TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ TRI THỨC

BÁO CÁO ĐỒ ÁN 1

Môn học: Khai thác dữ liệu văn bản và ứng dụng

Giảng viên lý thuyết: TS. Lê Thanh Tùng
Giảng viên thực hành: ThS. Nguyễn Trần Duy Minh
Nhóm: 01
Thành viên: 20127258 - Hoàng Phước Nguyên
20127655 - Trần Quốc Trung
20127625 - Nguyễn Trương Hoàng Thái
20127597 - Bùi Tấn Phương

Thành phố Hồ Chí Minh – Năm 2024

I. Mục lục

II. Tự đánh giá

A. Đánh giá thành viên

Họ và tên	MSSV	Công việc	Mức độ hoàn thành
Hoàng Phước Nguyên	20127258	- Thực hiện việc thử nghiệm mô hình ở Q2 bằng cách dùng FastText. - Tìm hiểu phương pháp huấn luyện mô hình ở Q3.	100%
Trần Quốc Trung	20127655	- EDA dữ liệu. - Tìm hiểu phương pháp encode mô hình ở Q3.	100%
Nguyễn Trương Hoàng Thái	20127625	- EDA dữ liệu. - Fine-tune và khảo sát mô hình ở Q3.	100%
Bùi Tấn Phương	20127597	- Thử nghiệm mô hình ở Q2 bằng phương pháp Word2Vec - PhoW2V. - Đánh giá và so sánh các models Q2.	100%

B. Mức độ hoàn thành các yêu cầu

Yêu cầu	Mức độ hoàn thành
EDA dữ liệu	100%
Word embedding & Classifier	100%
Thử nghiệm mô hình đã có sẵn	100%

III. Nội dung báo cáo

A. EDA dữ liệu

1. Phân tích non-graphical

- **EDA 1**: Thống kê số dòng, số cột của dữ liệu. Giải thích ý nghĩa các thuộc tính.

	id	question	title	text	label
9051	u3-1558514876_0	Myanmar trước đây có tên là gì	Paul Zinghtung Grawng	Paul Zinghtung Grawng (tên tương tự:Paul Zingt...	False
5936	u5-1562069524_0	Loại thuốc kháng sinh đầu tiên trên thế giới r...		Nhà khoa học mất vài tuần nữa để nuôi thêm mốc...	True
13245	u1-1557194099_0	Người đặt quốc hiệu Đại Cồ Việt là ai	Đại Cồ Việt	Quốc hiệu Đại Cồ Việt do vua Đinh Tiên Hoàng đ...	True
6188	u8-1561103640_1	Ai phát minh ra đèn điện	Phát minh	Phát minh, hay khám phá, phát hiện là việc tìm...	False
5108	u6-1567493405_2	bermuda ở đâu	Tam giác Bermuda	Tam giác Bermuda (Tam giác Béc-mu-đa) , còn ...	True

Hình 1: Một vài mẫu dữ liệu

- Bộ dữ liệu có **18108** dòng và ***5*** cột tương ứng với 5 thuộc tính.
- Mỗi hàng là một mẫu dữ liệu được xác định bằng 1 giá trị định danh và cung cấp một văn bản ngữ cảnh cụ thể để trả lời câu hỏi. Nếu cột **text** trong ngữ cảnh nhất định có thể trả lời thông tin của cột **question** thì **label** sẽ là *True* hoặc ngược lại.
- Theo thông tin quan sát trên dữ liệu, nhóm rút ra được ý nghĩa của từng thuộc tính dữ liệu như sau:

Thuộc tính	Ý nghĩa	Kiểu dữ liệu
id	Giá trị định danh của từng mẫu dữ liệu.	Object (String)
question	Câu hỏi của mẫu dữ liệu.	Object (String)
title	Chủ đề của mẫu dữ liệu.	Object (String)
text	Ngữ cảnh cung cấp thông tin để trả lời cho câu hỏi.	Object (String)
label	Nhãn để xác định ngữ cảnh có thể được dùng để trả lời cho câu hỏi hay không. Nhận giá trị là True hoặc False.	Bool

- **EDA 2**: Thống kê các giá trị bị khuyết và giá trị trùng lặp của dữ liệu.
  - Thống kê giá trị trùng lặp: Dữ liệu có một dòng trùng lặp với thuộc tính là **id,** giá trị trùng lặp là *u8-1560855262_1*. Vì chỉ có 1 giá trị, nhóm sẽ tiến hành loại bỏ mẫu dữ liệu này.

	id	question	title	text	label
6294	u8-1560855262_1	Thủ lĩnh của cuộc khởi nghĩa Yên Thế là ai	Khu di tích khởi nghĩa Yên Thế	Năm 1984, nhân dịp kỷ niệm 100 năm khởi nghĩa ...	True

Hình 2: Mẫu dữ liệu bị trùng lặp

- Thống kê về số lượng giá trị bị khuyết của từng thuộc tính:

	Non-Null Count
id	18108
question	18108
title	18108
text	18108
label	18108

Hình 3: Bảng thống kê số lượng dữ liệu khuyết ở mỗi thuộc tính

+ Từ bảng trên, nhóm thấy rằng tất cả các thuộc tính đều không bị thiếu giá trị, chứng tỏ bộ dữ liệu đã được làm sạch một phần từ trước. Tuy nhiên, khi đọc trong file dữ liệu gốc, nhóm thấy có có nhiều mẫu ở thuộc tính **title** đều mang một chuỗi rỗng, điều này có thể được xem là bị thiếu giá trị vì nó không cung cấp bất kỳ thông tin gì. Nhóm thực hiện kiểm tra xem các thuộc tính có gặp tình trạng tương tự hay không.

	Number of missing value
id	0
question	0
title	995
text	0
label	0

Hình 4: Bảng thống kê số lượng dữ liệu khuyết ở mỗi thuộc tính khi so với chuỗi rỗng

+ Như vậy, bộ dữ liệu chỉ có **995** dòng có giá trị thuộc tính **title** là rỗng. Do số lượng các giá trị bị khuyết từ thuộc tính **title** đều nhỏ hơn 10% trên tổng số dữ liệu, nhóm tiến hành xóa các mẫu chứa giá trị **title** là chuỗi rỗng.

- **EDA 3**: Thống kê số lượng giá trị phân biệt của từng thuộc tính.

	Unique values
id	18107
question	5070
title	9457
text	15957
label	2

Hình 5: Bảng thống kê số lượng dữ liệu phân biệt của từng thuộc tính.

- Từ bảng số liệu, nhóm rút ra rằng có tổng cộng **5070** câu hỏi trên **18107** mẫu dữ

liệu. Điều này có nghĩa là trung bình mỗi câu hỏi sẽ lặp lại 3 lần. Một câu hỏi có thể có nhiều giá trị **title** và **text.**

- **EDA 4**: Chiều dài tối đa và tối thiểu của câu hỏi và ngữ cảnh:

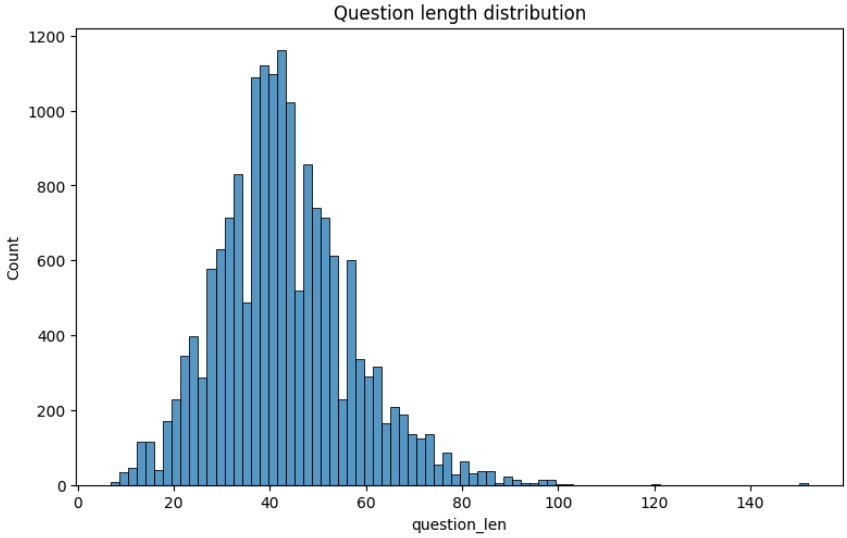
	Column	Maximum Length	Minimum Length
0	question_len	152	7
1	text_len	2609	4

Hình 6: Bảng thống kê chiều dài tối đa và tối thiểu của câu hỏi và ngữ cảnh

- Từ bảng số liệu, ta thấy Chiều dài tối đa của câu hỏi là 152, tối thiểu là 7. Về phía ngữ cảnh thì chiều dài tối đa là 2609, tối thiểu là 4.

2. Phân tích graphical

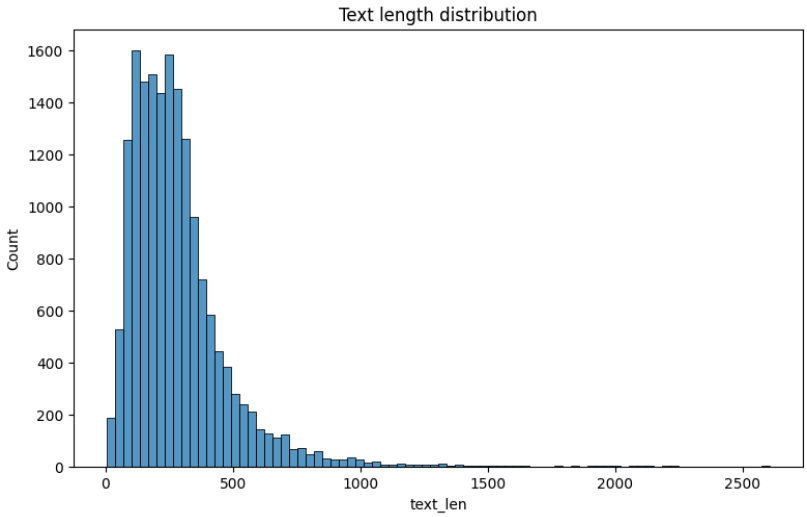
- **EDA 1**: Phân phối độ dài câu hỏi và độ dài ngữ cảnh.
  - Phân phối độ dài câu hỏi:



Hình 7: Biểu đồ thể hiện phân phối của độ dài câu hỏi

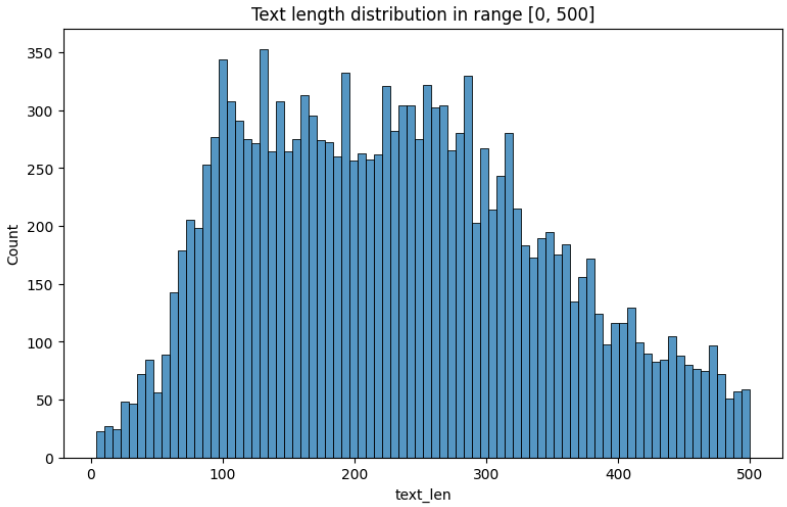
+ Từ biểu đồ trên, nhóm có nhận xét như sau: Các câu hỏi có độ dài tập trung chủ yếu từ 20 đến 60. Có một số câu hỏi có độ dài từ 120 đến hơn 140, các câu này có thể được xem là các câu ngoại lai, có thể xem xét lược bỏ khỏi tập dữ liệu.

