TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG_HCM

KHOA CÔNG NGHỆ THÔNG TIN

MÔN HỌC THỐNG KÊ

GVHD: Ngô Minh Nhựt

Lê Long Quốc

Final Semester Project – Text Classification

MSSV	:	HỌ VÀ TÊN
20120083		Nguyễn Trọng Hiếu
20120105		Lê Hoàng Huy
20120149		Phạm Sỹ Nguyên
20120515		Luân Mã Khương

Mục lục

I. Lời mở đầu

Xử lý ngôn ngữ tự nhiên (NLP - Natural Language Processing) là lĩnh vực nghiên cứu và ứng dụng công nghệ để máy tính có khả năng tương tác và hiểu được ngôn ngữ tự nhiên của con người. NLP tập trung vào việc xử lý, phân tích, và tạo ra thông tin từ văn bản và ngôn ngữ nói.

Một trong những ứng dụng của NLP là phân loại văn bản (text classification) là quá trình tự động gán nhãn cho các mẫu văn bản dựa trên nội dung và tính chất của chúng. Mục tiêu của phân loại văn bản là xác định và phân loại các văn bản vào các danh mục hay chủ đề khác nhau.

Mọi thông về project và data được lưu trữ tại link sau: *onedrive*

Source code được lưu trữ tại link sau: *github*

II. Nền tảng lý thuyết

Trong học máy, phân loại văn bản được xem là một nhiệm vụ học có giám sát, trong đó mô hình được huấn luyện trên tập dữ liệu huấn luyện có nhãn để có thể dự đoán nhãn cho các văn bản chưa được gán nhãn.

Các phương pháp phân loại văn bản phổ biến bao gồm

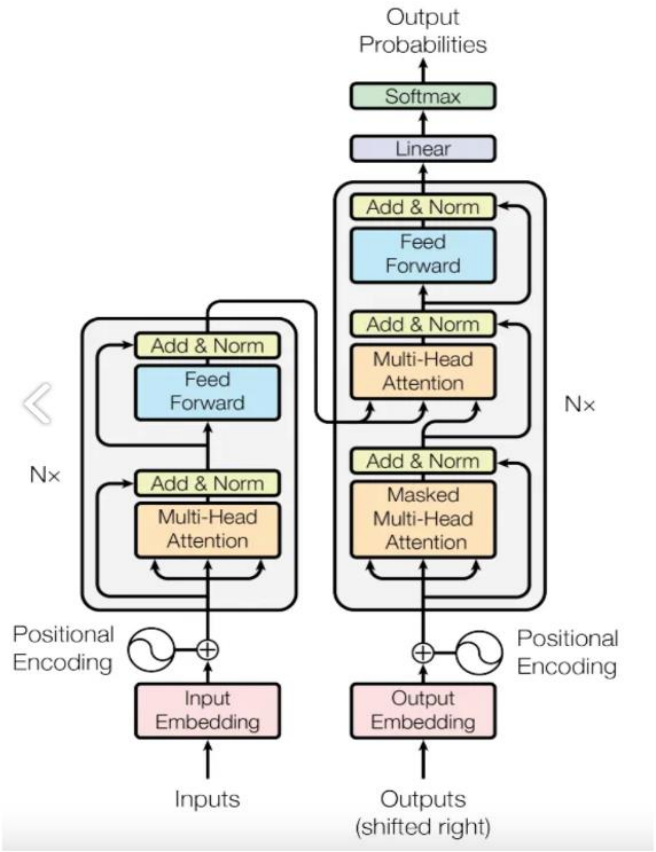
- máy học vector hỗ trợ (*SVM*): bị hạn chế về mặt tính toán đối với dữ liệu có số chiều hớn $\rightarrow$ không hoạt động tốt, kết quả đáp ứng đầu ra không cao.
- cây quyết định (*Decision tree*): đối với độ sâu cây cao $\rightarrow$ thường dẫn đến việc overfitting.
- naive Bayes: giả định về các yếu tố dự đoán độc lập, điều mà hầu như không thể có được trong các bài toán thực tế.
- mạng nơ-ron (*RNN*): bị hạn chế về mặt trí nhớ ngắn hạn – mặc dù có tính thứ tự tính toán do thực hiện recurent trên một token nhiều lần (điều này gây ảnh hướng về tốc độ xử lý) - không có khả năng xử lý song song. Đồng thời RNN còn bị mặt hạn chế về "vanishing gradient".

Từ các mặt hạn chế đối với các mô hình máy học, mô hình transfomer đã được đề xuất cho bài toán phân loại văn bản. Đây được coi như một cuộc cách mạng với kiến trúc áp dụng cho nlp và **AI** nói chung.

