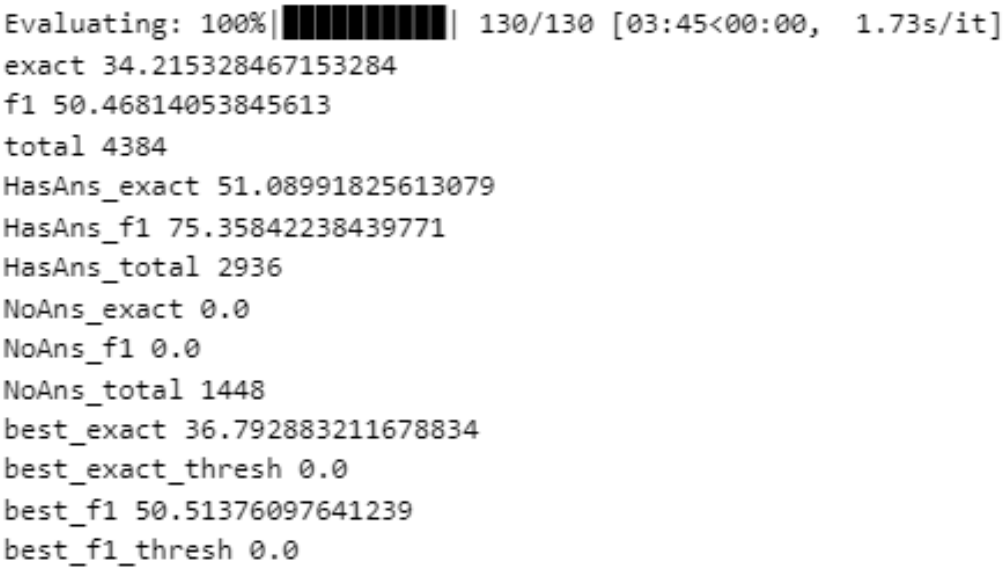
+ F1 = 52.65
+ Ex = 35.58



Hình *50*: Độ đo trên tập validate

Đối với tập test kết quả như hình 51:

+ F1 = 50.5
+ Ex = 34.2



Hình *51*: Độ đo trên tập test

CHƯƠNG 5: ỨNG DỤNG WEBSITE TRẢ LỜI CÂU HỎI

5.1. Ý tưởng dựng sản phẩm

Chúng tôi lấy ý tưởng xuất phát từ nhu cầu công nghệ ngày càng phát triển, con người đòi hỏi ngày càng tiếp thu thêm nhiều kiến thức mới do đó có nhiều câu hỏi được đặt ra hơn mỗi ngày. Đặt biệt là hiện khi ChatGPT ra đời đã và cũng đã có ít nhiều ảnh hưởng đến Google bởi vì tính tiện lời mà nó đem lại. Chính vì thế chúng tôi muốn đưa ra một sản phẩm ở mức độ áp dụng mô hình mà chúng tôi đã huấn luyện được và có giúp ích được cho mọi người.

5.2. Công nghệ sử dụng cho website

Chúng tôi sử dụng Flask để xây dựng website. Flask là một Framework web Python nhẹ, đơn giản và dễ sử dụng, được sử dụng để xây dựng các ứng dụng web nhỏ và trung bình. Flask cung cấp một cách tiếp cận tối giản cho việc xây dựng các ứng dụng web bằng cách cung cấp các công cụ cần thiết để xử lý các yêu cầu HTTP, quản lý phiên, tạo các mẫu HTML động, xử lý dữ liệu biểu mẫu và thực hiện các tác vụ khác liên quan đến web. Ngoài ra, Flask có một cộng đồng lớn, đầy đủ các tài liệu và hỗ trợ từ cộng đồng, điều này giúp cho việc phát triển ứng dụng web trở nên nhanh chóng và dễ dàng hơn. Flask cũng rất linh hoạt và có thể được tích hợp với các công cụ và thư viện khác của Python để tăng khả năng và tính linh hoạt của ứng dụng web.

Chúng tôi sử dụng Python 3.8 bởi vì tính ổn định và dễ sử dụng của nó.

