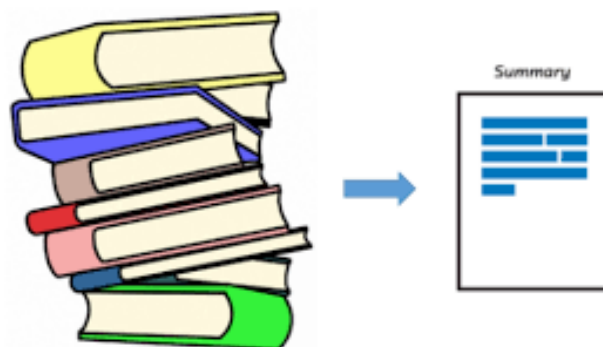




Đề Tài

Text Summarization sử dụng phương pháp Abstractive method



Giáo Viên Hướng Dẫn: Nguyễn Hồng Bửu Long

Sinh Viên Thực Hiện:

Mai Huy 43.01.104.079

Nguyễn Trung Hiếu 43.01.104.052

Nguyễn Thành Đạt 43.01.104.020

Lớp : Khai thác nội dung văn bản

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM HỒ CHÍ MINH**

Nội dung

I. Giới thiệu đề tài:	3
II. Hướng tiếp cận:	4
1. Abtractive summarization:	4
2. Long Short-Term Memory (LSTM):	5
3. Mô hình seq2seq:	9
3.1) Training phrase: (giai đoạn huấn luyện mô hình)	9
3.2) Inference phrase (giai đoạn dự đoán)	11
4. Attention mechanism:	12
III. Đánh giá kết quả trên ngữ liệu / Demo:	12
1) Thu thập Dataset (Amazon fined food reviews):	12
2) Tiền xử lý dữ liệu (Data preprocessing)	13
3) Tokenization:	15
4) Xây dựng mô hình:	17
5) Đánh giá mô hình (evaluation):	18
6) Tiến hành dự đoán	19
IV. Hướng phát triển:	21
V. Hướng dẫn sử dụng các file python đã được xây dựng cấu trúc thông qua môi trường Command Prompt:	21
File 1: File Processing_data.py	21
File 2: File split_data_tokenization.py	21
File 3: File visualize.py	21
File 4: Train_model.py	22
File 5: predict_model.py	23
VI. Tài liệu tham khảo:	24

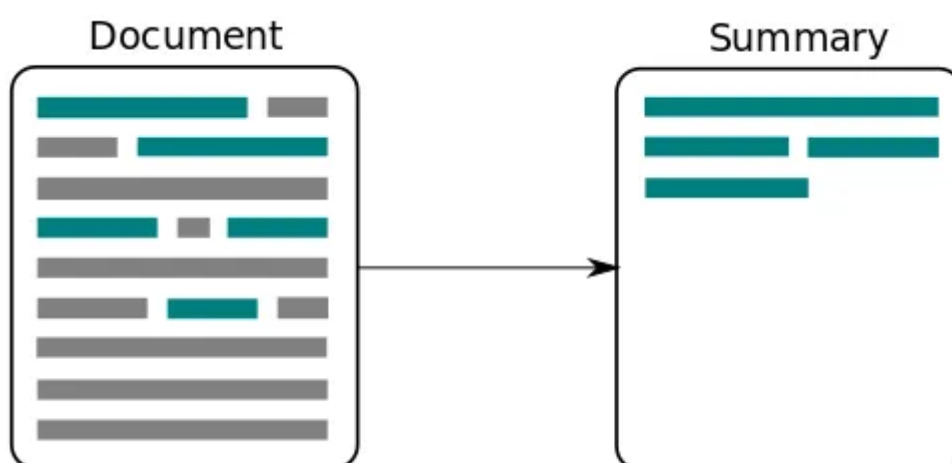
I. Giới thiệu đề tài:

Trong một thời đại mà mỗi ngày, mỗi giờ, mỗi phút đều có một lượng thông tin khổng lồ được sinh ra, nhưng giới hạn về thời gian, về khả năng đọc và tiếp thu của con người là có hạn, việc hiểu và nắm bắt thật nhiều thông tin một cách nhanh chóng không phải là vấn đề đơn giản với bất kỳ ai.

Đã bao giờ bạn tìm kiếm các kiến thức trên internet, hay đọc một cuốn sách mà nội dung của nó dài "lê thê", khiến cho bạn cảm thấy một chút khó khăn để có thể nắm bắt được nó chưa?

Đứng trước xu hướng con người ngày càng mất nhiều thời gian đọc email, báo điện tử và mạng xã hội, các thuật toán sử dụng machine learning để tự động tóm tắt các văn bản dài một cách gãy gọn và chính xác ngày càng trở nên cần thiết và có vai trò to lớn đối trong bất kỳ lĩnh vực nào.

Tự động tóm tắt sẽ là một trong những công nghệ quan trọng có thể giúp con người giảm thiểu thời gian đọc email và thông tin, kiến thức mới để dành thời gian cho các công việc khác, mà vẫn có thể nắm bắt được gãy gọn những nội dung của nó.

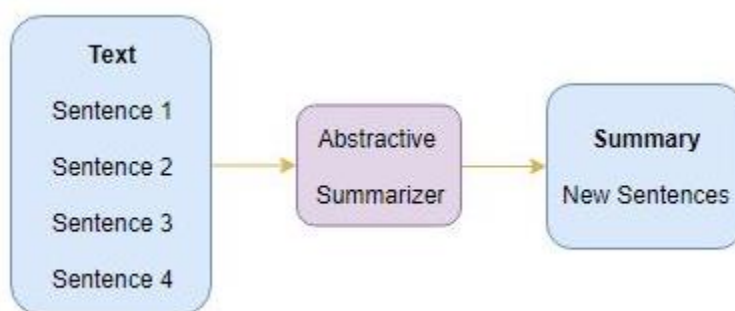


Tóm tắt văn bản là quá trình rút trích những thông tin quan trọng nhất từ một hoặc nhiều nguồn để tạo ra phiên bản cô đọng, ngắn gọn phục vụ cho một hoặc nhiều người dùng cụ thể, hay một hoặc nhiều nhiệm vụ cụ thể.

II. Hướng tiếp cận:

1. Abtractive summarization:

Tóm tắt trừu tượng là chúng ta sẽ tóm tắt thông tin từ văn bản lúc đầu thành một câu hoàn toàn mới. Phương pháp này hoàn toàn trái ngược với phương pháp trích xuất mà ta chỉ tóm tắt thông tin dựa trên các câu và từ có sẵn.



