Đại học Quốc gia Hà Nội
Đại học Khoa học tự nhiên

Máy tính và Khoa học thông tin (CLC)

Báo cáo cuối kỳ môn Lập trình nâng cao

PHÂN LOẠI BÌNH LUẬN TỪ CÁC TRANG THƯƠNG MẠI ĐIỆN TỬ

Sinh viên:
Phạm Đức Anh - 2001527
Trần Văn Hanh - 20001542
Phan Vũ Nguyên Hoàng - 20001552

Hà Nội, Ngày 17 Tháng 12 Năm 2022

Mục lục

1 Đặt vấn đề

Ngày nay, doanh nghiệp đặc biệt quan tâm đến sự hài lòng và thỏa mãn của khách hàng. Đây là yếu tố liên quan đến lợi ích của doanh nghiệp như thị phần, doanh thu và lợi nhuận. Đồng thời, sự hài lòng đồng nghĩa với việc khách hàng sẽ trở lại mua hàng nhiều hơn và giới thiệu thương hiệu của bạn đến nhiều người khác hơn. Cách tốt nhất để doanh nghiệp đáp ứng được mong muốn của khách hàng là lắng nghe, ghi nhận những ý kiến, khiếu nại từ họ.

Hằng ngày ở các trang thương mại điện tử có rất nhiều comment về sản phẩm được người dùng đưa ra đánh giá về những sản phẩm đó. Việc phân tích thống kê lại xem những bình luận đó là tích cực hay tiêu cực sẽ giúp cho doanh nghiệp biết được chất lượng sản phẩm, tâm lý khách hàng và từ đó đưa ra những thay đổi hợp lý trong kinh doanh. Vì có rất nhiều lượt bình luận, trên các trang thương mại điện tử lớn có thể lên tới hàng chục triệu lượt bình luận trong 1 ngày nên việc phân tích bằng tay truyền thống là điều không thể. Đây là lúc các mô hình học máy thể hiện sức mạnh của mình.

Vì vậy chúng em đã tìm hiểu và lựa chọn đề tài **"Phân tích bình luận từ các trang thương mại điện"** tử để làm tiểu luận kết thúc môn học.

2 Các lý thuyết và mô hình được sử dụng

2.1 Convolutional Neural Network

2.1.1 Khái niệm

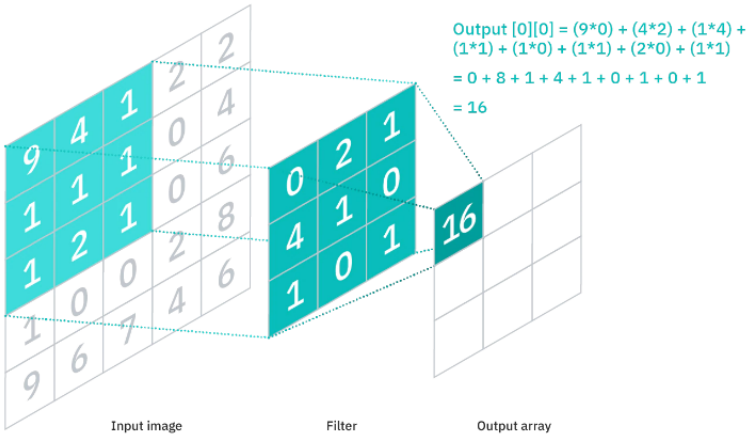
Trong các mạng thần kinh, mô-đun Convolutional Neural Network (CNN) hay được gọi mạng thần kinh tích chập là một trong những mô-đun nhận dạng và phân loại hình ảnh. Trong đó, nhận dạng đối tượng và nhận dạng khuôn mặt là một trong những lĩnh vực CNN được ứng dụng rộng rãi.

Convolutional Neural Network là một loại mạng thần kinh nhân tạo chuyên dụng sử dụng một phép toán gọi là phép tích chập thay cho phép nhân ma trận chung trong ít nhất một trong các lớp của chúng. Chúng được thiết kế đặc biệt để xử lý dữ liệu pixel và được sử dụng trong quá trình nhận dạng và xử lý hình ảnh. CNN có các lớp ẩn được gọi là lớp tích chập (convolutional layers) và các lớp này là thứ tạo nên một CNN.

2.1.2 Lớp tích chập - Convolution Layer

Giống như bất kỳ lớp nào khác, lớp tích chập nhận đầu vào, biến đổi đầu vào theo một cách nào đó, sau đó xuất đầu vào đã chuyển đổi sang lớp tiếp theo.

Với lớp tích chập, phép biến đổi xảy ra được gọi là phép tích chập. Đây là thuật ngữ được sử dụng bởi cộng đồng deep learning. Về mặt toán học, các hoạt động tích chập được thực hiện bởi các lớp tích chập thực sự được gọi là cross-correlations(tương quan chéo).

