HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN

BỘ MÔN: NHẬP MÔN TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI: XÂY DỰNG MÔ HÌNH PHÂN LOẠI THÁI ĐỘ TRONG VĂN BẢN

Giảng viên : **TS. Nguyễn Thị Mai Trang**

Nhóm học phần : **13**

Nhóm bài tập lớn : **14**

Thành viên :

Nguyễn Tuyết Mai	**B22DCCN392**
Cấn Đức Khôi	**B22DCKH069**
Nguyễn Thị Hiền	**B22DCCN288**
Trần Bá Thành	**B22DCCN798**
Hoàng Việt Hùng	**B22DCCN361**

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT

STT	Từ viết tắt	Ý nghĩa tiếng Anh	Ý nghĩa tiếng Việt
1	BERT	Bidirectional Encoder Representations from Transformers	Mô hình học sẵn
2	PhoBERT	Pre-trained BERT models for Vietnamese	Mô hình học sẵn dành riêng cho tiếng Việt
3	LSTM	Long Short-Term Memory	Bộ nhớ dài-ngắn hạn
4	BiLSTM	Bidirectional Long Short-Term Memory	Bộ nhớ dài-ngắn hạn song hướng
5	NLP	Natural Language Processing	Xử lý ngôn ngữ tự nhiên
6	RNN	Recurrent Neural Network	Mạng nơ-ron hồi quy
7	JSON	Javascript Object Notation	Định dạng trao đổi dữ liệu văn bản dung lượng nhẹ
8	HTML	Hyper Text Markup Language	Ngôn ngữ Đánh dấu Siêu văn bản
9	URL	Uniform Resource Locator	Địa chỉ web
10	ReLU	Rectified Linear Unit	Hàm kích hoạt nền tảng trong lĩnh vực học sâu và mạng nơ-ron

DANH MỤC HÌNH ẢNH

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1.1. Đặt vấn đề

Trong kỷ nguyên số hóa hiện nay, sự phát triển bùng nổ của internet và các nền tảng truyền thông xã hội đã tạo ra một khối lượng dữ liệu văn bản tiếng Việt khổng lồ và vô cùng đa dạng. Từ những dòng trạng thái cá nhân, các bình luận trên diễn đàn, cho đến nội dung từ các trang báo điện tử và blog, nguồn thông tin này phản ánh một cách phong phú ý kiến, quan điểm và các sắc thái thái độ của người dùng. Việc khai thác và phân tích hiệu quả nguồn dữ liệu này không chỉ mang lại những cơ hội to lớn mà còn đặt ra nhiều thách thức đáng kể cho lĩnh vực xử lý ngôn ngữ tự nhiên. Trong bối cảnh đó, phân tích thái độ nổi lên như một hướng nghiên cứu và ứng dụng quan trọng. Đây là một lĩnh vực của xử lý ngôn ngữ tự nhiên, tập trung vào việc tự động xác định và trích xuất các thông tin mang tính chủ quan từ văn bản, bao gồm quan điểm, đánh giá, và thái độ của người viết đối với một chủ thể, sự kiện hay vấn đề cụ thể. Các ứng dụng của phân tích thái độ rất rộng rãi, từ việc theo dõi và nắm bắt dư luận xã hội, phân tích phản hồi của khách hàng nhằm cải thiện sản phẩm và dịch vụ, đến việc đánh giá hiệu quả của các chiến dịch truyền thông và hỗ trợ quá trình ra quyết định trong nhiều ngành nghề.

Tuy nhiên, việc phân tích thái độ đối với văn bản tiếng việt gặp phải không ít trở ngại do đặc thù của ngôn ngữ. Tiếng Việt là một ngôn ngữ giàu sắc thái, với sự đa dạng trong cách biểu đạt, các hiện tượng đồng âm và đa nghĩa phổ biến, cùng với sự phức tạp của cấu trúc ngữ pháp và cú pháp. Bên cạnh đó, sự xuất hiện thường xuyên của tiếng lóng, từ ngữ địa phương, các biến thể ngôn ngữ, cũng như ảnh hưởng của các yếu tố văn hóa và ngữ cảnh cụ thể là những thách thức đặc thù mà các mô hình phân tích cần phải vượt qua để đạt được độ chính xác cao.

1.2. Định hướng giải pháp

Để giải quyết những thách thức nêu trên và xây dựng một hệ thống phân loại thái độ hiệu quả cho văn bản tiếng Việt, dự án của chúng em đề xuất một mô hình học sâu được xây dựng dựa trên sự kết hợp của ba thành phần công nghệ cốt lõi: mô hình ngôn ngữ tiền huấn luyện PhoBERT, kiến trúc mạng nơ-ron hồi quy hai chiều sử dụng các đơn vị nhớ ngắn-dài (BiLSTM), và cơ chế tập trung (attention).

Cụ thể, PhoBERT, một mô hình ngôn ngữ đã được huấn luyện trước trên một lượng lớn dữ liệu tiếng Việt, sẽ đóng vai trò trích xuất các vector đặc trưng

mang thông tin ngữ cảnh phong phú từ văn bản đầu vào. Tiếp theo, kiến trúc BiLSTM được sử dụng để khai thác thông tin tuần tự từ cả hai chiều (xuôi và ngược) của chuỗi embedding, giúp mô hình hiểu rõ hơn các mối quan hệ phụ thuộc xa giữa các từ trong câu. Cuối cùng, một lớp attention được tích hợp để cho phép mô hình tự động xác định và gán trọng số cao hơn cho những từ hoặc cụm từ mang thông tin cảm xúc quan trọng nhất trong ngữ cảnh, từ đó tối ưu hóa quá trình phân loại.

1.3. Mục tiêu và phạm vi đề tài

Mục tiêu chính của đề tài này là tìm hiểu, thiết kế và cài đặt một mô hình học sâu có khả năng phân loại thái độ trong văn bản tiếng việt một cách chính xác và hiệu quả. Mô hình sẽ phân loại văn bản thành ba lớp cảm xúc chính: tích cực, tiêu cực và trung tính.

Phạm vi của bài tập lớn tập trung vào việc xử lý dữ liệu văn bản tiếng Việt xoay quanh việc ứng dụng và kết hợp mô hình PhoBERT, kiến trúc BiLSTM và cơ chế attention. Nhóm lựa chọn hướng tiếp cận này xuất phát từ sự hiểu tính cấp thiết và tiềm năng ứng dụng thực tiễn của bài toán phân tích thái độ trong bối cảnh bùng nổ thông tin số hiện nay. Đồng thời, các kỹ thuật được đề xuất được chúng em đánh giá là phù hợp và có nhiều ưu điểm để giải quyết những thách thức đặc thù của tiếng Việt.

Cuối cùng, nhóm em sử dụng các hàm tạo nút, textfield và sự kiện cơ bản để thiết kế một giao diện nho nhỏ cho phép nhập vào văn bản và in ra nhãn đánh giá cùng với độ tin cậy đánh giá.

Qua bài tập lớn này, nhóm chúng em kỳ vọng sẽ lan tỏa tinh thần, cảm hứng đối với lĩnh vực xử lý ngôn ngữ tự nhiên đặc biệt là cho tiếng Việt nói chung và trong mảng phân tích thái độ nói riêng.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Tổng quan về PhoBERT

PhoBERT (vinai/phobert-base-v2) là một mô hình ngôn ngữ tiếng Việt được phát triển bởi VinAI Research, dựa trên nền tảng của BERT (Bidirectional Encoder Representations from Transformers). BERT, một mô hình học sâu do Google phát triển, đã mở ra một kỷ nguyên mới trong lĩnh vực xử lý ngôn ngữ tự nhiên. Khác với các mô hình truyền thống, BERT cho phép hiểu ngữ cảnh của từ trong câu thông qua cơ chế self-attention, giúp mô hình này có khả năng xử lý thông tin hai chiều và nhận diện các mối quan hệ phức tạp giữa các từ trong câu. Mô hình phobert-base-v2 là phiên bản tốt hơn, cập nhật hơn, được huấn luyện trên dữ liệu lớn hơn và đa dạng hơn nên hiệu quả thực thế cao hơn so với bản cũ.

PhoBERT là một biến thể của BERT, được huấn luyện riêng biệt trên một tập dữ liệu lớn của tiếng Việt, từ đó nó có thể giải quyết các vấn đề đặc thù của ngôn ngữ này. Đặc biệt, PhoBERT giúp cải thiện việc xử lý các đặc điểm ngữ nghĩa của tiếng Việt, một ngôn ngữ có rất nhiều từ đồng âm và từ đa nghĩa, cũng như cấu trúc câu linh hoạt. Các khả năng này là yếu tố quan trọng giúp chúng ta giải quyết các vấn đề phức tạp về ngữ nghĩa trong ngữ cảnh tiếng Việt.

PhoBERT, nhờ vào cấu trúc và cách huấn luyện đặc biệt của nó, có thể hiểu và xử lý các câu văn phức tạp, đồng thời cung cấp những đặc trưng (embeddings) ngữ nghĩa cho mỗi từ trong câu, từ đó hỗ trợ tốt cho các tác vụ như phân loại văn bản, phân tích thái độ, và nhận diện thực thể.

2.1.1. Kiến trúc Transformer và PhoBERT

PhoBERT được xây dựng dựa trên kiến trúc Transformer, một mô hình học sâu được giới thiệu lần đầu vào năm 2017 bởi Vaswani và cộng sự. Transformer là một kiến trúc mạng nơ-ron sâu được thiết kế đặc biệt để xử lý dữ liệu tuần tự, nổi bật nhất là văn bản trong xử lý ngôn ngữ tự nhiên. Trước transformer, các mô hình như mạng nơ-ron hồi quy hay LSTM thường gặp khó khăn trong việc xử lý các phụ thuộc xa trong câu và khó song song hóa tính toán. Transformer ra đời đã giải quyết được những vấn đề này.

Điểm cốt lõi và đột phá nhất của transformer chính là cơ chế self-attention. Bản chất của self-attention cho phép mỗi phần tử trong một chuỗi đầu vào (ví dụ mỗi từ trong một câu) có thể "chú ý" và đánh giá mức độ liên quan đến tất cả các phần tử khác trong cùng chuỗi đó, bất kể vị trí xa gần, kể cả đánh giá chính nó. Thông qua việc này, mô hình có thể xây dựng được một biểu diễn ngữ cảnh phong phú hơn cho từng phần tử. Ví dụ trong câu "Tôi thích học lập trình", để hiểu rõ ý nghĩa của từ "thích", mô hình cần liên kết nó với các từ

"học" và "lập trình". Cơ chế self-attention giúp mô hình thực hiện chính xác việc này bằng cách xác định mối quan hệ và tầm quan trọng tương đối giữa các từ trong câu, thay vì chỉ xử lý ngữ nghĩa của từng từ một cách độc lập.

Quá trình tính toán của cơ chế self-attention, cụ thể là dạng attention tích vô hướng mà chúng em sử dụng trong mô hình, thường được thực hiện qua các bước sau:

1. Từ chuỗi các vector đầu vào, ba loại vector khác nhau được tạo ra cho mỗi token:
    a. Query (Q) => Mỗi vector query tương ứng với một từ trong câu. Query được coi như một truy vấn mà từ hiện tại đặt ra, đại diện cho thông tin mà từ hiện tại đang tìm kiếm hoặc muốn tập trung vào từ các phần khác của câu.
    b. Key (K) => Mỗi vector key tương ứng với một từ trong câu đầu vào. Key được hiểu như một nhãn hoặc một đại diện của từ đó dùng để được so khớp. Khi một từ (query) muốn tìm kiếm sự liên quan trong câu, nó sẽ so sánh query của mình với tất cả các key của các từ khác và cả chính nó.
    c. Value (V) => Đại diện cho tất cả các token trong chuỗi, chứa thông tin hoặc giá trị thực sự của token đó. Khi một key được xác định là quan trọng đối với một query, value tương ứng của nó sẽ đóng góp nhiều hơn vào kết quả đầu ra. Về mặt kỹ thuật, các ma trận Q, K, V này được tạo ra bằng cách nhân ma trận đầu vào với ba ma trận trọng số riêng biệt. Các ma trận trọng số này được học trong quá trình huấn luyện mô hình.
2. Để xác định mức độ một token, vector query chú ý đến một token khác là vector key, một điểm số tương đồng được tính toán giữa chúng bằng cách nhân tích vô hướng giữa Q và K. Đối với toàn bộ chuỗi, thao tác này tương đương với việc nhân ma trận Q với ma trận chuyển vị của K.
3. Các điểm số attention sau đó được chia cho căn bậc hai của số chiều của các vector key.
4. Các điểm số chú ý đã được điều chỉnh tỷ lệ sau đó được đưa qua hàm softmax. Hàm softmax sẽ chuẩn hóa các điểm này thành một tập hợp các trọng số attention, có giá trị từ 0 đến 1 và tổng bằng 1. Các trọng số này thể hiện mức độ quan trọng tương đối của mỗi token trong chuỗi đối với token hiện tại đang được xem xét.
5. Cuối cùng, vector đầu ra cho mỗi token được tính bằng tổng có trọng số của tất cả các vector value (V) trong chuỗi, trong đó trọng số chính là các attention weights đã được tính ở bước trên.