Ví dụ:

Source Text: Peter and Elizabeth took a taxi to attend the night party in the city.

While in the party, Elizabeth collapsed and was rushed to the hospital.

Summary: Elizabeth was hospitalized after attending a party with Peter. 

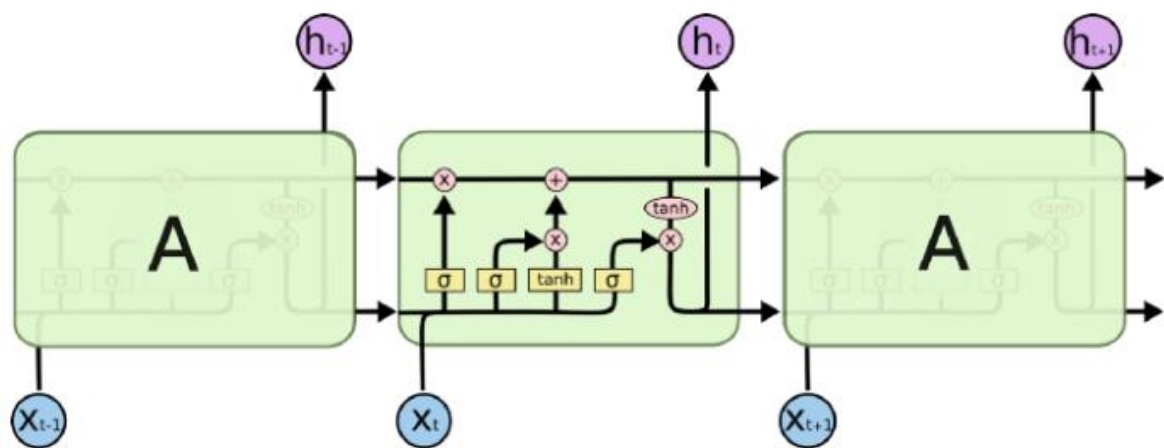
Ứng dụng tóm tắt trừu tượng quan sát toàn bộ các từ trong văn bản và nhớ các keyword- các từ mà ứng dụng đánh giá là quan trọng và nắm các ý chính trong

đoạn văn bản, như trong ví dụ trên, các từ mà ứng dụng đánh giá là quan trọng bao gồm các danh từ và động từ như: Peter, Elizabeth, attend, party, hospital.

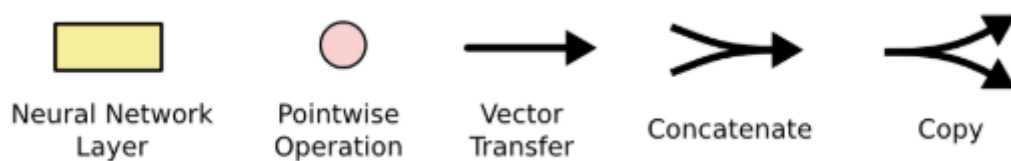
Từ các keyword này ứng dụng mới đoán được ý chính của văn bản và cho ra bản tóm tắt cuối cùng.

2. Long Short-Term Memory (LSTM):

Long Short-Term Memory (LSTM), một biến thể nổi tiếng của RNN, được đề xuất như là một giải pháp cho vấn đề vừa được nêu trên. Điểm chính trong kiến trúc mạng của LSTM chính là các **memory cell** với các cổng cho phép lưu trữ hoặc truy xuất thông tin. Các cổng này cho phép ghi đè (input gate), loại bỏ dư thừa (forget gate) và truy xuất (output gate) các thông tin được lưu trữ bên trong các memory cell.



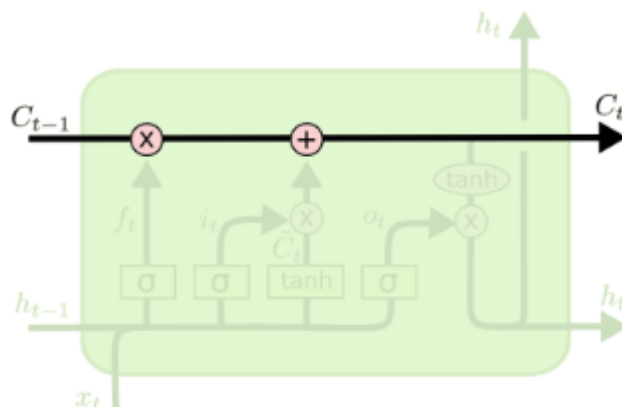
Hình 2: Long Short-Term Memory (nguồn: Colah's blog)



Từng Bước LSTM chạy qua:

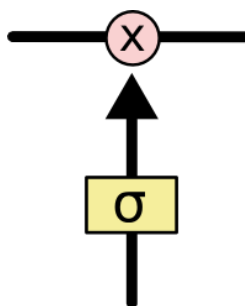
Chìa khóa của LSTM là trạng thái ô, đường ngang chạy qua đỉnh sơ đồ.

Trạng thái tế bào giống như một băng chuyền. Nó chạy thẳng xuống toàn bộ chuỗi, chỉ với một số tương tác tuyến tính nhỏ. Rất dễ dàng để thông tin chỉ chảy dọc theo nó



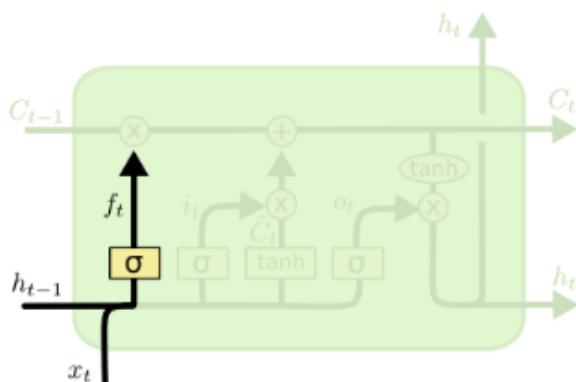
LSTM có khả năng loại bỏ hoặc thêm thông tin vào trạng thái tế bào, được điều chỉnh cẩn thận bởi các cấu trúc gọi là cổng vào

Gates là một cách để tùy ý để thông tin thông qua. Chúng được cấu tạo từ một lớp lưới thần kinh sigmoid và một phép toán nhân theo điểm.



