ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

KHOA TOÁN - TIN
Falcuty of Mathematics and Informatics

Báo cáo bài tập lớn

Hệ hỗ trợ quyết định

Chủ đề: Airline Sentiment

Giảng viên hướng dẫn: TS. Lê Hải Hà

Nhóm sinh viên: Doãn Chí Thường 20210831
Trần Sỹ Toàn 20210846

Mã lớp: 150330

Mục lục

1 Lời mở đầu

Hệ hỗ trợ quyết định (Decision support system - DSS) là một học phần quan trọng trong ngành công nghệ thông tin. Trong cuộc sống chúng ta luôn phải đưa ra những quyết định chẳng hạn như sáng nay sẽ đi làm bằng phương tiện nào, ăn gì, cuối tuần này có nên đi chơi hay không, nếu đầu tư vào dự án này thì có khả thi hay không, . . . Đôi khi việc đưa ra quyết định sẽ gặp những khó khăn nhất định và gặp phải rủi ro đối với quyết định đó. Hệ hỗ trợ quyết định ra đời nhằm giúp con người dễ dàng đưa ra các quyết định cũng như có một cái nhìn tổng quan về vấn đề mình gặp phải. DSS là một cách tiếp cận (hoặc phương pháp) hỗ trợ ra quyết định sử dụng hệ thống thông tin dựa trên máy tính (computer-based information system - CBIS) có tính tương tác, mềm dẻo và có khả năng thích nghi. Một hệ thống hỗ trợ quyết định thường được phát triển bởi người dùng cuối để hỗ trợ các phương án cho các vấn đề quản lý phi cấu trúc đặc biệt, nó sử dụng dữ liệu, mô hình và tri thức cùng với giao diện người dùng (thường là đồ họa, trên nền tảng Web) thân thiện. DSS kết hợp với các hiểu biết sâu sắc của người ra quyết định từ đó hỗ trợ tất cả các pha của quá trình ra quyết định. Hệ hỗ trợ quyết định có thể được ứng dụng trong rất nhiều lĩnh vực khác nhau và trong phạm vi của bài báo cáo này em đã lựa chọn xây dựng một hệ thống phân tích cảm xúc (Sentiment analysis) dựa trên việc phân loại các văn bản (Text classification) nhằm mục đích hỗ trợ các nhà làm phim có thể biết được bộ phim của họ có được yêu thích hay không từ những bài đánh giá phim của khán giả.

Nội dung của bài báo cáo bao gồm các mục sau: Mục 1 giới thiệu bài toán phân tích cảm xúc và các ứng dụng. Mục 2 trình bày một số lý thuyết cần thiết trong việc ứng dụng các model vào bài toán phân tích cảm xúc. Mục 3 trình bày về các model được ứng dụng trong việc giải bài toán phân tích cảm xúc. Tiếp đó, chúng em sẽ đưa ra chương trình kiểm thử và đánh giá kết quả trong việc phân tích cảm xúc trong một lĩnh vực cụ thể đó là dịch vụ hàng không (Airline Sentiment). Cuối cùng là kết luận về những điều đã làm được và danh mục tài liệu tham khảo.

Chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất tới **TS. Lê Hải Hà**, giảng viên khoa Toán - Tin, Đại học Bách khoa Hà Nội đã tận tình giảng dạy và đưa ra những góp ý, lời khuyên bổ ích để chúng em có thể hoàn thiện bài báo cáo một cách chỉn chu nhất. Mặc dù vậy bài báo cáo của chúng em có thể vẫn còn những thiếu sót, rất mong sẽ nhận được những góp ý và chỉnh sửa của Thầy để chúng em có thể rút kinh nghiệm cho những môn học sau.

Chúng em xin chân thành cảm ơn!

2 Giới thiệu bài toán phân tích cảm xúc

2.1 Sentiment Analysis là gì

Sentiment Analysis, hay còn gọi là phân tích cảm xúc là việc sử dụng xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP), phân tích văn bản, ngôn ngữ học tính toán và sinh trắc học để xác định, trích xuất, định lượng và nghiên cứu các trạng thái cảm xúc và thông tin chủ quan một cách có hệ thống. Hệ thống phân tích cảm xúc giúp các tổ chức thu thập thông tin chi tiết từ văn bản không có tổ chức và không có cấu trúc đến từ các nguồn trực tuyến như email, bài đăng trên blog, phiếu hỗ trợ, trò chuyện trên web, kênh truyền thông xã hội, diễn đàn và nhận xét. Các thuật toán thay thế việc xử lý dữ liệu thủ công bằng cách thực hiện các phương pháp dựa trên quy tắc, tự động hoặc kết hợp. Các hệ thống phân tích cảm xúc thực hiện phân tích cảm tính các quy tắc dựa trên từ vựng, được xác định trước trong khi các hệ thống tự động học từ dữ liệu bằng các kỹ thuật máy học. Phân tích cảm xúc hỗn hợp kết hợp cả hai cách tiếp cận. Với sự gia tăng của các mô hình ngôn ngữ sâu, chẳng hạn như RoBERTa, các miền dữ liệu khó hơn cũng có thể được phân tích, chẳng hạn như các văn bản tin tức trong đó các tác giả thường bày tỏ quan điểm cảm xúc của họ ít rõ ràng hơn.

2.2 Tại sao phân tích cảm xúc lại quan trọng

Ngày nay, mọi người có rất nhiều cách để thể hiện cảm xúc của mình trực tuyến, vì vậy các tổ chức và doanh nghiệp cần có các công cụ mạnh mẽ để giám sát những gì đang được nói về họ và sản phẩm cũng như dịch vụ của họ gần như theo thời gian thực. Khi các công ty áp dụng phân tích cảm xúc và bắt đầu sử dụng nó để phân tích nhiều cuộc trò chuyện và tương tác hơn, việc xác định các điểm xung đột của khách hàng ở mọi giai đoạn trong hành trình của khách hàng sẽ trở nên dễ dàng hơn.

- **Cung cấp kết quả khách quan hơn từ các đánh giá của khách hàng**

  Các công cụ phân tích cảm xúc bằng trí tuệ nhân tạo (AI) mới nhất giúp các công ty lọc các đánh giá và điểm số NPS (Net Promoter Score) để loại bỏ sự thiên vị cá nhân và nhận được ý kiến khách quan hơn về thương hiệu, sản phẩm và dịch vụ của họ. Ví dụ, nếu một khách hàng thể hiện ý kiến tiêu cực cùng với ý kiến tích cực trong một đánh giá, một người đánh giá có thể dán nhãn là tiêu cực trước

khi đọc đến các từ ngữ tích cực. Phân loại cảm xúc được nâng cao bằng AI giúp sắp xếp và phân loại văn bản một cách khách quan, tránh tình trạng này và phản ánh cả hai loại cảm xúc.

- **Đạt được khả năng mở rộng lớn hơn cho các chương trình thông minh kinh doanh**

  Phân tích cảm xúc cho phép các công ty với lượng dữ liệu không cấu trúc khổng lồ phân tích và trích xuất thông tin có ý nghĩa từ nó một cách nhanh chóng và hiệu quả. Với lượng văn bản được tạo ra bởi khách hàng trên các kênh kỹ thuật số, dễ dàng khiến các đội ngũ nhân viên bị quá tải thông tin. Các công cụ phân tích cảm xúc khách hàng mạnh mẽ, dựa trên đám mây và được nâng cao bằng AI giúp các tổ chức cung cấp thông tin thông minh kinh doanh từ dữ liệu khách hàng của họ ở quy mô lớn mà không tiêu tốn nguồn lực không cần thiết.

- **Thực hiện giám sát danh tiếng thương hiệu theo thời gian thực**

  Các doanh nghiệp hiện đại cần phản ứng nhanh chóng trong các tình huống khủng hoảng. Những ý kiến được thể hiện trên mạng xã hội, dù đúng hay không, có thể phá hủy một danh tiếng thương hiệu được xây dựng qua nhiều năm. Các công cụ phân tích cảm xúc mạnh mẽ, được nâng cao bằng AI giúp các giám đốc điều hành giám sát cảm xúc tổng thể xung quanh thương hiệu của họ để có thể phát hiện các vấn đề tiềm ẩn và giải quyết chúng nhanh chóng.

2.3 Phân loại bài toán phân tích cảm xúc

Ngoài các phương pháp khác nhau được sử dụng để xây dựng các công cụ phân tích cảm xúc, còn có các loại phân tích cảm xúc khác nhau mà các tổ chức sử dụng tùy thuộc vào nhu cầu của họ. Ba loại phổ biến nhất, phân tích cảm xúc dựa trên cảm xúc, phân tích cảm xúc chi tiết và phân tích cảm xúc theo khía cạnh (ABSA), đều dựa vào khả năng của phần mềm để đánh giá cái gọi là cực tính, tức là cảm giác tổng thể được truyền tải bởi một đoạn văn bản.

Nói chung, tính chất của một đoạn văn bản có thể được mô tả là tích cực, tiêu cực hoặc trung lập, nhưng bằng cách phân loại văn bản chi tiết hơn nữa, ví dụ như thành các nhóm con như "rất tích cực"hoặc "rất tiêu cực,"một số mô hình phân tích cảm xúc có thể nhận diện được các cảm xúc tinh tế và phức tạp hơn. Cực tính của một đoạn văn bản là thước đo phổ biến nhất để đánh giá cảm xúc văn bản và được phần mềm biểu thị dưới

dạng xếp hạng số trên thang điểm từ một đến 100. Số không đại diện cho cảm xúc trung lập và 100 đại diện cho cảm xúc cực đoan nhất.

Dưới đây là ba loại phân tích cảm xúc được sử dụng rộng rãi nhất:

1. **Phân tích cảm xúc chi tiết (graded)**

   Phân tích cảm xúc chi tiết, hay phân loại theo cấp độ, là một loại phân tích cảm xúc nhóm các văn bản thành các cảm xúc khác nhau và mức độ cảm xúc được biểu hiện. Cảm xúc sau đó được xếp hạng trên thang điểm từ zero đến 100, tương tự như cách các trang web tiêu dùng triển khai xếp hạng sao để đo lường sự hài lòng của khách hàng.

2. **Phân tích cảm xúc theo khía cạnh (ABSA)**

   Phân tích cảm xúc theo khía cạnh (ABSA) thu hẹp phạm vi những gì được xem xét trong một văn bản thành một khía cạnh duy nhất của sản phẩm, dịch vụ hoặc trải nghiệm khách hàng mà doanh nghiệp muốn phân tích. Ví dụ, một ứng dụng du lịch giá rẻ có thể sử dụng ABSA để hiểu giao diện người dùng mới có dễ sử dụng hay không hoặc để đánh giá hiệu quả của chatbot dịch vụ khách hàng. ABSA có thể giúp các tổ chức hiểu rõ hơn sản phẩm của họ đang thành công hay thiếu sót gì so với mong đợi của khách hàng.

3. **Phát hiện cảm xúc**

   Phân tích cảm xúc nhằm phát hiện cảm xúc tìm cách hiểu trạng thái tâm lý của cá nhân đứng sau một văn bản, bao gồm cả tâm trạng của họ khi viết và ý định của họ. Nó phức tạp hơn cả phân tích chi tiết và ABSA và thường được sử dụng để có cái nhìn sâu sắc hơn về động lực hoặc trạng thái cảm xúc của một người. Thay vì sử dụng cực tính như tích cực, tiêu cực hoặc trung lập, phân tích phát hiện cảm xúc có thể xác định các cảm xúc cụ thể trong văn bản như thất vọng, thờ ơ, bồn chồn và sốc.

2.4 Phân tích cảm xúc hoạt động như thế nào

Phân tích cảm xúc là một ứng dụng thuộc công nghệ xử lý ngôn ngữ tự nhiên (NLP) có khả năng đào tạo để giúp phần mềm máy tính hiểu văn bản theo các cách tương tự như con người. Quá trình phân tích thường trải qua môt số giai đoạn trước khi đưa ra kết quả cuối cùng.

1. **Tiền xử lý**

   Trong giai đoạn tiền xử lý, phân tích cảm xúc xác định các từ khóa để nêu bật thông điệp chủ đạo trong văn bản.

- Quá trình token hóa sẽ phân tách một câu thành nhiều yếu tố hoặc token.
- Việc phục hồi nguyên thể từ sẽ chuyển đổi các từ về dạng gốc của chúng. Ví dụ: dạng gốc của "am" là "be" trong tiếng Anh.
- Hoạt động loại bỏ từ dừng sẽ lọc ra các từ không mang đến giá trị ý nghĩa cho câu.

2. **Phân tích từ khóa**

   Các công nghệ NLP phân tích sâu hơn các từ khóa được trích xuất và cho chúng một số điểm cảm xúc. Điểm cảm xúc là một thang đo cho biết yếu tố cảm xúc trong hệ thống phân tích cảm xúc. Thang đo này cung cấp nhận thức tương đối về cảm xúc được thể hiện trong văn bản nhằm phục vụ mục đích phân tích. Ví dụ: khi phân tích đánh giá của khách hàng, các nhà nghiên cứu sử dụng số 10 để đại diện cho sự hài lòng và số 0 để đại diện cho sự thất vọng.

2.5 Một số ứng dụng của phân tích cảm xúc

Công nghệ phân tích cảm xúc được sử dụng trong nhiều lĩnh vực khác nhau. Bao gồm:

- **Marketing:** Doanh nghiệp sử dụng công nghệ phân tích cảm xúc để phân tích phản hồi và review của khách hàng. Từ đó, có được sự hiểu biết sâu sắc về những gì khách hàng thực sự nghĩ. Điều này có ích với công việc tìm kiếm insight khách hàng.
- **Dịch vụ khách hàng:** Phân tích cảm xúc phát hiện nhanh chóng những cuộc gọi tiêu cực theo thời gian thực. Điều này hỗ trợ đội ngũ chăm sóc khách hàng phản ứng kịp thời và giải quyết vấn đề.
- **Phân tích thị trường:** Phân tích những nguồn thông tin trên mạng xã hội để nắm bắt phản ứng của thị trường mục tiêu với một chủ đề nào đó.

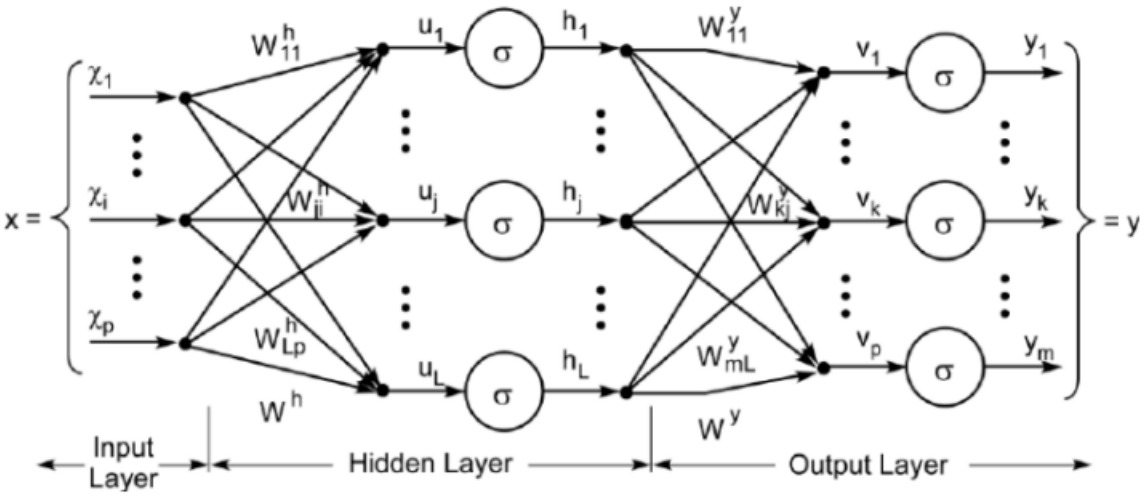
3 Cơ sở lý thuyết

Trong chương này, chúng em sẽ phân tích lý thuyết của 2 mô hình mạng Neural hồi quy (Recurrent Neural Network) mà cụ thể là mạng LSTM và mô hình BERT (Bidirectional Encoder Representations from Transformers), đây là 2 mô hình thường được sử dụng để xử lý ngôn tự nhiên. Và cũng là 2 mô hình mà chúng em hướng đến để tiếp cận bài toán của chủ đề này.