  - Độ dài ngữ cảnh:



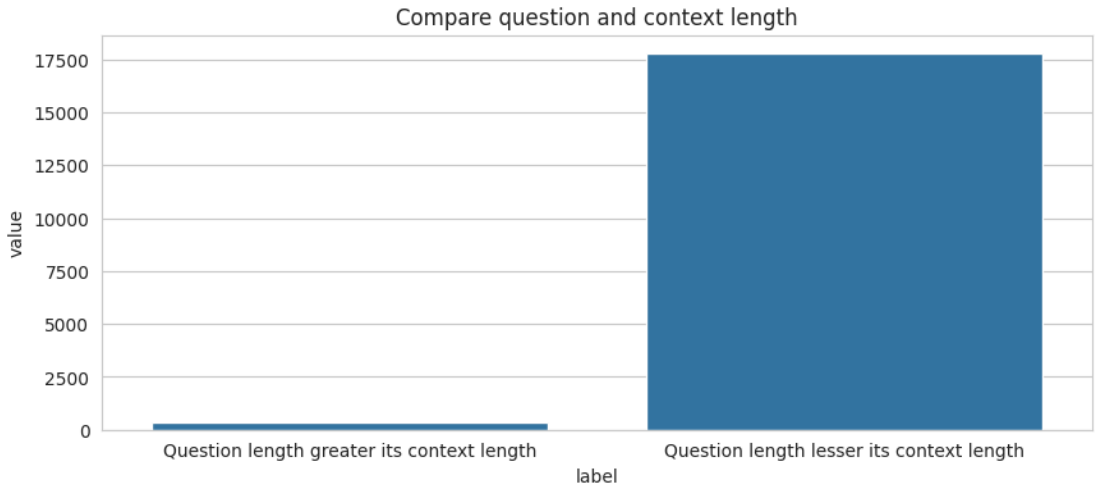
Hình 8: Biểu đồ thể hiện phân phối của độ dài ngữ cảnh

+ Các ngữ cảnh có độ dài ngữ cảnh chủ yếu dao động từ 0 đến 500. Biểu đồ dưới đây sẽ phân tích rõ hơn độ dài ngữ cảnh trong khoảng 0 đến 500:



Hình 9: Biểu đồ thể hiện phân phối của độ dài ngữ cảnh trong khoảng 0 đến 500

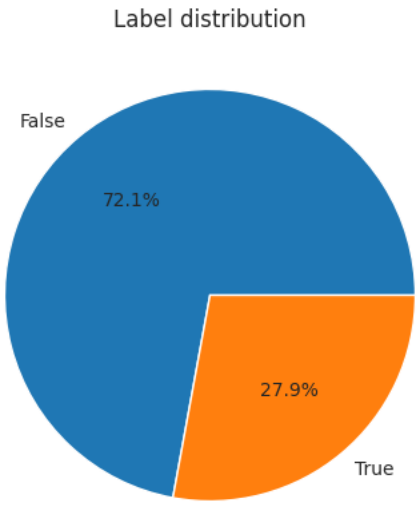
+ Từ hai biểu đồ trên, nhóm có nhận xét như sau: Một số bối cảnh có độ dài dưới 100. Điều đó có nghĩa là độ dài câu hỏi có thể lớn hơn độ dài ngữ cảnh. Nhìn chung, phần lớn ngữ cảnh có độ dài lớn, có thể chứa đủ thông tin để trả lời câu hỏi. Biểu đồ so sánh dưới đây sẽ làm rõ hơn về vấn đề này



Hình 10: Biểu đồ so sánh chiều dài của ngữ cảnh và chiều dài câu hỏi

+ Từ biểu đồ 9, ta thấy đa số chiều dài câu hỏi đều nhỏ hơn ngữ cảnh. Chỉ một số câu hỏi có độ dài lớn hơn ngữ cảnh nhưng tỉ lệ không quá cao. Biết được điều này để chọn ra chiều dài tối đa cho câu hỏi và ngữ cảnh phù hợp để kết hợp chúng lại đưa vào mô hình huấn luyện.

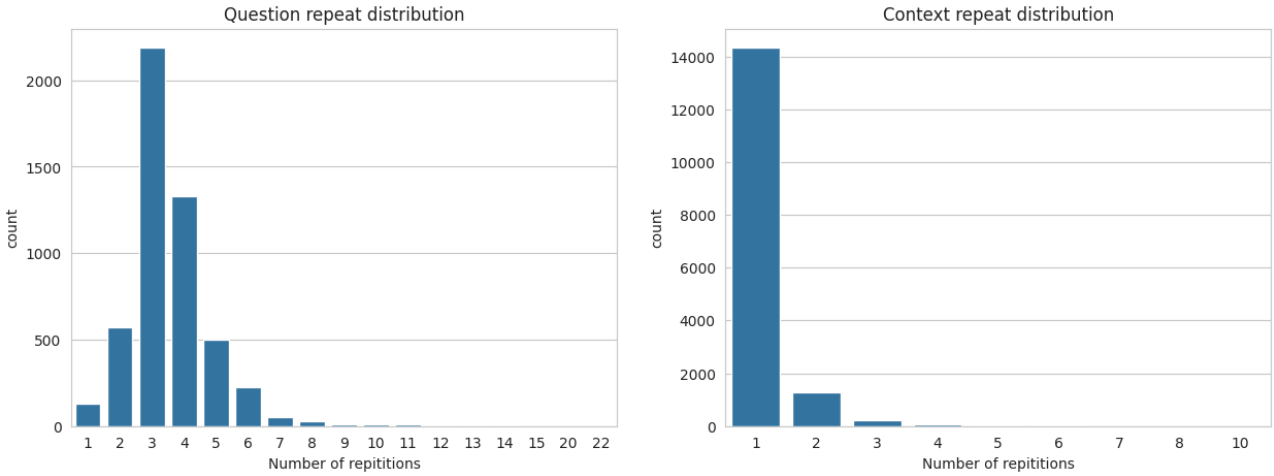
- **EDA 2:** Phân bố nhãn đầu ra.



Hình 11: Biểu đồ so sánh tỷ lệ các nhãn trong dữ liệu đầu ra

- Từ biểu đồ có thể thấy rằng dữ liệu khá mất cân bằng. Tỉ lệ *False*:*True* là 7:3. Vậy nên khi đưa vào mô hình huấn luyện, ta nên cân nhắc việc sampling lại các nhãn để cân bằng.

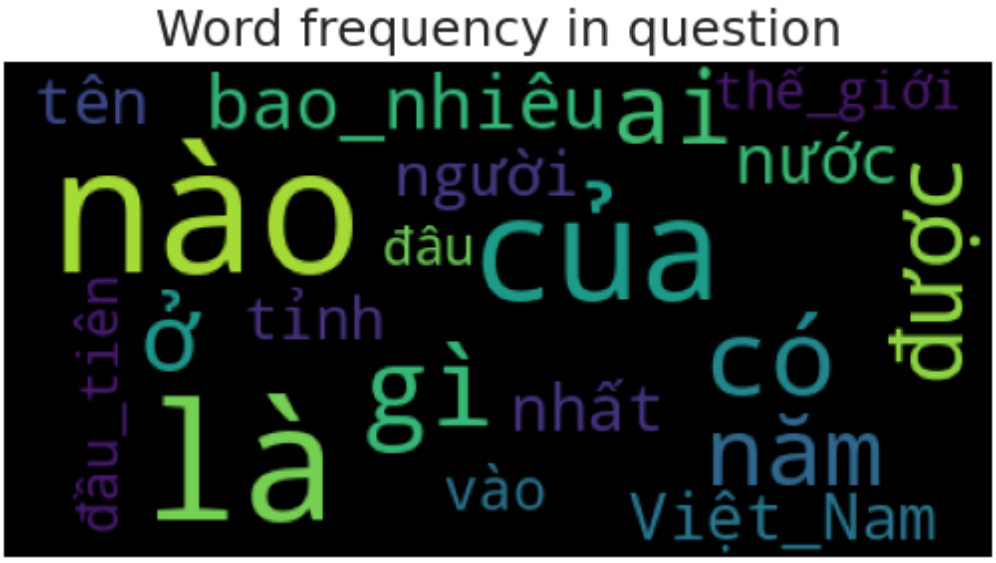
- **EDA 3**: Phân phối của số lần lặp lại của một câu hỏi và ngữ cảnh.



Hình 12: Biểu đồ phân phối số lần lặp lại của câu hỏi và ngữ cảnh

- Từ biểu đồ, ta thấy các câu hỏi lặp lại nhiều trong khoảng từ 1 đến 6 lần. Một số ít câu hỏi lặp lại nhiều hơn 7 lần. Ngữ cảnh lặp lại chỉ từ 1 đến 3 lần. Một số ít lặp lại nhiều hơn 4 lần. Nhìn chung các câu hỏi lặp lại ở mức vừa phải. Các ngữ cảnh lặp lại ít.

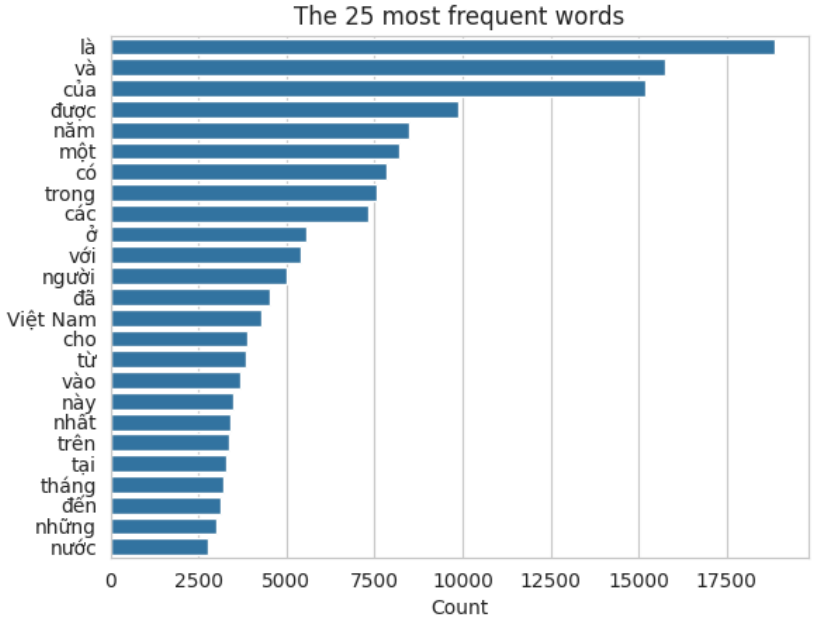
- **EDA 4**: Những từ xuất hiện nhiều trong câu hỏi:



Hình 12: Biểu đồ thể hiện những từ xuất hiện nhiều trong câu hỏi

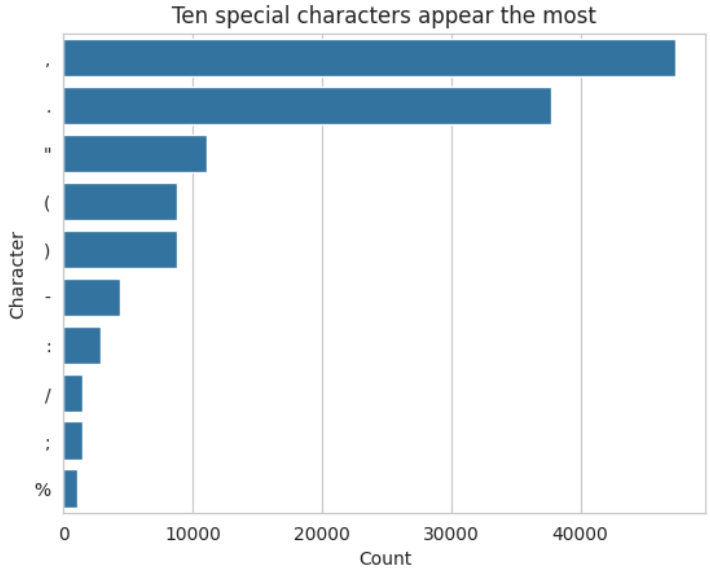
- Từ ảnh trên, ta thấy những từ *nào, là, của, ai, gì, bao nhiêu* ... là những từ xuất hiện nhiều. Dễ hiểu vì đó là những từ nghi vấn trong tiếng Việt. Ngoài ra có những danh từ như *Việt Nam, tỉnh, nước, thế giới* cũng xuất hiện nhiều trong câu hỏi do bộ dữ liệu có thể được lấy từ nhiều bài báo Việt Nam khác nhau.

- **EDA 5**: Những từ xuất hiện nhiều trong ngữ cảnh



Hình 13: Biểu đồ thể hiện 25 từ xuất hiện nhiều nhất trong ngữ cảnh

- Biểu đồ cho thấy những từ *là, và, của, có, ...* là những từ nối trong tiếng Việt, xuất hiện nhiều nhất trong ngữ cảnh. Những danh từ xuất hiện nhiều là *năm, một, người, Việt Nam, tháng, ...*
- 10 ký tự đặc biệt xuất hiện nhiều nhất trong các ngữ cảnh được thể hiện qua biểu đồ như sau:



Hình 13: Biểu đồ thể hiện 10 ký tự đặc biệt xuất hiện nhiều trong ngữ cảnh

- *Dấu chấm, phẩy, ngoặc kép và ngoặc đơn* là các ký tự xuất hiện nhiều nhất. Ngoài ra còn có các ký tự đặc biệt khác: *gạch ngang, hai chấm, xẹt chéo, chấm phẩy, phần trăm.*
- Những dấu này và những từ xuất hiện nhiều trước đó sẽ được xử lý bằng cách xóa bỏ khi thực hiện bước tiền xử lý dữ liệu để đưa vào mô hình.

B. Thử nghiệm trên mô hình tự xây dựng

Dựa vào MSSV của thành viên trong nhóm, các mô hình được sử dụng ở phần này là **Word2Vec** cho Embedding và **CNN + Fully Connected Layer** cho Classifier.