Lớp sigmoid xuất ra các số từ 0 đến 1, mô tả mức độ của mỗi thành phần sẽ được cho qua. Giá trị bằng 0 có nghĩa là không để bất cứ thứ gì qua, trong khi giá trị của 1 nghĩa là có thể cho phép mọi thứ thông qua!

Một LSTM có ba trong số các cổng này, để bảo vệ và kiểm soát trạng thái tế bào.

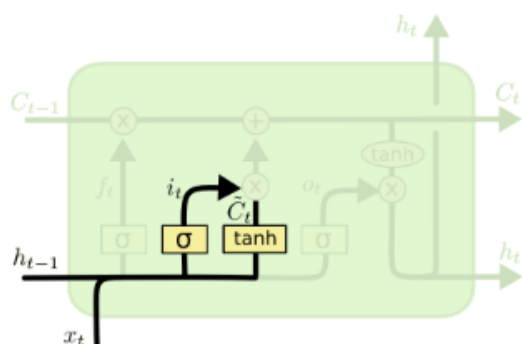


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Bước tiếp là quyết định thông tin mới nào chúng em sẽ lưu trữ trong trạng thái tế bào. Điều này có hai phần. Đầu tiên, một lớp sigmoid được gọi là lớp cổng đầu vào của Google, quyết định giá trị nào chúng ta sẽ cập nhật. Tiếp theo, một lớp

tanh tạo ra một vector các giá trị ứng cử viên mới \tilde{C}_t , có thể được thêm vào. Trong bước tiếp theo, chúng em sẽ kết hợp cả hai để tạo ra một bản cập nhật cho trạng thái.

Trong ví dụ về mô hình ngôn ngữ của chúng em, chúng em muốn thêm trạng thái của chủ thể mới vào trạng thái tế bào, để thay thế mô hình cũ mà nó đang quên

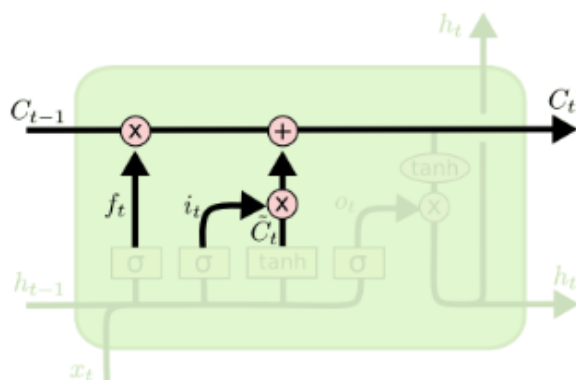


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Chúng em nhân lên trạng thái cũ bằng f_t , quên đi những điều chúng em quyết định quên trước đó. Sau đó, chúng em thêm $i_t * \tilde{C}_t$. Đây là giá trị ứng cử viên mới, được chia tỷ lệ theo mức độ chúng em quyết định cập nhật từng giá trị trạng thái.

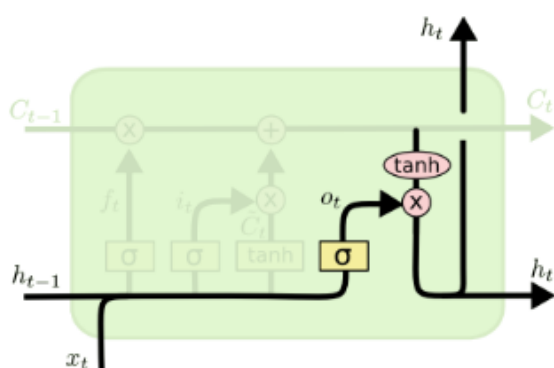
Trong trường hợp của mô hình ngôn ngữ, đây là nơi chúng em thực sự bỏ thông tin về giới tính của đối tượng cũ và thêm thông tin mới, như chúng em đã quyết định trong các bước trước.



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

Cuối cùng, chúng ta cần quyết định những gì chúng ta sẽ đầu ra. Đầu ra này sẽ dựa trên trạng thái ô của chúng em, nhưng sẽ là phiên bản được lọc. Đầu tiên, chúng em chạy một lớp sigmoid quyết định phần nào của trạng thái tế bào chúng ta sẽ xuất ra. Sau đó, chúng em đưa trạng thái tế bào thông qua tanh (để đẩy các giá trị ở giữa -1 và 1) và nhân nó với đầu ra của cổng sigmoid, do đó chúng em chỉ xuất ra các phần chúng em quyết định.

Đối với ví dụ về mô hình ngôn ngữ, vì nó chỉ nhìn thấy một chủ đề, nó có thể muốn xuất thông tin liên quan đến một động từ, trong trường hợp đó là những gì sắp diễn ra. Ví dụ, nó có thể xuất ra cho dù chủ ngữ là số ít hay số nhiều, để chúng ta biết dạng động từ nên được kết hợp thành gì nếu đó là những gì tiếp theo.



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

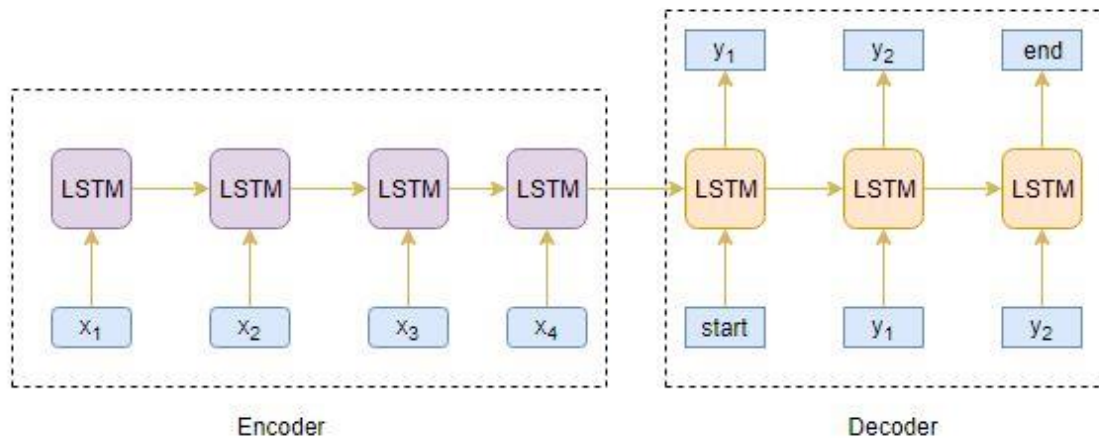
Việc nhận dạng chính xác tên riêng trong một đoạn văn bản phụ thuộc không chỉ vào các thông tin phía trước của từ đang xét mà còn cả các thông tin phía sau.