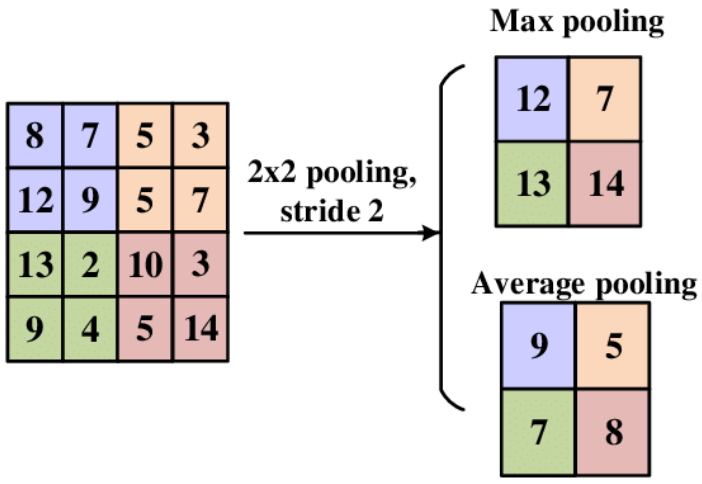


Hình 1: Các bước tích chập

2.1.3 Lớp gộp - Pooling Layer

Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau:

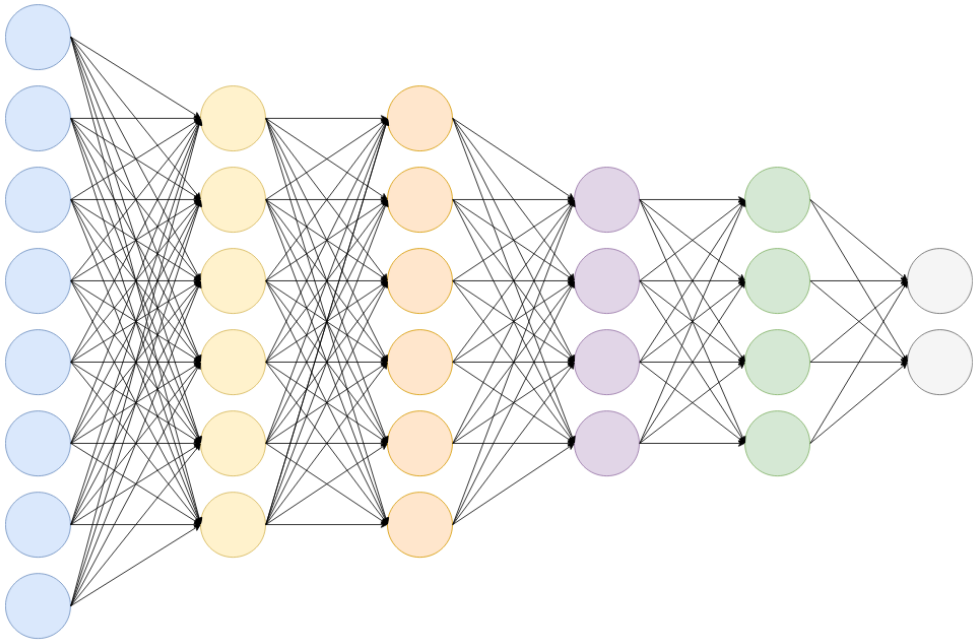
- Max Pooling
- Average Pooling
- Sum Pooling



Hình 2: Hai loại pooling: Max Pooling và Average Pooling

2.1.4 Lớp được kết nối đầy đủ - Fully connected layer

The fully connected layer (FC) hay lớp được kết nối đầy đủ (còn được gọi là dense layer) là một lớp trong đó mọi nơ-ron trong lớp được kết nối với mọi nơ-ron trong lớp trước đó. Điều này có nghĩa là đầu ra của mỗi nơ-ron trong lớp trước được sử dụng làm đầu vào cho từng nơ-ron trong lớp được kết nối đầy đủ.



Hình 3: Lớp được kết nối đầy đủ

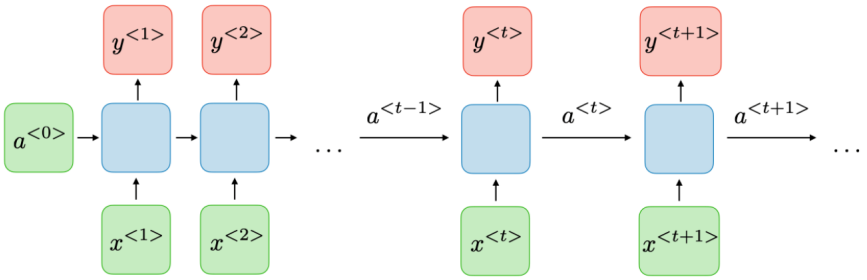
2.2 Recurrent Neural Network

2.2.1 Khái niệm

Mạng nơ-ron hồi quy (RNN) là một lớp mạng nơ-ron nhân tạo trong đó các kết nối giữa các nút có thể tạo ra một chu kỳ cho phép đầu ra từ một số nút ảnh hưởng đến đầu vào tiếp theo của cùng một nút. Bắt nguồn từ các mạng nơ-ron chuyển tiếp, RNN có thể sử dụng trạng thái bên trong của chúng (bộ nhớ) để xử lý các chuỗi đầu vào có độ dài thay đổi. Do đó, điều này giúp chúng có thể áp dụng cho các tác vụ như không phân đoạn, tức là xử lý dữ liệu theo trình tự, như văn bản hoặc lời nói, video,...