Ngoài ra còn sử dụng các thư viện của Python như:

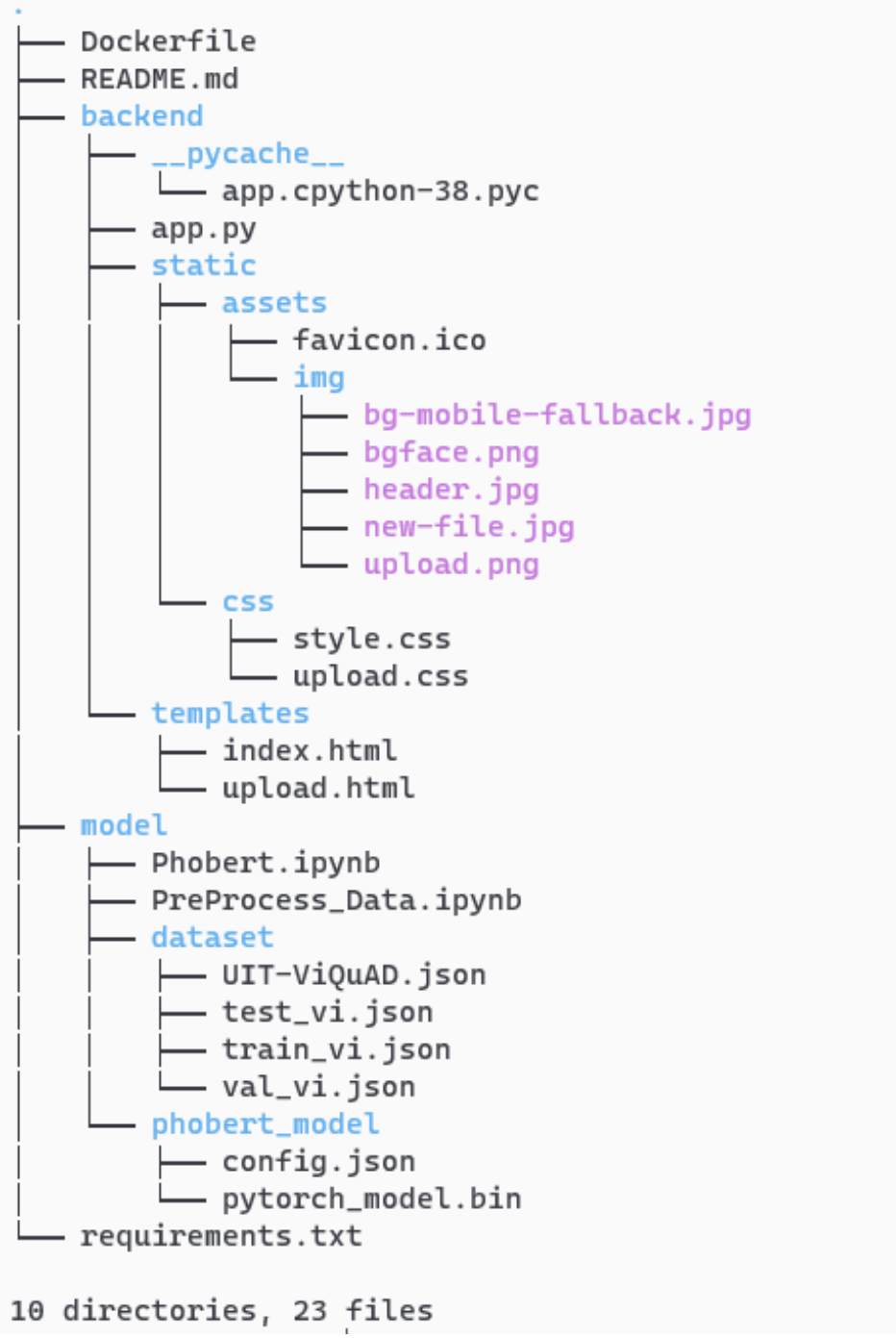
- Transformer: thư viện mã nguồn mở của Hugging Face hỗ trợ các tác vụ xử lý ngôn ngữ tự nhiên như phân loại văn bản, dịch máy, tạo ngữ liệu và trích xuất thông tin, sử dụng các mô hình học sâu như BERT, GPT-2, RoBERT, DistilBERT.
- Docx2txt: thư viện mã nguồn mở cho phép trích xuất nội dung từ file định dạng Microsoft Word (.docx) và chuyển đổi nó sang văn bản thuần túy (.txt).
- Flask: thư viện mã nguồn mở cho phép xây dựng các ứng dụng web và API REST bằng Python, đơn giản và dễ sử dụng.
- Vncorenlp: thư viện mã nguồn mở cho phép xử lý ngôn ngữ tự nhiên tiếng Việt, bao gồm các tác vụ như tách từ, gán nhãn từ loại, phân tích cú pháp, rút gọn từ và chuyển đổi chữ viết tắt.
- Torch: thư viện mã nguồn mở hỗ trợ các tác vụ học máy và học sâu bằng Python, bao gồm mô hình mạng nơ-ron sâu (Deep Neural Networks), thuật toán tối ưu hóa và các công cụ phân tích dữ liệu.