Công thức tổng quát cho attention tích vô hướng chia tỉ lệ được biểu diễn như sau:

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

Trong đó: Q, K, V là các ma trận query, key, và value; d_k là số chiều của vector key.

Như vậy, cơ chế self-attention cho phép mô hình tập trung một cách linh hoạt vào những phần tử quan trọng nhất trong chuỗi đầu ra của lớp BiLSTM khi hình thành biểu diễn cho mỗi từ và đưa ra quyết định phân loại. Bằng cách tính toán "trọng số chú ý" cho mỗi token dựa trên mức độ tương quan của nó với tất cả các token khác, những token mang thông tin cảm xúc then chốt sẽ được nhấn mạnh hơn. Kết quả là một chuỗi các vector biểu diễn mới, trong đó mỗi vector token được làm giàu thông tin từ các token khác dựa trên mức độ chú ý đã được tính toán, giúp làm nổi bật các đặc trưng quan trọng cho việc xác định thái độ của toàn bộ văn bản.

1. Bi-directional encoding: Khác với các mô hình RNN hay LSTM truyền thống, Transformer sử dụng cơ chế bi-directional, nghĩa là mô hình có thể xử lý văn bản từ cả hai chiều (từ trái sang phải và từ phải sang trái) vì thế giúp mô hình hiểu được ngữ nghĩa của từ trong ngữ cảnh toàn diện hơn, không bị giới hạn bởi thứ tự từ trong câu.
2. Pre-training và Fine-tuning: PhoBERT, giống như BERT, sử dụng chiến lược huấn luyện hai giai đoạn. Trong giai đoạn pre-training, mô hình được huấn luyện trên một tập dữ liệu lớn (trong trường hợp này là tiếng Việt), học được các mối quan hệ ngữ nghĩa chung của ngôn ngữ. Sau đó, trong giai đoạn fine-tuning, PhoBERT được tinh chỉnh trên các tác vụ cụ thể như phân loại văn bản hay phân tích thái độ, giúp mô hình cải thiện hiệu suất trên các bài toán thực tế

2.1.2. PhoBERT và xử lý tiếng Việt

Tiếng Việt là một ngôn ngữ có đặc điểm cấu trúc khá phức tạp và linh hoạt, với rất nhiều từ đồng âm và từ đa nghĩa. Do đó nó rất là khó khăn cho các mô hình ngôn ngữ truyền thống khi xử lý. PhoBERT, với việc được huấn luyện đặc biệt trên một tập dữ liệu tiếng Việt lớn, đã vượt qua các khó khăn này bằng cách hiểu ngữ nghĩa của từ trong ngữ cảnh của câu.

Ví dụ, từ "bàn" có thể có nghĩa là "một vật dụng" trong câu "Bàn học của tôi rất rộng", nhưng lại mang nghĩa "bờ sông" trong câu "Bờ bàn của con sông

này rất đẹp". Với cơ chế self-attention, PhoBERT có thể phân biệt giữa các nghĩa khác nhau của từ này tùy thuộc vào ngữ cảnh của câu.

Ngoài ra, tiếng Việt còn có đặc điểm là cấu trúc câu linh hoạt, nghĩa là thứ tự từ trong câu có thể thay đổi mà vẫn giữ nguyên ý nghĩa. PhoBERT có khả năng hiểu được những thay đổi này nhờ vào khả năng xử lý song song và ngữ nghĩa của các từ trong toàn bộ câu.

PhoBERT sử dụng thư viện Transformers, đây là thư viện hỗ trợ tải và sử dụng mô hình huấn luyện ngôn ngữ. Hai thành phần chính được sử dụng trong thư viện này:

- AutoTokenizer: Tự động tải tokenizer của mô hình PhoBERT, hỗ trợ biến đổi văn bản đầu vào thành chuỗi các token.
- TFAutoModel: Tự động tải encoder của mô hình PhoBERT, vì PhoBERT gốc được huấn luyện với PyTorch nên cần truyền tham số để chuyển đổi sang TensorFlow.

```
from transformers import AutoTokenizer, TFAutoModel

phobert_model_name = "vinai/phobert-base-v2"
tokenizer = AutoTokenizer.from_pretrained(phobert_model_name)
phobert_encoder = TFAutoModel.from_pretrained(phobert_model_name, from_pt=True)
```

Hình 1: Tải mô hình PhoBERT và tokenizer, encoder tương ứng.

1. Tokenizer của PhoBERT - Bộ tách và mã hóa từ

Tokenizer là công cụ chịu trách nhiệm chuyển đổi văn bản tự nhiên (dạng chuỗi) thành chuỗi các ID mà mô hình có thể hiểu được. PhoBERT sử dụng kỹ thuật tokenization dựa trên Byte-Pair Encoding cho phép tách từ thành các đơn vị nhỏ hơn gọi là subword — rất hiệu quả đối với tiếng Việt, vốn có nhiều từ ghép và từ chưa xuất hiện trong từ điển.

Các tệp liên quan đến Tokenizer sau khi tải từ pretrained model:

i. Tệp vocab.txt: là một danh sách chứa toàn bộ các token (các từ, subword hoặc ký hiệu đặc biệt) mà mô hình PhoBERT đã được huấn luyện để nhận diện. Mỗi dòng trong tệp tương ứng với một token và có một ID thứ tự theo dòng, bắt đầu từ 0. Có vai trò ánh xạ văn bản thành số (ID). Khi tokenizer xử lý văn bản, nó sẽ tách câu thành các token. Mỗi token sẽ được tìm trong vocab.txt để lấy ra chỉ số ID tương ứng. vocab.txt giới hạn số lượng token mà mô hình có thể hiểu. Những từ không có trong từ

điển sẽ bị ánh xạ thành token <unk> (unknown). PhoBERT sử dụng kỹ thuật Byte-Pair Encoding (BPE), nên nếu một từ không có trong từ điển thì nó sẽ được chia nhỏ thành các subword có trong vocab.txt.

ii. Tệp vocab.txt = Từ điển của PhoBERT, chứa toàn bộ token mà mô hình hiểu được. Nó được sử dụng để chuyển từ sang ID số → đầu vào cho mô hình. Tokenizer sẽ dựa vào vocab.txt để tra cứu, kết hợp với file bpe.codes để xử lý từ chưa gặp.

iii. Tệp bpe.codes là quy tắc phân tách từ sử dụng kỹ thuật BPE, để chia từ thành các subword giúp xử lý hiệu quả những từ hiếm hoặc từ chưa từng thấy trong tập huấn luyện.

iv. Tệp tokenizer.json là một tệp cấu hình tổng hợp chứa toàn bộ thông tin cần thiết để tái tạo lại quá trình tokenization một cách đầy đủ, nhanh chóng và đồng nhất. Bên trong tokenizer.json bao gồm: Từ điển vocab (token → ID) giống như trong vocab.txt, nhưng được lưu dưới dạng JSON. Bảng merge rules cho BPE tương tự bpe.codes, nhưng viết lại theo dạng dễ đọc bằng máy hơn (thường là danh sách các cặp subword cần ghép lại theo thứ tự). Thông tin về các token đặc biệt gồm các token như: [CLS]: token đầu câu, [SEP]: token phân cách, [PAD]: dùng để padding, [MASK]: dùng cho nhiệm vụ masked language modeling. Cấu hình chuẩn hóa văn bản (normalization). Cấu hình post-processing.

v. Tệp config.json lưu trữ cấu hình của mô hình transformer mà PhoBERT sử dụng. Nó không chỉ định nghĩa kiến trúc mạng (số lớp, kích thước ẩn...) mà còn chứa thông tin giúp đồng bộ giữa tokenizer và model.

Đầu ra từ tokenizer là các trường quan trọng như:

input_ids => dãy số biểu diễn các token trong văn bản.

attention_mask => đánh dấu đâu là token thực, đâu là padding.

2. Encoder của PhoBERT

Trong mô hình PhoBERT, Encoder đóng vai trò là thành phần cốt lõi chịu trách nhiệm trích xuất đặc trưng ngữ nghĩa từ văn bản tiếng Việt đầu vào. Cấu trúc Encoder trong PhoBERT được xây dựng dựa trên kiến trúc Roberta, bao gồm nhiều lớp Transformer Encoder chồng lên nhau. Cụ thể, Encoder trong PhoBERT thực hiện các bước sau:

i. Nhận chuỗi các token đã được mã hóa từ tokenizer (dưới dạng chỉ số ID).

ii. Chuyển đổi các ID này thành vector embedding thông qua bảng tra từ vựng (embedding lookup).

iii. Áp dụng chuỗi các lớp attention và feed-forward để học ra các đặc trưng ngữ nghĩa ngữ cảnh (contextualized embeddings).
iv. Xuất ra ma trận embedding đầu ra (last_hidden_state), trong đó mỗi vector biểu diễn ý nghĩa của một token, đã được làm giàu ngữ cảnh.

Đặc biệt, token đầu tiên của mỗi chuỗi thường được sử dụng để biểu diễn toàn bộ câu – vector của nó là embedding tổng hợp ngữ nghĩa toàn văn bản và thường dùng cho các tác vụ như phân loại văn bản. Để Encoder hoạt động chính xác và đồng nhất giữa lúc huấn luyện và suy luận, PhoBERT sử dụng một số tệp cấu hình sau:

v. Tệp config.json: Chứa toàn bộ thông tin về cấu trúc mô hình encoder: số lớp, kích thước ẩn, số đầu attention, kích thước từ vựng… Tệp này đảm bảo việc khởi tạo Encoder đúng theo kiến trúc đã huấn luyện.
vi. Tệp vocab.txt: Là bảng từ vựng dùng trong embedding layer. Nó ánh xạ token ID → từ tương ứng. Đây là nơi mô hình tra cứu vector embedding ban đầu cho mỗi token.

Tokenizer và Encoder đóng vai trò quan trọng trong việc chuyển văn bản thuần thành đầu vào dữ liệu cho mô hình deep learning. Sự đồng nhất và đúng định dạng trong tokenizer và encoder (qua các tệp config) là yếu tố quan trọng để đảm bảo hiệu quả.

2.1.3. PhoBERT với các phương pháp cũ

Trước khi PhoBERT ra đời, các phương pháp như Word2Vec và các lớp embedding trong Keras là những phương pháp phổ biến để chuyển từ ngữ thành vector. Tuy nhiên, chúng có những hạn chế rõ rệt khi so với PhoBERT:

1. Word2Vec:

   Word2Vec tạo ra một vector cố định cho mỗi từ trong từ điển. Các vector này không thay đổi dù từ đó xuất hiện trong bất kỳ ngữ cảnh nào nên Word2Vec không thể xử lý chính xác các từ đồng nghĩa hay từ đa nghĩa. Ví dụ, từ "bàn" có thể có nghĩa là "bàn học" hoặc "bàn bạc", nhưng trong Word2Vec, nó sẽ có cùng một vector bất kể ngữ cảnh nào.

2. Embedding trong Keras:

   Các lớp embedding trong Keras tương tự như Word2Vec, nhưng chúng học các vector từ các câu và đoạn văn. Tuy nhiên, chúng không có khả năng hiểu ngữ nghĩa của từ trong ngữ cảnh, nghĩa là khi từ "bàn" xuất

hiện trong các câu khác nhau, các vector này không thể phân biệt được giữa nghĩa "bàn học" và "bàn bạc".

PhoBERT, ngược lại, tạo ra vector động cho mỗi từ trong câu, dựa trên ngữ cảnh mà từ đó xuất hiện.Các vector động giúp mô hình hiểu được nghĩa của từ trong từng trường hợp cụ thể, từ đó cải thiện độ chính xác trong các tác vụ như phân loại văn bản, phân tích thái độ hay nhận diện thực thể.

2.1.4. Ứng dụng của PhoBERT trong bài tập lớn

Việc chọn PhoBERT cho quy trình tiền xử lý có nhiều lý do hợp lý:

1. Khả năng hiểu ngữ nghĩa theo ngữ cảnh:

   PhoBERT có thể phân biệt nghĩa của từ trong các ngữ cảnh khác nhau, điều mà các phương pháp như Word2Vec không thể làm được. Đó được coi là một lợi thế trong xử lý tiếng Việt, nơi các từ có thể mang nhiều nghĩa khác nhau.

2. Huấn luyện đặc biệt cho tiếng Việt:

   PhoBERT đã được huấn luyện đặc biệt trên dữ liệu tiếng Việt, giúp mô hình này có thể xử lý chính xác hơn các đặc điểm ngữ nghĩa của ngôn ngữ này.