2.2.2 Mô hình

RNN cho phép đầu ra được sử dụng như đầu vào trong khi có các trạng thái ẩn. Thông thường là như sau:

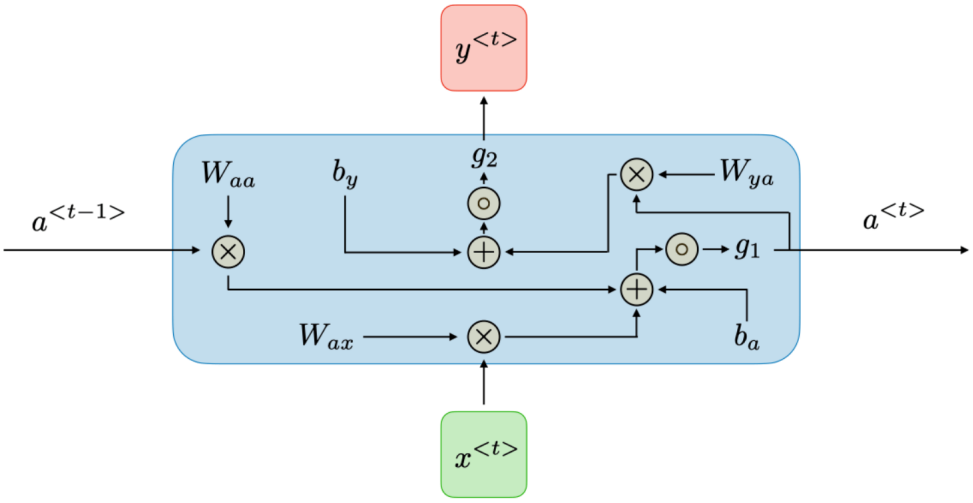


Hình 4: Kiến trúc của một mạng RNN truyền thống

- Tại mỗi bước t, giá trị kích hoạt $a^{<t>}$ và đầu ra $y^{<t>}$ được biểu diễn như sau:

  $a^{<t>} = g_1(W_{aa}a^{<t>} + W_{ax}x^{<t>} + b_a)$ và $y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$

- Với $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ là các hệ số được chia sẻ tạm thời và g_1, g_2 là các hàm kích hoạt.



- Các mô hình RNN sử dụng các hàm kích hoạt như tanh hoặc sigmoid có thể dẫn đến độ dốc mất mát (gradients). Bởi vì hai hàm trên biến một không gian đầu vào lớn thành một không gian đầu vào nhỏ như [0, 1] đối với hàm sigmoid và [-1, 1] đối với hàm tanh và điều đặc biệt hàm tanh lẫn sigmoid đều có đạo hàm bằng 0 tại 2 đầu. Do đó, một thay đổi lớn trong đầu vào của hàm sẽ gây ra một thay đổi nhỏ trong đầu ra.

- Đạo hàm bị triệt tiêu (Vanishing gradient)

  Ở trên, có một vấn đề là, hàm tanh lẫn sigmoid đều có đạo hàm bằng 0 tại 2 đầu. Mà khi đạo hàm bằng 0 thì nút mạng tương ứng tại đó sẽ bị bão hòa. Lúc đó các nút phía trước cũng sẽ bị bão hoà theo. Nên với các giá trị nhỏ trong ma trận, khi ta thực hiện phép nhân ma trận sẽ đạo hàm tương ứng sẽ xảy ra Vanishing gradient, tức đạo hàm bị triệt tiêu chỉ sau vài bước nhân. Như vậy, các bước ở xa sẽ không còn tác dụng với nút hiện tại nữa, làm cho RNN không thể học được các phụ thuộc xa.

- Để xử lý Vanishing Gradient, Ta cần thiết kế một kiến trúc có thể nhớ dài hạn hơn, đó là LSTM.

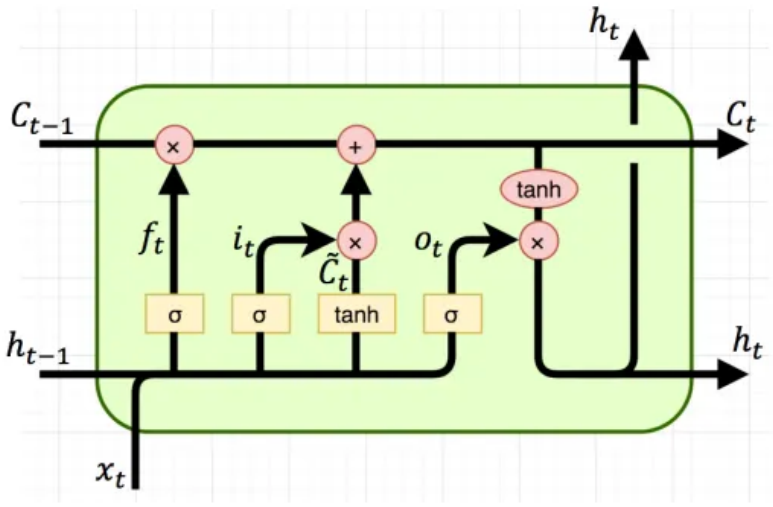
2.3 Long Short-Term Memory

2.3.1 Khái niệm

Long Short-Term Memory (LSTM) là một loại RNN có khả năng học hỏi và dự đoán dữ liệu tuần tự mạnh mẽ. Nghiên cứu cho thấy RNN bị hạn chế trong việc duy trì trí nhớ dài hạn. Do đó, LSTM đã được phát minh để khắc phục hạn chế này bằng cách bổ sung cấu trúc bộ nhớ, có thể duy trì trạng thái của nó theo thời gian, với các cổng để quyết định những gì cần nhớ, những gì cần quên và những gì cần xuất. LSTM cho thấy kết quả hiệu quả trong nhiều ứng dụng vốn có trình tự như nhận dạng giọng nói, tổng hợp giọng nói, mô hình hóa ngôn ngữ, dịch thuật và nhận dang chữ viết tay.