Mục đích của chúng tôi muốn có thể sử dụng sản phẩm ở bất cứ đâu do đó chúng tôi sử dụng Docker để tạo ra thùng chứa (Container) và mang sản phẩm có thể phát triển

lên các Server khác. Hiện tại chúng tôi sử dụng dịch vụ Machine Virtual của Azure để phát triển website.

5.3. Cấu trúc source code

Sản phẩm được quản lý và lưu trữ trên Github theo link: Source code đồ án



Hình *52*: Cấu trúc Source Code

Hình 52, chúng ta thấy bên ngoài cùng sẽ có hai thư mục chính là model và backend:

- Thư mục model bao gồm:
  + Thư mục dataset chứa các file dữ liệu hỗ trợ cho việc huấn luyện và kiểm tra kết quả của model.

+ Thư mục phober_model chưa model Pho-bert sẽ dụng vào phần backend để xử lý tìm câu trả lời cho bài toán trả lời câu hỏi.
+ Tệp PreProcess_Data.ipynb dùng để xử lý dữ liệu, chia dữ liệu để làm đầu vào cho model.
+ Tệp PhoBERT.ipynb dùng để tạo token cho đầu vào, huấn luyện, đánh giá mô hình so với các validate và tập test.

- Thư mục backend gồm có:
  + Một thư mục là template chứa các tệp giao diện như trang upload file, dùng để đăng tải các tệp word hoặc pdf chứa nội dung là các văn bản. Một tệp khác dùng để phản hồi đáp án của các câu hỏi.
  + Thư mục css dùng để chứa các file dùng để tinh chỉnh cho giao diện nhìn bắt mắt hơn cho người dùng.
  + Thư mục img dùng để chứa các hình ảnh hỗ trợ cho website
  + Tệp app.py dùng để chạy chương trình, tệp này sẽ tải lên mô hình và xử lý các API từ giao diện trả về.

5.4. Các lệnh dùng để chạy ứng dụng

5.4.1. Chạy ở môi trường máy cá nhân (local)

Bước 1: Trước hết, chúng tôi cần chạy file requirement.txt để cài đặt các thư viện như hình 53:

```
tranq@TuanPC MINGW64 ~
$ pip install -r requirements.txt
```

Hình 53: Lệnh cài đặt các thư viện

Bước 2: Chúng tôi sẽ kiểm tra xem máy tính sử dụng được GPU hay không (nếu không thì bỏ qua bước này) bằng việc vào tệp app.py và tiến hành thay thế như hình 54:

```
DEVICE = torch.device('cpu')
```

Hình 54: Có GPU

Thành lệnh giống như trong hình 55:

```
DEVICE = torch.device('cuda')
```

Hình 55: Không có GPU

Bước 3: Tiếp theo, chúng tôi sẽ chạy tệp backend với các lệnh sau:

+ Chúng tôi chuyển vào thư mục backend như hình 56:

```
tranq@TuanPC MINGW64 ~/Documents/Semeter1_2022_2023/kltn (master)
$ cd backend/
```

Hình *56*: Lệnh vào thư mục backend

+ Chúng tôi tiến hành chạy ứng dụng bằng cách chạy tệp app.py như hình 57:

```
tranq@TuanPC MINGW64
$ flask run
```