Tuy nhiên, một kiến trúc LSTM truyền thống với một lớp duy nhất chỉ có thể dự đoán nhãn của từ hiện tại dựa trên thông tin có được từ các từ nằm trước đó.

3. Mô hình seq2seq:

Đây là mô hình giúp giải quyết những vấn đề liên quan đến thông tin tuần tự, có một số ứng dụng rất phổ biến như : sentiment classification, machine translation, named entity recognition.

Mô hình bao gồm 2 thành phần chính là encoder và decoder:



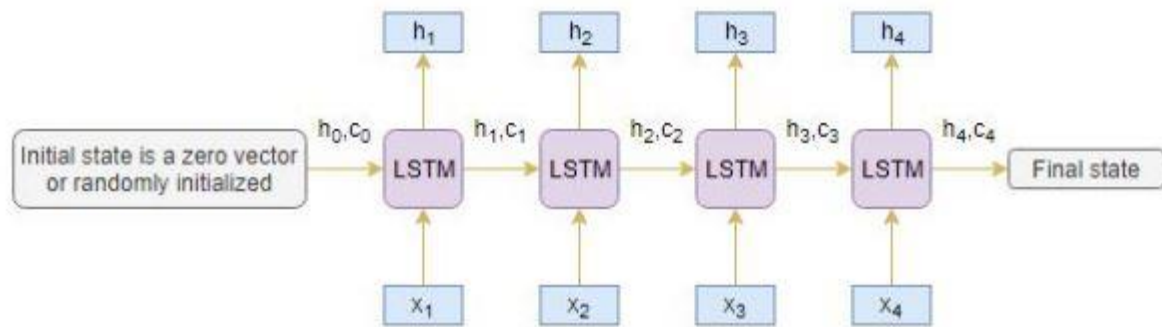
Kiến trúc Encoder-Decoder chủ yếu được sử dụng để giải quyết các vấn đề liên quan đến chuỗi (Seq2Seq) trong đó các chuỗi đầu vào và đầu ra có độ dài khác nhau.

Encoder được sử dụng dùng trích xuất các thông tin và ngữ cảnh của văn bản và encoder dùng dự đoán từng từ trong văn bản sau khi đã biết từ đứng trước nó.

3.1) Training phrase: (giai đoạn huấn luyện mô hình)

Encoder

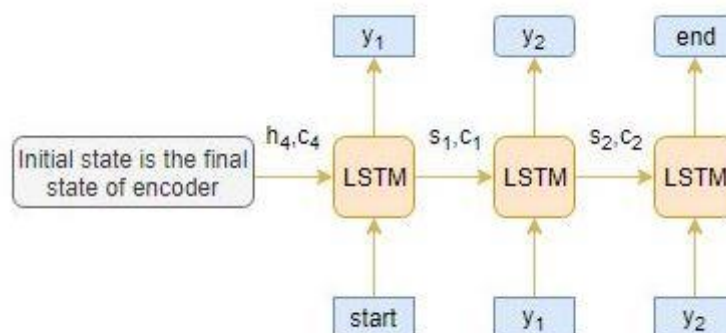
Lấy mô hình LSTM làm ví dụ, sử dụng encoder dùng để đọc toàn bộ chuỗi kí tự đầu vào, tại mỗi bước timestep, từng từ được đưa vào bộ encoder. Encoder giúp mô hình hiểu được rõ ý nghĩa và ngữ cảnh từng từ của câu input.



H_0 và c_0 được khởi tạo bằng các vector random, sau mỗi timestep, các h (hidden state) và c (cell state) của mô hình LSTM được thay đổi. h và c cuối cùng (ở mô hình trên là h_4 và c_4) được dùng để làm giá trị khởi tạo đầu vào cho decoder.

Decoder

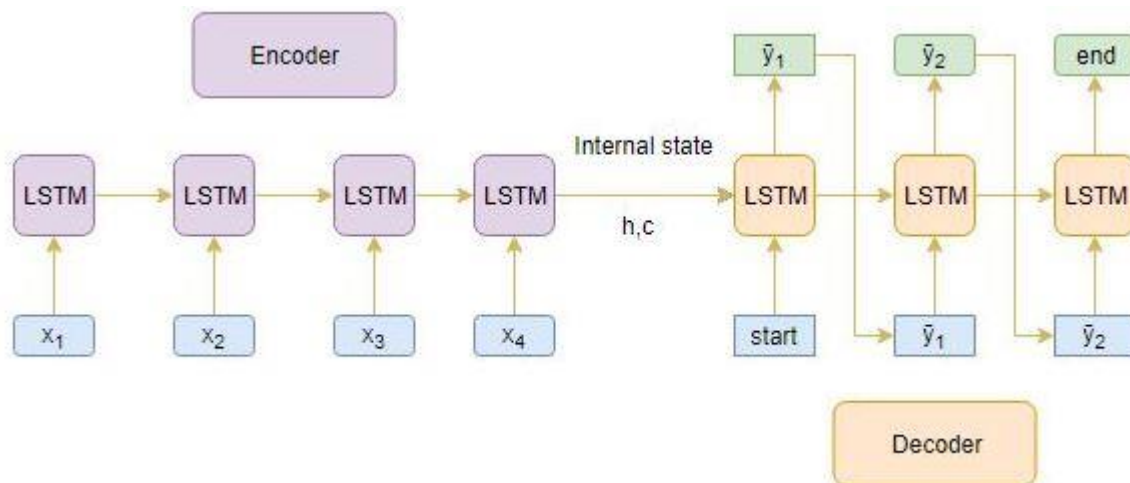
Decoder cũng là một mạng LSTM được huấn luyện để dự đoán từng từ trong câu output khi đã được biết trước từ trước đó trong mỗi timestep.



Token start và end là các token đặc biệt được thêm vào các chuỗi target y trước khi đưa vào decoder để mô hình training biết được đầu câu và cuối câu của target (câu summary trong data được huấn luyện).

3.2) Inference phrase (giai đoạn dự đoán)

Sau giai đoạn training, mô hình được thực nghiệm trên một câu mới. Nên ta cần encode và decode lại văn bản mới này.



Tương tự như training phrase, khi thực hiện dự đoán, văn bản mới được đưa vào encoder model để rút trích đặc trưng và ngữ cảnh, sau đó được đưa vào decoder model để thực hiện dự đoán.