3.1 Mô hình mạng Neural

3.1.1 Tổng quan về mạng Neural

Mô hình Neural Network được xây dựng dựa trên mô hình tổng quát Multilayer Perceptrion- MLP. Một MLP là một ánh xạ $y = f * (x, \theta)$ với input là dữ liệu và output là giá trị dự đoán. Hàm f sẽ được xây dựng dựa trên các hàm tuyến tính, logistic, tanh,... Bằng việc lựa chọn các hàm thích hợp và dữ liệu đầu vào mà mô hình MLP có thể xấp xỉ tham số θ và từ đó giúp máy tính xác định được đầu ra.



Hình 1: Kiến trúc cơ bản của MLP

Kiến trúc cơ bản của MLP gồm 3 thành phần sau:

1. **Input Layer:** Dữ liệu đầu vào.
2. **Hidden Layer:** Lớp ẩn tính toán trung gian và lưu trữ thông tin tại các Neural với các *activation function* khác nhau.
3. **Output Layer:** Lớp đầu ra giá trị dự đoán.

Mỗi layer sẽ là một tầng của mạng, trong mỗi layer đều có thể có nhiều Neural. Các Neural được liên kết với nhau bằng kết nối có trọng số $\mathbf{w}$. Nếu mỗi Neural tại lớp thứ $k+1$ được liên kết với toàn bộ các Neural tại lớp thứ k ta gọi đó là *Fully – Connected* và ngược lại ta gọi đó là *Pooling – Connected*. Giá trị của Neural sẽ được tính bằng tổng tuyến tính của tích các trọng số với layer đầu vào cộng với bias hiệu chỉnh sai số, sau đó đưa kết thu được qua *activation function* thu được giá trị đầu ra tại mỗi Neural.

3.1.2 Recurrent Neural Networks

Trong các mô hình mạng Neural khác, việc tính toán đầu ra của một đầu vào hoàn toàn độc lập với đầu ra trước đó. Tuy nhiên trong thực tế, nhiều bài toán cần lấy kết quả đầu ra trước đó để đưa ra kết quả tiếp theo. Ví dụ như để tạo ra một câu văn, mô hình cần biết câu văn trước đó là gì trước khi sinh ra một câu văn khác đúng ngữ pháp và có ý nghĩa. Để giải quyết vấn đề này, mạng Neutral hồi quy (Recurrent Neural Networks - RNN) được tạo ra.

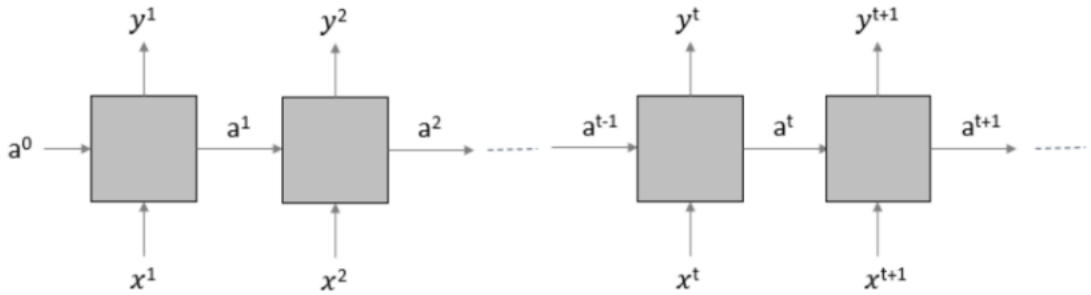
Mạng RNN được sử dụng rộng rãi trong nhiều ứng dụng xử lý dữ liệu chuỗi. Sau đây là một số ứng dụng phổ biến của mạng RNN:

- **Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP)**: Mạng RNN được sử dụng rộng rãi trong các bài toán NLP như dịch máy, phân loại văn bản, phát hiện cảm xúc trong văn bản, sinh văn bản tự động, ... Nhờ khả năng lưu trữ thông tin từ các input trước đó để sử dụng cho việc dự đoán output tiếp theo, mạng RNN có thể học được các mối quan hệ phức tạp giữa các từ trong câu hoặc các câu trong đoạn văn.
- **Nhận dạng giọng nói**: Mạng RNN cũng được sử dụng để nhận dạng giọng nói, nơi các đầu vào là các tín hiệu âm thanh. Mạng RNN có thể học được các đặc trưng của giọng nói và giúp xác định người nói là ai.
- **Phân tích chuỗi thời gian (Time series analysis)**: Mạng RNN có thể được sử dụng để phân tích chuỗi thời gian như dữ liệu chứng khoán, dữ liệu thời tiết, ... Mạng RNN có thể học được các mẫu trong dữ liệu chuỗi thời gian và dự đoán các giá trị trong tương lai.
- **Nhận dạng hành vi người dùng (User behavior recognition)**: Mạng RNN cũng có thể được sử dụng để nhận dạng hành vi người dùng trên các nền tảng trực tuyến. Mạng RNN có thể học được các mẫu trong hành vi người dùng và giúp xác định những hành vi bất thường hoặc gian lận.

- **Xử lý dữ liệu thị giác (Computer Vision)**: Mạng RNN cũng được sử dụng trong xử lý dữ liệu thị giác để giải quyết các bài toán như nhận dạng hình ảnh, phân loại đối tượng và dự báo chuỗi thời gian trong video.

Trên đây là một số ứng dụng phổ biến của mạng RNN. Mạng RNN được sử dụng rộng rãi trong nhiều lĩnh vực và ngày càng trở nên quan trọng trong xử lý dữ liệu chuỗi.

Kiến trúc mạng RNN



Hình 2: Kiến trúc cơ bản của RNN

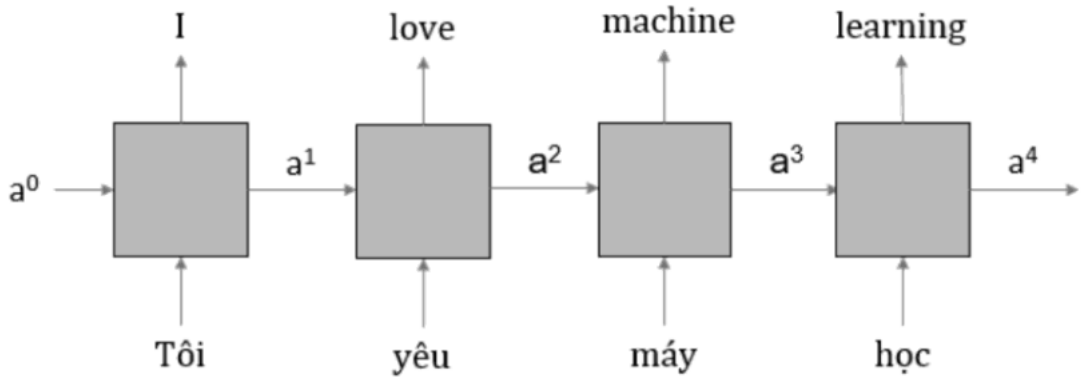
Mạng RNN dùng để xử lý dữ liệu chuỗi: chuỗi các từ, chuỗi các ký tự, chuỗi các âm thanh, chuỗi các hình ảnh,... Và dữ liệu chuỗi được xử lý tuần tự từng bước 1, mỗi bước xử lý từng phần tử (từng từ, từng âm thanh, ...). Mỗi bước như vậy được gọi là một bước thời gian (timestep).

- $x^1, x^2, \ldots$ là chuỗi dữ liệu đầu vào. Ví dụ, để dịch câu "Tôi yêu máy học" từ tiếng Việt sang tiếng Anh, dữ liệu x^1 là "Tôi", x^2 là "yêu", ...

- $y^1, y^2, \ldots$ là chuỗi đầu ra. Với ví dụ về việc dịch ở trên, thì đầu ra sẽ là chuỗi "I love machine learning". Và y^1 là "I", y^2 là "love"...

- $a^1, a^2, \ldots$ là chuỗi các trạng thái ẩn. Trạng thái ẩn đầu ra của một lớp sẽ là trạng thái ẩn đầu vào của lớp tiếp theo. Đây chính là việc sử dụng đầu ra ở bước trước trong việc xử lý bước tiếp theo. Tên gọi "trạng thái ẩn"có thể gây bối rối. Thực tế, nó cũng chỉ là kết quả của một hàm kích hoạt như sigmoid hay relu. Trạng thái ẩn đầu tiên, a^0, được khởi tạo là 0. Các a này chính là sự hồi quy, là bộ nhớ; mạng RNN nhớ một thông tin nào đó ở bước trước để xử lý bước tiếp theo.

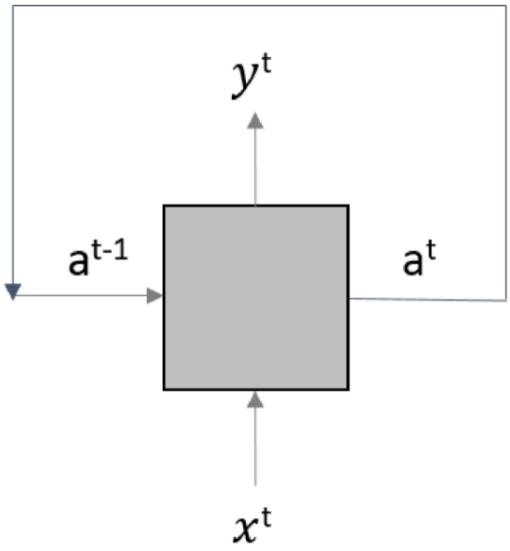
Sau đây là một hình minh họa về kiến trúc của mạng RNN khi dịch một câu từ tiếng Việt sang tiếng Anh:



Hình 3: Minh họa về kiến trúc RNN

Từ "yêu"trong tiếng Việt không phải lúc nào dịch sang tiếng Anh cũng là "love", mà nó phụ thuộc vào từ đứng trước nữa. Nếu từ đứng trước là "họ"chẳng hạn, thì từ "yêu"lúc này sẽ được dịch thành "loves". Đó là lý do tại sao chúng ta cần thêm một đầu ra cho trạng thái ẩn.

Trên đây là kiến trúc RNN đã được trải ra theo các bước thời gian cho chúng ta dễ hình dung bước đầu. Thực tế thì RNN chỉ gồm một nút trong mỗi thời gian, và được xây lặp đi lặp lại cho từng phần từ của một chuỗi, và cũng lặp lại cho các phần tử của các chuỗi khác. Kiến trúc của RNN chính xác như sau:



Hình 4: Kiến trúc RNN

Ban đầu, a^0 được khởi tạo bằng 0. Tại một bước t nào đó, với đầu vào x^t và trạng thái ẩn a^{t-1} đã được tính toán từ bước $t-1$ và lưu trữ sẵn bên trong RNN, việc tính toán trạng thái ẩn a^t như sau:

$$a^t = \sigma_1(W_1 a^{t-1} + W_2 x^t + b_1)$$

Trong đó, W_1 và W_2 là 2 ma trận trọng số, b_1 là một vector bias. σ_1 là một hàm kích hoạt nào đó, có thể là sigmoid, relu, tanh...

Và đầu ra y^t được tính như sau:

$$y^t = \sigma_2(W_3 a^t + b_2)$$

Trong đó, W_3 là một ma trận trọng số, b_2 là một vector bias và σ_2 thường được sử dụng là *Softmax*. Ta có thể biểu diễn đầu ra đó một cách tường minh như sau:

$$y^t = softmax(W_3 a^t + b_2)$$

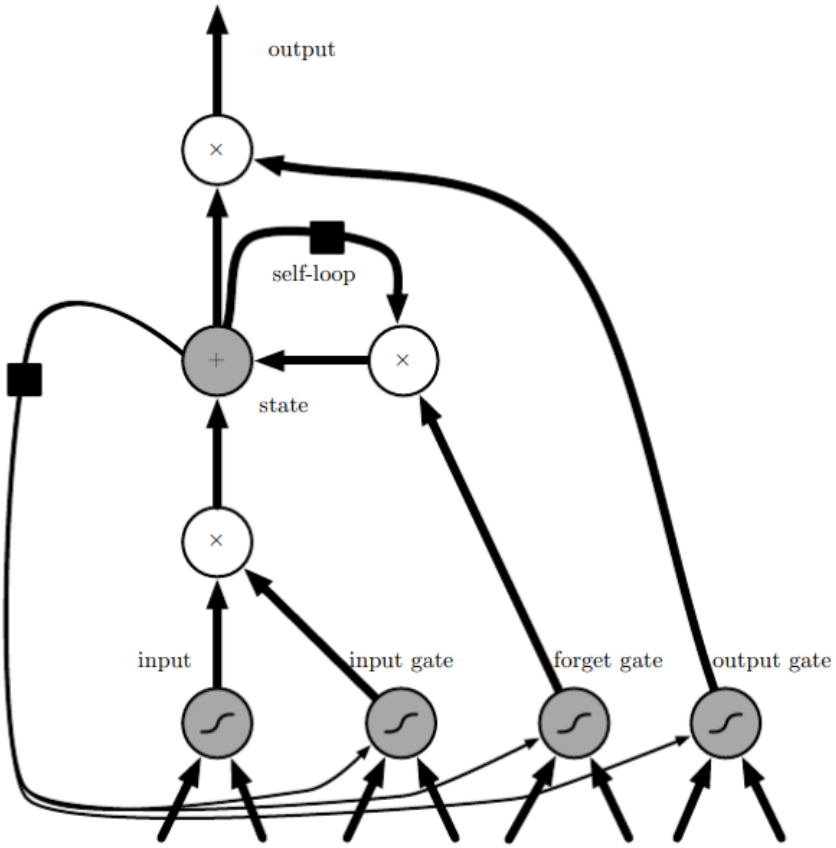
Như vậy, không giống như một lớp neuron bình thường có 1 ma trận trọng số và 1 vector bias, một lớp RNN có đến 3 ma trận trọng số và 2 vector bias. Giả sử, chọn hàm mất mát $L^{(t)}$ là *negative log-likelihood* của $y^{(t)}$ cho chuỗi $x^{(1)}, ..., x^{(t)}$.

$$\begin{aligned} L\big(\{x^{(1)}, ..., x^{(t)}\}, \{y^{(1)}, ..., y^{(t)}\}\big) &= \sum_t L^{(t)} \\ &= -\sum_t logp_{model}\big(y^{(t)} | \{x^{(1)}, ..., x^{(t)}\}\big) \end{aligned}$$

Với p_{model} được tính qua mô hình khi so sánh giá trị mục tiêu $y^{(t)}$ với giá trị dự đoán $\hat{y}^{(t)}$.