3. Mô hình đã được pre-trained:

   PhoBERT là một mô hình pre-trained, giúp nhóm tiết kiệm được thời gian huấn luyện và tài nguyên tính toán, chỉ cần fine-tune mô hình trên dữ liệu đặc thù của dự án để đạt được hiệu quả tối ưu.

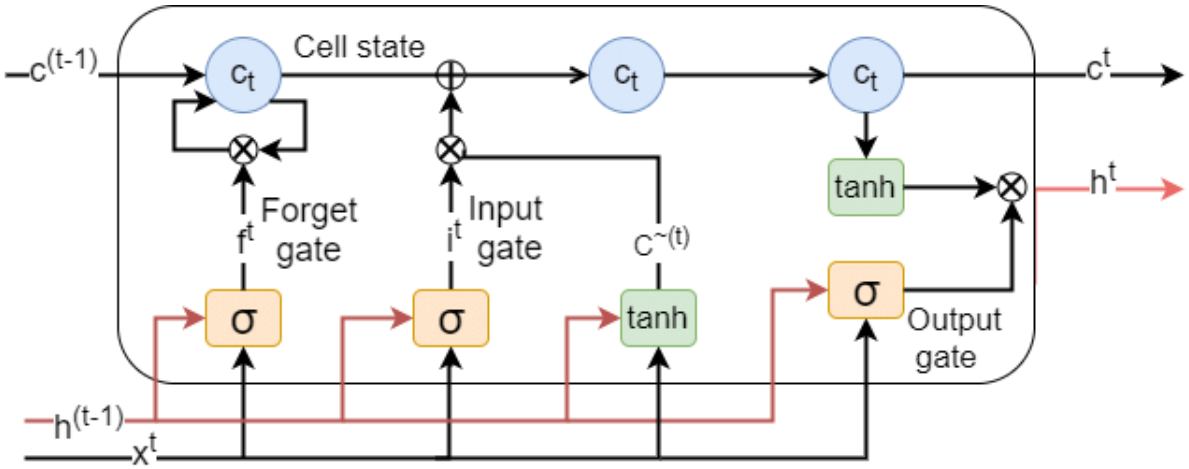
2.2. Tổng quan về BiLSTM

2.2.1. Cấu trúc cơ bản của LSTM

Bộ nhớ dài hạn ngắn hạn (LSTM), được Sepp Hochreiter và Jürgen Schmidhuber giới thiệu vào năm 1997, là một loại kiến trúc mạng nơ-ron hồi quy (RNN) được thiết kế để xử lý các phụ thuộc dài hạn. Điểm cải tiến chính của LSTM nằm ở khả năng lưu trữ, cập nhật và truy xuất thông tin một cách có chọn lọc trên các chuỗi mở rộng, khiến nó đặc biệt phù hợp với các tác vụ liên quan đến dữ liệu tuần tự. Đồng thời hạn chế được hiện tượng vanishing gradient và exploding gradient.

Cấu trúc của mạng LSTM bao gồm các ô nhớ, cổng vào, cổng quên và cổng ra. Các ô nhớ đóng vai trò là nơi lưu trữ dài hạn, cổng vào kiểm soát luồng

thông tin mới vào các ô nhớ, cổng quên điều chỉnh việc loại bỏ thông tin không liên quan và cổng ra xác định đầu ra dựa trên trạng thái hiện tại của các ô nhớ. Kiến trúc này cho phép LSTM nắm bắt và ghi nhớ hiệu quả các mẫu trong dữ liệu tuần tự đồng thời giảm thiểu các vấn đề về vanishing gradient và exploding gradient thường gây khó khăn cho các RNN truyền thống.



Hình 2: Sơ đồ kiến trúc 1 cell state của LSTM

2.2.1.1. Cell state

Là trạng thái ô nhớ - dòng chảy thông tin chính, lưu trữ thông tin bền vững trong thời gian dài qua nhiều bước thời gian, nó giống như một "băng chuyền" truyền thông tin xuyên suốt chuỗi đầu vào, được điều chỉnh bởi các cổng.

Cell state được cập nhật theo công thức:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Trong đó

C_t : cell state tại thời điểm hiện tại

f_t : forget gate — quyết định giữ lại bao nhiêu từ Ct−1C_{t-1}Ct−1

i_t : input gate — quyết định thêm bao nhiêu thông tin mới

$\tilde{C}_t$: candidate cell state — thông tin mới được đề xuất để ghi nhớ

Từ đó cho thấy dòng chảy thông tin được lọc và tích lũy dần theo thời gian.

2.2.1.2. Các cổng

1. **Forget gate (Cổng quên)**

Cổng quên có nhiệm vụ quyết định xem thông tin nào từ trạng thái tế bào trước đó (C_{t-1}) cần được loại bỏ hoặc "quên đi". Đầu vào cho cổng này bao gồm trạng thái ẩn của bước thời gian trước (h_{t-1}) và đầu vào của bước thời gian hiện tại (x_t).

Các đầu vào này được đưa qua một lớp mạng nơ-ron với hàm kích hoạt sigmoid. Hàm sigmoid sẽ cho ra các giá trị trong khoảng [0, 1].

$$f_t = sigmoid\left(W_f \cdot \left[h_{t-1,} x_t \right] + b_f \right)$$

Kết quả f_t là một vector chứa các giá trị từ 0 đến 1. Nếu một giá trị trong f_t gần bằng 0, điều đó có nghĩa là thông tin tương ứng trong trạng thái tế bào C_{t-1} sẽ bị quên đi phần lớn.

Ngược lại, nếu giá trị gần bằng 1, thông tin đó sẽ được giữ lại.

2. **Input gate (Cổng đầu vào)**

Cổng đầu vào có nhiệm vụ quyết định xem thông tin mới nào từ đầu vào hiện tại xt và trạng thái ẩn trước đó h_{t-1} sẽ được lưu trữ vào trạng thái tế bào C_t . Quá trình này gồm hai bước:

1. Quyết định thông tin cần cập nhật bằng một lớp sigmoid (gọi là "cổng đầu vào") xác định những giá trị nào trong trạng thái tế bào sẽ được cập nhật.

$$i_t = sigmoid\left(W_i \cdot \left[h_{t-1,} x_t \right] + b_i \right)$$

2. Tạo thông tin mới tiềm năng bằng một lớp mạng nơ-ron với hàm kích hoạt tanh tạo ra một vector chứa các giá trị mới tiềm năng $\tilde{C}_t$ (candidate cell state), có thể được thêm vào trạng thái tế bào. Hàm tanh giúp đưa giá trị về khoảng [-1, 1].

$$\tilde{C}_t = \tanh\left(W_c \cdot \left[h_{t-1,} x_t \right] + b_c \right)$$

Trong đó W_C và b_C cũng là các tham số được học.

Thông tin mới thực sự được thêm vào trạng thái tế bào là kết quả của phép nhân Hadamard (element-wise product) $i_t \odot \tilde{C}_t$.

3. Cập nhật cell state

Sau khi cổng quên và cổng đầu vào đã thực hiện vai trò của mình, trạng thái tế bào mới C_t được cập nhật từ trạng C_{t-1} thái tế bào cũ theo công thức:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Trong đó:

$f_t \odot C_{t-1}$ là phần thông tin từ quá khứ được giữ lại (những gì cổng quên quyết định không quên).

$i_t \odot \tilde{C}_t$ là phần thông tin mới được thêm vào (những gì cổng đầu vào quyết định cập nhật và giá trị mới tiềm năng).

4. Output gate (Cổng đầu ra)

Cổng đầu ra quyết định xem thông tin nào từ trạng thái tế bào C_t (đã được cập nhật) sẽ được xuất ra ngoài dưới dạng trạng thái ẩn h_t tại bước thời gian hiện tại. Trạng thái ẩn h_t này vừa là đầu ra của đơn vị LSTM tại bước t, vừa được truyền cho bước thời gian tiếp theo (t+1). Quá trình này cũng gồm hai bước:

1. Quyết định phần thông tin nào của cell state sẽ được xuất ra bằng một lớp sigmoid

$$O_t = sigmoid\left(W_o \cdot \left[h_{t-1,} x_t \right] + b_o \right)$$

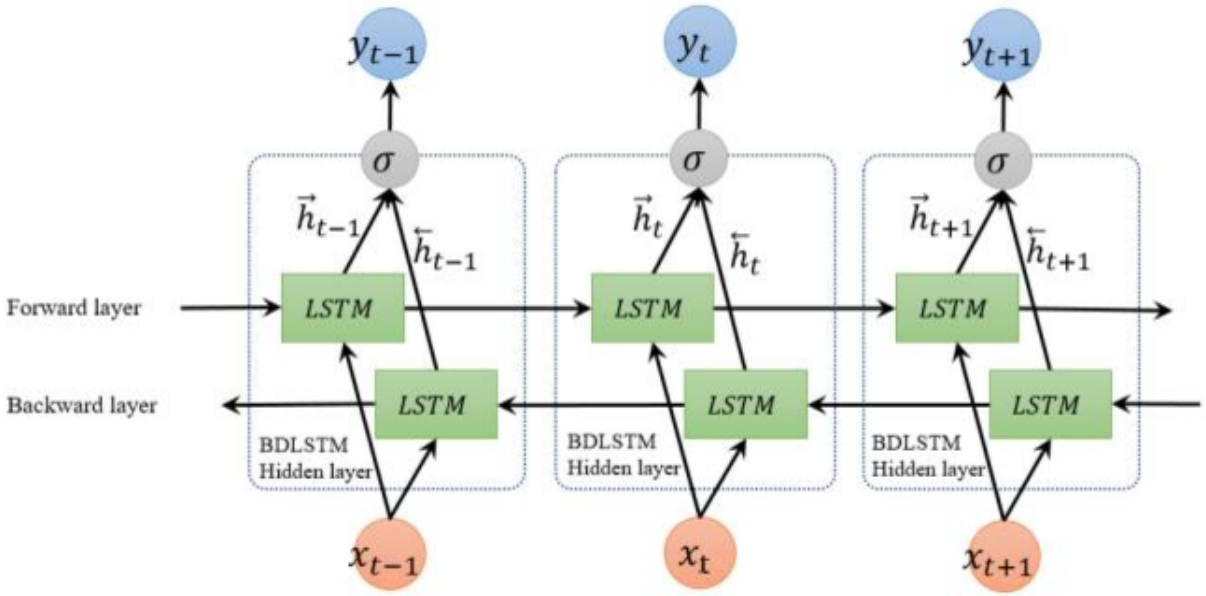
2. Tính toán trạng thái ẩn đầu ra h_t bằng cách đưa trạng thái tế bào C_t được đưa qua hàm kích hoạt tanh (để các giá trị nằm trong khoảng [-1, 1]), sau đó được nhân với vector O_t thông qua phép nhân Hadamard.

$$h_t = O_t \odot \tanh\left(C_t \right)$$

Đầu ra chính là vector ẩn h_t, truyền cho bước tiếp theo và cho output.Nó chứa thông tin đã được chọn lọc từ quá khứ (thông qua Ct) và thông tin từ đầu vào hiện tại, sẵn sàng để sử dụng cho các tính toán tiếp theo hoặc để đưa ra dự đoán.

Thông qua sự phối hợp của ba cổng này và trạng thái tế bào, các đơn vị LSTM có khả năng kiểm soát dòng chảy thông tin một cách tinh vi, cho phép chúng ghi nhớ các phụ thuộc dài hạn và quên đi những thông tin không còn cần thiết, từ đó trở nên rất hiệu quả trong nhiều tác vụ xử lý ngôn ngữ tự nhiên.

2.2.2. Cấu trúc lớp BiLSTM



Hình 3:Cấu trúc lớp BiLSTM

Bộ nhớ dài hạn song hướng (BiLSTM) là phần mở rộng của kiến trúc LSTM truyền thống kết hợp xử lý song hướng để tăng cường khả năng nắm bắt thông tin theo ngữ cảnh từ cả dữ liệu đầu vào trong quá khứ và tương lai. Được giới thiệu như một cải tiến so với LSTM đơn hướng, BiLSTM đặc biệt hiệu quả trong các tác vụ mà việc hiểu ngữ cảnh của một chuỗi theo cả hai hướng là rất quan trọng, chẳng hạn như xử lý ngôn ngữ tự nhiên và nhận dạng giọng nói.

Cấu trúc của BiLSTM bao gồm:

1. **Input Layer (Tầng đầu vào)**

Các ký hiệu x_{t-1}, x_t, x_{t+1} đại diện cho đầu vào tại các bước thời gian liên tiếp.

Mỗi đầu vào được đưa vào hai LSTM, một theo chiều xuôi (forward), một theo chiều ngược (backward).