Hình 57: Lệnh chạy dự án

5.4.2. Phát triển ứng dụng lên các môi trường khác

Bước 1: Chúng ta kiểm tra xem đã cài đặt Docker chưa như hình 58:

```
$ docker version
```

Hình *58*: Lệnh kiểm tra phiên bản *Docker*

Nếu có thì thì tiếp tục bước 2, nếu không thì cài đặt Docker.
Bước 2: Chúng tôi tạo images lệnh như hình 59:

```
tranq@TuanPC MINGW64 ~/Documents
$ docker build -t app .
```

Hình 59: Lệnh build Docker

Bước 3: Tiếp theo chúng tôi chạy container chạy lệnh như hình 60:

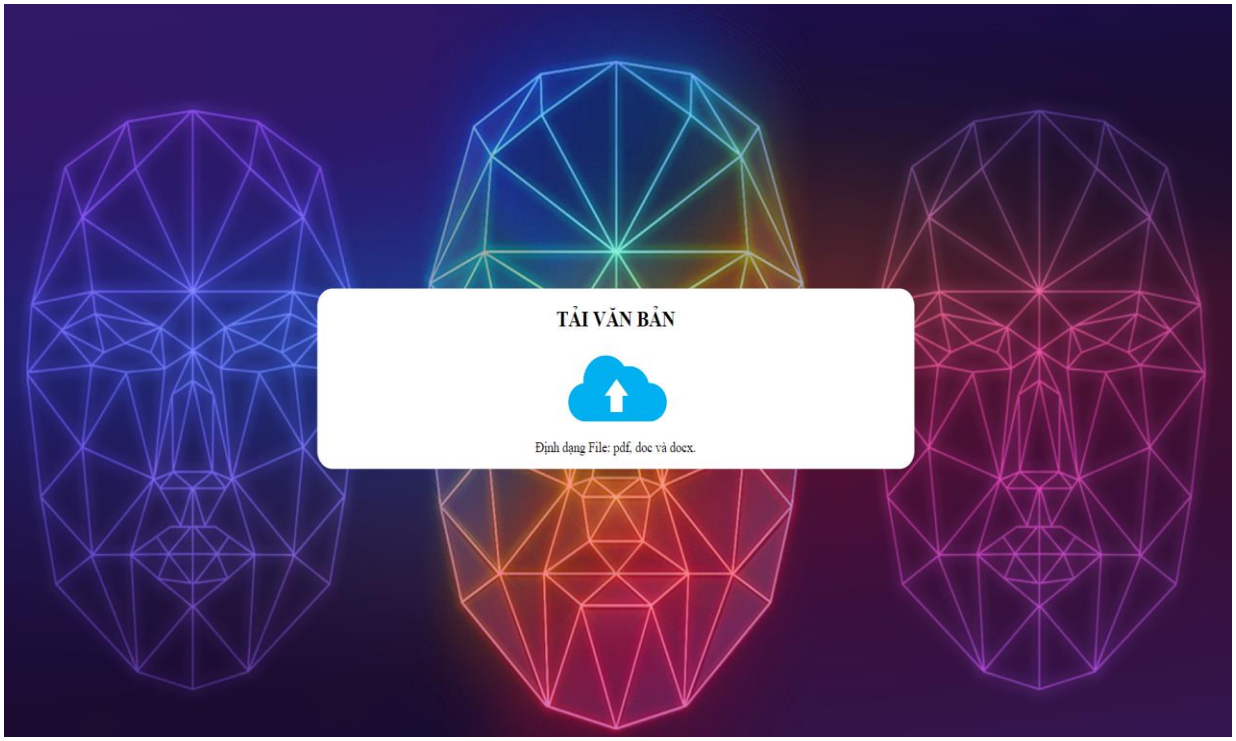
```
tranq@TuanPC MINGW64 ~/Documents/Semeter1
$ docker run -d -p 80:5000 app
```

Hình 60: Lệnh chạy Docker

5.5. Giao diện của website

5.5.1. Trang upload file

Đây là trang được chúng tôi dùng để tải các tệp word và pdf, cho người dùng tải lênh các tệp có phần đuôi tệp là docx, pdf khi bấm vào biểu tượng tải tệp như hình 61. Các tệp này chính là các văn bản mà người dùng muốn hệ thống sẽ dựa theo để trả lời các câu hỏi.