Mạng LSTM (Long-short term memory)

Về mặt lý thuyết thì mạng RNN có thể đưa thông tin từ *layer* trước đến các *layer* phía sau. Nhưng thực tế là thông tin chỉ mang được qua một số lượng state nhất định, điều đó dẫn đến hiện tượng *vanishing gradient*, hay nói cách khác mạng RNN sẽ chỉ đọc các *state* gần đó. Đây là hiện tượng *short term memory*. Với bài toán dịch văn bản trong xử lý ngôn ngữ tự nhiên thì input của mạng là những câu văn dài và ta cần thông tin của những câu văn trước đó để *output* đầu ra dịch đúng ngữ cảnh nhất. Chính vì vậy, để khắc phục hiện tượng *vanishing gradient* một mạng RNN cải tiến ra đời chính là mạng *Long-short term memory LSTM*.



Hình 5: Kiến trúc mạng LSTM

Điểm đặc biệt trong cấu trúc tế bào của mạng LSTM là nó có cơ chế *self-loop* để làm giảm hiện tượng *vanishing gradient*, tế bào LSTM sẽ quyết định tỷ lệ lưu giữ thông tin của các state trước đó, các tham số của *self-loop* được điều khiển thông qua cổng *(forget gate)* $f_i^{(t)}$ (ứng với bước lặp thứ t và tế bào thứ i), cổng này sẽ điều chỉnh tỷ lệ quên thông tin trước trong khoảng từ 0 tới 1 thông qua hàm *Sigmoid*:

$$f_i^{(t)} = \sigma\Big(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\Big)$$

trong đó, $x^{(t)}$ là vector input hiện tại và $h^{(t)}$ là vector *hidden layer* hiện tại chứa output của toàn bộ tế bào LSTM trước đó. b^f, U^f và W^f lần lượt là tham số *bias* của mạng, trọng số của dữ liệu đầu vào và trọng số cho tầng cổng quên *(forget gates)*. Thành phần quan trọng nhất của *State unit* $s_i^{(t)}$ lưu chữ thông tin của mạng. Khi đó trình cập nhật giá trị của mạng được diễn ra như sau:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\Big(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\Big)$$

với b, U, W lần lượt là hệ số của *bias*, tham số dữ liệu input và tham số *reccurent* của tế bào LSTM. Ta thấy giá trị của $s_i^{(t)}$ được cập nhật cùng với

giá trị của tầng cổng quên $f_i^{(t)}$. Giá trị *external input gate* $g_i^{(t)}$ được tính toán tương tự như tầng cổng quên với các tham số riêng để tính toán tỷ lệ ảnh hưởng của dữ liệu đầu vào ở bước thứ t là $x_j^{(t)}$ và thông tin từ *state* trước đó là $h_j^{(t-1)}$:

$$g_i^{(t)} = \sigma\Big(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\Big)$$

Giá trị output $h_i^{(t)}$ của tế bào LTSM cũng có thể bị loại bỏ thông qua cổng *output gate* $q_i^{(t)}$ cũng được tính thông qua hàm sigmoid:

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$
$$q_i^{(t)} = \sigma\Big(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\Big)$$

có chứa các tham số b^o, U^o, W^o lần lượt là *bias*, trọng số input và trọng số của tầng cổng quên đối với kết quả output dự đoán của các *state* trước đó.

3.2 Bidirectional Encoder Representations from Transformers

BERT *(Bidirectional Encoder Representations from Transformers)* là một mô hình ngôn ngữ *(Language Model)* được tạo ra bởi Google AI. BERT được coi như là đột phá lớn trong Machine Learning bởi vì khả năng ứng dụng của nó vào nhiều bài toán NLP khác nhau: Question Answering, Natural Language Inference,... với kết quả rất tốt.

Một trong những thách thức lớn nhất của NLP là vấn đề dữ liệu. Trên internet có rất nhiều dữ liệu, nhưng những dữ liệu đó không đồng nhất; mỗi phần của nó chỉ được dùng cho một mục đích riêng biệt, do đó khi giải quyết một bài toán cụ thể, ta cần trích ra một bộ dữ liệu thích hợp cho bài toán của mình, và kết quả là ta chỉ có một lượng rất ít dữ liệu. Nhưng có một nghịch lý là, các mô hình *Deep Learning* cần lượng dữ liệu rất lớn - lên tới hàng triệu, để có thể cho ra kết quả tốt. Do đó một vấn đề được đặt ra: làm thể nào để tận dụng được nguồn dữ liệu vô cùng lớn có sẵn để giải quyết bài toán của mình. Đó là tiền đề cho một kỹ thuật mới ra đời: *Transfer Learning*. Với *Transfer Learning* các mô hình *(model)* "chung"nhất với tập dữ liệu khổng lồ trên *internet (pre-training)* được xây dựng và có thể được "tinh chỉnh"*(fine-tune)* cho các bài toán cụ thể. Nhờ có kỹ thuật này mà kết quả cho các bài toán được cải thiện rõ rệt, không chỉ trong NLP mà còn trong các lĩnh vực khác như Computer Vision,... BERT là

một trong những đại diện ưu tú nhất trong *Transfer Learning* cho NLP, nó gây tiếng vang lớn không chỉ bởi kết quả mang lại trong nhiều bài toán khác nhau, mà còn bởi vì nó hoàn toàn miễn phí, tất cả chúng ta đều có thể sử dụng BERT cho bài toán của mình.

Nền tảng của BERT

BERT sử dụng *Transformer* là một mô hình attention *(attention mechanism)* học mối tương quan giữa các từ (hoặc 1 phần của từ) trong một văn bản. *Transformer* gồm có 2 phần chính: *Encoder* và *Decoder, encoder* thực hiện đọc dữ liệu đầu vào và decoder đưa ra dự đoán. Ở đây, BERT chỉ sử dụng *Encoder*.

Khác với các mô hình directional (các mô hình chỉ đọc dữ liệu theo 1 chiều duy nhất từ *trái* $\rightarrow$ *phải, phải* $\rightarrow$ *trái*) đọc dữ liệu theo dạng tuần tự, Encoder đọc toàn bộ dữ liệu trong 1 lần, việc này làm cho BERT có khả năng huấn luyện dữ liệu theo cả hai chiều, qua đó mô hình có thể học được ngữ cảnh *(context)* của từ tốt hơn bằng cách sử dụng những từ xung quanh nó.

3.2.1 Attention Mechanism

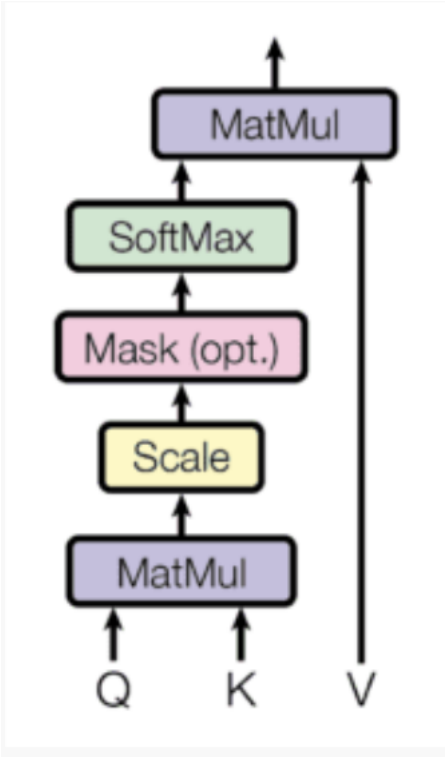
Attention là cơ chế liên kết các vị trí khác nhau của một chuỗi hoặc một câu để có được sự thể hiện chính xác hơn. Trình đọc máy là một thuật toán có thể tự động hiểu văn bản được cung cấp cho nó. Khi chúng ta đọc hoặc xử lý câu từng chữ, trong đó các từ nhìn thấy trước đó cũng được nhấn mạnh, được suy ra từ các sắc thái và đây chính xác là điều mà tính năng *attention* của máy đọc thực hiện. Để tính mức độ chú ý cho một từ trong câu, cơ chế tính điểm là sử dụng tích chấm hoặc một số chức năng khác của từ với biểu diễn trạng thái ẩn của các từ đã thấy trước đó.

Các thành phần chính được sử dụng bởi *Transformer-Attention* như sau:

- q và k biểu thị vecto kích thước, d_k chứa các truy vấn và khóa tương ứng.
- v biểu thị một vecto thứ nguyên, d_v chứa các giá trị.
- Q, K và V biểu thị các ma trận đóng gói các bộ truy vấn, khóa và giá trị tương ứng.
- W^Q, W^K, W^V biểu thị các ma trận chiếu được sử dụng để tạo ra các biểu diễn không gian con khác nhau của ma trận truy vấn, khóa và giá trị.

- W^O biểu thị ma trận chiếu cho đầu ra nhiều đầu.

Về bản chất, hàm *attention* có thể được coi là ánh xạ giữa một truy vấn và một tập hợp các cặp *key-value* tới đầu ra. Hàm *attention* sẽ thực hiện chấm điểm để chia tỷ lệ dựa trên tích số chấm cho mỗi truy vấn q với tất cả các phím k, sau đó chia mỗi kết quả cho $\sqrt{d_K}$ và tiến hành áp dụng hàm *Softmax*. Khi làm như vậy, nó thu được các trọng số được sử dụng để chia tỷ lệ các giá trị v.



Hình 6: Tính toán cơ chế chú ý theo tỷ lệ

Trong thực tế, các tính toán được thực hiện bởi *cơ chế chú ý* số chấm được chia tỷ lệ có thể được áp dụng một cách hiệu quả cho toàn bộ bộ truy vấn cùng một lúc. Để làm được điều đó, các ma trận Q, K, V được coi là input cho hàm *attention*:

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

Quy trình để tính toán hàm *attention* theo tỷ lệ chấm như sau:

1. Tính điểm căn chỉnh bằng cách nhân tập hợp các truy vấn được đóng gói trong ma trận Q. Với các khóa trong ma trận K. Trong đó, Q là

ma trận có kích thước $m \times d_k$, K có kích thước $n \times d_k$ thì ma trận kết quả có kích thước $m \times n$:

$$QK^T = \begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}$$

2. Chia tỷ lệ cho từng điểm căn chỉnh:

$$\frac{QK^T}{\sqrt{d_k}} = \begin{bmatrix} \frac{e_{11}}{\sqrt{d_k}} & \frac{e_{12}}{\sqrt{d_k}} & \dots & \frac{e_{1n}}{\sqrt{d_k}} \\ \frac{e_{21}}{\sqrt{d_k}} & \frac{e_{22}}{\sqrt{d_k}} & \dots & \frac{e_{2n}}{\sqrt{d_k}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{e_{m1}}{\sqrt{d_k}} & \frac{e_{m2}}{\sqrt{d_k}} & \dots & \frac{e_{mn}}{\sqrt{d_k}} \end{bmatrix}$$

3. Và làm theo quy trình chia tỷ lệ bằng cách áp dụng thao tác softmax để có được một tập hợp trọng số:

$$Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) = Softmax\left(\begin{bmatrix} \frac{e_{11}}{\sqrt{d_k}} & \frac{e_{12}}{\sqrt{d_k}} & \dots & \frac{e_{1n}}{\sqrt{d_k}} \\ \frac{e_{21}}{\sqrt{d_k}} & \frac{e_{22}}{\sqrt{d_k}} & \dots & \frac{e_{2n}}{\sqrt{d_k}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{e_{m1}}{\sqrt{d_k}} & \frac{e_{m2}}{\sqrt{d_k}} & \dots & \frac{e_{mn}}{\sqrt{d_k}} \end{bmatrix}\right)$$

4. Cuối cùng, áp dụng các trọng số thu được cho các giá trị trong ma trận V:

$$Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)\begin{bmatrix} v_{11} & v_{12} & \dots & v_{1n} \\ v_{21} & v_{22} & \dots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \dots & v_{mn} \end{bmatrix}$$

Multi-Head Attention

Ý tưởng đằng sau cơ chế chú ý nhiều đầu là cho phép hàm chú ý trích xuất thông tin từ các không gian con biểu diễn khác nhau, điều này là không

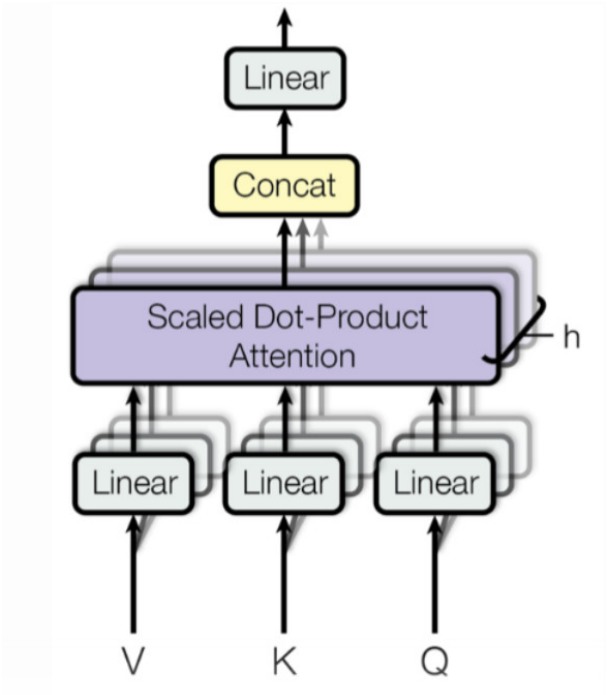
thể với một đầu chú ý duy nhất.