2.3.2 Mô hình

Dưới đây là tổng hợp các phương trình đặc trưng của kiến trúc LTSM:



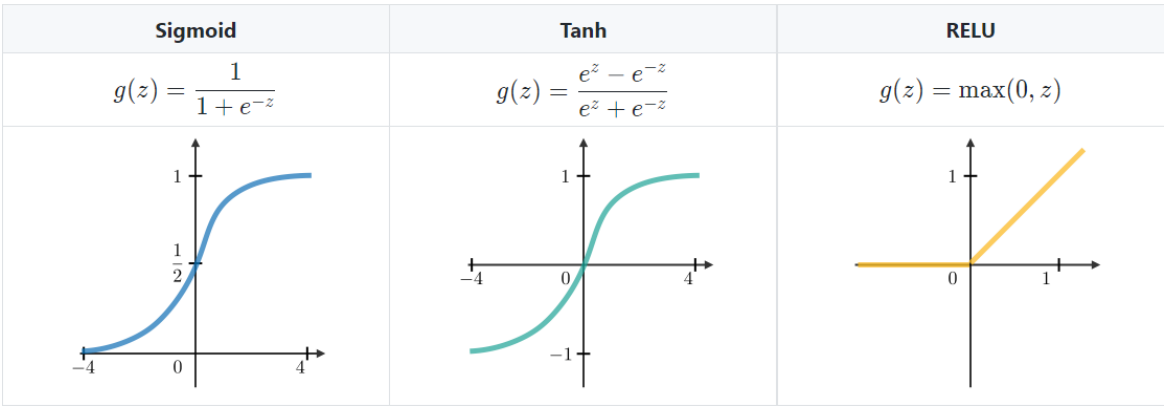
Hình 5: LSTM cells

- Mạng LSTM được tạo thành từ nhiều ô nhớ (cells) LSTM được liên kết hoạt động tốt về hiệu quả học tập. Nội dung của ô nhớ được điều chỉnh bởi cổng vào và cổng quên. Giả sử rằng cả hai cổng này đều đóng, nội dung của ô nhớ sẽ không thay đổi qua các bước thời gian. Cấu trúc gating cho phép thông tin được lưu giữ qua nhiều bước thời gian, cho phép độ dốc (gradients) cũng chảy qua nhiều bước thời gian. Điều này cho phép mô hình LSTM tránh được vấn đề vanishing gradient gây khó chịu cho hầu hết các mô hình RNN.
- Giống với RNN, mô hình sử dụng các hàm kích hoạt như tanh và sigmoid để tính toán. Chúng em sẽ mô tả hàm kích hoạt rõ hơn ở phần 2.3.3.

2.4 Hàm kích hoạt - Activation Function

Trong mạng thần kinh nhân tạo, hàm kích hoạt là hàm ánh xạ đầu vào của một nút với đầu ra tương ứng của nó. Các hàm kích hoạt phổ biến nhất được sử dụng trong các mạng thần kinh nhân tạo là Sigmoid, Tanh, ReLU.

Các chức năng kích hoạt phổ biến nhất được sử dụng trong các mô-đun RNN được mô tả bên dưới:



Hình 6: Các hàm kích hoạt

- Hàm sigmoid nhận đầu vào và thực hiện như sau: Đối với hầu hết các đầu vào âm, sigmoid sẽ biến đổi đầu vào thành một số rất gần với 0. Đối với hầu hết các đầu vào dương, sigmoid sẽ chuyển đổi đầu vào thành một số rất gần với 1. Đối với các đầu vào tương đối gần với 0, sigmoid sẽ chuyển đổi đầu vào thành một số giữa 0 và 1.
- Hàm tanh được sử dụng nhiều trong các mạng neural vì nó có khả năng tạo ra các giá trị đầu ra trong khoảng từ -1 đến 1, và có khả năng phân chia tốt các giá trị đầu vào khác nhau. Về mặt lịch sử, hàm tanh trở nên được ưa chuộng hơn hàm sigmoid vì nó mang lại hiệu suất tốt hơn cho các mạng thần kinh nhiều lớp. Nhưng nó không giải quyết được vấn đề biến mất độ dốc mà sigmoids gặp phải, vấn đề này đã được giải quyết hiệu quả hơn với việc giới thiệu kích hoạt ReLU.

- Rectified Linear Units, hay ReLUs, là một loại hàm kích hoạt tuyến tính theo chiều dương, nhưng bằng 0 trong chiều âm. Đường gấp khúc trong hàm là nguồn gốc của tính phi tuyến. Tính tuyến tính trong chiều dương có đặc tính hấp dẫn là nó ngăn chặn sự không bão hòa của độ dốc (ngược lại với kích hoạt sigmoid), mặc dù đối với một nửa đường thực, độ dốc của nó bằng không.