Trước khi đi vào huấn luyện từng phần cụ thể, nhóm thực hiện các bước tiền xử lý trên dữ liệu như sau:

- Loại bỏ những mẫu trùng nhau và những mẫu bị thiếu dữ liệu như đã đề cập ở phần EDA dữ liệu.
- Chuẩn hóa Unicode tương thích với tiếng Việt.
- Kiểm tra một từ có phải là từ tiếng việt hay không và chuẩn hóa từ đó về cách gõ dấu tiếng Việt giống như các từ có trong mô hình pretrain embedding.
- Loại bỏ các ký tự đặc biệt và các từ stopwords.
    - Các stopwords được lấy từ liên kết này.
- Chuyển các chữ cái viết hoa về chữ cái thường.
- Đưa các nhãn True và False về 1 và 0.

1. Phần Embedding:

Trong phần này, nhóm sử dụng 2 mô hình pretrain embedding là Word2Vec và FastText, lý do chúng được chọn là đều hỗ trợ tốt cho Tiếng Việt. Ngoài hai mô hình được kể trên, nhóm còn sử dụng các lớp Embedding có sẵn trong Pytorch. Sau đây là mô tả của hai mô hình embedding chính:

a. Word2Vec:

Nhóm chọn sử dụng các vector đã được pretrain từ model **PhoW2V**. **PhoW2V** là mô hình Word2Vec được huấn luyện trên 1 corpus tiếng Việt 20GB, mô hình này được huấn luyện với 4 kiểu embedding khác nhau có thông số như sau:

- PhoW2V syllables - level 100 dims.
- PhoW2V syllables - level 300 dims.
- PhoW2V word - level 100 dims.
- PhoW2V word - level 300 dims.

Trong đồ án này, nhóm sử dụng mô hình pretrain **PhoW2V** word - level 300 dims với các vector embedding có kích thước là 300.

b. FastText:

Ngoài PhoW2V hỗ trợ tốt phần embedding dành cho tiếng Việt, FastText cũng có chức năng tương tự. FastText là một thư viện mã nguồn mở được Facebook AI Research phát triển dành cho các tác word embeddings và word classification. Mô hình này cho phép tạo ra các thuật toán học có giám sát hoặc không giám sát để thu được các biểu diễn vector cho các từ. FastText hỗ trợ cả mô hình CBOW và Skip-gram.

Có thể coi FastText là phần mở rộng của Word2vec. Sở dĩ nhóm chọn mô hình này là bởi FastText có thể khắc phục nhược điểm là bỏ qua các cấu trúc nội bộ của ngôn ngữ, đặc biệt là

với các ngôn ngữ phong phú về mặt hình thái như tiếng Việt, đều này khiến cho việc hiểu quan hệ cú pháp giữa các từ trở nên bị hạn chế. FastText không chỉ đơn thuần cung cấp các biểu diễn vector cho từng từ mà nó còn tạo ra biểu diễn cho các nhóm ký tự (character n-grams) trong từ. Bằng cách tính trung bình của các biểu diễn này, mô hình đã tạo ra biểu diễn vector cho từng từ, không chỉ dựa trên từng đơn vị từ mà còn dựa trên cấu trúc nhúng ký tự bên trong.

Hiện tại, FastText có hỗ trợ mô hình word vector cho 157 ngôn ngữ, trong đó có tiếng Việt và có thể được tải về ở trang này. Mô hình FastText này được huấn luyện bằng cách sử dụng CBOW với position weight, kích thước là 300 chiều.

2. Phần Classifier:

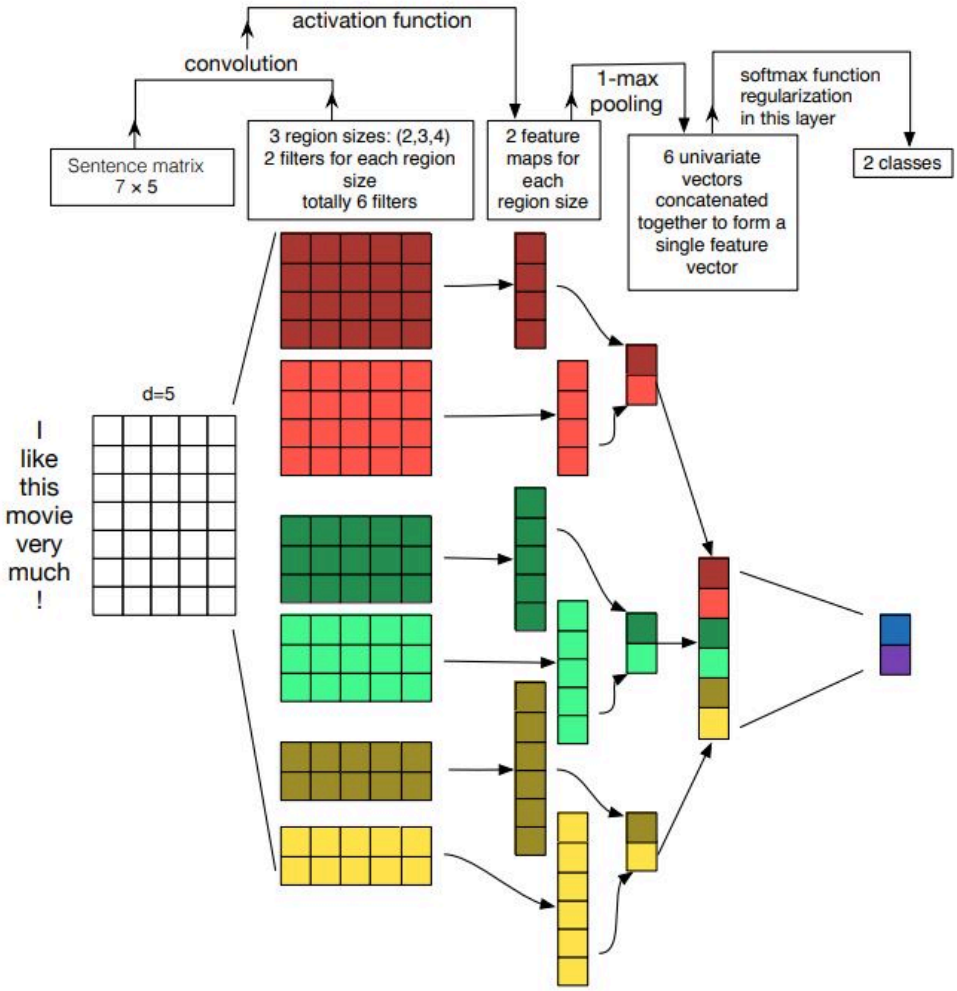
Mô hình classifier sẽ bao gồm 2 phần chính và lần lượt có cấu hình như sau:

- CNN (Convolutional Neural Network): Sử dụng một mạng CNN đơn giản để tiếp tục embedding các sentence, cài đặt với tham số như sau:
    - 3 lớp kernel (hoặc filter) với kích thước 100
    - Với mỗi kernel, ta sử dụng hàm pooling là 1-max pooling.
    - Output của CNN là một vector embedding của 1 câu với kích thước 300.
- Fully - connected layers: sau khi embedding câu, ta tiếp tục đẩy các vector câu đó vào một mạng kết nối đầy đủ để thực hiện nhiệm vụ phân loại. Cụ thể trong đồ án này sẽ có 1 lớp fully - connected với kích thước là kích thước vector embedding của câu - 300.
- Dropout: Cuối cùng, thực hiện dropout với tỉ lệ là 50% để giúp model tránh overfitting.

Sau đây là bảng tóm tắt tham số:

Mô tả kiến trúc	Giá trị
Word vector đầu vào	FastText/PhoW2
Embedding size	300
Filter size	(3, 4, 5)
Số các filters	(100, 100, 100)
Hàm kích hoạt	ReLU
Pooling	1-max pooling
Dropout rate	0.5

Cách thức hoạt động của mạng có thể được giải thích ở hình bên dưới đây (3 kích thước filter là 2, 3, 4 và mỗi kích thước có 2 filter).



Hình 14: KIến trúc của CNN (theo Zhang, 2015)

- Ví dụ, để phân loại câu input đầu vào là "I like this movie very much" (N = 7 tokens) và chiều của word vector là d = 5. Sau khi áp embedding layer vào token ids đầu vào, mẫu câu được biểu diễn dưới dạng tensor 2D có shape là (7, 5) giống như một hình ảnh. Ký hiệu: $x_{emb} \in \mathbb{R}^{7x5}$
- Sau đó, ta sử dụng 1D convolutional để trích xuất các đặc điểm từ câu. Trong ví dụ này, ta có tổng cộng 6 filter, mỗi filter có shape là (f_i, d) với f_i là kích thước filter, i chạy từ 1 đến 6. Mỗi filter sẽ quét qua x_{emb} và trả về một feature map:
  - $x_{conv_i} = Conv1D(x_{emb}) \in \mathbb{R}^{N - f_i + 1}$
- Tiếp đến sử dụng hàm kích hoạt ReLU cho x_{conv_i} và sử dụng max-pooling để giảm chiều các feature map thành 1 scalar. Ta nối các scalar này lại thành một vector rồi đưa đến một lớp fully connected layer để tính điểm cuối cùng cho các lớp:

- $x_{pool_i} = MaxPool(ReLU(x_{conv_i})) \in \mathbb{R}$
- $x_{fc} = concat(x_{pool_i}) \in \mathbb{R}^6$

- Cuối cùng, ta sử dụng một fully connected layer với trọng số ma trận là $W_{fc} \in \mathbb{R}^{2x6}$ và dropout để tính toán logits - một vector có độ dài 2 để giữ điểm số cho 2 lớp.
  - $logits = Dropout(W_{fc}x_{fc}) \in \mathbb{R}^2$

3. Mô tả thí nghiệm và độ hiệu quả:

Nhóm tiến hành thử nghiệm trên các mô hình và chúng chỉ khác nhau ở phương pháp embedding và giống nhau ở các lớp CNN (sentence embedding) + classifier (fully-connected). Cụ thể:

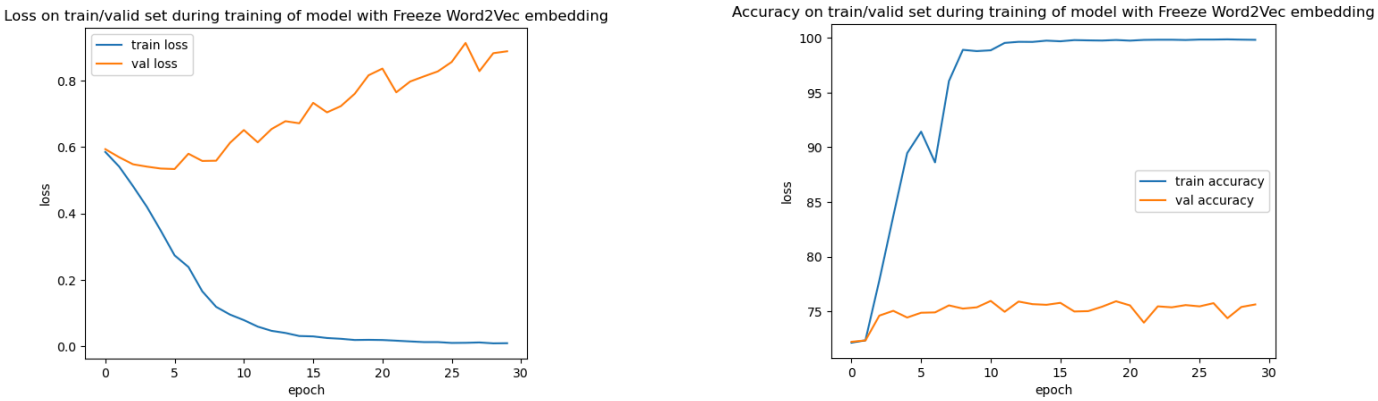
- Model Freeze Word2Vec: Model này embedding bằng Word2Vec và không hiệu chỉnh trong số embedding suốt quá trình huấn luyện.
- Model Fine-tune Word2Vec: Ngược lại với mô hình trên, model này sẽ hiệu chỉnh trọng số lớp embedding trong quá trình huấn luyện
- Model Freeze FastText: Tương tự Freeze Word2Vec, nhưng các vector embedding là từ model FastText.
- Model Fine-tune FastText: Cho phép cập nhật trọng số trong quá trình huấn luyện.
- Model Random Embedding: sử dụng lớp embedding mặc định có trong Pytorch.