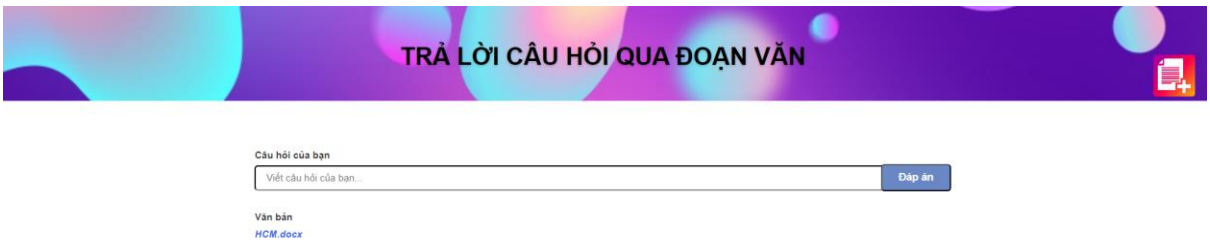
Hình *61*: Giao diện tải tệp lệnh

Trong hình 62 một khi người dùng tải tệp lên không đúng định dạng tệp thì hệ thống sẽ tự động hiện lên thông báo để người dùng biết và tải lại tệp khác lên.

Hình *62*: Khi tải lên sai định dạng tệp

5.5.2. Trang đặt câu hỏi

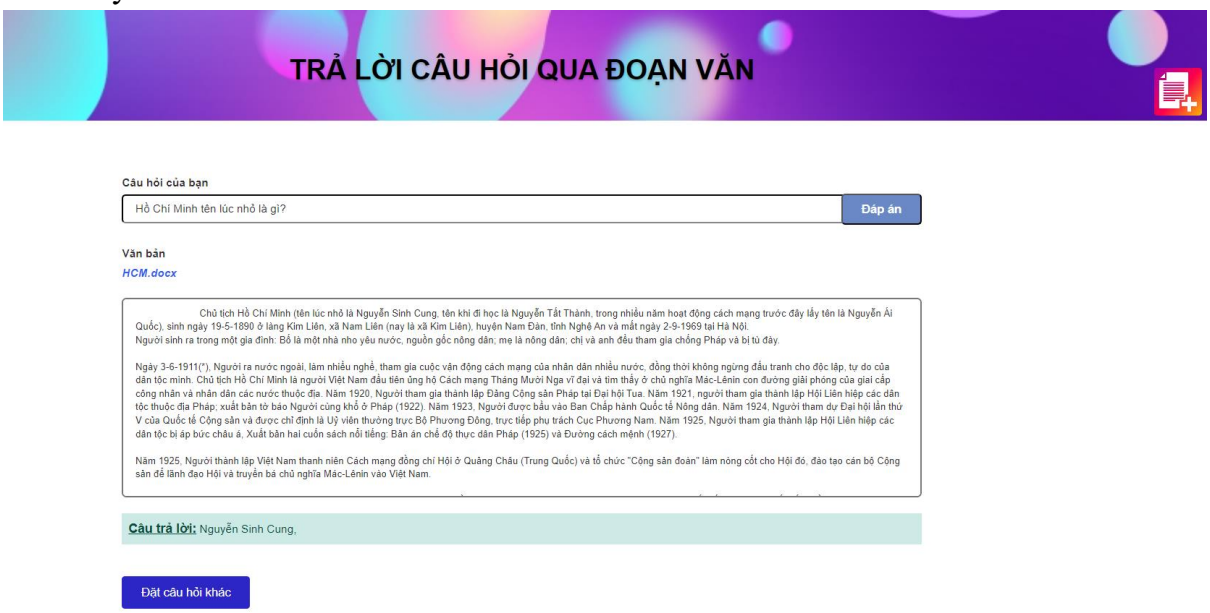
Sau khi hệ thống đã tải đúng định dạng tệp thì hệ thống sẽ chuyển sang trang câu hỏi và trả lời giống hình 63 để người dùng có thể đặt các câu hỏi. Ở đây người dùng sẽ nhập câu hỏi vào ô câu hỏi của bạn sau đó nhấn vào nút đáp án để hệ thống tìm câu trả lời cho câu hỏi đã đặt.