Hàm *Multi-Head Attention* có thể được biểu diễn như sau:

$$multihead(Q, K, V) = concat(head_1, head_2, ..., head_h)W^O \quad (1)$$

Ở đây mỗi hàm $head_i$ thực hiện một chức năng chú ý duy nhất được đặc trưng bởi ma trận chiếu đã học của chính nó:

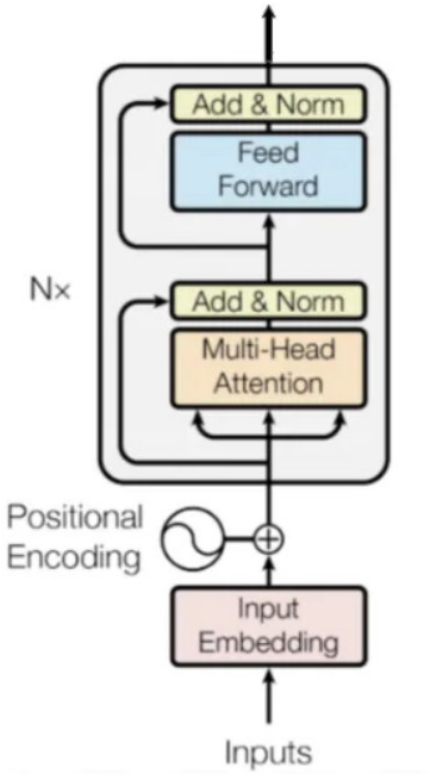
$$head_i = attention(QW_i^Q, KW_i^K, VW_i^K) \quad (2)$$



Hình 7: Cơ chế chú ý nhiều đầu vào

3.2.2 The Encoder

Bộ mã hóa *Encoder* sử dụng trong mô hình BERT là kiến trúc dựa trên sự chú ý *(attention)* cho việc xử lý ngôn ngữ tự nhiên (NLP). Như ở trên đã trình bày thì *Tranformer* xử dụng 2 bộ là bộ mã hóa và bộ giải mã, nhưng đối với BERT chỉ sử dụng bộ mã hóa. Chúng ta có thể quan sát kiến trúc tổng quát của bộ mã hóa *(Encoder)* như sau:



Hình 8: Bộ mã hóa

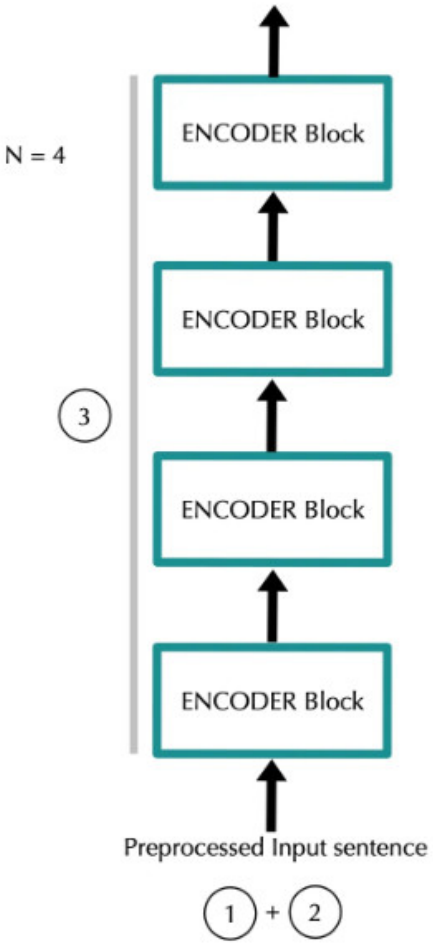
Luồng thông tin qua kiến trúc được thể hiện như sau:

1. Mô hình biểu thị mỗi mã thông báo dưới dạng vectơ có kích thước *emb_dim*. Với một vectơ nhúng cho mỗi mã thông báo đầu vào, chúng ta có ma trận kích thước *(input_length)* × *(emb_dim)* cho một chuỗi đầu vào cụ thể.

2. Sau đó nó thêm thông tin vị trí (mã hóa vị trí). Bước này trả về một ma trận có kích thước *(input_length)* × *(emb_dim)* , giống như bước trước.

3. Dữ liệu đi qua N khối mã hóa. Sau đó, chúng ta thu được ma trận có kích thước *(input_length)* × *(emb_dim)*.

Trong đó: *emb_dim, input_length* lần lượt là kích thước của phần nhúng mã thông báo và độ dài của chuỗi đầu vào.



Hình 9: Luồng thông tin trong bộ mã hóa

Kích thước đầu vào và đầu ra của khối mã hóa là như nhau. Do đó, sẽ hợp lý khi sử dụng đầu ra của một khối bộ mã hóa làm đầu vào của khối bộ mã hóa tiếp theo.

4 Ứng dụng các model trong việc giải bài toán Sentiment Analysis

4.1 Ứng dụng mạng LSTM giải bài toán Airline Sentiment

Ở đây, nhóm chúng em sẽ trình bày về phương pháp nghiên cứu áp dụng mô hình RNN/LSTM cho bài toán Sentiment Analysis.

Trước hết cần quan sát một cách trực quan về bộ dữ liệu của chúng ta để có thể đưa ra một phương án tiếp cận dữ liệu chính xác. Bộ dữ liệu *"Tweet"* từ *Kaggle* với 14640 hàng và 15 cột (tất cả các trường dữ liệu đều được định dạng văn bản). Mặc dù có rất nhiều các trường dữ liệu khác nhau như tên khách hàng, thời gian đánh giá, hay cả múi giờ nhưng ở đây chúng ta có thể bài toán này đang cần tập chung vào 2 mục chính. Đó là phản hồi (hay nhận xét) của khách hàng đối với dịch vụ hàng không và cả xúc của phản hồi đó là tích cực, tiêu cực hay trung tính. Đó chính là vấn đề mà chúng ta cần hướng đến. Điều đó có ý nghĩa rằng, chúng ra sẽ phân tích lời nhận xét khách hàng để tìm hiểu xem người ta đánh giá thế nào về trải nghiệm dịch vụ hàng không này.

Chúng ta cùng nhắc lại đôi nét về cơ sở lý thuyết của chúng ở phần trên. RNN là mạng neural được thiết kế dành cho việc xử lý chuỗi thời gian hay chuỗi văn bản. Đây là loại mạng đặc biệt khi có thể ghi nhớ được dữ liệu trước đó. Nhưng có một vấn đề rất nguy hiểm đó chính là RNN thường gặp phải *varnishing gradient*, chính vì thế mà LSTM ra đời và khắc phục được hiện tượng *varnishing gradient* một cách rất hiệu quả. Trước khi áp dụng mô hình chúng ta cần thực hiện tiền xử lý cho dữ liệu đầu vào.

4.1.1 Tiền xử lý văn bản

Trong quá trình xử lý văn bản, chúng ta cần thực hiện loại bỏ dấu câu, sau đó chuẩn hóa các chữ thành chữ thường cũng như loại bỏ các *Stop words*. *Stop Words* là những từ không có ích cho quyết định đánh giá là tích cực hay tiêu cực. Ví dụ các động từ tobe trong tiếng Anh như *am, is, are*, ...Để giảm độ phức tạp của mô hình chúng ta thực hiện loại bỏ tất cả các *stop words*. Toàn bộ những tác vụ này có thể được thực hiện thông qua framework của Python. Sau đó, chúng ta áp dụng kỹ thuật *word-stemming technique* để chuẩn hóa từ ngữ. Ngoài ra, chúng ta cũng phải thực hiện loại bỏ mật khẩu hay các ký tự đặc biệt mà không có lợi ích cho việc phân tích.

Trích xuất ánh xạ từ ngữ sang số nguyên

Sau khi xóa tất cả văn bản và ký hiệu không cần thiết khỏi tập dữ liệu, chúng em đã trích xuất các phép biến đổi từ ngữ thành số nguyên của tất cả các từ trong tập dữ liệu. Tần số của tất cả các từ trong tập dữ liệu đều sẽ được trích xuất.

Ánh xạ từ này sang từ khác

Sau đó, thực hiện ánh xạ từ ngữ trong câu đánh giá với từ tương ứng chỉ cảm xúc trong tập dữ liệu. Trong bước này, chúng ta sẽ chuyển đổi đánh giá thành một danh sách các danh sách. Trong đó, mỗi danh sách tạo thành ID của các từ có trong câu.

4.1.2 Phần đệm (Padding)

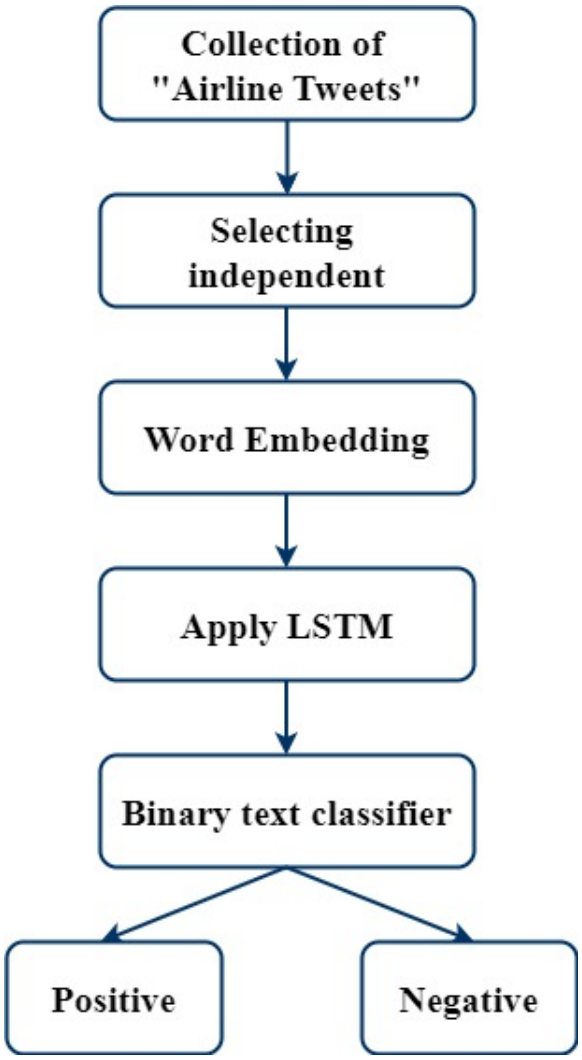
Vì độ dài của tất cả các câu trong tập dữ liệu không giống nhau. Nhưng mạng RNN lại dự kiến số lượng thời gian các bước để mỗi đầu vào đều giống nhau. Để đạt được yêu cầu này, Chúng ta phải chuẩn hóa tất cả các câu để có cùng độ dài. Đối với điều này, trước tiên phải tìm độ dài của câu với chiều dài tối đa. Sau đó, áp dụng phần đệm (pad bằng 0) cho những câu có độ dài nhỏ hơn mức tối đa từ. Sau bước này, tất cả độ dài của câu sẽ trở thành giống nhau, có nghĩa là tất cả các đầu vào có độ dài tương tự nhau.

4.1.3 Thiết lập tập huấn luyện và kiểm tra

Trong máy học, chúng ta sẽ chia bộ dữ liệu làm 2 phần. Một tập train và một tập test, trên thực tế khi chia như vậy chúng ta có thể hiểu một cách đơn giản đó là sẽ dạy cho mô hình theo tập train và dùng tập test (tập mà có dữ liệu chính xác trong tương lai) để kiểm tra tính đúng đắn của mô hình. Để có thể đào tạo mô hình một cách hiệu quả người ta thường chi 2 tập train và test với tỷ lệ 70% và 30%.

Nhãn mã hóa (Label Encoding)

Như đã phân tích, chúng ta có biến phụ thuộc trong tập dữ liệu là *airline-setiment*, đây là loại biến phân loại khi có 2 loại cảm xúc chủ đạo là tích cực *(positive)* và tiêu cực *(negative)*. Bởi vì, các mô hình học máy thực hiện xử lý dữ liệu số. Vì vậy mà chúng ta sẽ thực hiện thay đổi nhãn thành 1 và 0, trong đó 1 tương ứng với *"positive"* và 0 tương ứng với *"negative"*. Việc này sẽ được thực hiện thông qua Kera - một framework thông dụng của Python. Chúng ta cũng có thể thực hiện mã hóa nhãn với lớp bộ mã hóa nhãn trong gói sklearn của Python.



Hình 10: Mô hình đề xuất

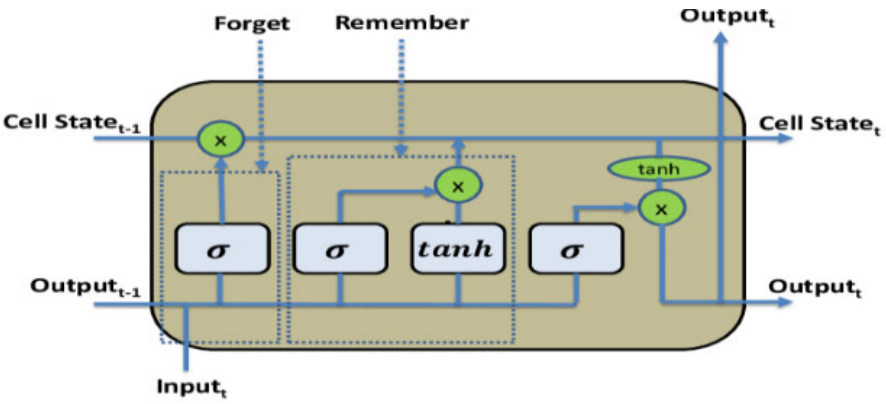
Chuẩn bị nhúng từ (Prepare Word Embeddings)

Việc nhúng từ được tạo ra để thể hiện tính dày đặc của từ. Phải mất một số từ khác nhau làm đầu vào và tạo ra một vectơ cho mỗi từ. Đây là một lớp quan trọng mà nằm giữa lớp đầu vào và lớp RNN/LSTM. Trọng lượng lớp nhúng có thể là giá trị ngẫu nhiên hoặc chúng có thể được tạo từ các phương pháp khác như wrod2vec hoặc Glove. Vì lớp Nhúng *(Embeddings)*, chúng ta cần cung cấp kích thước đầu vào, kích thước đầu ra và chiều dài đầu vào. Ở đây, kích thước đầu ra cho biết số chiều trong mỗi từ. Đầu vào thứ nguyên đại diện cho tổng số từ khác nhau trong tổng số dữ liệu. Đầu vào *Input_length* đại diện cho số lượng từ trong mỗi câu.

4.1.4 Áp dụng LSTM Model

Quá trình tính toán cho mạng LSTM bao gồm 4 bước chính:

1. Tính toán giá trị của tầng cổng quên *forget gate* và cổng vào *input gate*.
2. Cập nhật trạng thái của tế bào LSTM.
3. Tính toán giá trị đầu ra *output gate*
4. Cập nhật đầu ra toàn bộ.



Hình 11: Kiến trúc các cổng LSTM

Việc tính toán chi tiết được thể hiện dưới đây:

Input Gate:

$$a_l^t = \sum_{i=1}^{I} w_{il} x_i^t + \sum_{h=1}^{H} w_{hl} b_h^{t-1} + \sum_{c=1}^{C} w_{cl} s_c^{t-1}$$

$$b_l^t = f(a_l^t)$$

Forget Gates:

$$a_\theta^t = \sum_{i=1}^{I} w_{i\theta} x_i^t + \sum_{h=1}^{H} w_{h\theta} b_h^{t-1} + \sum_{c=1}^{C} w_{c\theta} s_c^{t-1}$$

$$b_\theta^t = f(a_\theta^t)$$

Cells:

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1}$$

$$s_c^t = b_\theta^t s_c^{t-1} + b_l^t . Sigmoid(a_c^t)$$

$$b_w^t = f(a_w^t)$$

Cell Outputs:

$$b_c^t = b_w^t . tanh(s_c^t)$$

4.2 Ứng dụng mô hình BERT giải bài toán Airline Sentiment

Như đã giới thiệu ở chương cơ sở lý thuyết, mô hình BERT sẽ chỉ quan tâm đến bộ mã hóa nhiều tầng *Encoder*, đồng thời kết hợp với cơ chế chú ý *Attention mechanism*. Trọng số *chú ý* của tất cả các yếu tố sau đó là được sử dụng để tính tổng có trọng số và xuất ra *vectơ ngữ cảnh* cuối cùng, cho phép máy biến đổi *(tranformer)* để nắm bắt cả sự phụ thuộc đoạn ngắn và đoạn dài trong kho văn bản dài.