Quá trình huấn luyện có sử dụng GPU do Google Colab cung cấp để tận dụng tính toán song song, giảm thời gian huấn luyện. Huấn luyện trên 30 epochs.

a. Quá trình huấn luyện:

Sự thay đổi độ lỗi (loss) và độ chính xác (accuracy) của các models trên tập train và valid được ghi nhận sau đây:

- Freeze Word2Vec:

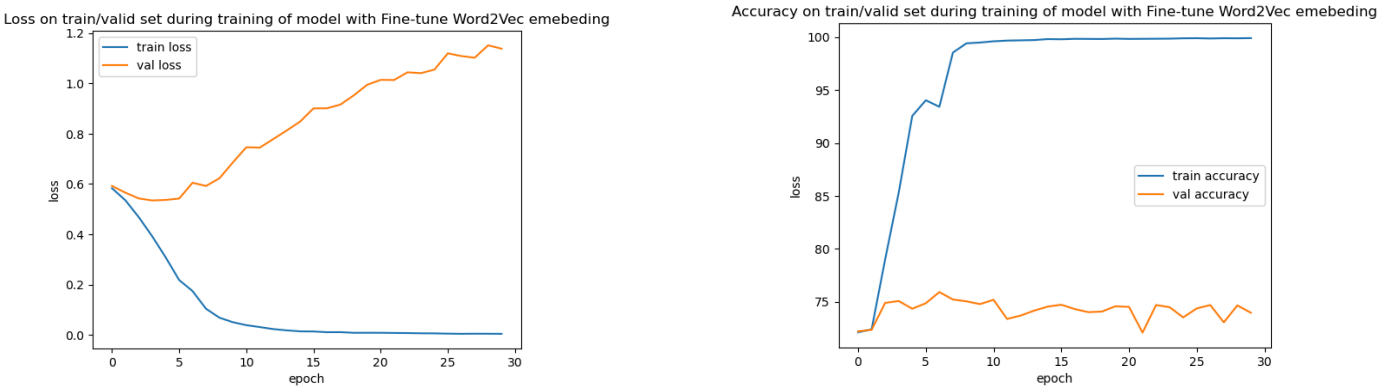


Hình 15: Sự thay đổi độ lỗi và độ chính xác của model Freeze Word2Vec

Epoch	Train Loss	Val Loss	Val Acc	Elapsed
1	0.595312	0.593538	72.21	22.82
2	0.573455	0.569253	72.32	22.84
3	0.537008	0.547996	74.60	27.38
4	0.493169	0.540973	75.04	22.36
5	0.437581	0.535182	74.43	22.64
6	0.380491	0.533716	74.87	23.25
7	0.319665	0.579802	74.90	22.23
8	0.264221	0.558051	75.54	29.39
9	0.215058	0.558764	75.25	30.31
10	0.174893	0.612665	75.36	29.19
11	0.147353	0.651197	75.95	24.33
12	0.124286	0.614181	74.96	25.02
13	0.105306	0.654253	75.89	27.72
14	0.094258	0.677807	75.65	23.33
15	0.079786	0.671516	75.59	22.02
16	0.074879	0.733155	75.77	21.31
17	0.068117	0.704452	74.98	25.90
18	0.065985	0.723230	75.01	28.58
19	0.058987	0.760386	75.42	23.97
20	0.049636	0.816153	75.92	23.33
21	0.052981	0.836343	75.54	26.04
22	0.044532	0.764848	73.96	26.57
23	0.046721	0.797381	75.45	30.28
24	0.042554	0.812887	75.36	30.45
25	0.035399	0.827897	75.57	27.52
26	0.036258	0.856294	75.45	24.76
27	0.032387	0.913572	75.74	22.05
28	0.034608	0.828629	74.37	27.12
29	0.026467	0.882708	75.39	25.85
30	0.029785	0.888453	75.62	24.38

Hình 16: Bảng thay đổi độ lỗi và độ chính xác của model Freeze Word2Vec qua các epoch.

- Fine-tune Word2Vec:

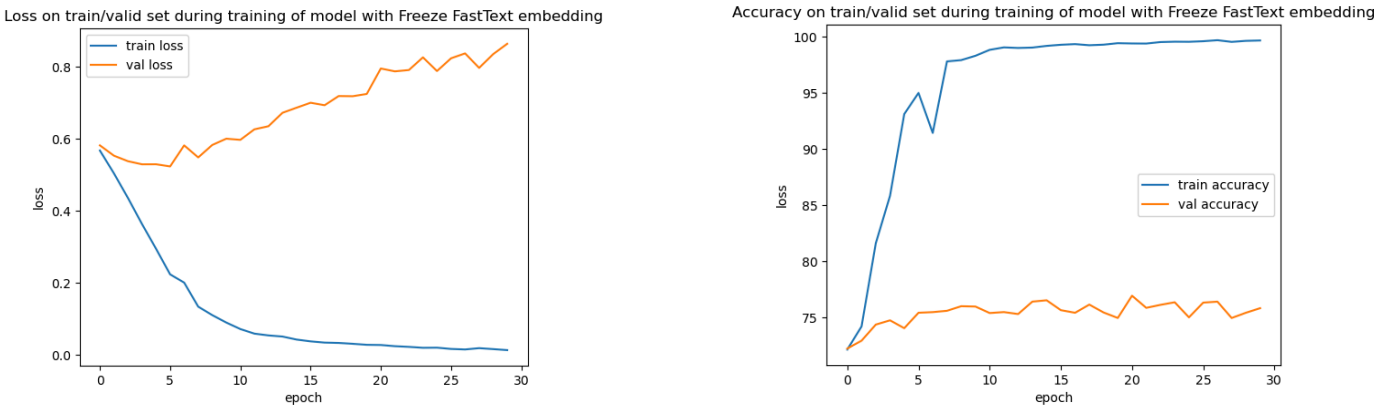


Hình 17: Sự thay đổi độ lỗi và độ chính xác của model fine-tune Word2Vec

Epoch	Train Loss	Val Loss	Val Acc	Elapsed
1	0.594908	0.592211	72.21	39.15
2	0.570151	0.565274	72.35	42.13
3	0.529568	0.542466	74.90	46.45
4	0.477616	0.534413	75.07	46.14
5	0.409223	0.536582	74.34	41.87
6	0.336441	0.541959	74.86	44.71
7	0.260913	0.604808	75.92	48.19
8	0.197695	0.592079	75.21	49.29
9	0.146746	0.622652	75.04	49.12
10	0.110503	0.685614	74.78	50.84
11	0.083465	0.745851	75.19	44.55
12	0.067800	0.744461	73.38	46.94
13	0.057294	0.778319	73.70	48.07
14	0.048788	0.812318	74.16	50.23
15	0.037407	0.847752	74.54	43.63
16	0.037544	0.900582	74.72	41.59
17	0.033299	0.900965	74.31	44.86
18	0.029207	0.915500	74.02	42.65
19	0.026919	0.952378	74.08	44.99
20	0.021358	0.994398	74.57	48.56
21	0.025087	1.013811	74.52	44.59
22	0.018505	1.013288	72.09	43.63
23	0.020708	1.043894	74.69	48.72
24	0.017449	1.040547	74.49	42.49
25	0.015698	1.054666	73.52	42.87
26	0.015183	1.119281	74.37	42.84
27	0.014926	1.108483	74.69	43.16
28	0.015717	1.101882	73.05	45.28
29	0.011795	1.151295	74.66	57.51
30	0.012285	1.137773	73.96	45.56

Hình 18: Bảng thay đổi độ lỗi và độ chính xác của model fine-tune Word2Vec qua các epoch.

- Freeze FastText

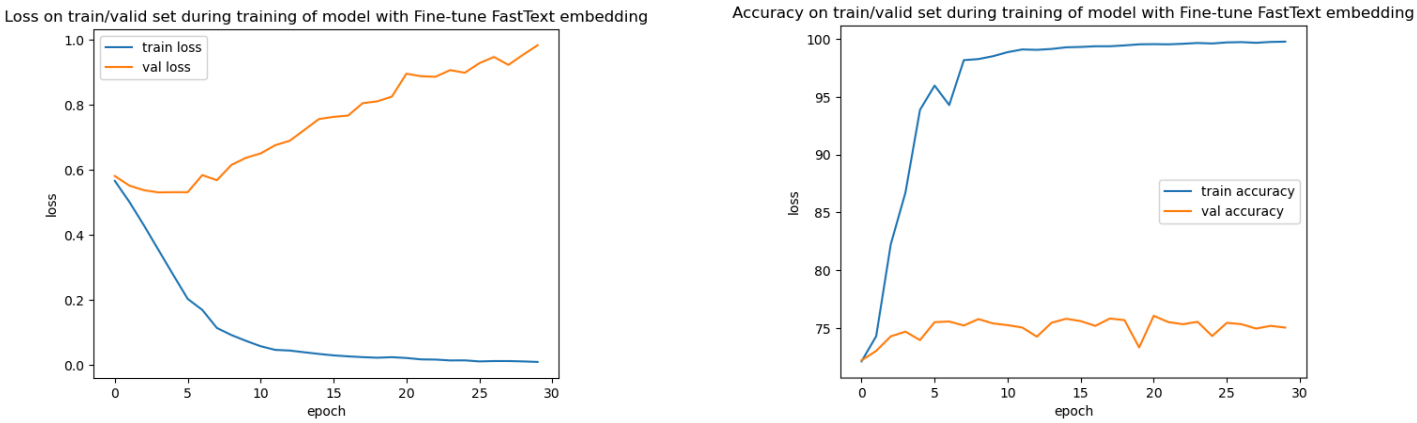


Hình 19: Sự thay đổi độ lỗi và độ chính xác của model Freeze FastText

Epoch	Train Loss	Val Loss	Val Acc	Elapsed
1	0.591206	0.581741	72.21	27.91
2	0.554577	0.553084	72.91	24.59
3	0.511715	0.537747	74.34	23.08
4	0.461943	0.529165	74.72	22.50
5	0.403980	0.529301	74.02	22.68
6	0.351248	0.523270	75.40	22.19
7	0.298141	0.581679	75.45	24.99
8	0.253167	0.548269	75.57	25.74
9	0.217909	0.582779	75.98	26.22
10	0.187944	0.600250	75.95	25.17
11	0.166772	0.597009	75.37	23.81
12	0.151760	0.626224	75.46	23.35
13	0.133317	0.634789	75.28	23.77
14	0.123970	0.672126	76.39	25.55
15	0.111917	0.686195	76.51	28.78
16	0.102675	0.699985	75.63	25.06
17	0.096801	0.693107	75.40	23.73
18	0.099699	0.718568	76.13	26.90
19	0.089254	0.718102	75.42	30.65
20	0.080565	0.724398	74.93	27.57
21	0.075000	0.795044	76.92	24.86
22	0.069207	0.787105	75.84	25.93
23	0.068806	0.790810	76.10	24.74
24	0.065747	0.825949	76.33	24.62
25	0.058770	0.788017	74.99	23.92
26	0.055998	0.823301	76.30	28.10
27	0.058970	0.836991	76.39	26.01
28	0.059221	0.796712	74.93	26.98
29	0.050357	0.834930	75.40	28.77
30	0.051343	0.863644	75.81	24.80

Hình 20: Bảng thay đổi độ lỗi và độ chính xác của model Freeze FastText qua các epoch.

- Fine-tune FastText

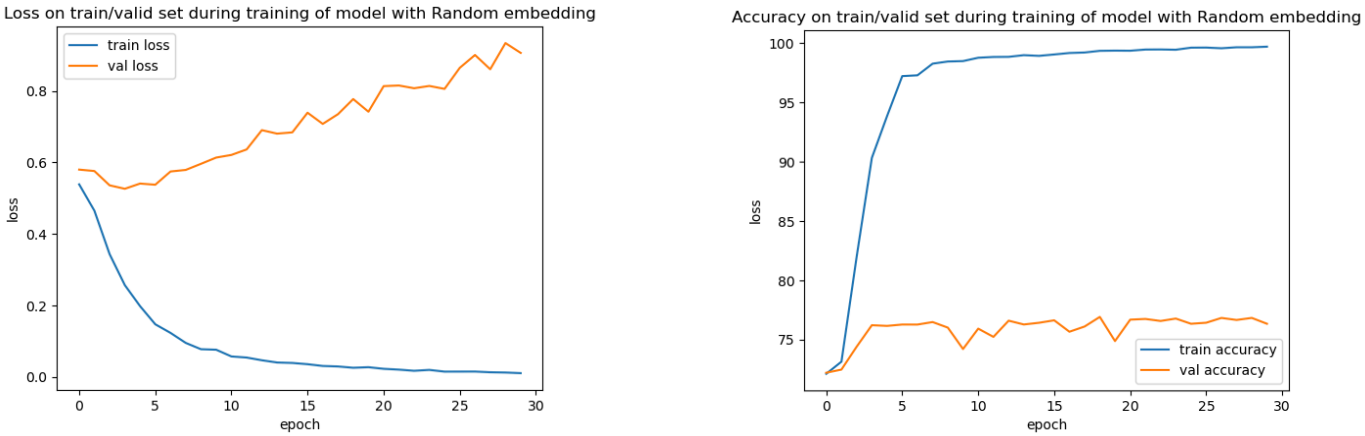


Hình 21: Sự thay đổi độ lỗi và độ chính xác của model Fine-tune FastText

Epoch	Train Loss	Val Loss	Val Acc	Elapsed
1	0.590769	0.580486	72.21	35.04
2	0.551789	0.551011	73.03	34.14
3	0.507150	0.536853	74.29	35.83
4	0.454858	0.530179	74.69	37.13
5	0.391838	0.530732	73.96	36.84
6	0.334670	0.530688	75.51	36.55
7	0.273062	0.583351	75.57	36.26
8	0.224666	0.567600	75.22	35.98
9	0.188565	0.614625	75.78	36.71
10	0.157541	0.636537	75.40	35.78
11	0.135409	0.649981	75.25	36.47
12	0.124271	0.675225	75.05	35.66
13	0.106067	0.688961	74.26	35.18
14	0.100641	0.722150	75.46	35.72
15	0.085295	0.755627	75.80	35.60
16	0.079821	0.762232	75.60	35.91
17	0.073781	0.766370	75.19	35.81
18	0.071572	0.804402	75.83	35.59
19	0.063625	0.810107	75.69	36.18
20	0.053950	0.824442	73.32	36.23
21	0.052811	0.895051	76.07	35.36
22	0.047014	0.887238	75.51	35.78
23	0.050729	0.885660	75.33	36.10
24	0.041998	0.906106	75.54	35.61
25	0.039621	0.897941	74.31	35.38
26	0.038711	0.927233	75.45	35.61
27	0.035862	0.946514	75.34	35.57
28	0.037901	0.922114	74.96	35.80
29	0.033129	0.953327	75.19	36.23
30	0.030030	0.982554	75.04	35.96

Hình 22: Bảng thay đổi độ lỗi và độ chính xác của model Fine-tune FastText qua các epoch.