Từng bước thực hiện của inference phrase:

- 1) Văn bản được đưa vào encoder model để rút trích đặc trưng, các hidden state(h) và cell state(c) trong mỗi timestep đều thay đổi, h và c ở timestep cuối cùng cũng chính là giá trị cho decoder.
- 2) Đưa token start như là input vào decoder model
- 3) Chạy mô hình decoder với mỗi timestep sẽ ứng với h và c khác nhau.
- 4) Decoder model sẽ dự đoán xác suất của các từ sẽ xuất hiện tiếp theo sau token start và sẽ chọn từ có xác suất xuất hiện cao nhất.
- 5) Đưa từ được dự đoán trước đó vào timestep kế tiếp và sẽ update h và c theo timestep trước đó như là giá trị đầu vào.
- 6) Lặp lại bước 3->5 cho tới khi tiến tới token end hoặc vượt quá độ dài chuỗi summary cho phép.

4. Attention mechanism:

Do kiến trúc encoder và decoder có một số điểm yếu khi giải quyết những câu quá dài do:

- Decoder cần xem xét toàn bộ các từ trong câu input dài để đưa ra dự đoán
- Khó cho encoder để có thể mã hoá những câu input dài thành một vector số nguyên.

Nên ta có cơ chế attention (attention mechanism) dùng để khắc phục vấn đề này.

Ý tưởng chính của attention mechanism là thay vì nhìn và xem xét toàn bộ các từ trong câu input, ta có thể chỉ cần nhìn vào một phần quan trọng nhất trong câu input

Ví dụ

- **Source sequence:** “Which sport do you like the most?”
- **Target sequence:** “I love cricket”

Từ đầu ‘I’ trong câu output target sequence có mối liên hệ với từ ‘you’ ở vị trí thứ 4 trong câu input. Tương tự vậy, từ thứ 2 ‘love’ trong output cũng có mối liên hệ với từ ‘like’ trong câu input.

III. Đánh giá kết quả trên dữ liệu / Demo:

Dưới đây là từng bước thực hiện đề án text summerization:

1) Thu thập Dataset (Amazon fined food reviews):

Link: <https://www.kaggle.com/snap/amazon-fine-food-reviews>



- Dataset gồm 500000 bài review về thức ở amazon đã được tóm tắt lại
- Thời gian từ 10-1999 đến 10-2012
- Mục tiêu : sử dụng dataset này để tạo ra một mô hình có thể tóm tắt lại các bài review về đồ ăn cho người tiêu dùng đọc dễ hiểu và nhanh chóng

2) Tiền xử lý dữ liệu (Data preprocessing)

Sau khi đã thu thập xong tập dữ liệu (dataset), bước tiếp theo tụi em thực hiện là sẽ xử lý lại dữ liệu ban đầu (raw) này.

Các bước tiền xử lý bao gồm:

- 1) Bỏ đi các giá trị null

Drop Duplicates and NA values

```
data.drop_duplicates(subset=['Text'],inplace=True)#dropping duplicates
data.dropna(axis=0,inplace=True)#dropping na
```

- 2) Lọc các cột không cần chỉ giữ lại 2 cột chính là text(văn bản ban đầu) và summary(văn bản sau khi tóm tắt)

```
data = data[['Text','Summary']]
```

```
data.head(10)
```

	Text	Summary
0	I have bought several of the Vitality canned dog food products and have found them all to be of good quality. The product looks more like a stew than a processed meat and it smells better. My Labr...	Good Quality Dog Food
1	Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actually small sized unsalted. Not sure if this was an error or if the vendor intended to represent the product as "Jumbo".	Not as Advertised
2	This is a confection that has been around a few centuries. It is a light, pillowy citrus gelatin with nuts - in this case Filberts. And it is cut into tiny squares and then liberally coated with ...	"Delight" says it all
3	If you are looking for the secret ingredient in Robitussin I believe I have found it. I got this in addition to the Root Beer Extract I ordered (which was good) and made some cherry soda. The fl...	Cough Medicine
4	Great taffy at a great price. There was a wide assortment of yummy taffy. Delivery was very quick. If your a taffy lover, this is a deal.	Great taffy
5	I got a wild hair for taffy and ordered this five pound bag. The taffy was all very enjoyable with many flavors: watermelon, root beer, melon,	Nice Taffy

- 3) Tiến hành xử lý, làm gọn văn bản:

```
stop_words = set(stopwords.words('english'))

def text_cleaner(text,num):
    newString = text.lower()
    newString = BeautifulSoup(newString, "lxml").text
    newString = re.sub(r'\s+', ' ', newString)
    newString = re.sub('\"', '', newString)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in newString.split(" ")])
    newString = re.sub(r'\s+', ' ', newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    newString = re.sub('[m]{2,}','mm', newString)
    if(num==0):
        tokens = [w for w in newString.split() if not w in stop_words]
    else:
        tokens=newString.split()
    long_words=[]
    for i in tokens:
        if len(i)>1:
            long_words.append(i)
    return (" ".join(long_words)).strip()
```

Quy trình này bao gồm các task :

- Chuyển chữ hoa thành chữ thường
- Bỏ đi html tags
- Ghi đầy đủ các từ, do một từ vẫn còn ghi tắt gây khó khăn trong quá trình training, bước này ta sẽ chuyển các từ ghi tắt thành dạng đầy đủ
- Bỏ đi 's
- Bỏ các văn bên trong dấu ngoặc đơn ()
- Xóa chấm câu và kí tự đặc biệt
- Xóa từ dừng (stopwords)
- Xóa từ ngắn (short words)

4) Thiết lập độ dài từ dài nhất cho văn bản summary và văn bản text ban đầu

```
cnt=0
for i in data['cleaned_summary']:
    if(len(i.split())<=8):
        cnt=cnt+1
print(cnt/len(data['cleaned_summary']))
```

0.9424907471335922

We observe that 94% of the summaries have length below 8. So, we can fix maximum length of summary to 8.

Let us fix the maximum length of review to 30

Do ta thấy gần 95% tập dữ liệu có độ dài tập summary < 8 nên ta cũng sẽ giữ lại những sample có tập summary < 8 để rút gọn tập dữ liệu và tránh những outlier không đáng.