2. **Bidirectional Layer (Tầng hai chiều)**

Bao gồm hai mạng LSTM song song: (i) LSTM forward (trên) giúp xử lý chuỗi từ trái sang phải (từ quá khứ đến hiện tại), (ii) LSTM backward (dưới) giúp xử lý chuỗi từ phải sang trái (từ tương lai về hiện tại). Mỗi LSTM tạo ra một vector trạng thái ẩn riêng cho mỗi bước thời gian.

3. **Kết hợp hai chiều**

Các đầu ra từ forward và backward LSTM được nối (concatenate) hoặc cộng (sum) lại. Kết quả sau khi kết hợp được chuyển đến tầng Output.

4. **Output Layer (Tầng đầu ra)**

Tạo ra đầu ra y_t tại mỗi bước thời gian dựa trên kết quả của hai chiều LSTM. Có thể dùng cho nhiều tác vụ khác nhau: phân loại chuỗi, gán nhãn từng từ v.v.

Bằng cách xử lý các chuỗi theo cả hai hướng, BiLSTM vượt trội trong các tác vụ như nhận dạng thực thể được đặt tên, phân tích tình cảm và dịch máy, trong đó việc hiểu ngữ cảnh của một từ hoặc cụm từ đòi hỏi phải xem xét cả ngữ cảnh quá khứ và tương lai của nó. Bản chất song hướng của BiLSTM khiến chúng trở nên linh hoạt và phù hợp với nhiều ứng dụng phân tích dữ liệu tuần tự.

Vì nghĩa của một từ phụ thuộc và cả hai phía xung quanh trong câu, BiLSTM giúp nắm bắt ngữ cảnh của từ tốt hơn.

Ví dụ minh họa: "Món ăn này **không tệ** chút nào, **rất ngon**."

Một LSTM xuôi khi đến "tệ" có thể bị ảnh hưởng bởi "không" và tạm hiểu là tiêu cực hoặc trung tính. Nó chưa "thấy" được vế "rất ngon" để xác nhận ý nghĩa thực sự.

BiLSTM, nhờ có luồng xử lý ngược, sẽ sớm "thấy" được "rất ngon" và "chút nào", giúp cho biểu diễn của "không tệ" được đặt trong ngữ cảnh chính xác hơn là "thực ra là tốt".

Kết hợp cả hai chiều, mô hình sẽ hiểu chính xác hơn nghĩa của từ “đánh” trong từng ngữ cảnh.

2.3. Tổng quan về Attention

2.3.1. Ý tưởng cốt lõi của Attention

Attention là một cơ chế trong học sâu cho phép mô hình tập trung vào những phần thông tin quan trọng nhất trong chuỗi đầu vào khi thực hiện một tác vụ như dịch máy, tóm tắt văn bản hay trả lời câu hỏi.

Thay vì xử lý tất cả các từ trong câu một cách ngang nhau, Attention tính toán mức độ liên quan giữa từng từ hiện tại (truy vấn - query) với các từ còn lại (khóa - key), sau đó sử dụng các trọng số này để tổng hợp thông tin từ các giá trị tương ứng (value). Việc tổng hợp giúp mô hình linh hoạt hơn trong việc hiểu ngữ cảnh và mối quan hệ giữa các từ, từ đó tạo ra kết quả chính xác và tự nhiên hơn.

Cơ chế Attention là nền tảng của nhiều mô hình hiện đại như Transformer và đóng vai trò then chốt trong sự phát triển của trí tuệ nhân tạo trong xử lý ngôn ngữ tự nhiên.

2.3.2. Kiến trúc lớp Attention trong Transformer

1. **Self-Attention**

Self-Attention là cơ chế cho phép mỗi từ trong câu tự "chú ý" đến các từ khác để hiểu ngữ cảnh. Mỗi từ được biểu diễn bằng ba vector:

Query (Q) => đặt câu hỏi về mối quan hệ của từ với các từ khác.

Key (K) => trả lời câu hỏi từ Query.

Value (V) => chứa thông tin của từ cần chú ý.

Mỗi từ sẽ tính toán mức độ tương quan với các từ khác bằng cách tính tích vô hướng giữa Query và Key, sau đó chia cho căn bậc hai của kích thước Key để chuẩn hóa. Kết quả được đưa qua hàm softmax để chuyển thành xác suất chú ý. Cuối cùng, các trọng số này được nhân với Value để thu được biểu diễn ngữ nghĩa của từ trong ngữ cảnh toàn câu.

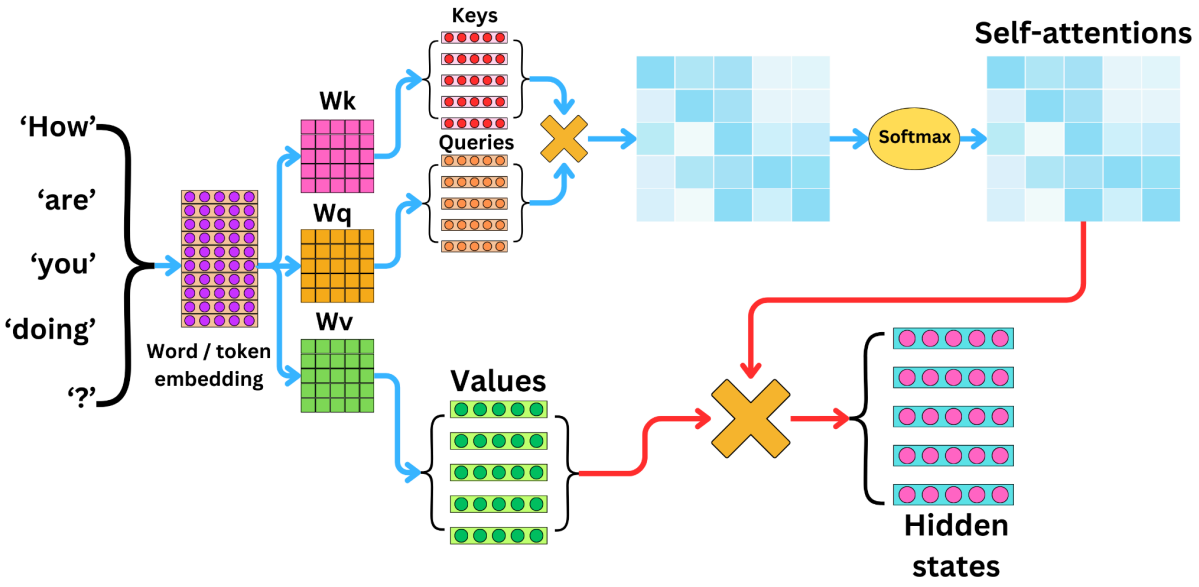
2. **Multi-head attention**

Để mô hình có thể học được nhiều mối quan hệ khác nhau trong câu, Multi-Head Attention chia các vector Q, K, V thành nhiều head và thực hiện các phép toán Attention song song. Kết quả từ các đầu này được ghép lại và đưa qua một lớp tuyến tính để thu được đầu ra cuối cùng. Việc này giúp mô hình học được các mối quan hệ phức tạp và đa dạng trong dữ liệu.

3. Positional encoding

Vì Transformer không sử dụng cấu trúc tuần tự như RNN, để mô hình nhận biết được thứ tự của các từ trong câu, Positional Encoding được thêm vào đầu vào. Kỹ thuật này sử dụng các hàm sin và cos với tần số khác nhau để tạo ra các vector vị trí, sau đó cộng chúng vào các vector embedding của từ, giúp mô hình nhận thức được vị trí của mỗi từ trong câu.

2.3.3. Nguyên tắc hoạt động:



Hình 4: Cơ chế Self-Attention

Cơ chế Attention hoạt động dựa trên việc tính toán trọng số cho mỗi từ trong chuỗi đầu vào, phản ánh mức độ quan trọng tương đối của từng từ đối với việc biểu diễn ngữ nghĩa tổng thể. Cụ thể, mỗi từ sẽ được so sánh với các từ còn lại để xác định mức độ đóng góp của nó vào việc tạo ra biểu diễn ngữ cảnh.

Các trọng số này sau đó được chuẩn hóa thông qua hàm Softmax, nhằm biến chúng thành một phân phối xác suất – đảm bảo tổng các trọng số bằng 1 và làm nổi bật những từ có mức ảnh hưởng lớn hơn.

Kết quả cuối cùng thu được là một tổng có trọng số của các vector từ đầu vào, trong đó những từ quan trọng nhất sẽ có ảnh hưởng lớn nhất đến vector đầu ra. Cơ chế này giúp mô hình tập trung hơn vào các phần thông tin có giá trị trong chuỗi đầu vào, từ đó nâng cao hiệu quả học và hiểu ngữ cảnh trong các bài toán xử lý ngôn ngữ tự nhiên.

Cụ thể hơn, thì nó thực hiện theo các nguyên tắc:

1. **Tính điểm attention:**

Mỗi phần tử trong chuỗi đầu vào được biểu diễn bằng ba vector: Query (Q), Key (K) và Value (V).

Đầu tiên, tính điểm tương quan (hay mức độ phù hợp) giữa Query và từng Key bằng phép nhân điểm (dot product):

$$score_{\{i,j\}} = Q_i \cdot K_j$$

(Nơi i là vị trí đang xét, j là các vị trí còn lại trong chuỗi.)

2. **Chia tích vô hướng theo tỉ lệ**

Đặc điểm của lớp attention mà chúng em sử dụng bao gồm việc áp dụng cơ chế attention tích vô hướng chia tỉ lệ (chia cho $\sqrt{d_k}$) để cải thiện sự ổn định trong quá trình huấn luyện vì khi số chiều của vector key và query lớn, nghĩa là d_k lớn thì phép nhân vô hướng có xu hướng tạo ra giá trị lớn làm cho hàm softmax trở nên nhọn và cực đoan. Nếu không chia tỉ lệ thì chỉ một vài phần tử có attention weight ≈ 1, còn lại ≈ 0, và mô hình cũng khó học vì gradient sẽ rất nhỏ (gần như không lan truyền được)

3. **Học ngữ cảnh qua giá trị của vector query và vector key**

Như đã nói ở trên, mỗi từ trong câu sẽ được biến đổi thành một vector query Q_i và một vector key K_j (cùng với một vector giá trị V_j).

Các ma trận trọng số W_q và W_k (được học trong quá trình huấn luyện) sẽ học cách chiếu các vector đầu vào ban đầu vào một không gian mới theo nguyên tắc:

Để A và B là hai từ có mối liên hệ với nhau, thì query của từ A (Q_A) và key của từ B (K_B) cần có tích vô hướng là một số dương càng lớn. Khi đó vector Q_A và vector K_B trong không gian mới này sẽ có xu hướng cùng chiều hoặc gần cùng chiều.

Nghĩa là: nếu tích vô hướng của một vector query của một từ với vector key của một từ khác có giá trị càng lớn về cực dương, thì hai từ đó có mối quan hệ càng mật thiết. Ngược lại nếu tích vô hướng của một vector query của từ đó với vector key của từ còn lại có giá trị càng lớn về cực âm, thì hai từ đó càng có mối quan hệ đối nghịch. Còn khi tích vô hướng càng gần 0 thì về mặt ngữ nghĩa hai từ đó càng không liên quan đến nhau.

4. **Chuẩn hóa bằng hàm softmax**

Các điểm tương đồng này (kết quả của Q · K) sau đó được đưa qua một hàm softmax.

Hàm softmax làm cho các điểm tương đồng dương lớn trở nên nổi bật hơn nhiều so với các điểm nhỏ hoặc âm, đồng thời chuẩn hóa các điểm này thành một phân phối xác suất, tức là các trọng số attention. Các trọng số này luôn dương và tổng của chúng bằng 1.

Công thức hàm softmax:

$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Một điểm tương đồng $Q_A \cdot K_B$ dương lớn sẽ dẫn đến một trọng số attention cao cho từ B khi từ A đang nhìn vào câu. Do đó mô hình hiểu được A và B là hai từ có liên quan đến nhau

Một điểm tương đồng âm (ví dụ $Q_A \cdot K_C$ là âm) sẽ dẫn đến một trọng số chú ý rất thấp (gần 0) cho từ C. Từ đó mô hình hiểu được rằng A và C là hai từ không có, hoặc ít liên quan đến nhau

5. **Trọng số attention nhấn mạnh và coi nhẹ/bỏ qua thông tin**

Sau khi có được các trọng số attention $\alpha_{i,j}$, chúng được sử dụng để tính toán vector đầu ra mới cho mỗi token i. Vector đầu ra này là một tổng có trọng số của tất cả các vector Value (V_j) trong chuỗi:

$$Output_i = \sum \alpha_{i,j}.V_j$$

Nếu một từ (token j) được xác định là có liên quan mật thiết và quan trọng đối với việc hiểu ngữ cảnh của từ hiện tại (token i) hoặc quan trọng đối với việc xác định thái độ chung của câu, nó sẽ nhận được trọng số attention $\alpha_{i,j}$ cao từ các query liên quan. Do đó, vector V_j của từ đó sẽ có đóng góp lớn hơn vào việc hình thành vector $output_i$. Trong bài toán phân loại thái độ, các từ như "tệ", "tuyệt vời", "rất", "không thể chấp nhận", v.v., hoặc các cụm từ mang ý nghĩa cảm xúc mạnh mẽ sẽ được mô hình học cách gán cho trọng số chú ý cao, giúp các đặc trưng của chúng được nổi bật hơn.