4.2.1 Fine-Tuning

Việc áp dụng các mô hình *Transformer* được đào tạo như BERT cho các nhiệm vụ phân loại và phân tích cảm xúc đòi hỏi phải tinh chỉnh BERT. Tinh chỉnh bao gồm việc sửa đổi mô hình BERT đã được huấn luyện trước để căn chỉnh nó với yêu cầu cụ thể của nhiệm vụ được đề cập. Quá trình này đòi hỏi sự chú ý tỉ mỉ để duy. trì sự cân bằng giữa khả năng hiểu ngôn ngữ chung vốn có trong mô hình được đào tạo trước và yêu cầu chức năng cụ thể của nhiệm vụ yêu cầu.

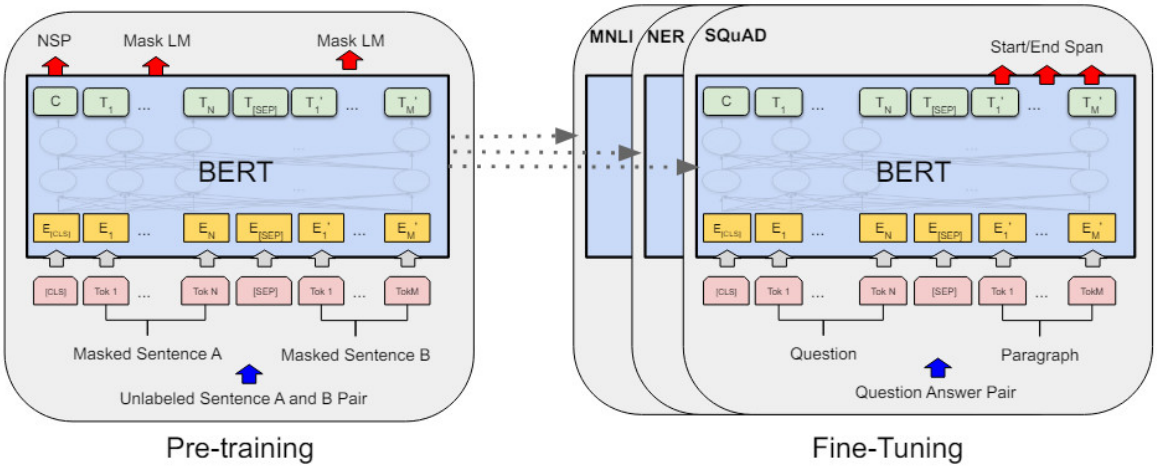
Đi sâu vào chiến lược tinh chỉnh BERT. Trong phân loại văn bản, Các lớp Softmax được sử dụng để lấy xác suất phần tử của lớp cho một dữ liệu nhất định quan sát bằng cách cung cấp cho mô hình mã thông báo đầu tiên về trạng thái ẩn cuối cùng của chuỗi. Khi được áp dụng cho tác vụ xuôi dòng, BERT có thể tự động cập nhật trọng số và cấu hình lớp đầu ra để đáp ứng yêu cầu cụ thể của nhiệm vụ yêu cầu. Ví dụ: Softmax lớp được sử dụng để phân loại nhiều nhãn hoặc lớp sigmoid được sử dụng để phân loại nhị phân.

Softmax sẽ sử dụng cho vecto Z đầu ra: $Z = [Z_1, Z_2, ..., Z_k]$:

$$Softmax(Z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, i = \overline{1, K}$$

Hay, phân loại nhị phân cho đầu ra Z (1 hoặc 0):

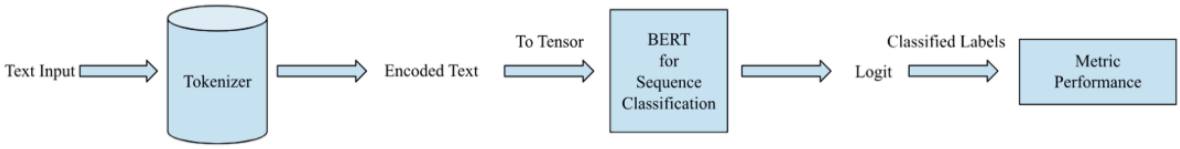
$$Sigmoid : \sigma(Z) = \frac{1}{1 + e^{-Z}}$$



Hình 12: Tổng thể quy trình pre-trainning và fine-tuning cho BERT

4.2.2 Áp dụng BERT cho bài toán Airline Sentiment

Chúng ta sẽ thực hiện mô hình BERT và thực hiện đánh giá hiệu suất của chúng bằng cách sử dụng các số liệu bao gồm độ chính xác, khả năng hội tụ và F1-Score.

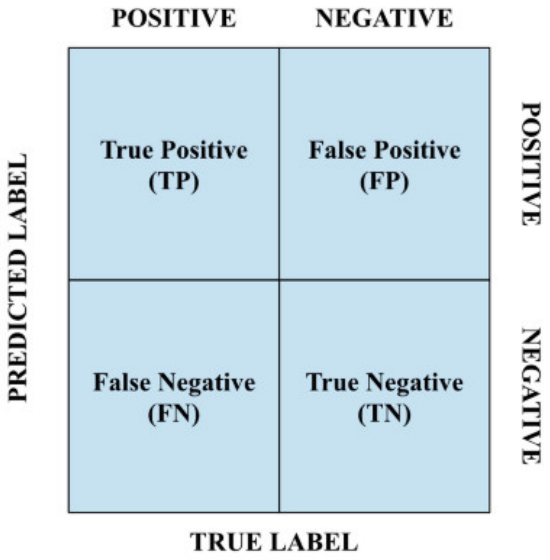


Hình 13: Biểu đồ tổng quan phương pháp

Quá trình tổng thể được thể hiện tổng thể trong sơ đồ trên. Ban đầu, chúng ta sẽ xử lý việc nhập văn bản thông qua tokenizer được thiết kế riêng cho mô hình tương ứng. Sau đó, văn bản được mã hóa sẽ được chuyển đổi thành tập dữ liệu tensor, dùng làm đầu vào cho mô hình phân loại. Sau đó, các nhật ký được tạo bởi mô hình phân loại sẽ được chuyển đổi thành các nhãn đã phân loại, cuối cùng được đánh giá bằng hiệu suất số liệu.

Đánh giá hiệu suất mô hình

Accuracy, precision, recall và F1 Score sẽ được dùng để đánh giá hiệu suất mô hình BERT. Chúng ta sẽ xây dựng một ma trận nhầm lẫn *confusion matrix* để quan sát điều đó.



Hình 14: Ma trận nhầm lẫn *(Confusion matrix)*

Trong đó, các đại lượng được thể hiện như sau:

1. **True Positives (TP):** Số lượng các mẫu mà mô hình dự đoán là đúng và thực tế cũng đúng.
2. **True Negatives (TN):** Số lượng các mẫu mà mô hình dự đoán là sai và thực tế cũng sai.
3. **False Positives (FP):** Số lượng các mẫu mà mô hình dự đoán là đúng nhưng thực tế sai (còn gọi là Type I error).
4. **False Negatives (FN):** Số lượng các mẫu mà mô hình dự đoán là sai nhưng thực tế đúng (còn gọi là Type II error).

Dựa vào *confusion matrix*, chúng ta có thể tính toán được nhiều chỉ số đánh giá khác nhau như sau:

- **Accuracy** (Độ chính xác): Tỉ lệ dự đoán đúng trên tổng số mẫu.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

- **Precision** (Độ chính xác trong các dự đoán dương): Tỉ lệ các dự đoán dương đúng trên tổng số dự đoán dương.

$$Precision = \frac{TP}{TP+FP}$$

- **Recall** (Độ nhạy, tỉ lệ đúng trong các giá trị thực tế dương): Tỉ lệ các dự đoán dương đúng trên tổng số giá trị dương thực tế.

$$Recall = \frac{TP}{TP+FN}$$

- **F1 Score**: Trung bình hài hòa của *Precision* và *Recall*, dùng để cân bằng giữa *Precision* và *Recall*.

$$F1\_Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Sử dụng kiến trúc *Transformer*, BERT thể hiện khả năng xử lý ngôn ngữ tự nhiên ưu việt với khả năng xử lý tốt so với nhiều mẫu máy cùng thời. Sự khác biệt của BERT là thích ứng kết hợp một bộ mã hóa hai chiều, cho phép nó xử lý các chuỗi đầu vào và nắm bắt toàn diện thông tin theo ngữ cảnh từ cả hai hướng của văn bản.

5 Chương trình kiểm thử và đánh giá kết quả

Ở chương này, chúng em sẽ thực hiện chạy chương trình kiểm thử, đồng thời đánh giá mức độ hiệu quả của cả 2 mô hình BERT và LSTM đã đề ra trước đó.

Môi trường và phần cứng chạy chương trình

1. Cả code nguồn cho quá trình *Trainning* cũng như *Testing* đều được thực hiện trên *Jupyter Notebook* của *Kaggle*.
2. Phần cứng bao gồm: CPU Ram 29GB, Disk space 73GB, GPU P100 memory 16GB, Output file size 19.5GB.
3. Ngôn ngữ lập trình: *Python*.
4. Source Code: *Airline Sentiment*

5.1 Chương trình kiểm thử

Khởi tạo dữ liệu

1. Tên dữ liệu: Tweets.csv
2. Dung lượng: 3.42Mb
3. Quy mô: 14460 dòng, 15 cột

5.1.1 Kiểm thử chương trình với mô hình BERT

Import thư viện

```
import random
from collections import defaultdict
import matplotlib.pyplot as plt
from textwrap import wrap
import pandas as pd
import numpy as np
import seaborn as sns
import torch
import os

from matplotlib import rc
```

```
from pylab import rcParams
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from transformers import (
    AdamW,
    BertModel,
    BertTokenizer,
    get_linear_schedule_with_warmup,
    )
```

Ở đây, chúng ta thực hiện import các framework quen thuộc dùng để thực hiện xử lý như *sklearn*, *transformers*. Thực hiện import bộ tối ưu *Adam* và *BertModel*.

Đọc DATA

```
%matplotlib inline
%config InlineBackend.figure_format='retina'

# Set color palette for plots
color = lambda: random.randint(0, 255)
colors = ['#%02X%02X%02X' %(color(), color(), color()) for i in range(6)
   ]
sns.set_palette(sns.color_palette(colors))
rcParams['figure.figsize'] = 12, 10

RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
torch.manual_seed(RANDOM_STATE)
```

Kiểm tra device xem có GPU hay không

```
device = torch.cuda.is_available()

if not device:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')
```

Nếu *device* có GPU sẽ thực hiện sử dụng GPU cho quá trình *trainning* cũng như *Testing*.

DATA EXPLORATION

```
from kaggle.api.kaggle_api_extended import KaggleApi
from zipfile import ZipFile
```

```
os.makedirs('data', exist_ok=True)

api = KaggleApi()
api.authenticate()
api.dataset_download_file('crowdflower/twitter-airline-sentiment', '
    Tweets.csv')
zip_file = ZipFile('Tweets.csv.zip')
zip_file.extractall('data')
zip_file.close()
os.remove('Tweets.csv.zip')
df = pd.read_csv('data/Tweets.csv')
df = df[df['airline_sentiment'] != 'neutral']
df.head()
```

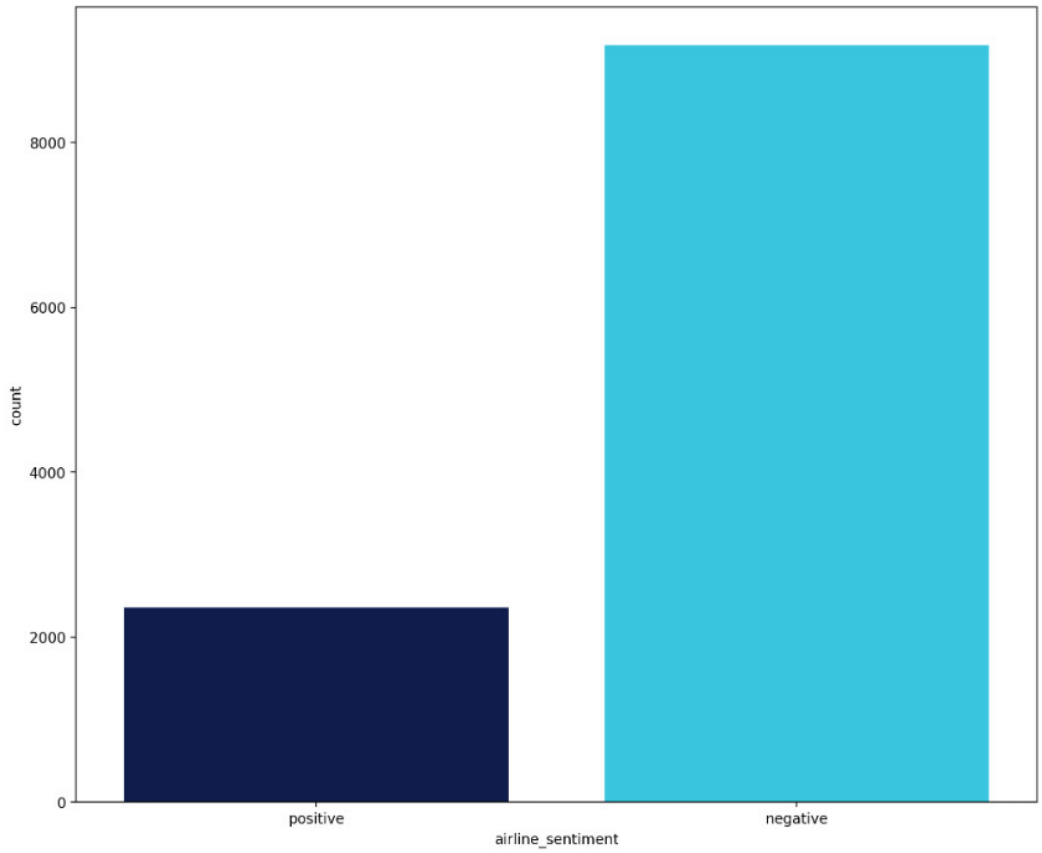
Chúng ta sẽ thực hiện in ra 5 dòng đầu tiên của bộ dữ liệu để quan sát, đồng thời sẽ quan tâm đến 2 lớp dữ liệu chính của trường *airline_setiment* là *positive* và *negative*. Việc quan sát dữ liệu là rất quan trọng, vì điều đó nhằm tiền xử lý dữ liệu một cách hợp lý và chính xác.

tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold
570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN
570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN
570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN
570300767074181121	negative	1.0000	Can't Tell	0.6842	Virgin America	NaN
570300616901320704	positive	0.6745	NaN	0.0000	Virgin America	NaN

Hình 15: Quan sát dữ liệu gốc

Thực hiện in ra biểu đồ thể hiện số lượng cảm xúc tích cực và tiêu cực. Như có thể thấy, tập dữ liệu rất mất cân bằng. Nó có nhiều tweet tiêu cực hơn so với các tweet tích cực. Nhưng không sao, chúng ta sẽ không đi sâu vào các kỹ thuật lấy mẫu quá mức hoặc lấy mẫu dưới mức để giải quyết vấn đề mất cân bằng lớp. Mặc dù các tweet tiêu cực nhiều hơn nhưng chúng ta vẫn có thể xử lý với một số tweet tích cực.

```
    sns.countplot(x=df.airline_sentiment)
```