Ta cũng thiết lập độ dài tập text lớn nhất là 30 để làm nhẹ dataset

```
max_text_len=30
max_summary_len=8
```

Let us select the reviews and summaries whose length falls below or equal to **max_text_len** and **max_summary_len**

```
cleaned_text =np.array(data['cleaned_text'])
cleaned_summary=np.array(data['cleaned_summary'])

short_text=[]
short_summary=[]

for i in range(len(cleaned_text)):
    if(len(cleaned_summary[i].split())<=max_summary_len and len(cleaned_text[i].split())<=max_text_len):
        short_text.append(cleaned_text[i])
        short_summary.append(cleaned_summary[i])

df=pd.DataFrame({'text':short_text,'summary':short_summary})
```

5) Thêm 2 token start và end vào tập summary

```
df['summary'] = df['summary'].apply(lambda x : 'sostok ' + x + ' eostok')
```

Ở đây ta thêm vào sostok tương ứng với start và eostok tương ứng end ở đầu và cuối các văn bản trong tập summary.

```
for i in range(5):
    print("Review: ", df['text'][i])
    print("Summary: ", df['summary'][i])
    print("\n")
```

Review: bought several vitality canned dog food products found good quality product looks like stew processed meat smells better labrador finicky appreciates product better
Summary: sostok good quality dog food eostok

Review: product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error vendor intended represent product jumbo
Summary: sostok not as advertised eostok

Review: looking secret ingredient robottussin believe found got addition root beer extract ordered made cherry soda flavor medicinal
Summary: sostok cough medicine eostok

Văn bản ta có được như sau.

3) Tokenization:

Ta chia tập train và tập validation theo tỉ lệ 9:1

```
from sklearn.model_selection import train_test_split
x_tr, x_val, y_tr, y_val = train_test_split(np.array(df['text']), np.array(df['summary']), test_size=0.1, random_state=0, shuffle=True)
```

Ta tiến hành xây dựng vocabulary chứa từng từ khác nhau (unique) trong các văn bản và chuyển các chuỗi từ trong văn bản thành các chuỗi số nguyên trước khi thực hiện quá trình training.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_tr))
```

Ta sẽ loại bỏ đi các từ hiếm xuất hiện trong các sample.

```

: thresh=4

cnt=0
tot_cnt=0
freq=0
tot_freq=0

for key,value in x_tokenizer.word_counts.items():
    tot_cnt=tot_cnt+1
    tot_freq=tot_freq+value
    if(value<thresh):
        cnt=cnt+1
        freq=freq+value

print("% of rare words in vocabulary:",(cnt/tot_cnt)*100)
print("Total Coverage of rare words:",(freq/tot_freq)*100)

% of rare words in vocabulary: 66.12339930151339
Total Coverage of rare words: 2.953684513790566

```

Ở đây ta đặt ngưỡng threshold là tức là những từ nào xuất hiện dưới 4 sample thì sẽ được xem là từ hiếm, ta thấy tỉ lệ từ hiếm trong gói vocabulary là 66% nhưng tỉ lệ xuất hiện các từ hiếm này trong toàn bộ văn bản chỉ là dưới 3%, nên các từ hiếm này thực sự dư thừa và không cần thiết đặt vào vocabulary nên ta sẽ loại bỏ đi.

Tot_cnt : kích thước vocabulary

Cnt : số lượng từ hiếm xuất hiện trong dưới 4 sample

Tot_cnt –cnt cho ta được các từ phổ biến nhất

```

#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
x_tokenizer.fit_on_texts(list(x_tr))

#convert text sequences into integer sequences
x_tr_seq = x_tokenizer.texts_to_sequences(x_tr)
x_val_seq = x_tokenizer.texts_to_sequences(x_val)

#padding zero upto maximum length
x_tr = pad_sequences(x_tr_seq, maxlen=max_text_len, padding='post')
x_val = pad_sequences(x_val_seq, maxlen=max_text_len, padding='post')

#size of vocabulary (+1 for padding token)
x_voc = x_tokenizer.num_words + 1

```

```

#prepare a tokenizer for reviews on training data
y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
y_tokenizer.fit_on_texts(list(y_tr))

#convert text sequences into integer sequences
y_tr_seq = y_tokenizer.texts_to_sequences(y_tr)
y_val_seq = y_tokenizer.texts_to_sequences(y_val)

#padding zero upto maximum length
y_tr = pad_sequences(y_tr_seq, maxlen=max_summary_len, padding='post')
y_val = pad_sequences(y_val_seq, maxlen=max_summary_len, padding='post')

#size of vocabulary
y_voc = y_tokenizer.num_words + 1

```


4) Xây dựng mô hình:

Trong giai đoạn training, ta sẽ xây dựng các mô hình encoder kết hợp embedding layer dùng để rút trích đặc trưng và đưa vào decoder để mô hình học các quy luật và ngữ cảnh của các từ theo thứ tự (Ví dụ như từ này đứng trước thì từ sau tỉ lệ cao sẽ là từ nào).

Ta cũng sẽ thêm một attention layer dùng để giúp mô hình encoder chỉ tập trung vào các từ keyword quan trọng chứ không tập trung vào toàn bộ các từ vì như vậy sẽ gặp nhiều khó khăn trong quá trình rút trích đặc trưng và chuyển đổi thành các vector số.

```

: from keras import backend as K
K.clear_session()

latent_dim = 300
embedding_dim=100

# Encoder
encoder_inputs = Input(shape=(max_text_len,)) # max_text_len = 30

#embedding layer
enc_emb = Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)

#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#encoder lstm 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))

#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurrent_dropout=0.2)
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[state_h, state_c])

# Attention Layer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

# Concat attention input and decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs, attn_out])

#dense layer
decoder_dense = TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.summary()

```

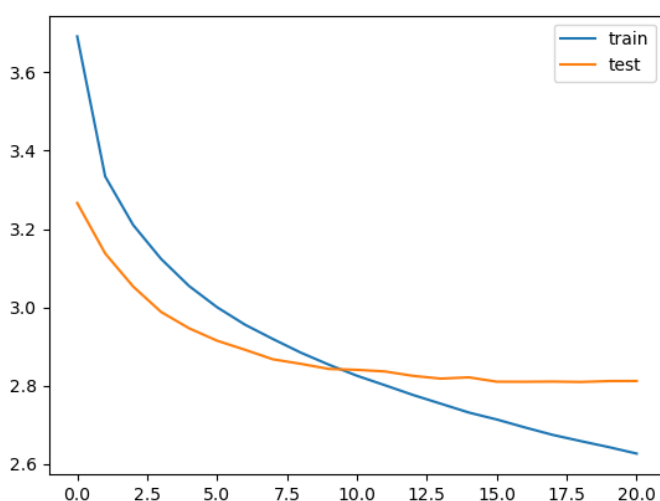
```
: es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
```

We'll train the model on a batch size of 128 and validate it on the holdout set (which is 10% of our dataset):