Hình 63: Giao diện đặt câu hỏi

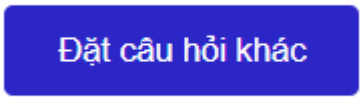
5.5.3. Trang trả lời câu hỏi

Đến với hình 64, hệ thống sau khi nhận được câu hỏi và văn bản sẽ dựa vào nội dung có trong văn bản để tìm câu trả lời đúng nhất và trả về màn hình cho người dùng nhìn thấy.



Hình *64*: Giao diện trả lời câu hỏi

Nếu người dùng muốn đặt câu hỏi khác thì có thể viết vào ô "Câu hỏi của bạn" hoặc bấm và nút "Đặt câu hỏi khác" như hình 65 để đặt câu hỏi mới.



Hình 65: Nút đặt câu hỏi

Nếu người dùng muốn thay đổi nội dung của văn bản, để có những câu hỏi khác chủ đề thì có thể chọn vào hình 66:

Hình *66*: Nút chọn file mới để hỏi

Khi người dùng bấm vào nút này hệ thống sẽ trở về trang tải tệp lên ban đầu để người chọn một văn bản khác tải lên.

PHẦN 3: KẾT LUẬN

CHƯƠNG 6: TỔNG KẾT VỀ ĐỒ ÁN

6.1. Những điều đã làm được

Trải qua quá trình nghiên cứu, dưới sự hướng dẫn tận tình của thầy Quách Đình Hoàng, chúng tôi đã hoàn thành luận văn "Tìm hiểu kiến trúc Transformer và ứng dụng cho bài toán trả lời câu hỏi" và đã đạt được những kết quả sau.

Tìm hiểu và hệ thống các kiến thức liên quan:

- Mô hình thuật toán Transformer.
- Tổng quan mô hình Transformer.
- Embedding Layer với Position Encoding.
- Encoder.
- Decoder.
- Cách hoạt động của Transformer trong bài toán trả lời câu hỏi.

Chúng tôi đã tìm được tập dataset tiếng Việt phù hợp cho bài toán trả lời câu hỏi, qua mô hình Bert và PhoBERT thì chúng tôi đã chọn mô hình PhoBERT để tiến hành huấn luyện và áp dụng dụng vào bài toán thực tế mà chúng tôi đã đặt ra.

Về phần mô hình chúng tôi đã tạo được mô hình PhoBERT để áp dụng cho bài toán trả lời câu hỏi và áp dụng mô hình vào website để có giao diện, và gần gũi hơn cho người sử dụng. Ngoài ra chúng tôi còn deploy website lên hosting để mọi người có thể dễ dàng sử dụng.

6.2. Những mặt cần khắc phục

Điểm thứ nhất là về mô hình PhoBERT do độ chính xác của mô hình vẫn còn chưa cao, chúng tôi đã tiến hành nhiều biện pháp tinh chỉnh về độ dài chuỗi đầu vào, các siêu tham số trong quá trình fine-tune lại mô hình, chúng tôi nhận ra có thể là do tập dữ liệu của chúng tôi vẫn chưa thật sự đầy đủ hoặc do giá hệ đánh giá mô hình truyền thống không phù hợp với mô hình của chúng tôi.

Điểm thứ hai là về phần ứng dụng, chúng tôi nhận thấy tốc độ trả lời câu hỏi của mô hình với các văn bản dài trên 10MB thì tốc độ phản hồi vẫn còn hơi lâu do phải xử lý cắt văn bản thành các đoạn nhỏ khoảng 256 tokens để làm đầu vào của mô hình, ở điểm này nếu có thể sử dụng hệ thống cụm Cluster để chia nhỏ đầu vào và phân đều trên các máy thì có thể rút ngắn đáng kể thời gian trả lời câu hỏi. Do quá trình cấu hình các máy để tạo cụm Cluster khá phức tạp và thời gian hoàn thành đồ án cũng khá là nhanh nên chúng tôi chỉ dừng lại ở việc tìm giải pháp cho mô hình.