2.5 Word Embeddings

Embedding là một kỹ thuật đưa một vector có số chiều lớn, thường ở dạng thưa, về một vector có số chiều nhỏ, thường ở dạng dày đặc. Phương pháp này đặc biệt hữu ích với những đặc trưng hạng mục có số phần tử lớn ở đó phương pháp chủ yếu để biểu diễn mỗi giá trị thường là một vector dạng one-hot. Một cách lý tưởng, các giá trị có ý nghĩa tương tự nhau nằm gần nhau trong không gian embedding.

Ví dụ nổi bật nhất là biểu diễn các từ trong một bộ từ điển dưới dạng số. Khi từ điển có hàng triệu từ, biểu diễn các từ dưới dạng one-hot vector dẫn tới số chiều vô cùng lớn. Hơn nữa, các từ này sẽ có khoảng cách đều nhau tới mọi từ khác (căn bậc hai của 2), dẫn đến việc thiếu thông tin giá trị cho việc huấn luyện mô hình machine learning. Chẳng hạn, một cách biểu diễn tốt các từ tiếng Việt cần mô tả tốt sự liên quan giữa cặp từ (vua, hoàng hậu) và (chồng, vợ) vì chúng có ý nghĩa gần nhau.

2.6 Dropout

Ý tưởng chung đằng sau việc bỏ học là nếu thêm nó vào một mô hình, nó sẽ ngẫu nhiên bỏ qua một số tập hợp con các nút trong một lớp nhất định trong quá trình đào tạo, tức là nó loại bỏ các nút khỏi lớp. Điều này sẽ ngăn các nút bị loại bỏ này tham gia đưa ra dự đoán về dữ liệu hay là tránh "học tủ" (Overfitting) vì một lớp fully connected có quá nhiều tham số và chiếm hầu hết tham số, các nút mạng trong lớp đó quá phụ thuộc lẫn nhau trong quá trình huấn luyện thì sẽ hạn chế sức mạnh của mỗi nút, dẫn đến việc kết hợp quá mức. Kỹ thuật này giúp mô hình khái quát hóa dữ liệu tốt hơn mà nó chưa từng thấy trước đây.

3 Phương pháp

3.1 Xử lý văn bản - Text processing

Dữ liệu thường đến từ nhiều nguồn khác nhau và thường ở các định dạng khác nhau. Vì lý do này, chuyển đổi văn bản là điều cần thiết. Tuy nhiên, quá trình chuyển đổi này không hề đơn giản vì dữ liệu văn bản của chúng ta thường chứa các từ thừa và lặp. Điều này có nghĩa là dữ liệu tiền xử lý là bước đầu tiên trong giải pháp của chúng em. Các bước thường được sử dụng để tiền xử lý dữ liệu văn bản là:

- Làm sạch dữ liệu thô
- Token hóa dữ liệu đã làm sạch

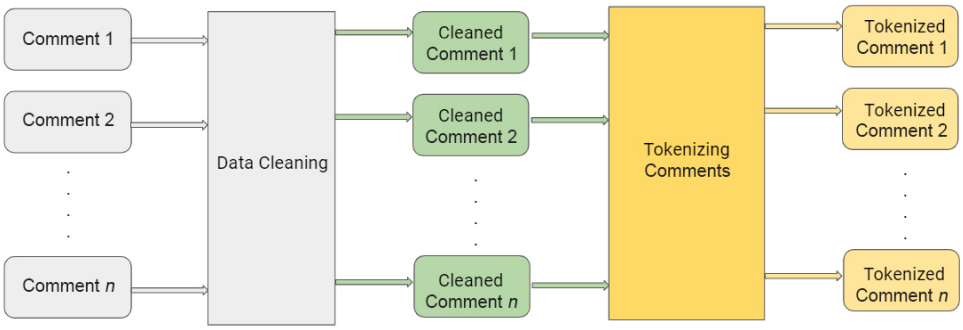
3.1.1 Làm sạch dữ liệu thô

Trong tiểu mục này, chúng ta phải xóa các từ hoặc ký tự không làm tăng ngữ nghĩa của văn bản. Đây là một số bước làm sạch tiêu chuẩn mà chúng em đã sử dụng trong đoạn này:

- Xóa văn bản trùng lặp
- Xóa liên kết trong văn bản
- Xóa dấu chấm câu
- Bỏ khoảng trắng ở đầu và đuôi văn bản
- Thay thế ký tự xuống dòng trong văn bản
- Viết thường hóa văn bản (lowering case)
- Loại bỏ các ký tự đặc biệt

3.1.2 Token hóa dữ liệu đã làm sạch

Mã thông báo là quá trình chia văn bản thành các phần nhỏ hơn, được gọi là mã thông báo. Mỗi mã thông báo là một đầu vào cho thuật toán học máy dưới dạng một tính năng.

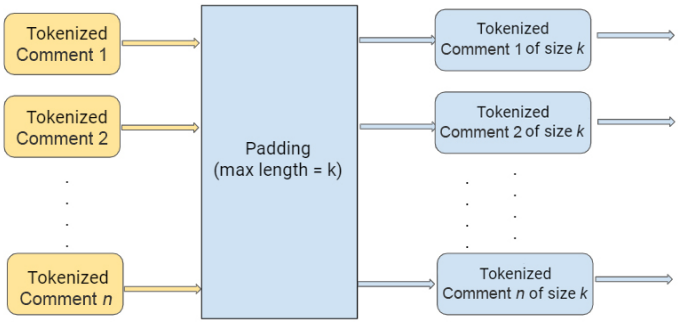


Hình 7: Các giai đoạn làm sạch và mã hóa dữ liệu của quá trình xử lý văn bản

3.2 Text Sequencing

3.2.1 Padding

Tạo mã thông báo cho tất cả các bình luận có kích thước bằng nhau được gọi là padding. Chúng em gửi thông tin đầu vào theo lô (batches) dữ liệu. Thông tin có thể bị mất khi đầu vào có kích thước khác nhau. Vì vậy, chúng em làm cho chúng có cùng kích thước bằng cách sử dụng phần đệm và điều đó giúp giảm bớt các bản cập nhật hàng loạt. Độ dài của tất cả các bình luận được mã hóa sau phần đệm được đặt bằng cách sử dụng 'max_len'.