Hình 16: Tổng quan số lượng loại cảm xúc

Quá trình xử lý dữ liệu

Hãy thực hiện một số xử lý trước một chút trên tập dữ liệu trước khi huấn luyện chúng trên mô hình. Có thể có một số biểu tượng cảm xúc, lượt đề cập, thẻ bắt đầu bằng # nên được coi là một thực thể duy nhất. Xóa URL, số vì chúng không truyền tải bất kỳ thông tin hữu ích nào. Hơn nữa, trong trường hợp kéo dài các từ phổ biến hơn trong các tweet sẽ được chuẩn hóa. Chúng ta sẽ không chuyển đổi nó thành token vì nó sẽ được BERT thực hiện.

```
    import re
def pre_processing(df):
    df['text'] = df['text'].apply(lambda x: re.sub('http[s]?://(?:[a-zA-
        Z]|[0-9]|[$-_@.&+#]|[!*\(\),]|'\
                        '(?:%[0-9a-fA-F][0-9a-fA-F]))+','', x))
    df['text'] = df['text'].apply(lambda x: re.sub('(@[A-Za-z0-9_]+)',''
        , x))
```

```
    pre_processing(df)

# Encode labels
possible_labels = df.airline_sentiment.unique()
labels_map = {possible_labels[idx]: idx for idx, label in enumerate(
    possible_labels)}
```

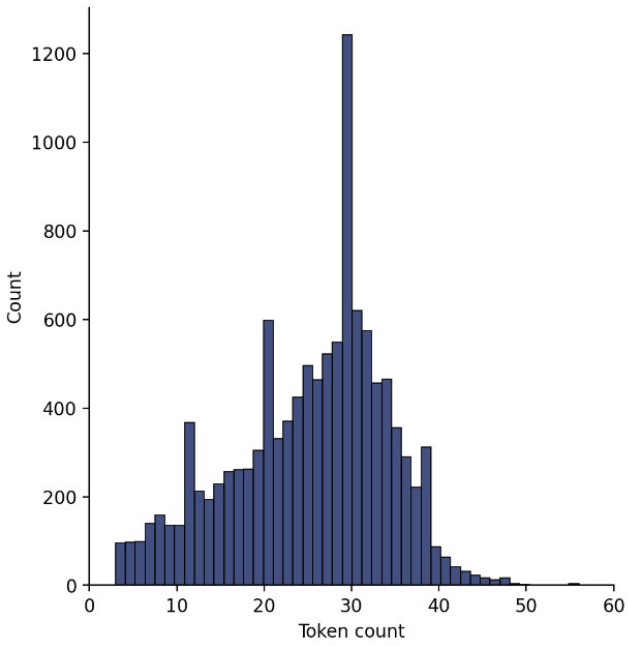
```
df['labels'] = df.airline_sentiment.replace(labels_map)
```

Trước khi chuyển sang nhiệm vụ phân loại, cần phải mã hóa văn bản thành dạng thích hợp để BERT có thể sử dụng nó làm đầu vào. Đây là nơi mà mã thông báo của BERT phát huy tác dụng, chúng em sẽ sử dụng encode_plus để thực hiện những việc sau:

1. Tokenize câu đầu vào.
2. Thêm mã thông báo [CLS] và [SEP]
3. Đệm hoặc cắt ngắn câu đến độ dài tối đa cho phép
4. Mã hóa các mã thông báo vào Bảng ID tương ứng của chúng hoặc cắt bớt tất cả các câu có cùng độ dài.
5. Tạo hàm chú ý để phân biệt rõ real tokens từ [PAD] tokens.

Thực hiện xem độ dài của tokens được sử dụng thường xuyên để có ý tưởng về độ dài tokens. Dựa trên biểu đồ dưới, chúng ta an toàn khi đặt 64 làm độ dài tokens.

```
    tokens = [len(tokenizer.encode(text, max_length=512, truncation=True
       )) for text in df.text]
sns.displot(tokens)
plt.xlim([0, 60]);
plt.xlabel('Token count');
```



Hình 17: Token count

Một mã đơn giản để xây dựng Tập dữ liệu tùy chỉnh và tải dữ liệu bằng DataLoader.

```
    from torch.utils.data import Dataset, DataLoader


class CustomDataset(Dataset):

    def __init__(
        self,
        tweets,
        labels,
        tokenizer,
        max_length
    ):
        self.tweets = tweets
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.tweets)

    def __getitem__(self, idx):
        tweet = self.tweets[idx]
        label = self.labels[idx]

        tokenize = self.tokenizer.encode_plus(
            tweet,
            add_special_tokens=True,
            max_length=self.max_length,
            return_token_type_ids=False,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='pt'
        )
        return {
            'tweet': tweet,
            'input_ids': tokenize['input_ids'].flatten(),
            'attention_mask': tokenize['attention_mask'].flatten(),
            'targets': torch.tensor(label, dtype=torch.long)
               }
```

Như ở trên đã phân tích, chúng ta sẽ lấy độ dài tối đa của tokens là 64. Chia tập train/test tỷ lệ 70/30 %, khởi tạo validation_size với hệ số 0.5, batch-size là 16. Tất cả các dữ liệu này sẽ định dạng số và chuyển dạng numpy array.

```
    MAX_LENGTH = 64
TEST_SIZE = 0.3
VALID_SIZE = 0.5
BATCH_SIZE = 16
NUM_WORKERS = 2
```

```
    train_sampler, test_sampler = train_test_split(df, test_size=
        TEST_SIZE, random_state=RANDOM_STATE)
valid_sampler, test_sampler = train_test_split(test_sampler, test_size=
    VALID_SIZE, random_state=RANDOM_STATE)

train_set = CustomDataset(
    train_sampler['text'].to_numpy(),
    train_sampler['labels'].to_numpy(),
    tokenizer,
    MAX_LENGTH
)
test_set = CustomDataset(
    test_sampler['text'].to_numpy(),
    test_sampler['labels'].to_numpy(),
    tokenizer,
    MAX_LENGTH
)
valid_set = CustomDataset(
    valid_sampler['text'].to_numpy(),
    valid_sampler['labels'].to_numpy(),
    tokenizer,
    MAX_LENGTH
)

train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, num_workers=
    NUM_WORKERS)
test_loader = DataLoader(test_set, batch_size=BATCH_SIZE, num_workers=
    NUM_WORKERS)
valid_loader = DataLoader(valid_set, batch_size=BATCH_SIZE, num_workers=
    NUM_WORKERS)
```

Hầu hết các công việc khó khăn đều do BERT thực hiện. Tuy nhiên, chúng ta cần sửa đổi lớp cuối cùng theo nhiệm vụ phân loại của chúng ta là phân loại nhị phân.

```
    from torch import nn
class AirlineSentimentClassifier(nn.Module):

    def __init__(self, num_labels):
        super (AirlineSentimentClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(MODEL)
        self.dropout = nn.Dropout(p=0.2)
        self.classifier = nn.Linear(self.bert.config.hidden_size,
            num_labels)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        out = self.classifier(pooled_output)
        return out
```

Thực hiện tạo hàm tính toán forward, với hệ số dropout là 0.2, đồng thời đưa vào hàm *attention*.

Loss Function và Optimizer

Chúng ta sẽ sử dụng mất mát Entropy chéo *Cross-Entropy* phân loại và trình tối ưu hóa *AdamW* (vì nó thực hiện sửa lỗi giảm trọng số cho thuật toán Adam thông thường) bằng *Hugging Face*. Ta sẽ trainning model với 20 epochs và hệ số learning rate là 2×10^{-5}.

```
    n_epochs = 20
learning_rate =2e-5

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimizer
optimizer = AdamW(model.parameters(), lr=learning_rate, correct_bias=
    False)

# Define scheduler
training_steps = len(train_loader)*n_epochs
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps=training_steps
)
```

Trainning Process

Thực hiện in ra màn hình từng epoch, với mỗi epoch sẽ thể hiện đủ 4 yếu tố: Trainning loss, Validation loss, Tranning accuracy và Validation accuracy.

```
    valid_loss_min = np.Inf

for epoch in range(1, n_epochs+1):

    # Setting training and validation loss
    train_loss = []
    validation_loss = []
    tr_predictions = 0
    acc = 0
    val_predictions = 0

    model = model.train()
    for data in train_loader:

        if device:
```

```
            input_ids, attention_mask, targets = data["input_ids"].cuda
                (), data["attention_mask"].cuda(), data["targets"].cuda()
        # Clear the gradients of variables
        optimizer.zero_grad()

        #### Forward pass
        output = model(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        loss = criterion(output, targets)
        _, pred = torch.max(output, 1)
        tr_predictions += torch.sum(pred == targets)

        #### Backward Pass
        # Compute gradients wrt to model parameters
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        scheduler.step()
        train_loss.append(loss.item())

    # Validate the model #
    model.eval()
    with torch.no_grad():
        for data in valid_loader:

            if device:
                input_ids, attention_mask, targets = data["input_ids"].
                    cuda(), data["attention_mask"].cuda(), data["targets"
                    ].cuda()

            output = model(
                input_ids=input_ids,
                attention_mask=attention_mask
            )
            # Compute batch loss
            loss = criterion(output, targets)
            _, pred = torch.max(output, 1)
            # Update loss per mini batches
            validation_loss.append(loss.item())
            val_predictions += torch.sum(pred == targets)

    train_accuracy = tr_predictions.double()/len(train_sampler)
    val_accuracy = val_predictions.double()/len(valid_sampler)

    print('Epoch: {}/{} \n\tTraining Loss: {:.6f} \n\tValidation Loss:
        {:.6f} \n\tTrain Accuracy: {:.6f} \n\tVal Accuracy: {:.6f}'.
        format(epoch,n_epochs, np.mean(train_loss), np.mean(
        validation_loss), train_accuracy, val_accuracy))

    if val_accuracy > acc:
         print('Saving model...')
         torch.save(model.state_dict(), 'bert_base_fine_tuned.pt')
         acc = val_accuracy
```

```
Epoch: 1/20
        Training Loss: 0.208853
        Validation Loss: 0.187671
        Train Accuracy: 0.926096
        Val Accuracy: 0.939919
Saving model...
Epoch: 2/20
        Training Loss: 0.086505
        Validation Loss: 0.214128
        Train Accuracy: 0.976108
        Val Accuracy: 0.948007
Saving model...
Epoch: 3/20
        Training Loss: 0.033488
        Validation Loss: 0.242103
        Train Accuracy: 0.991087
        Val Accuracy: 0.950895
Saving model...
Epoch: 4/20
        Training Loss: 0.014658
        Validation Loss: 0.275657
        Train Accuracy: 0.996905
        Val Accuracy: 0.953784
Saving model...
Epoch: 5/20
        Training Loss: 0.008908
        Validation Loss: 0.323875
        Train Accuracy: 0.998143
        Val Accuracy: 0.953206

Epoch: 6/20
        Training Loss: 0.005818
        Validation Loss: 0.311122
        Train Accuracy: 0.998514
        Val Accuracy: 0.954939
Saving model...
Epoch: 7/20
        Training Loss: 0.003973
        Validation Loss: 0.312415
        Train Accuracy: 0.998886
        Val Accuracy: 0.957250
Saving model...
Epoch: 8/20
        Training Loss: 0.004660
        Validation Loss: 0.296874
        Train Accuracy: 0.998638
        Val Accuracy: 0.957828
Saving model...
Epoch: 9/20
        Training Loss: 0.001389
        Validation Loss: 0.372451
        Train Accuracy: 0.999133
        Val Accuracy: 0.954939
Saving model...
Epoch: 10/20
        Training Loss: 0.001826
        Validation Loss: 0.372373
        Train Accuracy: 0.999133
        Val Accuracy: 0.953206

Epoch: 11/20
        Training Loss: 0.002153
        Validation Loss: 0.429960
        Train Accuracy: 0.999010
        Val Accuracy: 0.953206
Saving model...
Epoch: 12/20
        Training Loss: 0.001150
        Validation Loss: 0.408063
        Train Accuracy: 0.999133
        Val Accuracy: 0.955517
Saving model...
Epoch: 13/20
        Training Loss: 0.001069
        Validation Loss: 0.418787
        Train Accuracy: 0.999381
        Val Accuracy: 0.956095
Saving model...
Epoch: 14/20
        Training Loss: 0.001443
        Validation Loss: 0.441278
        Train Accuracy: 0.999133
        Val Accuracy: 0.952629
Saving model...
Epoch: 15/20
        Training Loss: 0.002316
        Validation Loss: 0.430669
        Train Accuracy: 0.999133
        Val Accuracy: 0.953206

Epoch: 16/20
        Training Loss: 0.002682
        Validation Loss: 0.425907
        Train Accuracy: 0.999010
        Val Accuracy: 0.954939
Saving model...
Epoch: 17/20
        Training Loss: 0.000831
        Validation Loss: 0.438467
        Train Accuracy: 0.999629
        Val Accuracy: 0.954362
Saving model...
Epoch: 18/20
        Training Loss: 0.000913
        Validation Loss: 0.435880
        Train Accuracy: 0.999133
        Val Accuracy: 0.952629
Saving model...
Epoch: 19/20
        Training Loss: 0.000849
        Validation Loss: 0.430354
        Train Accuracy: 0.999381
        Val Accuracy: 0.955517
Saving model...
Epoch: 20/20
        Training Loss: 0.000965
        Validation Loss: 0.430085
        Train Accuracy: 0.999133
        Val Accuracy: 0.955517
```

Quan sát kết quả qua từng *Epoch* ta rút ra nhận xét:

Nhận xét kết quả:

- Trainning Loss đã giảm từ 0.208853 ở *epoch 1* xuống 0.000965 ở *epoch thứ 20*
- Validation Loss có sự tăng, hoặc giảm nhẹ sau mỗi epoch. Nhưng giường như càng nhiều epoch thì nó đã tăng lên đôi chút
- Khi quan sát thấy *Trainning Loss* và *Validation Loss* có sự khác nhau về độ dốc, khi Trainning Loss đang giảm mà *Validation Loss* lại có sự tăng nhẹ qua nhiều *epoch* liên tiếp. Đó là biểu hiện của sự *overfitting*. Chính vì vậy mà chúng ta thực hiện dừng sớm *(Early Stopping)* để tránh hiện tượng *Overfitting* xảy ra.
- *Validation Accuracy* có sự tăng nhẹ sau mỗi *epoch*, và nó giữ ở mức ổn định cao trong khoảng 5 *epoch* cuối. Với hiệu suất trung bình khoảng 95,12% thì có thể thấy đấy mô hình hoạt động rất tốt, khả năng dự đoán cao, bước đầu thể hiện được mức độ hiệu quả của mô hình BERT trong bài toán phân tích cảm xúc khách hàng.

Kiểm tra Fine-Tuned Network

```
test_loss = 0.0
class_predictions = list(0. for i in range(3))
class_total = list(0. for i in range(3))
predictions = []
labels = []

model.eval()
with torch.no_grad():
    for data in test_loader:

        if device:
            input_ids, attention_mask, targets = data["input_ids"].cuda
               (), data["attention_mask"].cuda(), data["targets"].cuda()

        output = model(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        loss = criterion(output, targets)
        test_loss += loss.item()
        _, pred = torch.max(output, 1)

        predictions.extend(pred)
        labels.extend(targets)
```

```
predictions = torch.stack(predictions) if not device else torch.stack(
    predictions).cpu()
labels = torch.stack(labels) if not device else torch.stack(labels).cpu
    ()
```

Mặc dù độ chính xác của *Trainning Accuracy* đạt trên 99% nhưng hãy xác định một số chỉ số và xem mô hình hoạt động như thế nào trên dữ liệu không nhìn thấy được.