Ngược lại, những từ ít mang thông tin cảm xúc hoặc ít liên quan đến ngữ cảnh cần phân tích (ví dụ: các từ dùng, từ nối không mang sắc thái riêng) sẽ nhận được trọng số attention thấp. Do đó, các vector Value tương ứng của chúng sẽ có đóng góp không đáng kể vào vector đầu ra. Bằng cách này, mô hình hiệu quả "lọc bỏ" hoặc giảm thiểu ảnh hưởng của thông tin nhiễu hoặc không cần thiết.

Kết quả của lớp attention là một chuỗi các vector biểu diễn mới, trong đó mỗi vector đã được "tinh chỉnh" để phản ánh thông tin ngữ cảnh quan trọng nhất từ toàn bộ câu, đặc biệt là những thông tin hữu ích cho việc xác định thái độ. Các vector này sau đó được truyền đến các lớp tiếp theo để đưa ra dự đoán cuối cùng. Nhờ vậy, mô hình có thể tập trung vào những tín hiệu cảm xúc mạnh mẽ nhất, cải thiện độ chính xác và khả năng diễn giải của hệ thống.

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ MÔ HÌNH

3.1. Khái quát mô hình

Trong nghiên cứu này, chúng em trình bày việc phân tích, thiết kế và cài đặt một mô hình học sâu tiên tiến cho bài toán phân loại cảm xúc văn bản tiếng việt. Mô hình được xây dựng dựa trên sự kết hợp của các thành phần cốt lõi: mô hình ngôn ngữ PhoBERT được huấn luyện trước, kiến trúc mạng nơ-ron tuần tự hai chiều với các đơn vị nhớ ngắn-dài của BiLSTM, và cơ chế tập trung (attention). Mục tiêu chính là tăng cường khả năng xử lý hiệu quả dữ liệu tuần tự và nắm bắt các đặc trưng ngữ nghĩa phức tạp trong ngôn ngữ tự nhiên. Cụ thể, PhoBERT đóng vai trò trích xuất các embedding ngữ cảnh phong phú từ văn bản đầu vào. Tiếp đó, BiLSTM được sử dụng để khai thác thông tin từ cả hai chiều của chuỗi dữ liệu, giúp mô hình hiểu rõ hơn mối quan hệ phụ thuộc xa giữa các từ. Cuối cùng, lớp attention được tích hợp để cho phép mô hình tự động xác định và tập trung vào những từ hoặc cụm từ mang thông tin cảm xúc quan trọng nhất trong ngữ cảnh, từ đó cải thiện độ chính xác và hiệu suất tổng thể của mô hình trong việc phân loại cảm xúc thành các lớp tích cực, tiêu cực và trung tính.

3.2. Phân tích mô hình

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
embedding_input (InputLayer)	(None, 128, 768)	0	-
bidirectional (Bidirectional)	(None, 128, 128)	426,496	embedding_input[0][0]
attention (Attention)	(None, 128, 128)	1	bidirectional[0][0], bidirectional[0][0]
global_max_pooling1d (GlobalMaxPooling1D)	(None, 128)	0	attention[0][0]
dense (Dense)	(None, 64)	8,256	global_max_pooling1d[…
dropout (Dropout)	(None, 64)	0	dense[0][0]
sentiment_output (Dense)	(None, 3)	195	dropout[0][0]

Total params: 434,948 (1.66 MB)

Trainable params: 434,948 (1.66 MB)

Non-trainable params: 0 (0.00 B)

Hình 5: Mô hình phân loại cảm xúc sử dụng Attention và BiLSTM

3.2.1. Lớp embedding đầu vào của mô hình

Thành phần đầu tiên trong kiến trúc mô hình là lớp đầu vào, được định nghĩa để tiếp nhận dữ liệu đặc trưng đã được trích xuất. Đầu vào cho lớp này là các embedding của câu văn được trích xuất từ mô hình ngôn ngữ lớn PhoBERT. Dữ liệu đầu vào có hình dạng là (None, MAX_LEN, 768). Trong đó, None biểu thị kích thước lô (batch size) có thể linh hoạt thay đổi trong quá trình huấn luyện hoặc dự đoán; MAX_LEN là độ dài tối đa của một chuỗi token sau khi được mã hóa (trong bài này là 128); và 768 là số chiều của vector embedding tương ứng với mỗi token, đây là một đặc trưng của mô hình PhoBERT.

Lớp đầu vào này không thực hiện biến đổi dữ liệu mà chỉ định nghĩa khuôn dạng. Do đó, đầu ra của nó sẽ có hình dạng tương tự như đầu vào, tức là (None, 128, 768), và được truyền trực tiếp tới lớp xử lý tiếp theo trong mô hình. Kiểu dữ liệu của các embedding đầu vào là số thực 32-bit. Lớp này không có tham số nào cần huấn luyện; vai trò chính của nó là thông báo cho mô hình biết rằng mỗi mẫu dữ liệu sẽ là một ma trận có kích thước 128x768, tương ứng với 128 vector embedding (mỗi vector 768 chiều) đại diện cho một câu văn đã được mã hóa.

3.2.2. Lớp BiLSTM

Tiếp nhận đầu ra từ lớp embedding_input, kiến trúc mô hình sử dụng một lớp mạng nơ-ron tuần tự hai chiều. Đầu vào cho lớp BiLSTM này là chuỗi các embedding có hình dạng (None, 128, 768). Lớp LSTM bên trong được cấu hình với 64 đơn vị (units). Do được bao bọc bởi lớp Bidirectional, mô hình sẽ thực thi hai lớp LSTM một cách song song: một lớp xử lý chuỗi theo chiều từ trái sang phải (forward pass) và một lớp xử lý theo chiều ngược lại từ phải sang trái (backward pass). Kết quả đầu ra từ hai chiều này sau đó được ghép lại. Vì vậy, chiều đặc trưng của vector đầu ra tại mỗi bước thời gian là 128 (64 đơn vị từ chiều xuôi cộng với 64 đơn vị từ chiều ngược).

Hình dạng đầu ra của lớp BiLSTM này là (None, 128, 128). Tham số return_sequences của lớp LSTM được thiết lập là True, đảm bảo rằng lớp này sẽ trả về một chuỗi đầy đủ các vector đầu ra tại mỗi bước thời gian (tương ứng với mỗi token trong chuỗi 128 token), thay vì chỉ trả về vector đầu ra của bước thời gian cuối cùng. Chuỗi vector này cực kỳ quan trọng vì trong lớp attention tiếp theo trong kiến trúc cần xử lý toàn bộ chuỗi đầu ra này. Các hàm kích hoạt mặc

định của LSTM, bao gồm hàm tanh cho trạng thái cell và hàm sigmoid cho các cổng, được giữ nguyên.

Vai trò chính của lớp BiLSTM trong mô hình là học cách biến đổi chuỗi embedding đầu vào thành một chuỗi các vector đặc trưng mới, có kích thước (None, 128, 128). Các vector này chứa đựng thông tin ngữ cảnh phong phú hơn, đã được xử lý qua lại hai chiều, làm tiền đề cho các lớp xử lý sâu hơn trong mô hình.

3.2.3. Lớp Attention

Tiếp theo trong kiến trúc mô hình, sử dụng lớp attention để hiện thực hóa cơ chế self-attention. Lớp này nhận đầu vào là chuỗi các vector đặc trưng được tạo ra bởi lớp BiLSTM.

Cụ thể, đầu vào cho lớp attention này chính là bilstm_output, là kết quả đầu ra của lớp BiLSTM. bilstm_output có hình dạng là (None, MAX_LEN, 128), trong đó None đại diện cho kích thước lô (batch size), MAX_LEN là độ dài tối đa của chuỗi đầu vào (128 tokens), và 128 là số chiều của vector trạng thái ẩn tại mỗi token (kết hợp từ 64 chiều của LSTM xuôi và 64 chiều của LSTM ngược).

Khi sử dụng lớp attention trong Keras cho cơ chế self-attention, cả ba thành phần Query (Q), Key (K), và Value (V) đều được dẫn xuất từ cùng một nguồn đầu vào, chính là bilstm_output, có nghĩa là mô hình sẽ học cách để mỗi vị trí trong chuỗi bilstm_output "truy vấn" (query) và "so khớp" (key) với tất cả các vị trí khác trong chính chuỗi đó, sau đó tổng hợp thông tin (value) dựa trên mức độ phù hợp đã tính toán. Do Q, K, V cùng được tạo từ bilstm_output, chúng đều có hình dạng là (None, MAX_LEN, 128). Tham số use_scale=True được thiết lập khi khởi tạo lớp này, đảm bảo rằng mô hình áp dụng cơ chế attention tích vô hướng chia tỉ lệ giúp ổn định quá trình huấn luyện.

Đầu ra của lớp attention sẽ có cùng hình dạng với đầu vào, tức là (None, MAX_LEN, 128). Tuy nhiên, điểm khác biệt quan trọng nằm ở nội dung của các vector này. Mỗi vector tại mỗi bước thời gian trong chuỗi đầu ra đã được "trọng số hóa", nghĩa là, nó không chỉ chứa thông tin từ vị trí hiện tại mà còn là một sự tổng hợp có chọn lọc thông tin từ toàn bộ chuỗi đầu vào là toàn bộ output của BiLSTM, dựa trên mức độ quan trọng và liên quan mà cơ chế attention đã xác định. Các vector đầu ra này, do đó, mang thông tin ngữ cảnh tổng thể của câu một cách hiệu quả hơn, làm nổi bật những đặc trưng quan trọng đã được "chú ý" đến, sẵn sàng cho các lớp xử lý tiếp theo trong mô hình.

3.2.4. Lớp GlobalMaxPooling1D

Sau khi chuỗi các vector đặc trưng được xử lý và trọng số hóa bởi lớp attention (tạo ra attention_output với hình dạng (None, MAX_LEN, 128)), chúng em sử dụng một lớp gộp cực đại toàn cục một chiều. Lớp này có nhiệm vụ quan trọng là giảm chiều dữ liệu từ một chuỗi các vector đại diện cho từng token trong câu thành một vector duy nhất có kích thước cố định, đại diện cho toàn bộ câu.

Cụ thể, lớp GlobalMaxPooling1D nhận đầu vào là attention_output. Nó hoạt động bằng cách lấy giá trị lớn nhất dọc theo chiều thời gian (chiều MAX_LEN tương ứng với 128 token) cho từng chiều đặc trưng trong số 128 chiều đặc trưng của các vector đầu ra từ lớp Attention. Kết quả là một vector duy nhất có hình dạng (None, 128).

Mục đích chính của việc sử dụng lớp này là để tạo ra một vector "đại diện" súc tích cho toàn bộ câu, trong đó giữ lại những đặc trưng nổi bật nhất (những giá trị có độ lớn nhất) tại mỗi chiều đặc trưng. Những đặc trưng này chính là những thông tin đã được lớp Attention xác định là quan trọng và "nhấn mạnh". Bằng cách này, thông tin quan trọng nhất từ toàn bộ chuỗi được cô đọng lại thành một vector duy nhất, làm giảm độ phức tạp của dữ liệu và giúp các lớp phân loại tiếp theo hoạt động hiệu quả hơn. Vector đầu ra từ lớp GlobalMaxPooling1D này sau đó sẽ được đưa vào các lớp Dense để thực hiện việc phân loại cảm xúc cuối cùng.

3.2.5. Lớp Fully Connected (Dense)

Tiếp theo sau lớp gộp GlobalMaxPooling1D, mô hình của chúng em sử dụng một lớp kết nối đầy đủ. Lớp này đóng vai trò như một lớp ẩn trong phần mạng nơ-ron phục vụ cho việc phân loại cuối cùng. Đầu vào của lớp Dense này là pooled_output từ lớp GlobalMaxPooling1D, có hình dạng là (None, 128).

Lớp Dense này được cấu hình với 64 nơ-ron (units), và do đó, đầu ra của nó sẽ có hình dạng là (None, 64). Số lượng tham số có thể huấn luyện trong lớp này được tính bằng công thức (input_dim + 1) * units, tức là (128 + 1) * 64 = 8256 tham số (bao gồm trọng số và bias).