- Random Embedding



Hình 23: Sự thay đổi độ lỗi và độ chính xác của model random embedding

```
 Epoch   |  Train Loss  |  Val Loss  |  Val Acc  |  Elapsed
------------------------------------------------------------
/Users/btp712/anaconda3/lib/python3.11/site-packages/torch/r
  torch.embedding_renorm_(weight.detach(), input, max_norm,
   1    |   0.601797   |  0.579674  |   72.21   |   38.71
   2    |   0.536366   |  0.575703  |   72.47   |   40.94
   3    |   0.471735   |  0.535704  |   74.40   |   40.80
   4    |   0.408350   |  0.526052  |   76.21   |   40.82
   5    |   0.353386   |  0.540665  |   76.15   |   40.87
   6    |   0.312207   |  0.537418  |   76.27   |   41.10
   7    |   0.271452   |  0.574414  |   76.27   |   60.49
   8    |   0.244455   |  0.578779  |   76.48   |   60.14
   9    |   0.205888   |  0.595846  |   76.01   |   41.74
  10    |   0.190258   |  0.613364  |   74.20   |   45.01
  11    |   0.168242   |  0.620718  |   75.92   |   40.53
  12    |   0.157497   |  0.636101  |   75.23   |   39.74
  13    |   0.146311   |  0.690025  |   76.59   |   41.55
  14    |   0.130950   |  0.680279  |   76.27   |   40.94
  15    |   0.120118   |  0.683819  |   76.42   |   41.87
  16    |   0.108813   |  0.738784  |   76.63   |   48.59
  17    |   0.103897   |  0.707519  |   75.66   |   45.39
  18    |   0.093845   |  0.734706  |   76.10   |   44.54
  19    |   0.086104   |  0.776982  |   76.91   |   43.46
  20    |   0.090405   |  0.741578  |   74.87   |   43.01
  21    |   0.081071   |  0.813154  |   76.68   |   44.36
  22    |   0.075122   |  0.814888  |   76.74   |   42.29
  23    |   0.066650   |  0.807195  |   76.57   |   41.78
  24    |   0.064301   |  0.813628  |   76.77   |   42.10
  25    |   0.064892   |  0.805434  |   76.33   |   42.22
  26    |   0.055986   |  0.864389  |   76.42   |   42.49
  27    |   0.056843   |  0.900253  |   76.83   |   42.31
  28    |   0.049873   |  0.860374  |   76.65   |   42.51
  29    |   0.050791   |  0.933479  |   76.83   |   43.16
  30    |   0.042331   |  0.906012  |   76.33   |   42.71
```

Hình 24: Bảng thay đổi độ lỗi và độ chính xác của model Fine-tune FastText qua các epoch.

Bảng giá trị accuracy tốt nhất cho 5 mô hình**:**

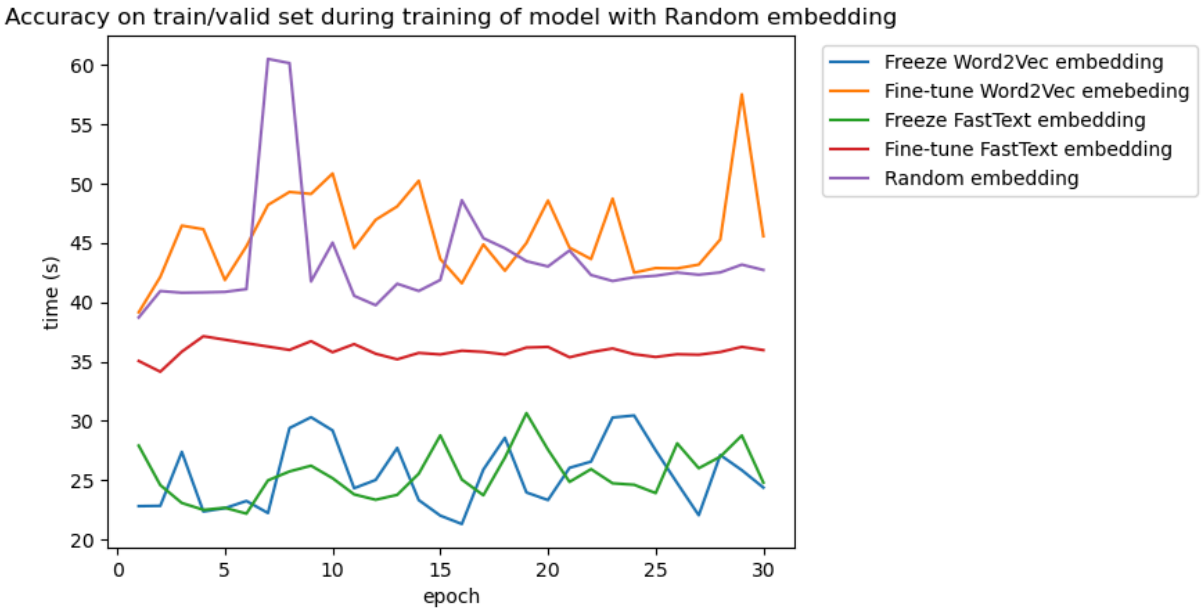
Tên mô hình thử nghiệm	Giá trị Accuracy tốt nhất
Freeze Word2Vec embedding	75.95
Fine-tune Word2Vec embedding	75.92
Freeze FastText embedding	76.92
Fine-tune FastText embedding	76.07
Random embedding	76.91

Nhận xét:

- Trong quá trình huấn luyện, tất cả các model trên đều giảm độ lỗi trên tập train.
- Tuy nhiên với tập valid thì độ lỗi chỉ giảm nhẹ ở vài epoch đầu và tăng mạnh ở các epoch sau.
- Tương tự độ lỗi, accuracy của các model cũng tăng gần tới mức tuyệt đối (100%) trên tập train. Ngoài ra trong quá trình train thì accuracy sẽ có xu hướng giảm lại tại vài epoch từ 5 - 10.
- Độ chính xác của các model cũng chỉ tăng ở vài epoch đầu và sẽ chững lại ở quanh mức 75%.

→ Các model đều overfitting bất chấp sử dụng phương pháp embedding nào.

Thời gian huấn luyện các models:



Hình 25: Biểu đồ thể hiện thời gian huấn luyện của từng model.

Nhận xét:

- 2 models giữ nguyên trọng số lớp embedding có thời gian huấn luyện ngắn, trong khoảng 20-30s/epoch.
- 3 models phải train trọng số cho lớp embedding có thời gian dài hơn, tuy nhiên với model Fine-tune FastText thì thời gian chỉ tầm 35s/epoch; 2 models còn lại (Fine tune Word2Vec, Random Embedding) có thời gian là dài nhất: chủ yếu 40-50s/epoch, có những epoch dài tới 60s.

→ Các model không cần huấn luyện lớp embedding có thời gian huấn luyện ngắn hơn đáng kể các model phải huấn luyện.

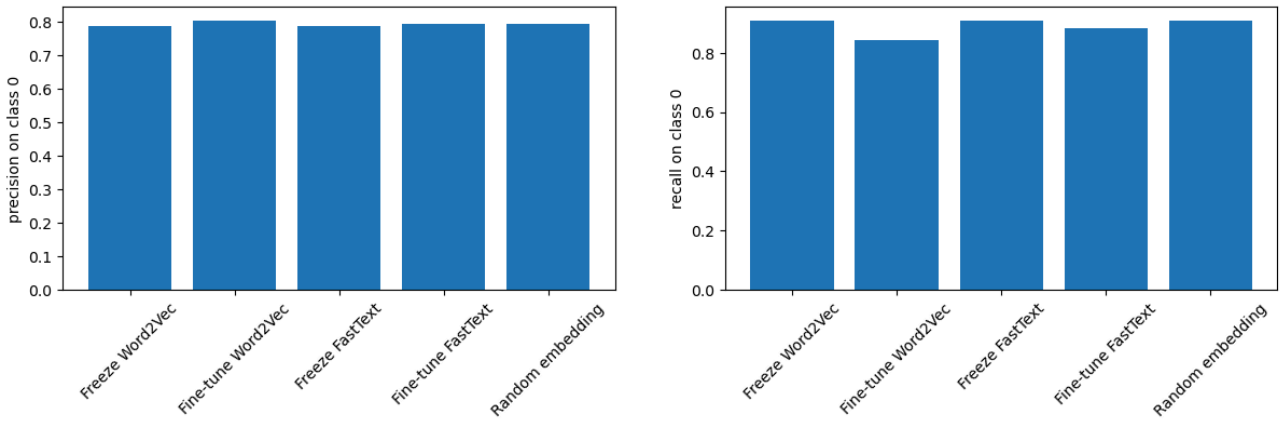
b. Đánh giá mô hình sau quá trình huấn luyện

Sau khi các models đã được huấn luyện, ta sẽ thực hiện đánh giá các mô hình trên tập valid/test bằng các độ đo dành cho bài toán phân loại như sau:

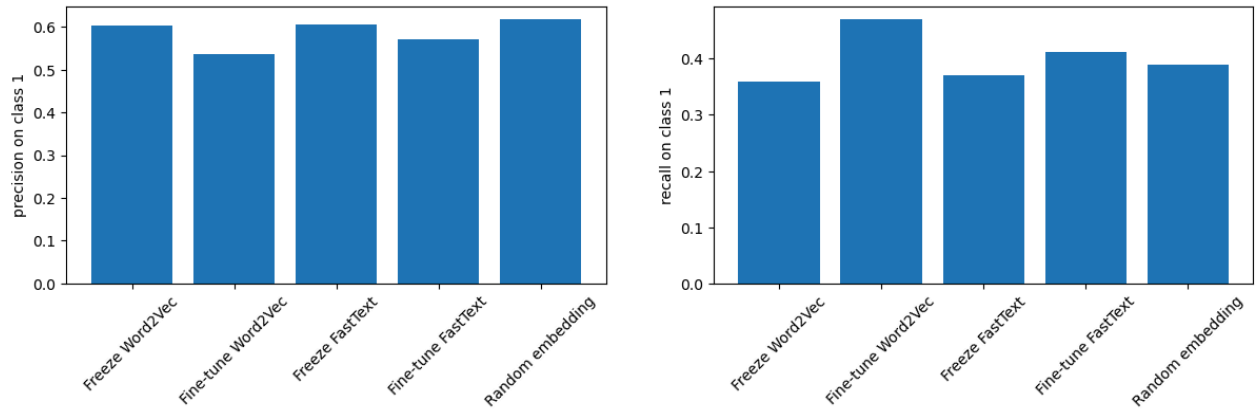
- Accuracy
- Precision trên class 1
- Precision trên class 0
- Recall trên class 1
- Recall trên class 0

Đánh giá trên tập valid:

	Freeze Word2Vec embedding	Fine-tune Word2Vec embedding	Freeze FastText embedding	Fine-tune FastText embedding	Random embedding
Accuracy	0.756283	0.739626	0.758036	0.750438	0.763296
Precision on class 0	0.786489	0.805255	0.789362	0.795396	0.794191
Recall on class 0	0.909348	0.843383	0.906920	0.881020	0.907325
Precision on class 1	0.603540	0.535971	0.605489	0.570803	0.617696
Recall on class 1	0.358570	0.470032	0.371188	0.411146	0.389064



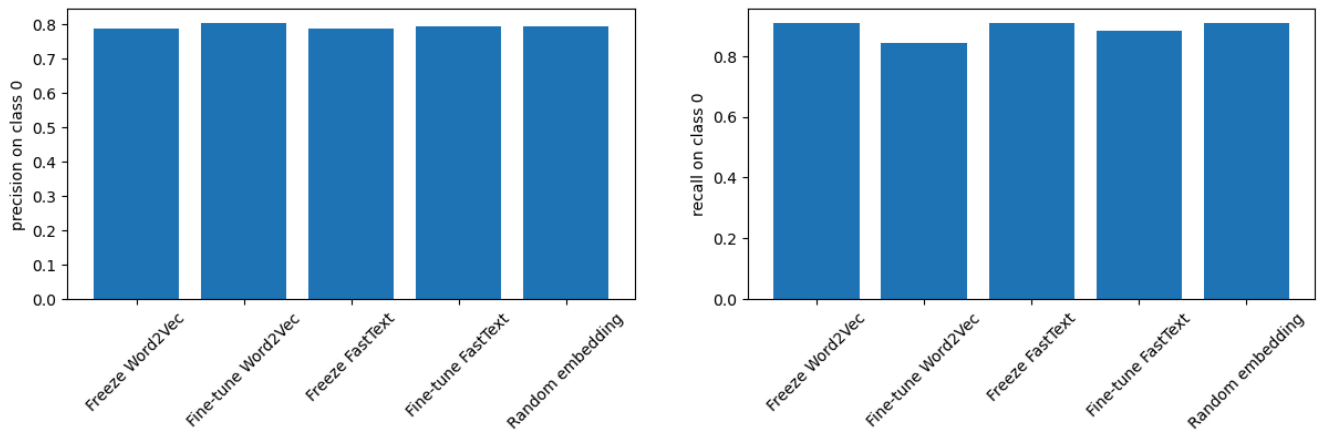
Hình 26: Biểu đồ so sánh độ đo Precision và Recall trên class 0 của từng model.