```
history=model.fit([x_tr,y_tr[:, :-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)[:,:1], epochs=50, callba
41346/41346 [=====] - 393s 10ms/sample - loss: 1.9570 - val_loss: 2.0780
Epoch 13/50
41346/41346 [=====] - 397s 10ms/sample - loss: 1.9359 - val_loss: 2.0632
Epoch 14/50
41346/41346 [=====] - 405s 10ms/sample - loss: 1.9157 - val_loss: 2.0508
Epoch 15/50
41346/41346 [=====] - 400s 10ms/sample - loss: 1.8935 - val_loss: 2.0619
Epoch 16/50
41346/41346 [=====] - 402s 10ms/sample - loss: 1.8761 - val_loss: 2.0456
Epoch 17/50
41346/41346 [=====] - 403s 10ms/sample - loss: 1.8596 - val_loss: 2.0444
Epoch 18/50
41346/41346 [=====] - 404s 10ms/sample - loss: 1.8426 - val_loss: 2.0449
Epoch 19/50
41346/41346 [=====] - 394s 10ms/sample - loss: 1.8280 - val_loss: 2.0372
Epoch 20/50
41346/41346 [=====] - 399s 10ms/sample - loss: 1.8151 - val_loss: 2.0441
Epoch 21/50
41346/41346 [=====] - 400s 10ms/sample - loss: 1.7996 - val_loss: 2.0389
Epoch 00021: early stopping
```

Ta tiến hành training và đặt điều kiện dừng sớm là khi giá trị loss của validation tăng mạnh.

5) Đánh giá mô hình (evaluation):



Em đã thử tuning các parameter : latent dim (latent space của encoder model), embedding dim (kích thước embedding layer) và rút ra kết luận rằng Latent dim= 300 và embedding dim =100 đưa ra giá trị loss mô hình tốt nhất.

6) Tiến hành dự đoán

Đây cũng là giai đoạn inference (inference phrase) như trong lý thuyết ở trên

```
# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=encoder_inputs, outputs=[encoder_outputs, state_h, state_c])

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_text_len, latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2 = dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decoder_state_input_h, decoder_state_input_c])

# attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input, decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn_out_inf])

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input, decoder_state_input_h, decoder_state_input_c],
    [decoder_outputs2] + [state_h2, state_c2])
```

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index['sostok']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if (sampled_token != 'eostok'):
            decoded_sentence += ' ' + sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eostok' or len(decoded_sentence.split()) >= (max_summary_len-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1, 1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

Ở đây là dùng thuật toán tham lam (argmax) để xác định từ có xác suất xuất hiện kế tiếp cao nhất.

Ta sẽ thực hiện từng bước inference phrase để dự đoán như trên:

- 1) Văn bản được đưa vào encoder model để rút trích đặc trưng, các hidden state(h) và cell state(c) trong mỗi timestep đều thay đổi, h và c ở timestep cuối cùng cũng chính là giá trị cho decoder.
- 2) Đưa token start như là input vào decoder model
- 3) Chạy mô hình decoder với mỗi timestep sẽ ứng với h và c khác nhau.
- 4) Decoder model sẽ dự đoán xác suất của các từ sẽ xuất hiện tiếp theo sau token start và sẽ chọn từ có xác suất xuất hiện cao nhất.
- 5) Đưa từ được dự đoán trước đó vào timestep kế tiếp và sẽ update h và c theo timestep trước đó như là giá trị đầu vào.
- 6) Lặp lại bước 3->5 cho tới khi tiến tới token end hoặc vượt quá độ dài chuỗi summary cho phép .

Do giá trị trả về của inference phrase vẫn là các vector số nên ta sẽ xây dựng 2 hàm để thành chuỗi string

```
def seq2summary(input_seq):
    newString=""
    for i in input_seq:
        if((i!=0 and i!=target_word_index['sostok']) and i!=target_word_index['eostok']):
            newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=""
    for i in input_seq:
        if(i!=0):
            newString=newString+reverse_source_word_index[i]+' '
    return newString
```

Ta thực nghiệm trên 100 samples của tập validation

```
for i in range(0,100):
    print("Review:",seq2text(x_val[i]))
    print("Original summary:",seq2summary(y_val[i]))
    print("Predicted summary:",decode_sequence(x_val[i].reshape(1,max_text_len)))
    print("\n")
```

Review: salt separate area pain makes hard regulate salt putting like salt go ahead get product
Original summary: tastes ok packaging
Predicted summary: great salt

Review: really like product super easy order online delivered much cheaper buying gas station stocking good long drives
Original summary: turkey jerky is great
Predicted summary: great product

Review: best salad dressing delivered promptly quantities last vidalia onion dressing compares made oak hill farms sometime
s find costco order front door want even orders cut shipping costs
Original summary: my favorite salad dressing
Predicted summary: best mustard ever

IV. Hướng phát triển:


- Sử dụng **mô hình BiLSTM**
- Sử dụng **beam search strategy** trong quá trình decode để dự đoán từ tiếp theo thay vì thuật toán tham lam (argmax)
- Sử dụng **Bleu Score** để đánh giá mô hình

V. Hướng dẫn sử dụng các file python đã được xây dựng cấu trúc thông qua môi trường Command Prompt:

File 1: File Processing_data.py

Đây là file phục vụ quá trình tiền xử lý dữ liệu raw ở trong thư mục data:

- Để sử dụng cần chú ý đặt đúng đường dẫn trong command prompt đến đúng vị trí thư mục /TextSummarization
- Mở command prompt và dùng lệnh `python src/data/Processing_data.py` để sử dụng
- File này sẽ xử lý dữ liệu và lưu dữ liệu mới ở đường dẫn `TextSummarization/data/processed/data_processed.csv`

 Command Prompt

```
E:\Slide subject\Text mining\TextSummarization>python src/data/Processing_data.py _
```

File 2: File split_data_tokenization.py

Đây là file dùng để phục vụ quá trình chia data và tokenization, đây chỉ file chứa hàm tokenization để các file tham chiếu tới để sử dụng nên ở file này ta không cần gọi file trong command prompt.

File 3: File visualize.py

Đây là file dùng để hiện đồ thị hiển thị distribution của văn bản text và văn bản summary. Nếu bạn muốn xem thử EDA của mô hình thì sử dụng file này

Cách sử dụng trong command prompt:

```
python src/visualization/visualize.py
```

File 4: Train_model.py

Đây là file dùng để training mô hình sử dụng kiến trúc seq2seq bao gồm các mô hình encoder và decoder.

Cách dùng:

- Để sử dụng cần chú ý đặt đúng đường dẫn trong command prompt đến đúng vị trí thư mục /TextSummarization
- Vào file train_model.py sửa đường dẫn của src sau tương ứng máy của bạn

```
sys.path.append("E:\\Slide subject\\Text mining\\TextSummarization\\src")
```

- Dùng lệnh python src/models/train_model.py mặc định là sẽ thiết lập mô hình latent_dim, embedding_dim, epochs mặc định lần lượt là 300, 100, 50, nên nếu muốn thay đổi các thông số này ta dùng lệnh:

```
python src/models/train_model.py -l {thông số latent_dim} -e {thông số embedding_dim} -ep {số epochs}
```

Ví dụ:

```
E:\\Slide subject\\Text mining\\TextSummarization>python src/models/train_model.py -l 300 -e 100 -ep 50
```

- Sau khi mô hình được training xong (sau khoảng 3 giờ đồng hồ) các mô hình encoder và decoder sử dụng để dự đoán trong giai đoạn inference (inference phrase) sẽ được lưu lại trong file .h5

Các file đó gồm :

encoder_model_inference.h5 được lưu tại TextSummarization/models
decoder_model_inference.h5 được lưu tại TextSummarization/models

```
Using TensorFlow backend.
Train on 40925 samples, validate on 4501 samples
Epoch 1/50 [=====] - 544s 13ms/sample - loss: 3.6914 - val_loss: 3.2664
Epoch 2/50 [=====] - 560s 14ms/sample - loss: 3.3341 - val_loss: 3.1376
Epoch 3/50 [=====] - 581s 14ms/sample - loss: 3.2103 - val_loss: 3.0533
Epoch 4/50 [=====] - 593s 14ms/sample - loss: 3.1236 - val_loss: 2.9883
Epoch 5/50 [=====] - 609s 15ms/sample - loss: 3.0545 - val_loss: 2.9467
Epoch 6/50 [=====] - 614s 15ms/sample - loss: 3.0005 - val_loss: 2.9151
Epoch 7/50 [=====] - 616s 15ms/sample - loss: 2.9559 - val_loss: 2.8918
Epoch 8/50 [=====] - 615s 15ms/sample - loss: 2.9192 - val_loss: 2.8675
Epoch 9/50 [=====] - 610s 15ms/sample - loss: 2.8843 - val_loss: 2.8559
Epoch 10/50 [=====] - 601s 15ms/sample - loss: 2.8538 - val_loss: 2.8429
Epoch 11/50 [=====] - 609s 15ms/sample - loss: 2.8256 - val_loss: 2.8407
Epoch 12/50 [=====] - 594s 15ms/sample - loss: 2.8014 - val_loss: 2.8365
Epoch 13/50 [=====] - 594s 15ms/sample - loss: 2.7767 - val_loss: 2.8253
Epoch 14/50 [=====] - 584s 14ms/sample - loss: 2.7543 - val_loss: 2.8182
Epoch 15/50 [=====] - 577s 14ms/sample - loss: 2.7317 - val_loss: 2.8215
Epoch 16/50 [=====] - 565s 14ms/sample - loss: 2.7140 - val_loss: 2.8104
Epoch 17/50 [=====] - 561s 14ms/sample - loss: 2.6938 - val_loss: 2.8103
Epoch 18/50 [=====] - 559s 14ms/sample - loss: 2.6747 - val_loss: 2.8107
Epoch 19/50 [=====] - 560s 14ms/sample - loss: 2.6589 - val_loss: 2.8098
Epoch 20/50 [=====] - 559s 14ms/sample - loss: 2.6436 - val_loss: 2.8120
Epoch 21/50 [=====] - 560s 14ms/sample - loss: 2.6270 - val_loss: 2.8121
Epoch 00021: early stopping
Encoder model for prediction has been saved
Decoder model for prediction has been saved
training completed
```

File 5: predict_model.py

Đây là file dùng để dự đoán sau khi đã training

Cách dùng:

- Để sử dụng cần chú ý đặt đúng đường dẫn trong command prompt đến đúng vị trí thư mục /TextSummarization
- Vào file train_model.py sửa đường dẫn của src sau tương ứng máy của bạn

```
sys.path.append("E:\\Slide subject\\Text mining\\TextSummarization\\src")
```

- Dùng lệnh python src/models/predict_model.py

```
E:\\Slide subject\\Text mining\\TextSummarization>python src/models/predict_model.py
```

```
Review: one smooth easy acidity without bland however tassimo retail amazon getting inflated retail price retail pack every disc  
Original summary: impressive  
Predicted summary: great product  
  
Review: would call coffee addict drink average double espresso per day coffee lacks everything like enjoy coffee period waste  
Original summary: huge bag very fresh  
Predicted summary: great coffee  
  
Review: wife ordered soup amazon chef buys food impressed order handled processed shipped quickly looking forward ordering product  
Original summary: very good  
Predicted summary: great tasting  
  
Review: bland dry lots salt little bit know like dusty taste love cheese really pleasant  
Original summary: gerber's fruits are cooked  
Predicted summary: great product  
  
Review: hard find particular chocolate essential creating marvelous second none chocolate cake  
Original summary: delicious tea  
Predicted summary: great product  
  
Review: kind bars delicious go wrong mangos macadamia nuts save flavor really want something special shared actually pleasure tasted first one food us  
Original summary: delicious  
Predicted summary: great snack  
  
Review: love gloria jeans coffee since closed store closest get fix butter toffee delicious  
Original summary: gloria jean's butter toffee k cup coffee  
Predicted summary: great coffee
```

Kết quả sẽ hiển thị 10 sample được dự đoán trong tập validation

VI. Tài liệu tham khảo:

- ❖ <https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/>
- ❖ <https://blog.floydhub.com/gentle-introduction-to-text-summarization-in-machine-learning/>
- ❖ <https://medium.com/jatana/unsupervised-text-summarization-using-sentence-embeddings-adb15ce83db1>
- ❖ <https://medium.com/sciforce/towards-automatic-text-summarization-extractive-methods-e8439cd54715>