Lớp kết nối đầy đủ thực hiện một phép biến đổi tuyến tính trên vector đầu vào (nhân với ma trận trọng số và cộng với vector bias), sau đó áp dụng một hàm kích hoạt phi tuyến. Trong mô hình của chúng em, hàm kích hoạt được sử dụng là ReLU, được định nghĩa là f(x)=max(0,x). Việc sử dụng hàm ReLU giúp

mô hình có khả năng học các mối quan hệ phi tuyến phức tạp tồn tại trong dữ liệu, cho phép nó biểu diễn các đặc trưng ở mức độ trừu tượng cao hơn.

Vai trò của lớp Dense này là học cách kết hợp và biến đổi các đặc trưng đã được trích xuất và cô đọng từ vector pooled_output. Nó giúp tạo ra một không gian đặc trưng mới, nhỏ hơn (từ 128 chiều xuống 64 chiều), mà ở đó các lớp thái độ có thể được phân tách tốt hơn, chuẩn bị cho lớp phân loại cuối cùng.

3.2.6. Lớp Dropout

Sau lớp Dense với 64 units, chúng em áp dụng một lớp Dropout. Lớp này nhận đầu vào là kết quả từ lớp Dense trước đó và tạo ra đầu ra có cùng hình dạng (None, 64).

Dropout là một kỹ thuật điều chuẩn được sử dụng rộng rãi để giảm thiểu hiện tượng quá khớp (overfitting) trong các mô hình học sâu. Hiện tượng quá khớp xảy ra khi mô hình học quá tốt trên dữ liệu huấn luyện, bao gồm cả nhiễu và các đặc điểm không tổng quát, dẫn đến việc mô hình hoạt động kém trên dữ liệu mới chưa từng thấy.

Trong quá trình huấn luyện, lớp Dropout với tỷ lệ được thiết lập (nhóm chọn mức 0.5, tương ứng 50%) sẽ ngẫu nhiên đặt một phần các giá trị đầu ra của lớp trước đó (tức là các nơ-ron trong lớp Dense 64 units) thành 0 tại mỗi bước huấn luyện. Việc tắt ngẫu nhiên các nơ-ron này buộc các nơ-ron còn lại trong mạng phải học các đặc trưng mạnh mẽ và độc lập hơn, thay vì phụ thuộc quá nhiều vào một vài nơ-ron cụ thể nào đó.

3.2.7. Lớp output

Lớp cuối cùng trong kiến trúc mô hình của chúng em là một lớp kết nối đầy đủ đóng vai trò là lớp đầu ra, chịu trách nhiệm đưa ra dự đoán phân loại thái độ cuối cùng.

Lớp này nhận đầu vào là kết quả từ lớp Dropout trước đó, một vector đặc trưng có hình dạng (None, 64). Số lượng nơ-ron (units) trong lớp đầu ra này bằng với số lượng lớp thái độ mà mô hình cần phân loại. Trong bài, chúng em phân loại thành ba lớp (tích cực, tiêu cực, trung tính), do đó, lớp đầu ra có 3 nơ-ron, và hình dạng đầu ra của nó là (None, 3). Số lượng tham số có thể huấn luyện trong lớp này được tính bằng (input_dim + 1) * units = (64 + 1) * 3 = 195 tham số.

Hàm kích hoạt được sử dụng cho lớp đầu ra này là softmax. Hàm softmax có vai trò quan trọng trong các bài toán phân loại đa lớp. Nó nhận một vector

các giá trị đầu vào (logits) từ các nơ-ron của lớp Dense và biến đổi chúng thành một vector các xác suất. Mỗi phần tử trong vector xác suất này tương ứng với xác suất mà mẫu đầu vào thuộc về một lớp cụ thể. Ví dụ, nếu ba lớp là "tích cực", "trung tính", "tiêu cực", thì vector đầu ra sẽ có dạng [P(tích cực), P(trung tính), P(tiêu cực)]. Các giá trị xác suất này luôn dương và tổng của chúng bằng 1.

Sau khi có được vector xác suất từ hàm softmax, lớp có xác suất cao nhất sẽ được chọn làm nhãn dự đoán cuối cùng cho câu văn đầu vào. Ví dụ, nếu vector xác suất là [0.1, 0.2, 0.7], mô hình sẽ dự đoán câu văn đó mang thái độ "tiêu cực".

CHƯƠNG 4: THỰC NGHIỆM VÀ ĐÁNH GIÁ

4.1. Dữ liệu và tiền xử lý

4.1.1. Nguồn dữ liệu

Trong nghiên cứu này, chúng em sử dụng một tập hợp các văn bản tiếng Việt đã được gán nhãn về thái độ để xây dựng mô hình phân loại. Bộ dữ liệu được phân thành ba nhóm nhãn chính, bao gồm: nhãn "pos" đại diện cho các văn bản có thái độ tích cực, nhãn "neu" (neutral) thể hiện thái độ trung lập, và nhãn "neg" (negative) dành cho các văn bản mang thái độ tiêu cực. Nguồn gốc của dữ liệu này được chúng em thu thập từ các diễn đàn trực tuyến, mạng xã hội, hoặc các đánh giá sản phẩm trên ứng dụng thương mại điện tử..

Về mặt kích thước, bộ dữ liệu ban đầu bao gồm 38854 dòng văn bản. Trong quá trình kiểm tra sơ bộ, chúng em đã phát hiện 24 giá trị rỗng (null) trong cột nội dung ('content') và không có giá trị rỗng nào trong cột nhãn ('label'). Sau khi thực hiện loại bỏ các dòng chứa giá trị rỗng, số lượng mẫu dữ liệu còn lại để chúng em sử dụng là 38830. Phân tích phân phối nhãn trên tập dữ liệu đã làm sạch cho thấy sự không cân bằng giữa các lớp: lớp "pos" (tích cực) chiếm số lượng lớn nhất với 20078 mẫu, tiếp theo là lớp "neg" (tiêu cực) với 11510 mẫu, và lớp "neu" (trung lập) có số lượng ít nhất là 7242 mẫu. Sự chênh lệch này sẽ được chúng em xem xét và xử lý trong các bước tiền xử lý tiếp theo để đảm bảo hiệu quả của mô hình huấn luyện.

4.1.2. Tiền xử lý train model:

4.1.2.1. Xử lý văn bản đầu vào

Dữ liệu văn bản thô thường chứa nhiều yếu tố nhiễu có thể ảnh hưởng đến chất lượng của mô hình học máy. Do đó, việc làm sạch và chuẩn hóa văn bản là một bước quan trọng trước khi đưa dữ liệu vào mô hình. Trong nghiên cứu này, chúng em đã áp dụng một chuỗi các thao tác xử lý văn bản như sau:

1. Đầu tiên, để đảm bảo tính nhất quán, toàn bộ văn bản đầu vào được chuyển đổi thành dạng chữ thường. Thao tác này giúp đồng nhất các từ viết hoa và viết thường, ví dụ như "Ngon" sẽ được chuyển thành "ngon".
2. Tiếp theo, các thẻ HTML, nếu có, được loại bỏ khỏi văn bản bằng cách sử dụng biểu thức chính quy để tìm và xóa các cấu trúc thẻ. Các thẻ như định dạng chữ đậm, chữ nghiêng, hay thẻ đoạn văn bản thường không mang thông tin ngữ nghĩa cần thiết cho bài toán phân loại cảm xúc và cần được loại bỏ.

3. Các đường dẫn URL cũng là một yếu tố nhiễu phổ biến trong dữ liệu văn bản thu thập từ internet. Chúng em đã sử dụng một biểu thức chính quy khác để phát hiện và loại bỏ tất cả các chuỗi ký tự bắt đầu bằng "http" hoặc "https".
4. Việc chuẩn hóa khoảng trắng cũng được thực hiện. Bước này giúp thay thế nhiều khoảng trắng liên tiếp bằng một khoảng trắng duy nhất, đồng thời loại bỏ các ký tự xuống dòng hoặc tab không cần thiết, ví dụ như chuỗi có nhiều khoảng trắng thừa sẽ được chuẩn hóa thành một chuỗi với các từ cách nhau bởi một khoảng trắng duy nhất.
5. Cuối cùng, các khoảng trắng thừa ở đầu và cuối mỗi chuỗi văn bản được loại bỏ.

Thông qua các bước xử lý này, dữ liệu văn bản trở nên gọn gàng và sạch sẽ hơn, loại bỏ được các thành phần không mang tính thông tin hoặc gây nhiễu, từ đó giúp mô hình dễ dàng hơn trong việc học và trích xuất các đặc trưng quan trọng.

4.1.2.2. Văn bản đầu vào được chuyển thành dạng vector trọng số thông qua PhoBERT-base

Trong bài tập lớn của nhóm, PhoBERT (vinai/phobert-base-v2) được sử dụng trong bước tiền xử lý để chuyển đổi các từ trong văn bản thành các vector đặc trưng (embeddings). Đây là bước quan trọng giúp các mô hình học máy có thể hiểu và phân tích văn bản.

Input của PhoBERT: Các câu hoặc đoạn văn tiếng Việt, sau khi được tokenized, sẽ là đầu vào cho PhoBERT. Quá trình tokenization sẽ tách các câu thành các từ hoặc cụm từ. Ví dụ, câu "Tôi thích học lập trình" sẽ được chia thành các token như ["Tôi", "thích", "học", "lập", "trình"].

Output của PhoBERT: PhoBERT sẽ chuyển đổi các token này thành các vector đặc trưng. Mỗi từ trong câu sẽ có một vector đại diện cho ngữ nghĩa của từ đó trong ngữ cảnh của toàn bộ câu. Các vector này là các đại diện số với d-dimensional embeddings (vector đặc trưng d chiều). Được biểu diễn dưới dạng nhiều chiều vì mỗi chiều trong vector thể hiện một khía cạnh ngữ nghĩa nào đó của từ, việc dùng nhiều chiều giúp mô hình có thể nắm bắt tốt hơn các đặc trưng phức tạp trong ngôn ngữ.

4.1.2.3. Mã hoá nhãn và chia tập dữ liệu

1. Mã hóa nhãn

Các mô hình học máy thường yêu cầu đầu vào ở dạng số để có thể xử lý hiệu quả. Do đó, các nhãn thái độ ban đầu của dữ liệu, vốn ở dạng chữ ("POS", "NEG", "NEU"), cần được chuyển đổi sang một dạng biểu diễn số tương ứng. Để thực hiện việc này, chúng em đã sử dụng công cụ LabelEncoder từ thư viện scikit-learn.

Quá trình mã hóa nhãn được tiến hành qua các bước sau:

1. Đầu tiên, một đối tượng LabelEncoder được khởi tạo. Tiếp theo, phương thức fit_transform() của đối tượng này được áp dụng trên dữ liệu nhãn của tập huấn luyện, giúp LabelEncoder học được ánh xạ duy nhất từ mỗi nhãn dạng chữ sang một nhãn dạng số nguyên, đồng thời thực hiện việc biến đổi trực tiếp trên tập dữ liệu nhãn huấn luyện.

   Sau khi đã học được ánh xạ từ tập huấn luyện, phương thức transform() được sử dụng để áp dụng cùng một ánh xạ đó lên dữ liệu nhãn của tập kiểm định và tập kiểm thử nhằm đảm bảo tính nhất quán trong việc mã hóa, tránh tình trạng cùng một nhãn chữ lại được mã hóa thành các số khác nhau giữa các tập

   Kết quả của quá trình mã hóa này là các nhãn thái độ được ánh xạ như sau: nhãn "NEG" (tiêu cực) được mã hóa thành số 0, nhãn "NEU" (trung tính) được mã hóa thành số 1, và nhãn "POS" (tích cực) được mã hóa thành số 2.
2. Cuối cùng, sau khi hoàn tất việc mã hóa, các nhãn dạng số nguyên này đã được chuyển đổi sang kiểu dữ liệu số thực 32-bit (numpy.float32) nhằm đảm bảo sự tương thích với yêu cầu đầu vào của một số hàm mất mát và các thành phần khác trong mô hình.

2. Chia tập dữ liệu

Sau khi làm sạch và mã hóa nhãn, bước tiếp theo trong quá trình tiền xử lý là phân chia toàn bộ tập dữ liệu đã được xử lý (bao gồm cả dữ liệu đặc trưng X và nhãn y) thành các tập con riêng biệt. Đây là một thông lệ chuẩn trong xây dựng mô hình học máy, nhằm mục đích: huấn luyện mô hình trên một phần dữ liệu (tập huấn luyện - train set), tinh chỉnh các siêu tham số và theo dõi quá trình học của mô hình trên một tập dữ liệu độc lập mà không làm rò rỉ thông tin từ tập kiểm thử (tập kiểm định - validation set), và cuối cùng là đánh giá hiệu năng tổng quát của mô hình trên dữ liệu hoàn toàn mới mà mô hình chưa từng nhìn thấy trong suốt quá trình huấn luyện và tinh chỉnh (tập kiểm thử - test set).