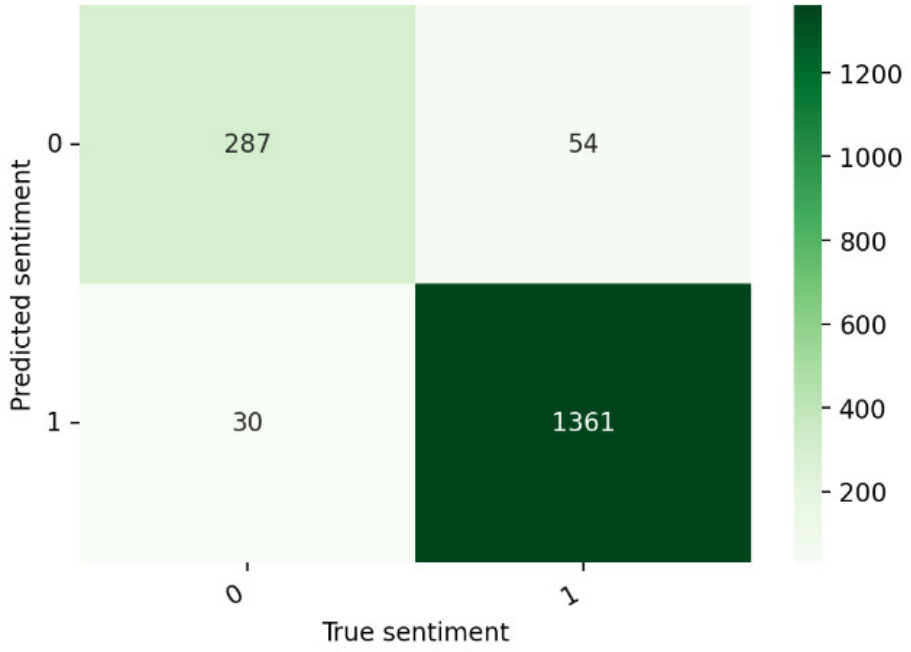
	precision	recall	f1-score	support
positive	0.84	0.91	0.87	317
negative	0.98	0.96	0.97	1415
accuracy			0.95	1732
macro avg	0.91	0.93	0.92	1732
weighted avg	0.95	0.95	0.95	1732

Hình 18: Kết quả Fine-Tuned

Visualizations

```
cm = confusion_matrix(labels, predictions)
heatmap = sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,
     ha='right')
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation
    =30, ha='right')
plt.xlabel('True sentiment')
plt.ylabel('Predicted sentiment');
plt.gcf().set_size_inches(6, 4)
plt.show()
```

Hãy trực quan hóa hiệu suất mô hình của chúng ta thông qua việc vẽ ma trận nhầm lẫn *(confussionmatrix)*. Chúng ta cũng thấy rằng, mô hình hoạt động tốt trong trường hợp các dòng *tweet negative*. Quan sát số liệu cũng có thể thấy rằng, 2 cảm xúc xác định đúng trong ma trận đều chiếm đa số của bộ dữ liệu testing. Sai lầm loại 1 và sai lầm loại 2 chiếm khoảng 4,85% giá trị. Chúng ta có thể kết luận rằng nó hoạt động ở mức khá tốt hay tương đối, đồng thời chúng ta không thể hoàn toàn chính xác khi nói đến sự mất cân bằng giữa các lớp.



Hình 19: BERT Visualizations

5.1.2 Kiểm thử chương trình với mô hình LSTM

Như đã phân tích, mạng LSTM thực hiện xử lý số, chính vì vậy mà chúng ta cần quan sát dữ liệu và chuyển các trạng thái cảm xúc thành dạng index 0 và 1. Thực hiện loại bỏ các ký tự, ký hiệu và các dấu chấm,...

Tiền xử lý DATA

```
data = pd.read_csv("/kaggle/input/dataset/Tweets.csv")
df = data[["text","airline_sentiment"]]
df['text'] = df['text'].map(lambda x: x.lstrip('
    @VirginAmerica@UnitedAir@Southwestairline@DeltaAir@USAirways@American
    ').rstrip('@'))
```

```
df = df[df.airline_sentiment!="neutral"] # To remove neutral responses
df['text'] = df['text'].apply(lambda x: x.lower()) # To lower
df['text'] = df['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))
    # To keep numbers and strings only
```

	text	airline_sentiment
1	plus youve added commercials to the experienc...	positive
3	its really aggressive to blast obnoxious ente...	negative
4	and its a really big bad thing about it	negative
5	seriously would pay 30 a flight for seats tha...	negative
6	yes nearly every time i fly vx this ear worm ...	positive

Hình 20: Đọc dữ liệu đầu

Sử dụng *tokenizer* xây dựng lấy ra giá trị của array cảm xúc với độ dài tối đa cho từ là 4000.

```
max_fatures = 4000
tokenizer = Tokenizer(num_words = max_fatures, split=' ')
tokenizer.fit_on_texts(df['text'].values)
X = tokenizer.texts_to_sequences(df['text'].values)
X = pad_sequences(X)
Y = df['airline_sentiment']
L = Y.values
X
L
```

Thực hiện định dạng nhị phân cho dữ liệu cảm xúc, với *"negative"* là 0 và *"positive"* là 1.

```
k = []
for i in range(6541):
    if L[i]=="negative":
                        k.append(0)
    elif L[i]=="positive":
                        k.append(1)
```

Train Test Split

Ở đây chúng ta sẽ chia tập train/test với tỷ lệ 70/30

```
X_train, X_test, Y_train, Y_test = train_test_split(X, k,test_size=0.3,
    shuffle=True,stratify = k,random_state = 1 )
```

Xây dựng model

Thực hiện sử dụng kết hợp 2 thư viện là tensoflow và keras để xây dựng cho bài toán phân loại 2 lớp. Đầu tiên, chúng ta sẽ nhúng chỉ số các bộ từ vựng thành các vecto nhúng *(embedding vectors)* có kích thước cố định là

128. Tiếp thep, sẽ giảm thiểu *overfitting* bằng cách ngẫu nhiên bỏ qua toàn bộ các *feature map* trong các bước time steps khác nhau với tỷ lệ 50%.

Tiếp theo thực hiện xây dựng *LSTM Layer*, xử lý dữ liệu tuần tự và nắm bắt các mối quan hệ dài hạn trong chuỗi đầu vào với 196 *lớp ẩn*, tỷ lệ *Drop out* cho các kết nối đầu vào cũng như kết nối tuần hoàn là 0.3. Tiếp theo, chúng ta sẽ bỏ qua 20% số neural trong quá trình huấn luyện hay hệ số *Drop out* là 0.2. Lớp đầu ra với 2 loại đơn vị vì đây là bài toán phân loại nhị phân. Cuối cùng là sử dụng *softmax* để chuyển đầu ra thành xác suất dự đoán.

```
embed_dim = 128
lstm_out = 196
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Embedding(max_fatures, 128, input_length=
    X_train.shape[1]))
model.add(tf.keras.layers.SpatialDropout1D(0.5))
model.add(tf.keras.layers.LSTM(196, dropout = 0.3, recurrent_dropout =
    0.3 ))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(100, activation = tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.4))
model.add(tf.keras.layers.Dense(2, activation = tf.nn.softmax))
```

Loss function và Optimizer

```
    model.compile(optimizer="adam", loss="
       sparse_categorical_crossentropy", metrics=["accuracy"])
```

- Sử dụng bộ tối ưu hóa Adam để cập nhật trọng số của mô hình.
- Sử dụng hàm mất mát *Sparse Categorical Crossentropy* để đo lường sự khác biệt giữa các dự đoán của mô hình và nhãn thực tế.
- Theo dõi độ chính xác của mô hình trong suốt quá trình huấn luyện và kiểm tra.

Chạy model và in kết quả

Chúng ta sẽ đưa vào model 2 tập train lần lượt là text và cảm xúc đã đưa về nhị phân, cũng giống với BERT chúng ta sẽ train với 20 epochs để tiện cho việc so sánh 2 model, Kích thước batch: 32. Sau mỗi epoch sẽ dùng 20% dữ liệu để đánh giá ($validation = 0.2$).

```
Model = model.fit(X_train, Y_train, epochs=20,batch_size=32,
    validation_split = 0.2,verbose = 2)
```

Kết quả model:

```
Epoch 1/20
115/115 - 11s - 98ms/step - accuracy: 0.7750 - loss: 0.4911 - val_accuracy: 0.8581 - val_loss: 0.3301
Epoch 2/20
115/115 - 5s - 47ms/step - accuracy: 0.8938 - loss: 0.2622 - val_accuracy: 0.8919 - val_loss: 0.2637
Epoch 3/20
115/115 - 6s - 49ms/step - accuracy: 0.9358 - loss: 0.1673 - val_accuracy: 0.9017 - val_loss: 0.2535
Epoch 4/20
115/115 - 10s - 89ms/step - accuracy: 0.9555 - loss: 0.1274 - val_accuracy: 0.9061 - val_loss: 0.2460
Epoch 5/20
115/115 - 6s - 49ms/step - accuracy: 0.9664 - loss: 0.0980 - val_accuracy: 0.9148 - val_loss: 0.2602
Epoch 6/20
115/115 - 6s - 51ms/step - accuracy: 0.9776 - loss: 0.0647 - val_accuracy: 0.9148 - val_loss: 0.2947
Epoch 7/20
115/115 - 5s - 48ms/step - accuracy: 0.9817 - loss: 0.0502 - val_accuracy: 0.9225 - val_loss: 0.3365
Epoch 8/20
115/115 - 6s - 50ms/step - accuracy: 0.9850 - loss: 0.0554 - val_accuracy: 0.9094 - val_loss: 0.3073
Epoch 9/20
115/115 - 6s - 49ms/step - accuracy: 0.9869 - loss: 0.0361 - val_accuracy: 0.9050 - val_loss: 0.3621
Epoch 10/20
115/115 - 6s - 51ms/step - accuracy: 0.9880 - loss: 0.0420 - val_accuracy: 0.9072 - val_loss: 0.3249
Epoch 11/20
115/115 - 6s - 48ms/step - accuracy: 0.9883 - loss: 0.0379 - val_accuracy: 0.8996 - val_loss: 0.3347
Epoch 12/20
115/115 - 6s - 49ms/step - accuracy: 0.9904 - loss: 0.0306 - val_accuracy: 0.8854 - val_loss: 0.3891
Epoch 13/20
115/115 - 6s - 49ms/step - accuracy: 0.9937 - loss: 0.0191 - val_accuracy: 0.9007 - val_loss: 0.3840
Epoch 14/20
115/115 - 5s - 48ms/step - accuracy: 0.9910 - loss: 0.0294 - val_accuracy: 0.9083 - val_loss: 0.3616
Epoch 15/20
115/115 - 6s - 49ms/step - accuracy: 0.9951 - loss: 0.0203 - val_accuracy: 0.8930 - val_loss: 0.5376
Epoch 16/20
115/115 - 5s - 47ms/step - accuracy: 0.9913 - loss: 0.0262 - val_accuracy: 0.8799 - val_loss: 0.4241
Epoch 17/20
115/115 - 6s - 48ms/step - accuracy: 0.9945 - loss: 0.0178 - val_accuracy: 0.9007 - val_loss: 0.5033
Epoch 18/20
115/115 - 6s - 49ms/step - accuracy: 0.9951 - loss: 0.0151 - val_accuracy: 0.9017 - val_loss: 0.6102
Epoch 19/20
115/115 - 6s - 48ms/step - accuracy: 0.9934 - loss: 0.0173 - val_accuracy: 0.9050 - val_loss: 0.5052
Epoch 20/20
115/115 - 6s - 48ms/step - accuracy: 0.9962 - loss: 0.0161 - val_accuracy: 0.9116 - val_loss: 0.6034
```

Quan sát kết quả, chúng ta có thể nhận thấy rằng, sau mỗi epoch đã có sự tối ưu kết quả. Trainning accuracy đã tăng từ 77,5% lên đến trên 99%, val_accuracy cũng đã tăng lên và có xu hướng ổn định từ trong 10 epoch sau. Ở đây, sau khi train khoảng 15 epochs và sau đó thấy val_loss đang có sự tăng lên đánh kể trong khi trainning loss lại giảm xuống rất sâu, chính vì vậy em đã thực hiện dừng để tránh hiện tượng *overfitting*. Chúng ta có một đánh giá tổng quát như sau:

- Mô hình có độ chính xác trung bình cao (90.83%), điều này cho thấy

mô hình có khả năng phân loại chính xác phần lớn các mẫu trong tập huấn luyện.

- Giá trị loss (0.6732) có thể chấp nhận được, nhưng có thể cải thiện thêm bằng cách điều chỉnh các siêu tham số hoặc thu thập thêm dữ liệu. (điều chỉnh siêu tham số epochs hay batch_size)

Hình 21: Trực quan hóa Loss và Accuracy theo từng epoch

Quan sát có thể thấy rằng, chúng ta có thể điều chỉnh số epoch xuống 10 hay 15, khi đó val_accuracy giữ ở mức cao hơn, đồng thời val_loss đang giữ ở mức thấp và chưa tăng nhiều như 10 epochs sau.