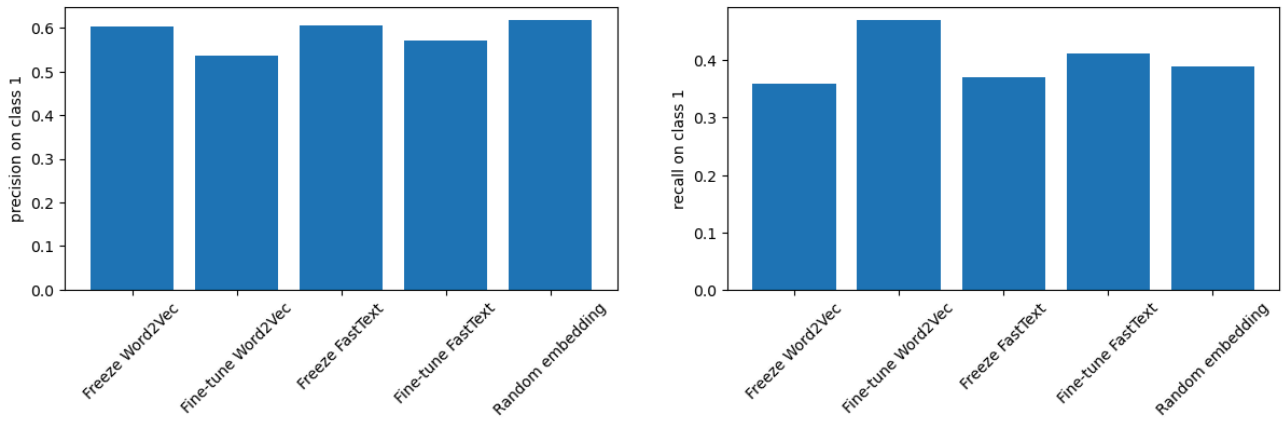
Hình 27: Biểu đồ so sánh độ đo Precision và Recall trên class 1 của từng model.

Đánh giá trên tập test:

	Freeze Word2Vec embedding	Fine-tune Word2Vec embedding	Freeze FastText embedding	Fine-tune FastText embedding	Random embedding
Accuracy	0.754673	0.728972	0.762850	0.761682	0.769276
Precision on class 0	0.779381	0.785026	0.784584	0.794853	0.788966
Recall on class 0	0.919708	0.858881	0.924574	0.901865	0.927818
Precision on class 1	0.614786	0.520661	0.640927	0.613419	0.660305
Recall on class 1	0.329854	0.394572	0.346555	0.400835	0.361169



Hình 28: Biểu đồ so sánh độ đo Precision và Recall trên class 0 của từng model.



Hình 29: Biểu đồ so sánh độ đo Precision và Recall trên class 1 của từng model.

Nhận xét:

- Như đã đề cập ở trên, các models chỉ có accuracy từ 72-76% trên cả tập valid lẫn test.
- Tuy nhiên, các models đạt được các giá trị recall trên class False khá tốt, từ 85-92% → Việc phát hiện các văn bản không liên quan của các models là khá tốt.
- Nhưng độ precision trên cả tập valid lẫn test đều chưa tốt khi chỉ đạt từ 78-80% → models dự đoán nhiều class False nên recall trên class 0 tốt.
- Và với class True, tất cả các models đều khá tệ khi precision chỉ đạt từ 50 tới 60% và recall đạt 30 đến gần 50%.

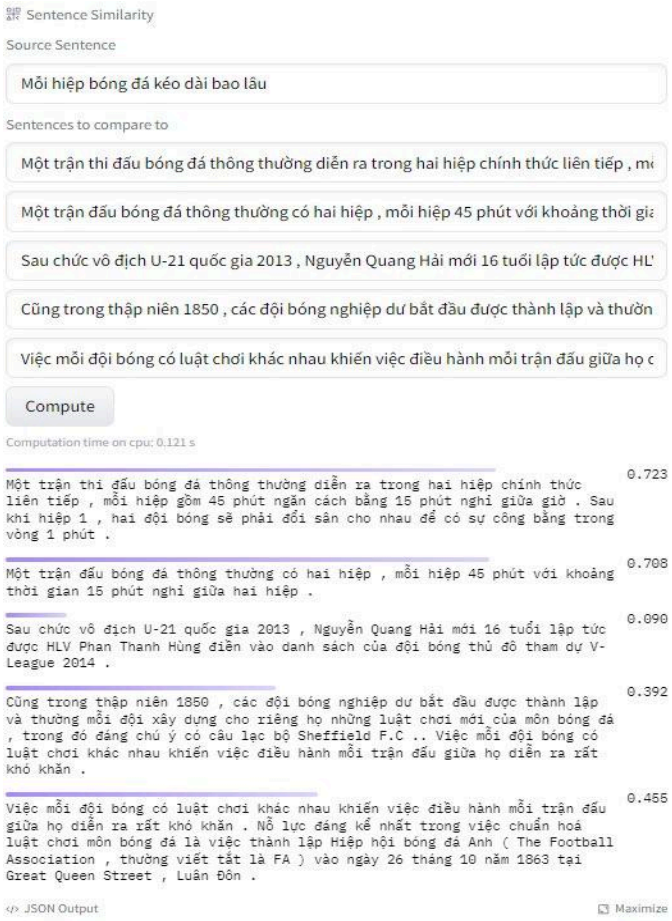
→ Các models đang bị ảnh hưởng bởi vấn đề bộ dữ liệu mất cân bằng (imbalanced data) khi các đánh giá trên class False tốt hơn class True bởi vì các mẫu False chiếm đa số.

C. Thử nghiệm trên mô hình có sẵn

1. Giới thiệu mô hình

SimeCSE Vietnamese viết tắt của cụm từ *Simple Contrastive Learning of Sentence Embeddings with Vietnamese,* là một mô hình Sentence Embeddings hiện đại trên tập dữ liệu tiếng Việt. SimeCSE_Vietnamese là mô hình tinh chỉnh dựa trên **SimCSE** giúp tối ưu hóa hiệu suất cũng như cho kết quả chính xác hơn. Mô hình encode input sentences bằng cách sử dụng mô hình ngôn ngữ tinh chỉnh **PhoBert**.

Đầu vào mô hình sẽ là các câu hoặc đoạn văn. Để dễ hình dung ta có hình bên dưới:



Hình 30: Ví dụ cho cách hoạt động mô hình

Kết quả mô hình trả về sẽ là độ tương đồng (từ 0 đến 1) giữa hai câu hoặc giữa câu và đoạn văn. Độ tương đồng càng lớn (gần bằng 1) thì sự giống nhau càng cao.

Điểm mạnh của mô hình:

- Vì được huấn luyện trên nhiều nguồn dữ liệu tiếng Việt, cho phép mô hình hiểu được nhiều mẫu câu và ngữ nghĩa của tiếng Việt.
- Có thể fine-tune trên các downstream tasks cụ thể, ví dụ như: phân loại văn bản, nhận dạng thực thể được đặt tên, trả lời câu hỏi,…
- Nhờ nhiều sự cải tiến trong quá trình training mô hình hiệu quả hơn giúp cải thiện hiệu suất vượt trội.

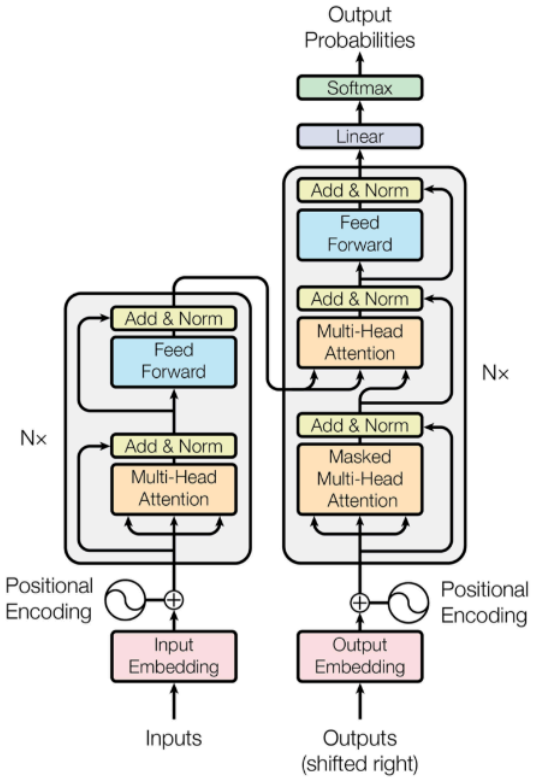
2. Cách hoạt động

2.1. Encode

Mô hình sử dụng PhoBert để encode input sentences. PhoBert là dự án của VinAI về huấn luyện biểu diễn từ (pre-train word embedding) sử dụng model RoBERTa. RoBERTa tối ưu hóa quy trình pre-trained để có năng suất mạnh mẽ hơn dựa trên mô hình BERT.

2.1.1. BERT

BERT là viết tắt của cụm từ *Bidirectional Encoder Representation from Transformer* có nghĩa là mô hình biểu diễn từ theo 2 chiều ứng dụng kỹ thuật Transformer. Hình dưới đây là kiến trúc của Transformer:



Hình 31: Kiến trúc Transformers

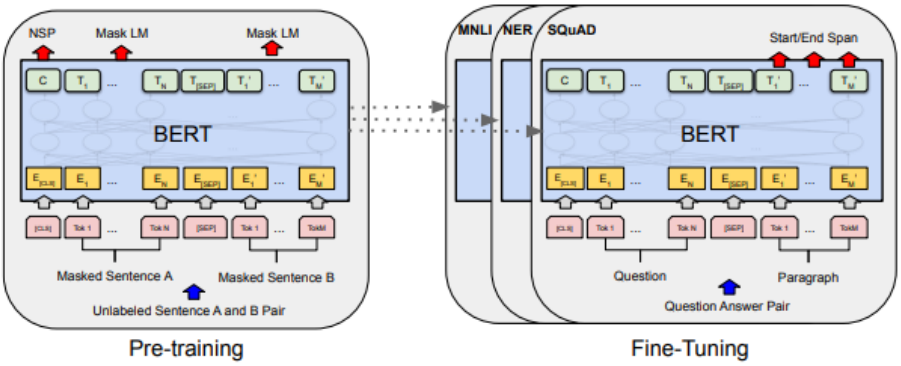
Mô hình gồm 2 phases:

- **Encoder**: Bao gồm các layers liên tiếp nhau. Mỗi một layer sẽ bao gồm một sub-layer là Multi-Head Attention kết hợp với fully-connected layer như mô tả ở nhánh encoder bên trái của hình vẽ. Kết thúc quá trình encoder ta thu được một vector embedding output cho mỗi từ.
- **Decoder**: Kiến trúc cũng bao gồm các layers liên tiếp nhau. Mỗi một layer của Decoder cũng có các sub-layers gần tương tự như layer của Encoder nhưng bổ sung thêm sub-layer đầu tiên là Masked Multi-Head Attention có tác dụng loại bỏ các từ trong tương lai khỏi quá trình attention.

Cơ chế attention của Transformer sẽ truyền toàn bộ các từ trong câu văn đồng thời vào

mô hình một lúc mà không cần quan tâm đến chiều của câu. Do đó Transformer được xem như là huấn luyện hai chiều (bidirectional). Đặc điểm này cho phép mô hình học được bối cảnh của từ dựa trên toàn bộ các từ xung quanh nó bao gồm cả từ bên trái và từ bên phải.