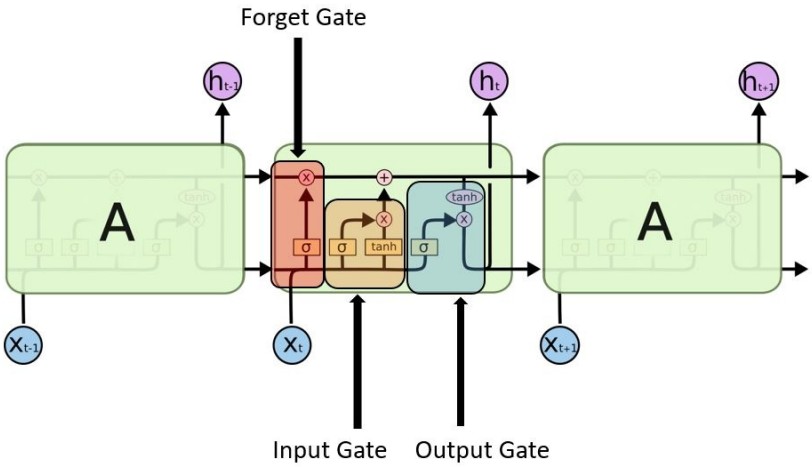
Hình 8: Tất cả các bình luận được mã hóa đều được chuyển đổi thành cùng một kích thước trong giai đoạn padding.

3.2.2 Gắn nhãn biến mục tiêu mã hóa

Mô hình sẽ mong đợi biến mục tiêu là một số chứ không phải một chuỗi. Khi chúng tôi sử dụng chuỗi để gắn nhãn dữ liệu của mình, nó sẽ kém linh hoạt hơn so với việc gắn nhãn theo số.

3.3 Lựa chọn mô hình

RNN có một mô-đun lặp lại lấy đầu vào từ giai đoạn trước và cung cấp đầu ra của nó làm đầu vào cho giai đoạn tiếp theo. Tuy nhiên, trong RNNs, chúng ta chỉ có thể giữ lại thông tin từ giai đoạn gần đây nhất. Để tìm hiểu các phụ thuộc dài hạn, mạng của chúng ta cần khả năng ghi nhớ. Đó là lý do chúng ta chọn Mạng bộ nhớ dài hạn ngắn hạn (LSTMs).



Hình 9: Hình minh họa mô hình LSTM

3.4 Thuật toán tối ưu hóa - Optimization Algorithm

Adaptive Moment Estimation là một thuật toán cho kỹ thuật tối ưu hóa để giảm dần độ dốc. Phương pháp này thực sự hiệu quả khi làm việc với bài toán lớn liên quan đến nhiều dữ liệu hoặc tham số. Nó đòi hỏi ít bộ nhớ hơn và hiệu quả. Theo trực giác, nó là sự kết hợp của thuật toán 'gradient descent with momentum' và thuật toán 'RMSP'.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

Hình 10: Adam Optimizer

3.5 Hàm mất mát - Loss Function

Mất mát là lỗi hoặc sự khác biệt giữa những gì mạng dự đoán so với nhãn thực và trình tối ưu sẽ cố gắng giảm thiểu lỗi này để làm cho mô hình của chúng tôi chính xác nhất có thể trong các dự đoán của nó. Chúng ta sẽ sử dụng hàm mất mát binary cross-entropy bởi vì vấn đề của chúng ta là bộ phân loại nhị phân.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i) \cdot log(1-p(y_i))$$

Hình 11: Binary Cross-Entropy

4 Triển khai, đánh giá và thực nghiệm

4.1 Bộ dữ liệu

Vì trên mạng khó tìm thấy bộ dữ liệu phù hợp với bài toán, nên chúng em sẽ tự tạo một bộ dữ liệu của riêng mình bằng cách lấy các bình luận, đánh giá của đa dạng các loại sản phẩm từ thời trang, mỹ phẩm, đồ dùng công nghệ đến các vật dụng hàng ngày như giày, dép được rao bán trên shopee và tự gán nhãn bằng tay, với tích cực được đánh là 0 và tiêu cực được đánh nhãn là 1.