Vẽ ma trận confusion

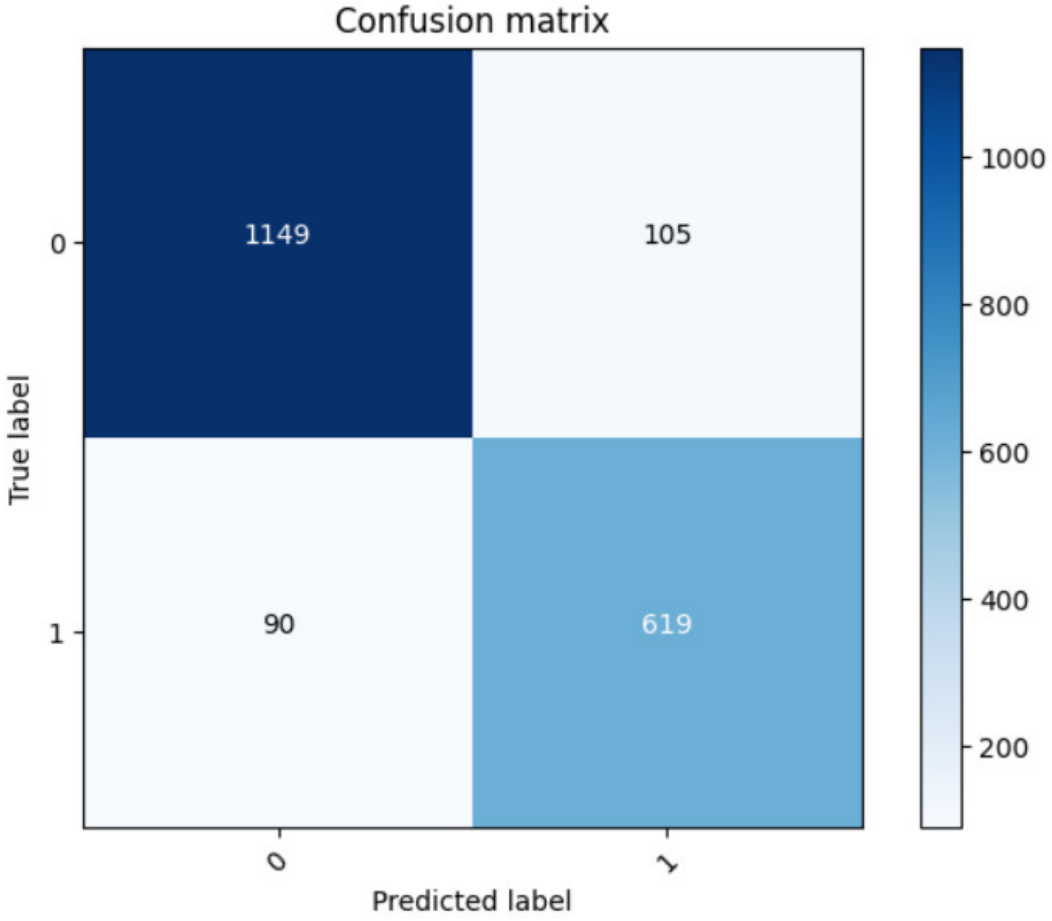
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
#     plt.figsize(6,4)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])
        ):
        plt.text(j, i, cm[i, j],
```

```
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred,axis = 1)
confusion_mtx = confusion_matrix(Y_test, y_pred_classes)
plot_confusion_matrix(confusion_mtx, classes = range(2))
```



Hình 22: Ma trận confusion

Chúng ta cũng thấy rằng, mô hình hoạt động tốt trong trường hợp các dòng *tweet positive*. Quan sát số liệu cũng có thể thấy rằng, 2 cảm xúc xác định đúng trong ma trận đều chiếm đa số của bộ dữ liệu testing. Sai lầm loại 1 và sai lầm loại 2 chiếm khoảng 9,17% giá trị. Chúng ta có thể kết luận rằng nó hoạt động ở mức khá tốt hay tương đối, đồng thời chúng ta không thể hoàn toàn chính xác khi nói đến sự mất cân bằng giữa các lớp.

5.1.3 Kiểm thử với những bộ dữ liệu khác

Sau khi trainning và thực hiện kiểm thử với chính bộ dữ liệu gốc cho chúng ta được một kết quả rất khả quan với accuracy lên đến 90 %. Để đánh giá một cách tổng thể hơn, chúng ta sẽ thực hiện áp dụng model cho 2 tập dữ liệu khác. Trong đó:

- Bộ dữ liệu 1: Airline_Sentiment test: Đây là bộ dữ liệu tự sinh mới dùng để kiểm tra cho trường hợp được train với dữ liệu về airline_sentiment và test với một airline_sentiment khác.
- Bộ dữ liệu 2: Customer_Sentiment test: Đây là bộ dữ liệu chung về đánh giá cảm xúc khách hàng trên nhiều lĩnh vực khác nhau.

Kiểm tra với bộ dữ liệu 1 *(Airline_sentiment_test1)*:

Bộ dữ liệu với 100 dòng tự sinh khác nhau, vẫn với 2 trường dữ liệu là *'Review'* và *'Sentiment'*.

Tên file: */kaggle/input/datatesst/airline_sentiment_differ_test.csv*.

	Review	Sentiment
0	The flight attendants were incredibly friendly...	Positive
1	My flight was delayed for hours with no explan...	Negative
2	The seats were very comfortable and spacious.	Positive
3	Lost my luggage and no one seemed to care.	Negative
4	Excellent in-flight entertainment options!	Positive
...	...	...
95	They did not handle the delay well.	Negative
96	The overall flight experience was great.	Positive
97	The staff was rude and unaccommodating.	Negative
98	Enjoyed the comfortable seating and good service.	Positive
99	The flight was delayed with no clear explanation.	Negative

100 rows × 2 columns

Hình 23: Dữ liệu test 1

```
rate_test = 0
for i in range(100):
    samples = [df_test.at[i,'Review']]
    samples = tokenizer.texts_to_sequences(samples)
    samples=pad_sequences(samples ,maxlen=31, dtype='int32', value=0)
```

```
#      print(sample)
    sentiment = model.predict(samples  ,batch_size=1,verbose = 2)[0]
    if(np.argmax(sentiment) == 0) and (df_test.at[i, 'Sentiment'] == '
       Negative'):
        rate_test += 1
    if (np.argmax(sentiment) == 1) and (df_test.at[i, 'Sentiment'] == '
       Positive'):
        rate_test += 1
print("Rate_test =", rate_test, "%")
```

Kết quả:

```
Rate_test = 87 %
```

Nhận xét:

Đối với bộ dự liệu hoàn toàn mới, chúng ta cũng thấy được rằng mô hình cho một kết quả là tương đối tốt, khi không có sự khác biệt quá nhiều. Đối với bộ dữ liệu gốc cho *accuracy* khoảng 90% còn với bộ dữ liệu *Test1* cho tỷ lệ dự đoán đúng đến 87%. Từ đó gián tiếp khẳng định, mô hình học khá tốt và không xảy ra hiện tượng *overfitting*.

Kiểm tra với bộ dữ liệu 2 *(Customer_sentiment_test2)*:

Bộ dữ liệu với 200 dòng tự sinh khác nhau, vẫn với 2 trường dữ liệu là *'Review'* và *'Sentiment'*.

Tên file: */kaggle/input/datatest2/customer_reviews.csv*.

	Review	Sentiment
0	The product is fantastic, I'm very satisfied!	Positive
1	The service is too slow, I'm not happy at all.	Negative
2	The staff is very friendly and helpful.	Positive
3	Poor quality, I will not buy again.	Negative
4	Fast and timely delivery.	Positive
...	...	...
195	The delivery man was very rude.	Negative
196	The event was very well-organized.	Positive
197	The TV has a blurry picture.	Negative
198	The perfume is very long-lasting.	Positive
199	The chair is very uncomfortable.	Negative

200 rows × 2 columns

Hình 24: Dữ liệu test 2

Kết quả:

```
Rate_test2 =  76.0 %
```

Nhận xét:

Đối với bộ dữ liệu có quy mô rộng hơn, không còn nằm trong phạm vi bài toán airline_setiment mà đã mở rộng ra nhiều lĩnh vực khác thì khả năng dự đoán của mô hình đã giảm đi một chút chỉ còn 76%. Tuy nhiên, đây cũng là một con số tương đối tốt với một bộ dữ liệu mang tính bao quát lớn. Ngoài ra, để tăng hiệu suất dự đoán cho bài toán này, chúng ta có thể áp dụng phương pháp tăng cường dữ liệu cho tập train *Data Augumentation*.

5.1.4 So sánh mô hình BERT và LSTM cho bài toán Airline Sentiment

Chúng ta sẽ thực hiện xây dựng một bảng đánh giá so sánh các chỉ số giữa 2 mô hình như sau:

	BERT	LSTM
Trainning accuracy	0.9991	0.9962
Validation accuracy	0.9512	0.9116
Trainning loss	0.00097	0.0161
Validation loss	0.4304	0.6034
Precision (P)	0.84	0.9163
Recall (P)	0.91	0.9274
F-1 Score (P)	0.87	0.9218
Precision (N)	0.98	0.8856
Recall (N)	0.96	0.8550
F-1 Score (N)	0.97	0.8700

(P): *Positive.*
(N): *Negative.*

Nhận xét:

Quát sát chung về các chỉ số, ta có thể nhận xét được rằng: về độ chính xác thì mô hình BERT đang cao hơn một chút so với mô hình mạng LSTM, đồng thời cả về khắc phục vấn đề *overfitting* thì BERT cũng đang cho thấy độ nhỉnh hơn về sự chênh lệch giữa *Trainning loss* và *Validation loss*.

Tuy nhiên, xét các chỉ số mở rộng như *Precision, Recall, F-1 Score* thì cả 2 mô hình lại cho thấy thế mạnh riêng của nó. Khi BERT đang cho thấy

độ chính xác với các trường hợp *'negative'* thì LSTM lại mang độ chính xác tốt hơn về *'positive'*. Từ đó, có thể nhận định rằng. Tuy đối với mỗi model có các ưu và nhược điểm khác nhau. Nhưng nhìn chung cả 2 mô hình đều cho chúng ta một kết quả tương đối tốt cả về dữ liệu gốc lẫn dữ liệu mới. Điều đó khẳng định tính đa dạng ứng dụng của bài toán *Airline_Sentiment*, đồng thời cũng khẳng định, độ hiệu quả của các mô hình học sâu *(Deep learning)* và *Transformer BERT* trong bài toán xử lý ngôn ngữ tự nhiên.

6 Kết luận

Bài báo cáo đã đạt được những mục tiêu đề ra

Bài báo cáo đã trình bày được các vấn đề sau sau:

1. Mô tả bài toán Phân tích cảm xúc, bao gồm đầu vào, đầu ra, yêu cầu xử lý.
2. Tiền xử lý dữ liệu đầu vào.
3. Trình bày về ứng dụng của các model trong việc giải bài toán Phân tích cảm xúc, với hai mô hình là LSTM và BERT.
4. Kiểm thử chương trình với mô hình BERT.
5. Kiểm thử chương trình với mô hình LSTM.
6. So sánh mô hình BERT và LSTM cho bài toán Airline Sentiment.

Kỹ năng đạt được

1. Nâng cao khả năng tìm kiếm, đọc, dịch tài liệu chuyên ngành liên quan đến nội dung bài tập lớn.
2. Biết tổng hợp các kiến thức đã học và kiến thức trong tài liệu tham khảo để viết báo cáo bài tập lớn.
3. Chế bản báo cáo bằng LaTeX.
4. Biết tóm tắt nội dung đã tìm hiểu được và trình bày dưới dạng một báo cáo khoa học.

Khả năng ứng dụng của kết quả nghiên cứu trong tương lai

Hiện nay, bài toán NLP đang được tiếp cận rất nhiều bởi các mô hình học máy, đặc biệt là các mô hình học sâu chính vì vậy mà các mô hình này sẽ còn được áp dụng cho nhiều bài toán hơn nữa. Có thể kể đến như:

1. **Tăng cường hiệu quả và độ chính xác của mô hình**

(a) **Hợp nhất mô hình**: Kết hợp sức mạnh của LSTM và BERT có thể tạo ra một mô hình mạnh mẽ hơn cho phân tích cảm xúc. LSTM có thể giúp trong việc xử lý dữ liệu tuần tự trong khi BERT cung cấp khả năng hiểu ngữ cảnh hai chiều sâu sắc hơn.

(b) **Tối ưu hóa cấu trúc mô hình**: Cải tiến cấu trúc hiện tại của LSTM và BERT, chẳng hạn như sử dụng các phiên bản tiên tiến như LSTM-CRF hoặc BERT với các tầng chuyên biệt cho từng nhiệm vụ cụ thể.

2. **Ứng dụng vào ngữ cảnh đa ngôn ngữ và đa văn hóa**

(a) **Phân tích cảm xúc đa ngôn ngữ**: Phát triển các mô hình có khả năng hiểu và phân tích cảm xúc trong nhiều ngôn ngữ khác nhau. Điều này đòi hỏi một sự kết hợp giữa LSTM và BERT để tận dụng khả năng xử lý ngôn ngữ tuần tự và ngữ cảnh sâu sắc.

(b) **Phân tích cảm xúc đa văn hóa**: Cảm xúc có thể được biểu đạt và hiểu khác nhau trong các nền văn hóa khác nhau. Phát triển các mô hình có khả năng điều chỉnh theo ngữ cảnh văn hóa sẽ làm tăng độ chính xác của phân tích cảm xúc.

3. **Tối ưu hóa cho thời gian thực và hiệu quả tính toán**

(a) **Tăng tốc độ xử lý**: Tối ưu hóa các mô hình LSTM và BERT để giảm thiểu thời gian xử lý và tăng hiệu suất làm việc trong các ứng dụng thời gian thực như chatbot hay hệ thống tư vấn khách hàng.

(b) **Giảm tài nguyên tính toán**: Phát triển các mô hình nhẹ hơn nhưng vẫn đảm bảo độ chính xác cao để có thể triển khai trên các thiết bị có tài nguyên hạn chế như điện thoại di động.

4. **Tích hợp với các dữ liệu không phải văn bản**

(a) **Phân tích cảm xúc đa phương tiện**: Kết hợp dữ liệu văn bản với dữ liệu hình ảnh và âm thanh để phân tích cảm xúc toàn diện hơn. Ví dụ, phân tích biểu cảm khuôn mặt hoặc giọng nói cùng với văn bản để đánh giá cảm xúc chính xác hơn.

(b) **Phân tích cảm xúc trong dữ liệu mạng xã hội**: Tích hợp dữ liệu từ các nền tảng mạng xã hội, bao gồm văn bản, hình ảnh và video, để phân tích cảm xúc của người dùng trong một bối cảnh rộng hơn.

5. **Học tập liên tục và thích ứng**

(a) **Mô hình học tập liên tục**: Phát triển các mô hình có khả năng học tập liên tục từ dữ liệu mới, cập nhật để thích ứng với sự thay đổi trong cách diễn đạt cảm xúc của người dùng theo thời gian.

(b) **Học thích ứng**: Phát triển các mô hình có khả năng tự điều chỉnh dựa trên phản hồi thực tế và dữ liệu mới để cải thiện độ chính xác và hiệu quả của phân tích cảm xúc.

7 Tài liệu tham khảo

[1] Aftan, S., & Shah, H. (2023, January). A Survey on BERT and Its Applications. In 2023 20th Learning and Technology Conference (L&T) (pp. 161-166). IEEE.

[2] A.Pro N.Sriram. (2019 April 2019). *"Us Airlines sentiment annalysis using LSTM"*, Volume 6, Issue 4, ISSN-2349-5162.

[3] B.Ramya & A.Jothi II M.SC CS1, Assistant Professor, Department of Computer Science1,2, K.R. College of Arts and Science1,2.(2023)."Sentiment Analysis using deep learning". Volume 10, Issue 3.

[4] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[5] [Ebook] Ian Goodfellow, Yoshua Bengio, Aaron Courville. (2016). *"Deep-Learning-The-MIT-Press"*.

[6] Hasib, K. M., Naseem, U., Keya, A. J., Maitra, S., Mithu, K., & Alam, M. G. R. (2023). A systematic review on airlines industries based on sentiment analysis and topic modeling.

[7] Kingma, D. P., & Ba, J. (2014). *"Adam: A method for stochastic optimization"*. arXiv preprint arXiv:1412.6980.

[8] Li, Z., Yang, C., & Huang, C. (2023). A Comparative Sentiment Analysis of Airline Customer Reviews Using Bidirectional Encoder Representations from Transformers (BERT) and Its Variants. Mathematics, 12(1), 53.

[9] Manchikanti, K., & Madhurika, B. (2020). Airline tweets sentiment analysis using RNN and LSTM techniques. International Journal of Advanced Trends in Computer Science and Engineering, 9(5), 8197-8201.

[10] Samir, H. A., Abd-Elmegid, L.,& Marie, M. (2023). *"Sentiment analysis model for Airline customers' feedback using deep learning techniques"*. International Journal of Engineering Business Management, 15, 18479790231206019.

[11] Ullah, M. A., Marium, S. M., Begum, S. A., & Dipa, N. S. (2020). An algorithm and method for sentiment analysis using the text and emoticon. ICT Express, 6(4), 357-360.