Một điểm đặc biệt ở BERT mà các model embedding trước đây chưa từng có đó là kết quả huấn luyện có thể fine-tuning được. Chúng ta sẽ thêm vào kiến trúc model một output layer để tùy biến theo tác vụ huấn luyện.



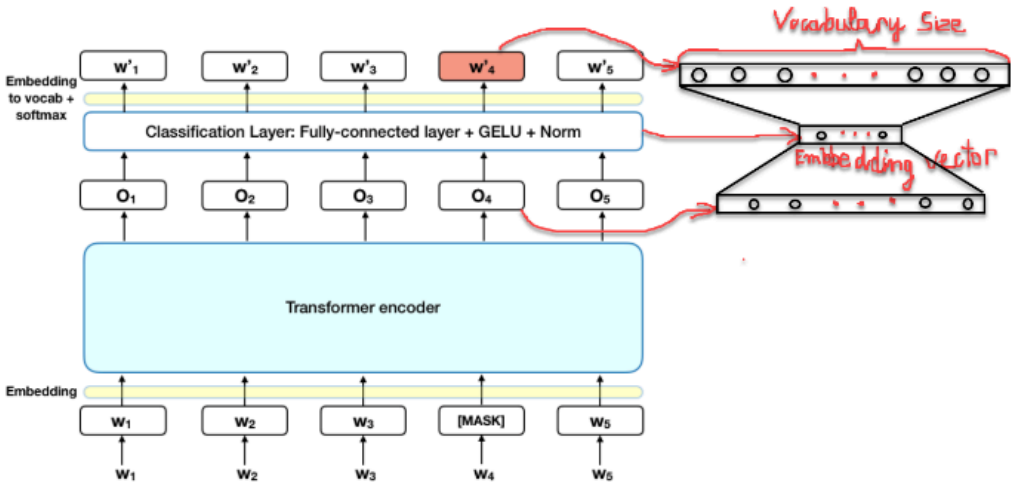
Hình 32: Pre-training & Fine-Tuning của BERT

Tiến trình áp dụng fine-tuning sẽ như sau:

- **Bước 1**: Embedding toàn bộ các token của cặp câu bằng các véc tơ nhúng từ pretrain model. Các token embedding bao gồm cả 2 token là [CLS] và [SEP] để đánh dấu vị trí bắt đầu của câu hỏi và vị trí ngăn cách giữa 2 câu. 2 token này sẽ được dự báo ở output để xác định các phần Start/End Spand của câu output.
- **Bước 2**: Các embedding véc tơ sau đó sẽ được truyền vào kiến trúc multi-head attention với nhiều block code (thường là 6, 12 hoặc 24 blocks tùy theo kiến trúc BERT). Ta thu được một véc tơ output ở encoder.
- **Bước 3**: Để dự báo phân phối xác suất cho từng vị trí từ ở decoder, ở mỗi time step chúng ta sẽ truyền vào decoder véc tơ output của encoder và véc tơ embedding input của decoder để tính encoder-decoder attention (cụ thể về encoder-decoder attention là gì các bạn xem lại mục 2.1.1). Sau đó projection qua liner layer và softmax để thu được phân phối xác suất cho output tương ứng ở time step t.
- **Bước 4**: Trong kết quả trả ra ở output của transformer ta sẽ cố định kết quả của câu Question sao cho trùng với câu Question ở input. Các vị trí còn lại sẽ là thành phần mở rộng Start/End Span tương ứng với câu trả lời tìm được từ câu input.

Trong quá trình training, BERT có 2 tác vụ là Masked ML và Next Sentence Prediction. Ở bài toán dự đoán sentence similarity, chúng ta sẽ áp dụng Masked ML.

Bên dưới là sơ đồ huấn luyện BERT theo tác vụ Masked ML



Hình 33: Sơ đồ huấn luyện của BERT theo tác vụ Masked

Theo đó:

- Khoảng 15% các token của câu input được thay thế bởi *[MASK]* token trước khi truyền vào model đại diện cho những từ bị che dấu (masked). Mô hình sẽ dựa trên các từ không được che (non-masked) dấu xung quanh *[MASK]* và đồng thời là bối cảnh của *[MASK]* để dự báo giá trị gốc của từ được che dấu. Số lượng từ được che dấu được lựa chọn là một số ít (15%) để tỷ lệ bối cảnh chiếm nhiều hơn (85%).
- Bản chất của kiến trúc BERT vẫn là một mô hình seq2seq gồm 2 phase encoder giúp embedding các từ input và decoder giúp tìm ra phân phối xác suất của các từ ở output. Kiến trúc Transfomer encoder được giữ lại trong tác vụ Masked ML. Sau khi thực hiện self-attention và feedforward ta sẽ thu được các véc tơ embedding ở output là O_1,O_2, …, O_5.
- Để tính toán phân phối xác suất cho từ output, chúng ta thêm một Fully connected layer ngay sau Transformer Encoder. Hàm softmax có tác dụng tính toán phân phối xác suất. Số lượng units của fully connected layer phải bằng với kích thước của từ điển.
- Cuối cùng ta thu được vector nhúng của mỗi một từ tại vị trí *MASK* sẽ là embedding véc tơ giảm chiều của véc tơ O_i sau khi đi qua fully connected layer như mô tả trên hình 33.

2.1.2. RoBERTa

RoBERTa viết tắt của "Robustly optimized BERT approach". Nó là một biến thể của BERT model, được giới thiệu bởi các nhà nghiên cứu tại Facebook AI. RoBERTa xây dựng dựa trên kiến trúc và mục tiêu đào tạo trước của BERT nhưng đưa ra một số sửa đổi và cải tiến để nâng cao hiệu suất của nó.

Những điểm khác biệt và cải tiến chính của RoBERTa so với BERT như sau:

- **Dynamic Masking**: Trong khi BERT random masking token trong quá trình pre-train. Model áp dụng masking pattern khác nhau cho từng training epoch, masking token khác nhau ở mỗi lần. Điều này giúp RoBERTa diễn đạt từ ngữ mạnh mẽ hơn và nắm bắt được nhiều thông tin theo ngữ cảnh hơn.

- **Batch Size lớn hơn và training lâu hơn**: Điều này cho phép model học hiệu quả hơn từ lượng lớn dữ liệu và nắm bắt được các cách trình bày theo ngữ cảnh phong phú hơn.
- **Loại bỏ thủ tục Next Sentence Prediction (NSP)**: Model chỉ sử dụng thủ tục MLM (Masked Language Model). Giúp đơn giản hóa quá trình training và cho phép RoBERTa tập trung nhiều hơn vào việc tìm hiểu các cách biểu diễn theo ngữ cảnh từ các masked token
- **Tăng dữ liệu training:** RoBERTa được train trên bộ dữ liệu lớn hơn và đa dạng hơn so với BERT. Kết hợp nhiều bộ dữ liệu từ các nguồn và lĩnh vực khác nhau, cho phép model học hỏi từ nhiều dữ liệu hơn và cải thiện khả năng khái quát hóa của nó.
- **Thay đổi siêu tham số**: RoBERTa tinh chỉnh một số siêu tham số so với BERT, chẳng hạn như rate schedules, optimizer settings, và dropout rates. Những thay đổi này được thiết kế để cải thiện tính ổn định và hiệu suất training.

2.1.3. PhoBERT

PhoBERT viết tắt của "Phở BERT" (Phở là món ăn truyền thống của Việt Nam). Cách pre-train của PhoBERT dựa vào model RoBERTa. PhoBERT tinh chỉnh lại huấn luyện được trên dữ liệu tiếng Việt.

Dữ liệu tiếng Việt sẽ thực hiện phân đoạn từ và câu trên tập dữ liệu huấn luyện. Khác với RoBERTa,model áp dụng *fastBPE* để phân khúc những câu có đơn vị từ phụ (subword), sử dụng từ vựng trong số 64000 loại từ phụ.

Thuật toán BPE(Byte Pair Encoding) là một kỹ thuật nén từ cơ bản giúp chúng ta index được toàn bộ các từ kể cả trường hợp từ mở (không xuất hiện trong từ điển) nhờ mã hóa các từ bằng chuỗi các từ phụ. Nguyên lý hoạt động của BPE dựa trên phân tích trực quan rằng hầu hết các từ đều có thể phân tích thành các thành phần con. Quá trình thuật toán BPE gồm các bước:

- Bước 1: Khởi tạo từ điển (vocabulary).
- Bước 2: Biểu diễn mỗi từ trong bộ văn bản bằng kết hợp của các ký tự với token <\w> ở cuối cùng đánh dấu kết thúc một từ
- Bước 3: Thống kê tần suất xuất hiện theo cặp của toàn bộ token trong từ điển.
- Bước 4: Gộp các cặp có tần suất xuất hiện lớn nhất để tạo thành một n-gram theo level character mới cho từ điển.
- Bước 5: Lặp lại bước 3 và bước 4 cho tới khi số bước triển khai merge đạt đỉnh hoặc kích thước kỳ vọng của từ điển đạt được.

2.2. Huấn luyện mô hình

Các bước để huấn luyện mô hình SimCSE như sau:

- Bước 1 - Chuẩn bị dữ liệu:
    - Sử dụng các checkpoints đã được tiền huấn luyện của BERT hoặc RoBERTa.
    - Sử dụng các câu văn bản được lấy mẫu ngẫu nhiên từ Wikipedia tiếng Anh (10^6 câu được lấy mẫu ngẫu nhiên) cho huấn luyện không giám sát.
    - Sử dụng kết hợp các tập dữ liệu MNLI và SNLI (314k cặp câu) cho huấn luyện giám sát.

- Bước 2 - Thông số huấn luyện mô hình:
    - Sử dụng gói transformers (Wolf et al., 2020) để triển khai mô hình SimCSE.
    - Đối với SimCSE giám sát, huấn luyện mô hình trong 3 epochs và đánh giá mô hình sau mỗi 250 bước huấn luyện trên tập phát triển của STS-B. Chọn checkpoint tốt nhất cho đánh giá cuối cùng trên tập kiểm tra.
    - Đối với SimCSE không giám sát, huấn luyện mô hình trong 1 epoch.
    - Thực hiện tìm kiếm trên lưới (grid-search) với các giá trị batch size $\in$ {64, 128, 256, 512} và learning rate $\in$ {1e-5, 3e-5, 5e-5} trên tập phát triển của STS-B và sử dụng cài đặt siêu tham số trong dưới đây:

	Unsupervised				**Supervised**	
	BERT		RoBERTa		base	large
	base	large	base	large		
Batch size	64	64	512	512	512	512
Learning rate	3e-5	1e-5	1e-5	3e-5	5e-5	1e-5

- Bước 3 - Cài đặt bên trong mô hình:
    - Sử dụng biểu diễn của token [CLS] với một lớp MLP (Multilayer Perceptron) trên đỉnh nó làm biểu diễn của câu.
    - Đối với SimCSE không giám sát, loại bỏ lớp MLP và chỉ sử dụng đầu ra của [CLS] trong quá trình kiểm tra vì nó dẫn đến hiệu suất tốt hơn.
- Bước 4 - Cải thiện hiệu suất:
    - Một biến thể mới được các tác giả giới thiệu là thêm một mục tiêu học trên masked language modeling (MLM) như một loss phụ, nghĩa là: $\ell + \lambda . \ell^{mlm}$ giúp cho mô hình này tránh quên thông tin ở mức token.
    - Ngoài ra, thêm mục tiêu này còn giúp cải thiện hiệu suất trên các tác vụ chuyển giao.

3. Thực nghiệm

Quá trình fine-tune và khảo sát mô hình được thực hiện như sau:

1. Đọc dữ liệu và tiền xử lý: tập tin dữ liệu *Project1_Data.json* được lưu trữ ở Github và được tải về Google Colab thông qua wget. Cách làm này sẽ giúp quá trình xử lý dữ liệu diễn ra nhanh hơn so với phương pháp sử dụng Google Drive như thông thường. Dữ liệu sau đó được lưu trữ ở pandas dataframe và được tiền xử lý (chuẩn hóa unicode, loại bỏ ký tự đặc biệt, loại bỏ stopword…). Dữ liệu thu được là một pandas dataframe với 18056 dòng (mẫu dữ liệu) và 3 cột (question, text, label) được sử dụng cho quá trình fine-tune và đánh giá mô hình.
2. Fine-tune và khảo sát mô hình: mô hình được khảo sát với các tham số test_ratio (tỉ lệ chia dữ liệu train/test), batch_size (kích thước của mỗi batch trong quá trình huấn luyện), epoch (số lần mô hình được huấn luyện trên toàn bộ tập huấn luyện). Tất cả 18056 mẫu dữ liệu được sử dụng để fine-tune và đánh giá mô hình với tỉ lệ tương ứng sử dụng API sentence_transformers. Kết quả thu được sau quá trình khảo sát là một bảng kết quả thể hiện các thông số (thời gian thực thi, độ chính xác, precision…) và một biểu đồ tương ứng với bảng kết quả.
3. Tập dữ liệu chỉ gồm hai phần là huấn luyện và kiểm thử. Mô hình được fine-tune trên tập huấn luyện, sau đó được khảo sát trên tập kiểm thử tương ứng. Khi tiến hành khảo

sát một tham số, các tham số còn lại sẽ có giá trị mặc định. Cấu hình mặc định: (test_ratio=0.4, shuffle=True, batch_size=32, epoch=5).