Bộ dữ liệu gồm có 618 đánh giá chủ quan của khách hàng, trong đó bao gồm 307 đánh giá mang tính tiêu cực và 311 đánh giá mang tính tích cực. Chúng ta chia bộ dữ liệu gốc ra làm hai phần: bộ dữ liệu huấn luyện chiếm 80% và bộ dữ liệu kiểm tra chiếm 20%.

	label	text
0	1	Hàng giao nhanh, bàn phím nặng, gói hàng cẩn t...
1	1	Led đẹp bấm phím rất thích với giá tiền này th...
2	0	Ủng hộ member hội macbook Việt, nghĩ là mua tr...
3	0	Giá đỡ gập khá khó khăn. Không đẹp như hình. N...
4	1	Khá là ưng sản phẩm, đúng với mô tả và đi ok d...
...	...	...
613	0	Sản phẩm okie, nhưng có chút k vừa ý khâu trao...
614	0	Máy xài dc 6 ngày thì bị đốm rồi, đang chờ đổi...
615	0	Sao nhanh nóng thế
616	0	Tạm được. Nhưng khó kết nối với k20p. Chất lượ...
617	0	Sản phẩm ổn, cũng được nhưng chưa đạt yêu cầu ...

4.2 Triển khai và đánh giá mô hình

Ta thu thập dữ liệu và chuyển đổi sang dạng file csv. Chúng ta sử dụng thư viện pandas để đọc dữ liệu từ file dữ liệu csv này.

```
df = pd.read_csv('comments.csv')
```

Tiếp theo ta loại bỏ các dòng trống và các dữ liệu bị trùng lặp:

```
df = df.dropna()
df = df.drop_duplicates()
```

Sau đó ta dùng hàm *train_test_split* trong thư viện *sklearn.model_selection* để chia dữ liệu ra làm hai bộ train và test:

```
comments_train, comments_test, label_train, label_test
= train_test_split(df['text'], df['label'], test_size=0.2)
```

Ta sử dụng biểu thức chính quy để làm sạch dữ liệu thô đã thu thập được:

```
#Xoa link
def remove_hyperlink(text):
    return re.sub(r"http\S+","",text)

#Chuyen ve chu cai viet thuong
def to_lower(text):
    return text.lower()
```

```
#Loai bo cac chu so
def remove_number(text):
    return re.sub(r'\d+',"", text)

#Loai bo dau cham cau:
def remove_punctuation(text):
    text = re.sub(r'[^\w\s]', ' ', text)
    return text

#Loai bo khoang trang hai ben van ban
def remove_whitespace(text):
    text = re.sub(r'\s+',' ', text)
    text.strip()
    return text

#Loai bo dau xuong dong
def replace_newline(text):
    return text.replace('\n',' ')

#Loai bo emoji
def remove_emoji(text):
    emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols and pictographs
        u"\U0001F680-\U0001F6FF"  # transport and map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r' ', text)

#loai bo cac ky tu khong co y nghia
def remove_unmeaning_word(text):
    text = re.sub(r'=[)]+', '', text) #Loai bo ky tu =)))
    text = re.sub(r':[D]+','', text) #loai bo ky tu :D :DDDDD
    return text

#Thay the cac tu viet tat trong tieng Viet
def replace_acronym(text):
    text = re.sub(r'\s+k\s+|\s+K\s+|^k\s+|^K\s+|\s+k$|\s+K$|\s+ko\s+|\s+Ko\s+|^ko\s+|^K
    text = re.sub(r'\s+đc\s+|\s+Đc\s+|^đc\s+|^Đc\s+|\s+đc$|\s+Đc$', ' được ', text)
    text = re.sub(r'\s+nhg\s+|\s+Nhg\s+|^nhg\s+|^Nhg\s+|\s+nhg$|\s+Nhg$',' nhưng ', tex
    text = re.sub(r'\s+mk\s+|\s+Mk\s+|^mk\s+|^Mk\s+|\s+mk$|\s+Mk$',' mình ',text)
    return text
```

```
#Tong hop cac ham lai de lam sach du lieu
def clean_up_pipeline(sentence):
    cleaning_utils = [remove_hyperlink,
                      to_lower,
                      remove_number,
                      remove_punctuation,
                     remove_emoji,
                      replace_acronym,
                     remove_whitespace,
                     replace_newline]
    for o in cleaning_utils:
        sentence = o(sentence)
    return sentence

x_train = [clean_up_pipeline(o) for o in comments_train]
x_test = [clean_up_pipeline(o) for o in comments_test]
```

Để tránh sai sót trong lúc gán nhãn (ví dụ như có chữ,..) ta sử dụng lớp *LabelEncoder* ở thư viện *sklearn.preprocessing*:

```
le = LabelEncoder()
y_train = le.fit_transform(label_train.values)
y_test = le.transform(label_test.values)
```

Sau khi xử lý xong dữ liệu thô, chúng ta tiến hành tokenizing dữ liệu thành các vector số:

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train)

x_train_features = np.array(tokenizer.texts_to_sequences(x_train))
x_test_features = np.array(tokenizer.texts_to_sequences(x_test))

x_train_features = pad_sequences(x_train_features, maxlen=50)
x_test_features = pad_sequences(x_test_features, maxlen=50)
```

Ta xây dựng mô hình cho bài toán, sử dụng các model có trong thư viện tensorflow:

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding, Dropout
from tensorflow.keras.models import Model

def LMTS(input_length, input_dim, x_train, x_test, y_train, y_test):
lstm_model = Sequential()
#Creating an embedding layer to vectorize
```

```
    lstm_model.add(Embedding(input_dim=input_dim+1, output_dim=20, input_length=input_lengt
    #Addding LSTM
    lstm_model.add(LSTM(64))
    # Relu allows converging quickly and allows backpropagation
    lstm_model.add(Dense(16, activation='relu'))
    #Deep Learninng models can be overfit easily, to avoid this, we add randomization usi
    lstm_model.add(Dropout(0.1))
    # Adding sigmoid activation function to normalize the output
    lstm_model.add(Dense(1, activation='sigmoid'))

    lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    lstm_model.summary()
    history = lstm_model.fit(x_train, y_train, epochs=20, batch_size=32, shuffle = True,
                validation_data=(x_test, y_test))
    #Save the model
    lstm_model.save('lstm_model.h5')
    y_predict = [1 if o>0.5 else 0 for o in lstm_model.predict(x_test)]
    return history, y_predict
```