TÀI LIỆU THAM KHẢO

[1] C. Zeng et al., "A Survey on Machine Reading Comprehension: Tasks, Evaluation Metrics and Benchmark Datasets," arXiv:2006.11880v2, 2020. [Online]. Available: https://arxiv.org/abs/2006.11880.

[2] P. Đ. Khánh, "Attention is all you need," 18 06 2019. [Online]. Available: https://phamdinhkhanh.github.io/2019/06/18/AttentionLayer.html.

[3] mrbo, "Giới thiệu về Transformers: Kiến trúc cải tiến trong xử lý ngôn ngữ tự nhiên," 10 2 2023. [Online]. Available: https://mrbo.site/thuat-toan-transformer/.

[4] "Transformer (machine learning model)," [Online]. Available: https://en.wikipedia.org/wiki/Transformer_(machine_learning_model).

[5] "Transformers," Hugging Face, [Online]. Available: https://huggingface.co/docs/transformers/index.

[6] vncorenlp, N. Q. Dat and V. Thanh, "VnCoreNLP," [Online]. Available: https://github.com/vncorenlp/VnCoreNLP.

[7] P. Lippe, "Tutorial 6: Transformers and Multi-Head Attention," [Online]. Available: https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html.

[8] M. Phi, "Illustrated Guide to Transformers- Step by Step Explanation," 30 04 2020. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0.

[9] . M. Saeed, "A Gentle Introduction to Positional Encoding in Transformer Models, Part 1," 20 09 2022. [Online]. Available: https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/.

[10] M. Phi, "Illustrated Guide to Transformers Neural Network: A step by step explanation," 28 04 2020. [Online]. Available: https://www.youtube.com/watch?v=4Bdc55j80l8&fbclid=IwAR1yYvR7eXckBEppTktTcgxBp6X3u_FdQPX9G2JVW7-X-eyVprwr6btfA_E.

[11] Q. Pham, "Tìm hiểu mô hình Transformer," 20 03 2020. [Online]. Available: https://pbcquoc.github.io/transformer.

[12] S. Cristina , "How to Implement Scaled Dot-Product Attention from Scratch in TensorFlow and Keras," 26 09 2022. [Online]. Available: https://machinelearningmastery.com/how-to-implement-scaled-dot-product-attention-from-scratch-in-tensorflow-and-keras/.

[10] N. C. Thắng, "Chương 2. Precision, Recall và F Score," 16 06 2020. [Online]. Available: https://www.miai.vn/2020/06/16/oanh-gia-model-ai-theo-cach-mi-an-lien-chuong-2-precision-recall-va-f-score/.

[14] N. Q. Dat, "PhoBERT: Pre-trained language models for Vietnamese," VinAI Research, 2020. [Online]. Available: https://github.com/VinAIResearch/PhoBERT.

[15] "IIIustrated Guilde to Transformer," 13 03 2021. [Online]. Available: https://trituenhantao.io/tin-tuc/minh-hoa-transformer/. [Accessed 27 05 2023].

[16] N. Kiet, N. Vu, N. Anh and N. Ngan, "A Vietnamese Dataset for Evaluating Machine Reading Comprehension," 12 2020. [Online]. Available: https://aclanthology.org/2020.coling-main.233.

[17] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for," 24 05 2019. [Online]. Available: https://arxiv.org/pdf/1810.04805.pdf.

[18] M. NAMVARPOUR , "How does Layer Normalization work?," 2021. [Online]. Available: https://www.kaggle.com/code/halflingwizard/how-does-layer-normalization-work.

[19] N. Q. Dat and N. T. Anh, "PhoBERT: Pre-trained language models for Vietnamese," VinAI Research, 10 05 2020. [Online]. Available: https://arxiv.org/pdf/2003.00744.pdf.

[20] Đ. Đ. Hùng, "Tìm hiểu về kiến trúc transformer," 16 07 2021. [Online]. Available: https://viblo.asia/p/tim-hieu-ve-kien-truc-transformer-Az45byM6lxY.

[21] A. Zhang et al., "Encoder-Decoder Seq2Seq for Machine Translation, in Dive into Deep Learning," 2021. [Online]. Available: http://d2l.ai/chapter_recurrent-modern/seq2seq.html.