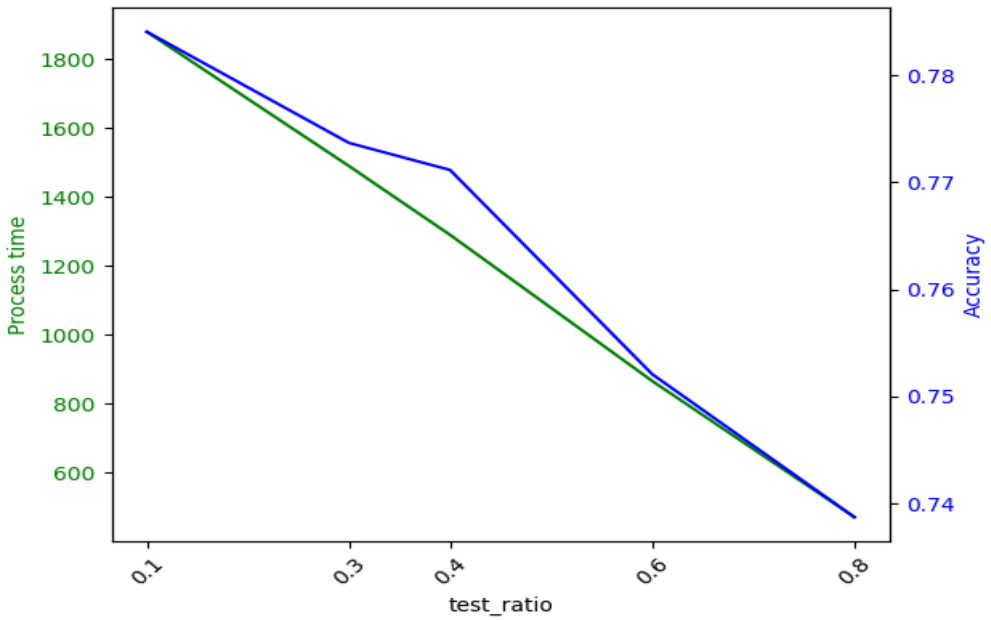
Các tham số khảo sát:

1. test_ratio (tỉ lệ chia dữ liệu train/test): mô hình được khảo sát với các tỉ lệ tập kiểm thử là [0.1, 0.3, 0.4, 0.6, 0.8]. Thời gian fine-tune cho từng batch và từng epoch được trình bày ở notebook tương ứng.
2. batch_size (kích thước của mỗi batch trong quá trình huấn luyện): mô hình được khảo sát với các giá trị [8, 16, 32].
3. epoch (số lần mô hình được huấn luyện trên toàn bộ tập huấn luyện): để đảm bảo giới hạn về tài nguyên của colab với phiên bản GPU Tesla T4, mô hình được khảo sát với các giá trị epoch là [1, 3, 10].

Kết quả khảo sát:

- **Tham số test_ratio**

Test Ratio	Process time	Accuracy	Recall	Precision	F1-Score
0.1	1879.588830	0.784053	0.517241	0.731707	0.606061
0.3	1490.133618	0.773675	0.510133	0.698652	0.589692
0.4	1290.562403	0.771148	0.504161	0.688397	0.582048
0.6	866.678664	0.752077	0.454334	0.658516	0.537694
0.8	471.967292	0.738733	0.377096	0.654820	0.478585



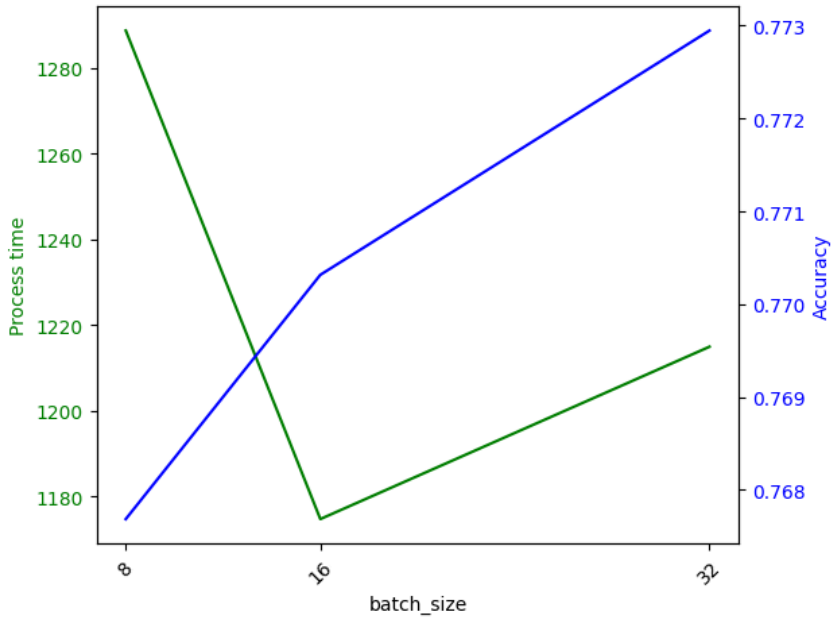
Hình 34: Test Ratio, Process time & Accuracy tương ứng

Nhận xét:

- Với giá trị test_ratio nhỏ nhất là 0.1, mô hình có độ chính xác cao nhất (78%) cùng thời gian xử lý lâu nhất (1879.59 giây). Với giá trị test_ratio lớn nhất là 0.8, mô hình có độ chính xác thấp nhất (73.8%) với thời gian xử lý nhanh nhất (471.96 giây).
- Dựa vào biểu đồ kết quả có thể dễ dàng nhận thấy rằng với sự tăng dần của giá trị test_ratio (0.1, 0.3…) là sự giảm dần của thời gian xử lý (1879.58 giây, 1490.13 giây,...) và độ chính xác của mô hình (78.4%, 77.3%, …).
- Hiện tượng này thể hiện rằng khi mô hình được huấn luyện với nhiều mẫu hơn thì vấn đề overfitting sẽ càng rõ ràng, và vì cần huấn luyện trên nhiều mẫu nên thời gian huấn luyện cũng sẽ dài hơn những tập dữ liệu nhỏ.

- **Tham số batch_size:**

Batch Size	Process time	Accuracy	Recall	Precision	F1-Score
8	1288.733133	0.767687	0.528252	0.667405	0.589731
16	1174.743665	0.770317	0.502409	0.686826	0.580319
32	1214.957929	0.772948	0.504161	0.693791	0.583968



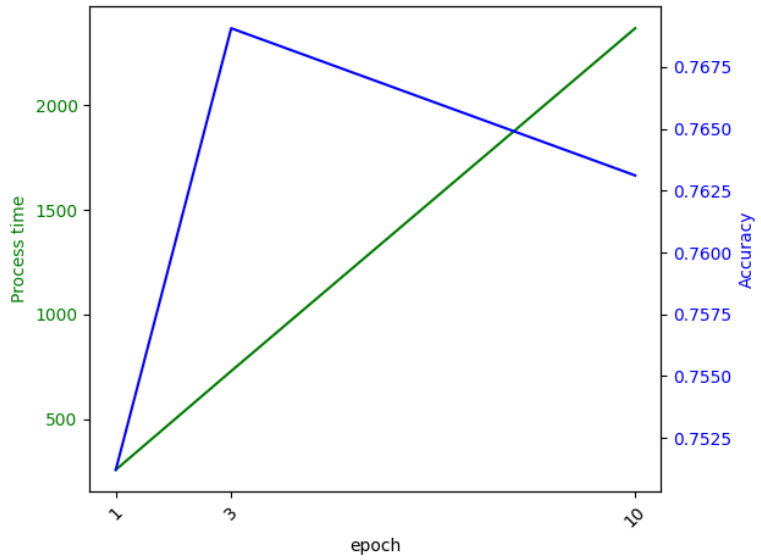
Hình 35: Batch Size, Process time & Accuracy tương ứng

Nhận xét:

- Với batch_size là 8 thì mô hình có độ chính xác thấp nhất (76.76%) và với batch_size là 32 thì mô hình có độ chính xác cao nhất (77.29%).
- Đồ thị thể hiện sự thay đổi gần như là tuyến tính của độ chính xác khi giá trị tham số tăng dần. Ngược lại với độ chính xác, giá trị thời gian xử lý có sự thay đổi không rõ ràng. Với batch_size là 8, thời gian xử lý tương ứng là 1288.73 giây và là thời gian dài nhất. Thời gian xử lý giảm với batch_size là 16 (1174.74 giây) và tăng trở lại với batch_size là 32 (1214.95 giây).

- **Tham số epoch**

Epoch	Process time	Accuracy	Recall	Precision	F1-Score
1	256.586392	0.751211	0.416995	0.671368	0.514456
3	728.933487	0.769071	0.468682	0.701639	0.561975
10	2369.249527	0.763118	0.5265	0.656114	0.584204



Hình 36: Epoch, Process time & Accuracy tương ứng

Nhận xét:

- Mô hình có độ chính xác thấp nhất với epoch là 1 (75.12%) và độ chính xác cao nhất với epoch là 3 (76.9%).
- Mô hình có độ chính xác thấp nhất với epoch là 1 (75.12%) và độ chính xác cao nhất với epoch là 3 (76.9%).
- Sự thay đổi của thời gian xử lý khi giá trị epoch tăng dần là tuyến tính. Với giá trị epoch nhỏ nhất là 1, thời gian xử lý của mô hình là 256.58 giây. Khi giá trị epoch là lớn nhất (10 epoch), mô hình yêu cầu thời gian xử lý là dài nhất (2369.24 giây). Đây là kết quả hiển nhiên vì với số lượng epoch lớn thì mô hình được yêu cầu xử lý một lượng tương ứng.
- Về thông số độ chính xác, giá trị epoch quá nhỏ (1 epoch) khiến việc huấn luyện mô hình không quá hiệu quả. Với 3 epoch huấn luyện, độ chính xác của mô hình được cải thiện nhưng có thể cảm nhận được hiện tượng overfitting của mô hình trên tập dữ liệu kiểm thử. Với 10 epoch huấn luyện, độ chính xác của mô hình không còn đạt được giá trị cao nhất nhưng hiện tượng overfitting đã được cải thiện giúp mô hình tiến gần hơn đến tính chất generalization.

So sánh giữa mô hình ở Q2 (mô hình tự xây dựng) và mô hình ở Q3 (mô hình có sẵn):

- Mô hình ở Q2 có độ chính xác cao nhất là 76.9% (random embedding) trên tập dữ liệu kiểm thử. Dựa vào kết quả có thể thấy rằng mô hình fine-tune có độ chính xác thấp hơn

mô hình freeze. Độ chính xác của các phiên bản dao động từ ≈75-76% (có sự chênh lệch ở phiên bản fine-tune Word2Vec là 72.8%).

- Mô hình ở Q3 có độ chính xác cao nhất là 78.4% trên tập dữ liệu kiểm thử khi khảo sát trên tham số test_ratio. Khi khảo sát với các tham số khác, độ chính xác của mô hình dao động từ ≈77-78%.
- Tổng quát, mô hình có sẵn có độ chính xác được cải thiện và ổn định hơn so với mô hình tự xây dựng. Cả hai phiên bản mô hình đều bị ảnh hưởng bởi tình trạng mất cân bằng của tập dữ liệu khi nhãn False chiếm tỉ lệ đáng kể so với nhãn True.

D. Tài liệu tham khảo

[1] Datquocnguyen. (n.d.). GitHub - datquocnguyen/PhoW2V: Pre-trained Word2Vec syllable- and word-level embeddings for Vietnamese. GitHub. https://github.com/datquocnguyen/PhoW2V
[2] FastText. (n.d.). https://fasttext.cc/
[3] Gao, T., Yao, X., & Chen, D. (2021, April 18). SIMCSE: Simple Contrastive Learning of Sentence Embeddings. arXiv.org. https://arxiv.org/abs/2104.08821
[4] Khánh, P. Đ. (2020, May 23). Khoa học dữ liệu. Khanh's Blog. https://phamdinhkhanh.github.io/2020/05/23/BERTModel.html#24-next-sentence-prediction-nsp
[5] Kim, Y. (2014, August 25). Convolutional neural networks for sentence classification. arXiv.org. https://arxiv.org/abs/1408.5882
[6] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019, July 26). ROBERTA: A robustly optimized BERT pretraining approach. arXiv.org. https://arxiv.org/abs/1907.11692
[7] Models - hugging face. (n.d.). https://huggingface.co/models Tran, C. (2020, February 1).
[8] A Complete Guide to CNN for Sentence Classification with PyTorch. Chris Tran. https://chriskhanhtran.github.io/posts/cnn-sentence-classification/#22-load-pretrained-vectors