Để thực hiện việc phân chia này, chúng em đã sử dụng hàm train_test_split từ thư viện scikit-learn.model_selection. Quá trình phân chia được thực hiện qua hai bước chính:

1. Đầu tiên, toàn bộ tập dữ liệu, sau khi loại bỏ các giá trị null và trước khi cân bằng, được chia thành một tập tạm thời và tập kiểm thử. Tỷ lệ dành cho tập kiểm thử được ấn định là 10% tổng số mẫu dữ liệu.
2. Tiếp theo, tập dữ liệu tạm thời (chiếm 90% dữ liệu ban đầu) được tiếp tục phân chia thành tập huấn luyện và tập kiểm định. Tỷ lệ dành cho tập kiểm định được tính toán là 0.1 chia cho 0.9 (tức là khoảng 11.11% của tập tạm thời), tương đương với việc tập kiểm định sẽ chiếm khoảng 10% tổng số mẫu dữ liệu ban đầu. Do đó, tập huấn luyện sẽ chiếm khoảng 80% tổng số mẫu dữ liệu ban đầu.

Kết quả của quá trình phân chia này cho ra số lượng mẫu là:

Số lượng mẫu trong tập huấn luyện (X_train): 31063

Số lượng mẫu trong tập kiểm định (X_val): 3884

Số lượng mẫu trong tập kiểm thử (X_test): 3883

4.2. Tham số huấn luyện

Độ dài chuỗi tối đa (MAX_LEN) là 128 tokens, các câu dài hơn sẽ được cắt bớt và các câu ngắn hơn sẽ được đệm thêm vào để đạt được độ dài này.

Hàm tối ưu Adam với tốc độ học ban đầu được thiết lập là 1×10^{-4}.

Do đây là bài toán phân loại đa lớp với nhãn dạng số nguyên (0, 1, 2), hàm mất mát sparse_categorical_crossentropy đã được lựa chọn.

Chỉ số chính được sử dụng để theo dõi hiệu năng của mô hình trong quá trình huấn luyện và đánh giá là độ chính xác (accuracy).

Dữ liệu huấn luyện được đưa vào mô hình theo từng lô có kích thước là 32 mẫu.

Mô hình được thiết lập để huấn luyện trong tối đa 30 epoch. Tuy nhiên, quá trình huấn luyện có thể dừng sớm hơn nhờ vào cơ chế EarlyStopping.

Để kiểm soát và tối ưu hóa quá trình huấn luyện, chúng em đã sử dụng một số cơ chế callbacks:

- EarlyStopping theo dõi giá trị hàm mất mát trên tập kiểm định. Nếu val_loss không cải thiện sau một số lượng epoch nhất định (set là 7), quá trình huấn luyện sẽ tự động dừng lại. Cơ chế này được cấu hình để khôi phục lại trọng số của mô hình từ epoch có val_loss tốt nhất.
- ReduceLROnPlateau: Cơ chế này cũng theo dõi val_loss. Nếu val_loss không có sự cải thiện sau patience=3 epoch, tốc độ học sẽ được giảm đi còn 0.2 (tức là còn 20% so với tốc độ học hiện tại). Tốc độ học sẽ không bị giảm xuống dưới một ngưỡng tối thiểu là 1×10^{-7}.
- ModelCheckpoint: Cơ chế này lưu lại toàn bộ mô hình (kiến trúc, trọng số và trạng thái của hàm tối ưu) mỗi khi tìm thấy một giá trị val_loss tốt hơn so với các epoch trước đó.
- TensorBoard: Cơ chế này được sử dụng để ghi lại các thông tin của quá trình huấn luyện (như hàm mất mát, độ chính xác qua các epoch, đồ thị mô hình). Các thông tin này có thể được trực quan hóa bằng công cụ TensorBoard, giúp theo dõi và phân tích chi tiết hơn về quá trình học của mô hình. Tần suất ghi histogram là mỗi epoch.

4.3. Kết quả và đánh giá mô hình

4.3.1. Kết quả huấn luyện

```
--- Bắt đầu huấn luyện model với các callbacks đã chuẩn bị ---
Epoch 1/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 443ms/step - accuracy: 0.5287 - loss: 0.9278
Epoch 1: val_loss improved from inf to 0.70123, saving model to /kaggle/working/model_checkpoints/mode
l_epoch-01_valloss-0.7012.keras
544/544 ━━━━━━━━━━━━━━━━━━━━ 263s 459ms/step - accuracy: 0.5289 - loss: 0.9277 - val_accuracy: 0.6820 -
val_loss: 0.7012 - learning_rate: 1.0000e-04
Epoch 2/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 444ms/step - accuracy: 0.6897 - loss: 0.6939
Epoch 2: val_loss improved from 0.70123 to 0.65278, saving model to /kaggle/working/model_checkpoints/
model_epoch-02_valloss-0.6528.keras
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 457ms/step - accuracy: 0.6897 - loss: 0.6939 - val_accuracy: 0.7128 -
val_loss: 0.6528 - learning_rate: 1.0000e-04
Epoch 3/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 446ms/step - accuracy: 0.7144 - loss: 0.6468
Epoch 3: val_loss improved from 0.65278 to 0.65035, saving model to /kaggle/working/model_checkpoints/
model_epoch-03_valloss-0.6504.keras
544/544 ━━━━━━━━━━━━━━━━━━━━ 250s 459ms/step - accuracy: 0.7144 - loss: 0.6468 - val_accuracy: 0.7073 -
val_loss: 0.6504 - learning_rate: 1.0000e-04
Epoch 4/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.7308 - loss: 0.6241
Epoch 4: val_loss improved from 0.65035 to 0.62015, saving model to /kaggle/working/model_checkpoints/
model_epoch-04_valloss-0.6202.keras
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 458ms/step - accuracy: 0.7308 - loss: 0.6241 - val_accuracy: 0.7230 -
val_loss: 0.6202 - learning_rate: 1.0000e-04
Epoch 5/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 446ms/step - accuracy: 0.7399 - loss: 0.6016
Epoch 5: val_loss improved from 0.62015 to 0.61523, saving model to /kaggle/working/model_checkpoints/
model_epoch-05_valloss-0.6152.keras
544/544 ━━━━━━━━━━━━━━━━━━━━ 250s 459ms/step - accuracy: 0.7399 - loss: 0.6015 - val_accuracy: 0.7239 -
val_loss: 0.6152 - learning_rate: 1.0000e-04
```

Hình 6: Kết quả huấn luyện

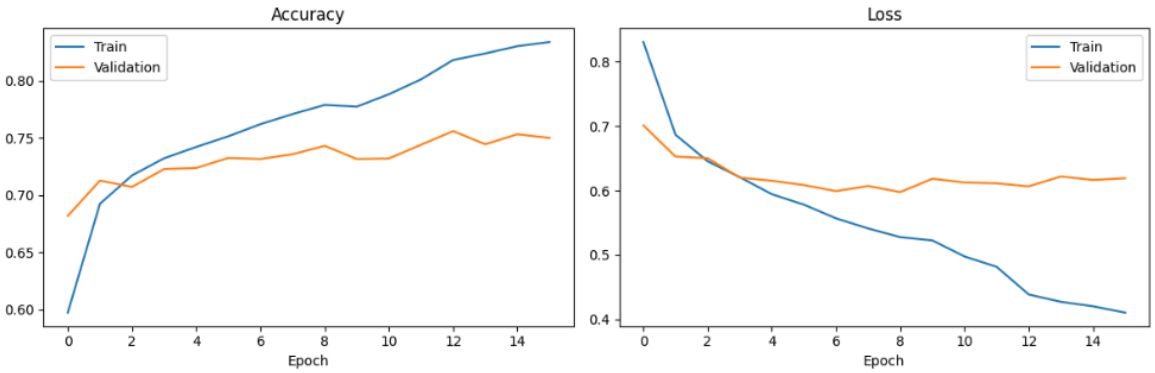
```
Epoch 6/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.7523 - loss: 0.5739
Epoch 6: val_loss improved from 0.61523 to 0.60853, saving model to /kaggle/working/model_checkpoints/
model_epoch-06_valloss-0.6085.keras
544/544 ━━━━━━━━━━━━━━━━━━━━ 250s 459ms/step - accuracy: 0.7523 - loss: 0.5739 - val_accuracy: 0.7326 -
val_loss: 0.6085 - learning_rate: 1.0000e-04
Epoch 7/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 447ms/step - accuracy: 0.7592 - loss: 0.5557
Epoch 7: val_loss improved from 0.60853 to 0.59912, saving model to /kaggle/working/model_checkpoints/
model_epoch-07_valloss-0.5991.keras
544/544 ━━━━━━━━━━━━━━━━━━━━ 254s 466ms/step - accuracy: 0.7593 - loss: 0.5557 - val_accuracy: 0.7317 -
val_loss: 0.5991 - learning_rate: 1.0000e-04
Epoch 8/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.7765 - loss: 0.5345
Epoch 8: val_loss did not improve from 0.59912
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 458ms/step - accuracy: 0.7765 - loss: 0.5345 - val_accuracy: 0.7358 -
val_loss: 0.6071 - learning_rate: 1.0000e-04
Epoch 9/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.7791 - loss: 0.5243
Epoch 9: val_loss improved from 0.59912 to 0.59764, saving model to /kaggle/working/model_checkpoints/
model_epoch-09_valloss-0.5976.keras
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 458ms/step - accuracy: 0.7791 - loss: 0.5243 - val_accuracy: 0.7432 -
val_loss: 0.5976 - learning_rate: 1.0000e-04
Epoch 10/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.7806 - loss: 0.5112
Epoch 10: val_loss did not improve from 0.59764
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 458ms/step - accuracy: 0.7806 - loss: 0.5113 - val_accuracy: 0.7317 -
val_loss: 0.6184 - learning_rate: 1.0000e-04
Epoch 11/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.7828 - loss: 0.4983
Epoch 11: val_loss did not improve from 0.59764
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 458ms/step - accuracy: 0.7828 - loss: 0.4983 - val_accuracy: 0.7322 -
val_loss: 0.6125 - learning_rate: 1.0000e-04
```

Hình 7: Kết quả huấn luyện

```
Epoch 12/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.7990 - loss: 0.4806
Epoch 12: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.

Epoch 12: val_loss did not improve from 0.59764
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 458ms/step - accuracy: 0.7990 - loss: 0.4806 - val_accuracy: 0.7441 -
val_loss: 0.6112 - learning_rate: 1.0000e-04
Epoch 13/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 442ms/step - accuracy: 0.8146 - loss: 0.4512
Epoch 13: val_loss did not improve from 0.59764
544/544 ━━━━━━━━━━━━━━━━━━━━ 248s 456ms/step - accuracy: 0.8146 - loss: 0.4512 - val_accuracy: 0.7561 -
val_loss: 0.6066 - learning_rate: 2.0000e-05
Epoch 14/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.8248 - loss: 0.4269
Epoch 14: val_loss did not improve from 0.59764
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 458ms/step - accuracy: 0.8248 - loss: 0.4269 - val_accuracy: 0.7446 -
val_loss: 0.6219 - learning_rate: 2.0000e-05
Epoch 15/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 445ms/step - accuracy: 0.8337 - loss: 0.4162
Epoch 15: ReduceLROnPlateau reducing learning rate to 3.999999898951501e-06.

Epoch 15: val_loss did not improve from 0.59764
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 458ms/step - accuracy: 0.8337 - loss: 0.4162 - val_accuracy: 0.7533 -
val_loss: 0.6165 - learning_rate: 2.0000e-05
Epoch 16/30
544/544 ━━━━━━━━━━━━━━━━━━━━ 0s 444ms/step - accuracy: 0.8410 - loss: 0.4033
Epoch 16: val_loss did not improve from 0.59764
544/544 ━━━━━━━━━━━━━━━━━━━━ 249s 457ms/step - accuracy: 0.8409 - loss: 0.4033 - val_accuracy: 0.7501 -
val_loss: 0.6191 - learning_rate: 4.0000e-06
Epoch 16: early stopping
Restoring model weights from the end of the best epoch: 9.

--- Hoàn thành huấn luyện model ---
```

Hình 8: Kết quả huấn luyện

Hiệu năng model:



Hình 9: Hiệu năng huấn luyện mô hình

4.3.2. Đánh giá mô hình

Độ chính xác trên tập huấn luyện, train accuracy, bắt đầu từ khoảng 0.5374 ở epoch đầu tiên, độ chính xác trên tập huấn luyện có xu hướng tăng khá nhanh và đều đặn qua các epoch. Đến epoch thứ 10, chỉ số này đạt khoảng

0.7816 và tiếp tục cải thiện ở các epoch sau, ví dụ như ở epoch 17 (epoch cuối cùng trước khi dừng sớm), độ chính xác huấn luyện đạt khoảng 0.8390. Sự tăng trưởng này cho thấy mô hình đang học tốt và hiệu quả trên dữ liệu huấn luyện.