Ở mô hình này ta sử dụng các công thức đánh giá precision, recall và f1 score để đánh giá và show kết quả thử với bộ dữ liệu test trên confusion matrix:

```
    from sklearn.metrics import confusion_matrix,f1_score, precision_score,recall_score
    import seaborn as sns
    import matplotlib.pyplot as plt

    def evaluating(test_y, y_predict):
    cf_matrix =confusion_matrix(test_y,y_predict)
    print("Precision: {:.2f}%".format(100 * precision_score(test_y, y_predict)))
    print("Recall: {:.2f}%".format(100 * recall_score(test_y, y_predict)))
    print("F1 Score: {:.2f}%".format(100 * f1_score(test_y,y_predict)))
    ax= plt.subplot()
    #annot=True to annotate cells
    sns.heatmap(cf_matrix, annot=True, ax = ax,cmap='Blues',fmt='')
    # labels, title and ticks
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix')
    ax.xaxis.set_ticklabels(['Positive', 'Negative']);
    ax.yaxis.set_ticklabels(['Positive', 'Negative'])
    plt.show()
```

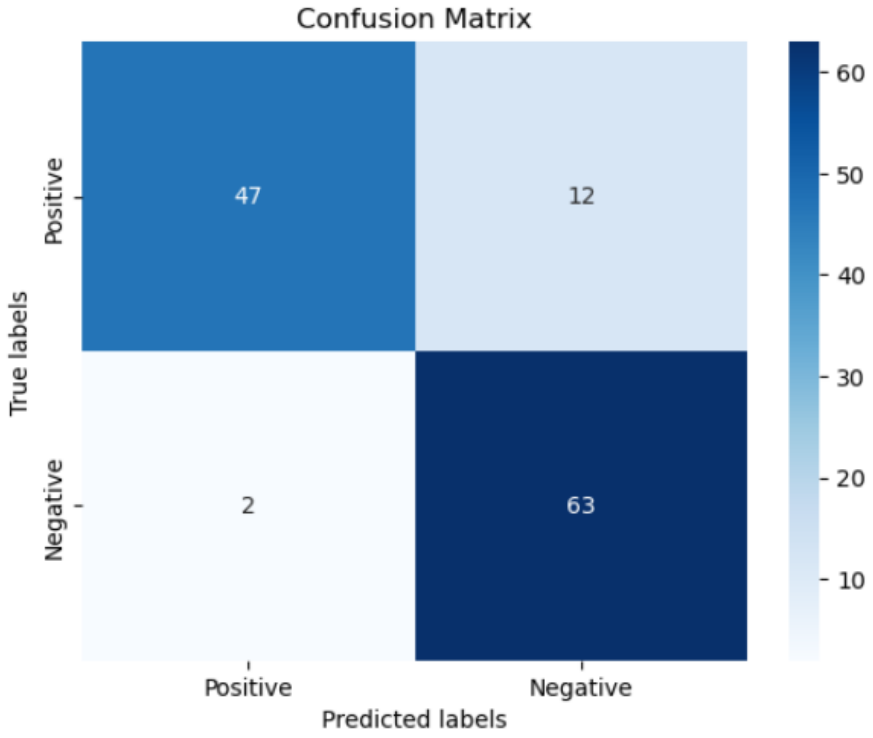
Cuối cùng ta triển khai mô hình với bộ dữ liệu huấn luyện và kiểm tra trên bộ dữ liệu test:

```
    lmts, y_predict = LMTS(50,1557, x_train_features, x_test_features, y_train, y_test)
    evaluating(y_test, y_predict)
```

4.3 Kết quả thực nghiệm

Kết quả thu được sau quá trình huấn luyện với epochs = 20 và batchsize = 32 như sau, và sau khi huấn luyện xong ta lưu mô hình dưới định dạng file h5 để tái sử dụng:

```
Precision: 84.00%
Recall: 96.92%
F1 Score: 90.00%
```



4.4 Tổng kết và hướng phát triển tiếp theo

Trong chủ đề này, chúng ta đã tạo một mô hình phân loại bình luận tích hay tiêu cực trên các trang thương mại điện tử bằng cách chuyển đổi các bình luận thành các vector, tạo một mô hình LSTM, và gắn mô hình với các vector. Chúng ta cũng sử dụng nhiều kỹ thuật xử lý văn bản, mô hình học sâu như RNN, LSTM.

Với mô hình và kỹ thuật chúng ta sử dụng có thể áp dụng trong nhiều lĩnh vực xử lý ngôn ngữ tự nhiên như xây dựng chatbot, mô hình dịch ngôn ngữ. Ngoài ra, có thể dụng để giải quyết một số chủ đề cụ thể như phân loại thư rác.