Độ chính xác trên tập kiểm định, validation accuracy, bắt đầu ở mức 0.6912 tại epoch đầu tiên. Nó có sự cải thiện và đạt giá trị cao nhất là 0.7547 ở epoch thứ 17. Tại epoch thứ 10, val_accuracy là 0.7464, đây là một trong những điểm tốt nhất.

Từ khoảng epoch thứ 2 trở đi, độ chính xác trên tập huấn luyện bắt đầu vượt qua độ chính xác trên tập kiểm định, và khoảng cách này có xu hướng mở rộng hơn ở các epoch sau. Đây là một dấu hiệu cho thấy mô hình có thể đang bắt đầu overfitting với dữ liệu huấn luyện, tức là mô hình học thuộc các đặc điểm nhiễu hoặc riêng biệt của tập huấn luyện, làm giảm khả năng tổng quát hóa trên tập train.

Hàm mất mát trên tập huấn luyện, train loss, giảm đều và khá nhanh từ khoảng 0.9272 ở epoch đầu tiên xuống còn khoảng 0.3988 ở epoch thứ 17, củng cố thêm nhận định rằng mô hình đang học tốt trên tập huấn luyện.

Hàm mất mát trên tập kiểm định , validation loss, bắt đầu ở mức 0.6880, val_loss giảm xuống giá trị thấp nhất là 0.5977 ở epoch thứ 10. Sau epoch này, val_loss có xu hướng không cải thiện thêm hoặc thậm chí tăng nhẹ

Dựa trên log huấn luyện, mô hình đã dừng lại ở epoch thứ 17. Cơ chế EarlyStopping được thiết lập để theo dõi val_loss với patience=7. Tuy nhiên, do có cơ chế ModelCheckpoint lưu lại mô hình tốt nhất dựa trên val_loss và restore_best_weights=True được kích hoạt trong EarlyStopping, mô hình cuối cùng sẽ được khôi phục về trọng số của epoch mà tại đó val_loss đạt giá trị thấp nhất. Theo log, val_loss tốt nhất là 0.5977, đạt được ở epoch thứ 10. Do đó, mô hình cuối cùng được sử dụng sẽ là mô hình tại epoch thứ 10.

Nhận xét chung về quá trình huấn luyện:

Mô hình đạt hiệu suất tốt nhất trên tập kiểm định ở epoch thứ 10 với val_accuracy khoảng 0.7464 và val_loss khoảng 0.5977. Có dấu hiệu của hiện tượng overfitting khi độ chính xác trên tập huấn luyện tiếp tục tăng mạnh và hàm mất mát trên tập huấn luyện tiếp tục giảm, trong khi các chỉ số tương ứng trên tập kiểm định không còn cải thiện nhiều hoặc bắt đầu có xu hướng xấu đi sau epoch thứ 10. Cơ chế EarlyStopping kết hợp với ModelCheckpoint đã hoạt động hiệu quả, giúp ngăn mô hình tiếp tục huấn luyện sâu vào vùng quá khớp và

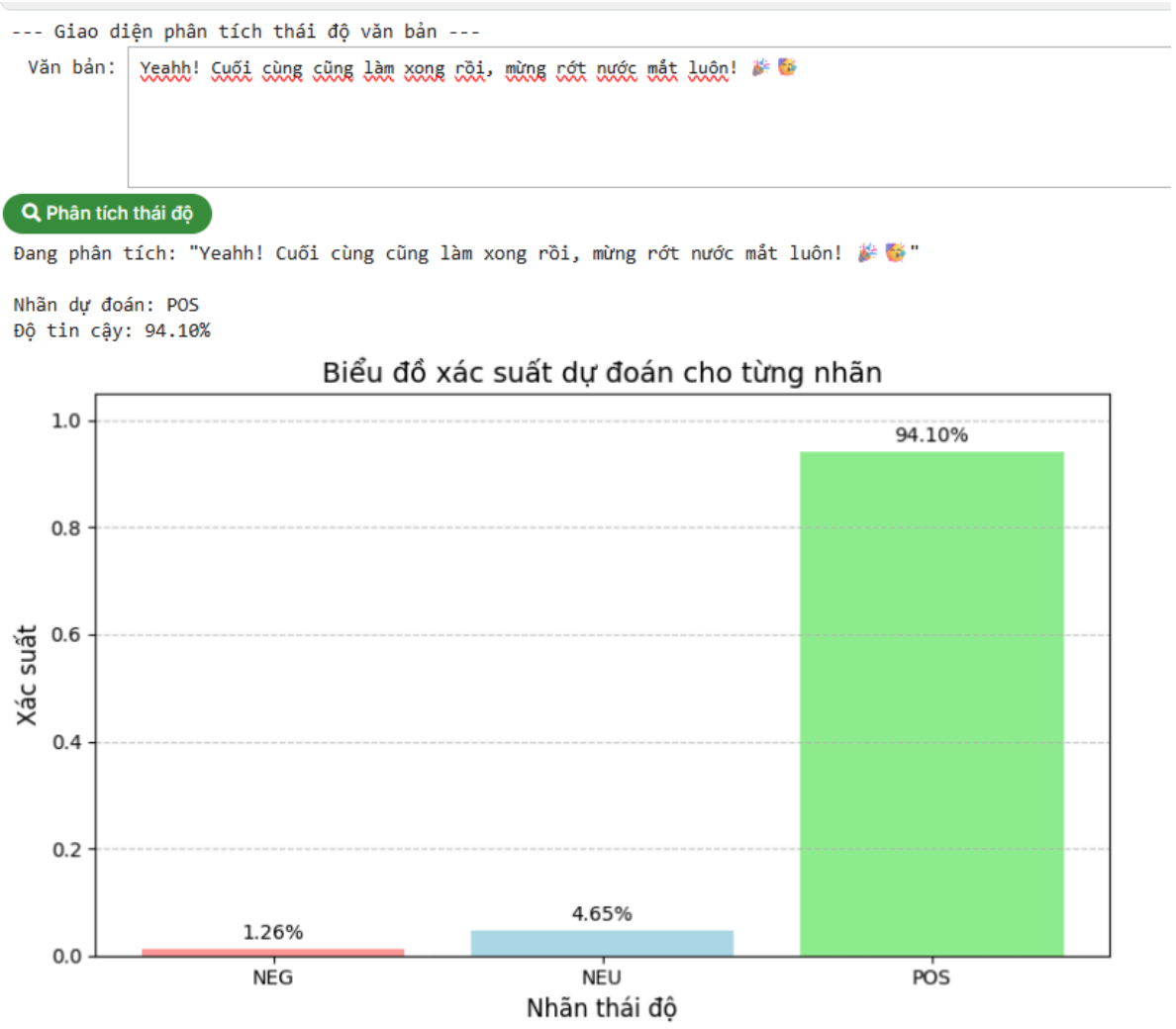
chọn ra được phiên bản mô hình có khả năng tổng quát hóa tốt hơn (tại epoch thứ 10). Thời gian huấn luyện cho mỗi epoch dao động trong khoảng 250-265 giây, đây là một khoảng thời gian hợp lý cho việc huấn luyện một mô hình với kiến trúc và lượng dữ liệu như trong bài tập lớn này.

4.4. Thử nghiệm mô hình

```
--- Bắt đầu dự đoán cho các câu test ---
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

Văn bản: Sản phẩm này thực sự tuyệt vời, tôi rất hài lòng với chất lượng
  Nhãn dự đoán: POS
  Độ tin cậy: 95.96%
---
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step

Văn bản: Dịch vụ khách hàng quá tệ, tôi sẽ không bao giờ quay lại nữa
  Nhãn dự đoán: NEG
  Độ tin cậy: 95.99%
---
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step

Văn bản: Món ăn bình thường, không có gì đặc biệt
  Nhãn dự đoán: NEU
  Độ tin cậy: 97.90%
---
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step

Văn bản: Quá thất vọng, không như quảng cáo
  Nhãn dự đoán: NEG
  Độ tin cậy: 97.50%
---
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step

Văn bản: Rất đáng tiền, sẽ mua lại ủng hộ shop!
  Nhãn dự đoán: POS
  Độ tin cậy: 92.68%
---
```

Hình 10: Thử nghiệm mô hình

Xây dựng giao diện đơn giản cho phép nhập câu muốn đoán:



Hình 11: Giao diện phân tích

4.5. Hạn chế và cải tiến mô hình

4.5.1. Hạn chế của mô hình hiện tại:

1. **Còn xảy ra hiện tượng overfitting**

Phân tích biểu đồ accuracy và loss trong quá trình huấn luyện cho thấy có sự chênh lệch đáng kể giữa hiệu suất trên tập huấn luyện và tập validation. Cụ thể, trong khi accuracy trên tập huấn luyện tiếp tục tăng và loss giảm sâu, các chỉ số tương ứng trên tập validation có xu hướng chững lại hoặc thậm chí tệ đi sau một số epoch nhất định. Qua đây cho thấy mô hình có xu hướng "học vẹt" dữ liệu huấn luyện, làm giảm khả năng tổng quát hóa trên dữ liệu mới. Mặc dù EarlyStopping đã được sử dụng để chọn ra trọng số tốt nhất dựa trên val_loss, việc giảm thiểu overfitting ngay từ đầu vẫn là một ưu tiên.

2. **Mất cân bằng dữ liệu**

Phân tích phân phối nhãn trong tập dữ liệu cho thấy có sự chênh lệch về số lượng mẫu giữa các lớp cảm xúc (ví dụ, lớp 'POS' có số lượng mẫu lớn hơn đáng kể so với lớp 'NEU'). Sự mất cân bằng này có thể khiến mô hình ưu tiên học các đặc trưng của lớp đa số và hoạt động kém hiệu quả hơn trên các lớp thiểu số, dẫn đến dự đoán thiếu chính xác cho các lớp ít dữ liệu hơn.

3. **Độ phức tạp và thời gian suy luận**

Kiến trúc kết hợp nhiều thành phần (Transformer, BiLSTM, Attention) làm cho mô hình trở nên khá phức tạp nên có thể dẫn đến thời gian suy luận lâu hơn so với các mô hình đơn giản hơn, đây là một yếu tố cần cân nhắc nếu ứng dụng yêu cầu tốc độ phản hồi nhanh.

4.5.2. Đề xuất hướng cải thiện và phát triển

Tăng kích thước tập dữ liệu huấn luyện để đảm bảo mô hình có thêm thông tin để cải thiện khả năng tổng quát hóa, mở rộng tập dữ liệu huấn luyện với nhiều mẫu hơn và đa dạng hơn về ngữ cảnh, sắc thái biểu cảm, giúp mô hình học được các đặc trưng mạnh mẽ và tổng quát hơn.

Giải quyết tình trạng overfitting bằng một số phương pháp ví dụ thử nghiệm tăng tỷ lệ dropout trong các lớp Dense hoặc thêm dropout sau lớp BiLSTM….

Phát triển thêm khả năng xử lý văn bản sắc thái tinh tế hoặc mơ hồ, nơi mà câu văn có sắc thái cảm xúc không rõ ràng, trung tính thực sự, hoặc chứa đựng sự mỉa mai, châm biếm mà đòi hỏi hiểu biết ngữ cảnh sâu hơn.

Đánh giá mô hình kỹ lưỡng hơn bằng cách sử dụng các bộ dữ liệu kiểm tra chuyên biệt để đánh giá khả năng xử lý các loại văn bản khác nhau.

Kết luận: Dự án này đã chứng minh tiềm năng của mạng BiLSTM trong phân tích cảm xúc văn bản. Mặc dù vẫn còn một số hạn chế, những kết quả đạt được là nền tảng để tiếp tục cải thiện mô hình, mở rộng phạm vi ứng dụng, và khai thác tiềm năng của các mô hình ngôn ngữ hiện đại trong tương lai.

TÀI LIỆU THAM KHẢO

1. **Nguyễn, V. S., & Nguyễn, T. L. (2020).** *PhoBERT: Pre-trained language models for Vietnamese.* Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings.
2. **Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019).** *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers).
3. **Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017).** *Attention is all you need.* Advances in neural information processing systems, 30.
4. **Hochreiter, S., & Schmidhuber, J. (1997).** *Long short-term memory.* Neural computation, 9(8), 1735-1780.
5. **Schuster, M., & Paliwal, K. K. (1997).** *Bidirectional recurrent neural networks.* IEEE Transactions on Signal Processing, 45(11), 2673-2681.
6. **Liu, B. (2012).** *Sentiment analysis and opinion mining.* Morgan & Claypool Publishers.
7. **Goldberg, Y. (2017).** *Neural network methods for natural language processing.* Morgan & Claypool Publishers.