Kiến Trúc transformer

Kiến trúc Transformer là một kiến trúc mạng nơ-ron đột phá được giới thiệu trong bài báo "Attention Is All You Need" của Vaswani và đồng nghiệp vào năm 2017. Được phát triển ban đầu cho các tác vụ dịch máy, Transformer đã trở thành một công cụ mạnh mẽ trong xử lý ngôn ngữ tự nhiên, bao gồm cả phân loại văn bản.

Kiến trúc Transformer dựa trên cơ chế attention (sự tập trung). Nó thay thế các lớp mạng nơ-ron truyền thống bằng các lớp tự chú ý (self-attention), cho phép mô hình tập trung vào các phần quan trọng của đầu vào trong quá trình xử lý.



Hình 1: The Transformer - model architecture [1]

2.1. Cái nhìn tổng quan

Encoder

Encoder stack là một tập các layer xác định với số lượng được chọn là 6. Mỗi layer bao gồm 2 sub-layers:

- sub-layer 1: Sử dụng cơ chế *multi-head self-attention*
- sub-layer 2: Position-wise fully connected *feed-forward network*

Đồng thời sử dụng các kết nối Residual giữa các 2 sub-layer có kết hợp layer-normalization.

Decoder

Bộ giải mã (Decoder) trong Transformer chịu trách nhiệm chuyển đổi đầu vào từ bộ mã hóa (Encoder) thành đầu ra cuối cùng. Decoder hoạt động qua nhiều lớp, mỗi lớp bao gồm hai thành phần chính: mô-đun "Self-Attention" và mạng "Feed-Forward".

2.2. Chi tiết bên trong

2.2.1. Embedding

Dùng embedding với input đầu vào để được các token với số chiều được quy định cùng với số chiều của model là $d_{\text{model}} = 512$. Input ở đây được miêu tả bởi 2 đầu vào là Inputs và Output (shift right) được mô tả như *hình 1*.

Có một số phương thức được dùng để chuyển hóa từ word thành vector (word2vec) được sử dụng phổ biến như sau:

- Continuous bag-of-words model: đúng như tên gọi, kiến trúc chỉ quan tâm đến 1 tập các từ chuyển hóa và tính thứ tự không được coi trọng.
- Continuous skip-gram model: dự đoán một từ ngữ với khoảng cách nhất định về phía trước hoặc sau so với vị trí hiện tại của từ đang xét.

Thông tin chi tiết được đính kèm trong phần reference [2].

2.2.2. Encoder

❖ ***Posional encodding***

Để một model (bao gồm cả các model không tích hợp recurrence và không convolution) nhận biết được tính thứ tự ngữ nghĩa chứa trong một *sequence* (chuỗi input trước khi tách thành các tokens), ta cần thêm một số thông tin liên quan hoặc đặc trưng về vị trí cho từng tokens ở trong một sequence.

Do đó, sau bước *Embedding* ta có thao tác cộng "*position encoding*" ở layer thấp nhất trong encoder và decoder stacks.

Có nhiều cách chọn đối với "*position encoding*" sau đây là một trong những phép toán được dùng:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

*Vì "*position encoding*" có cùng số chiều với "*embdding*" nên ta có thể thực hiện phép cộng. Cơ chế này cũng được áp dụng cho "*decoder"*.

2.2.3. Attention

Attention function có thể được miêu tả tương tự một phương pháp ánh xạ một câu truy vấn *(query)* và một tập các *key-value* thành một *ouput*. Ở đây các khái niệm *query*, *keys*, *values*, và *output* đều là các *vectors*.

Bằng cách sử dụng các phương pháp đánh giá sự tương thích giữa một *query* và một *key* tương ứng, ta sẽ thu được giá trị *value*. Tính toán dựa trên các *values* thu được đối với nhiều *query* và *key* tương ứng ta sẽ thu được giá trị *output*.

Bước đầu của việc tính toán self-attention yêu cầu tạo ra được 3 dạng biểu diễn khác nhau ứng với cùng nguồn là một token đầu vào. Để tạo được 3 vector $(\text{Key: } k,\ \text{Query: } q,\ \text{Value: } v)$ đặc trưng cho từng token, ta cần có các weight-matrix tương ứng (được khởi tạo random ban đầu hoặc có một số yêu cầu nhất định) được cải thiện thông qua quá trình huấn luyện model.

Trong trường hợp k, v cùng nguồn và q khác nguồn thì cơ chế được gọi là *cross-attention*

$$
\begin{aligned}
q_i &= W^q x_i \\
k_i &= W^k x_i \qquad \text{với } \forall x_i, i \in \mathrm{N}, i \in [1, n],\ n \text{ là số lượng token} \\
v_i &= W^v x_i
\end{aligned}
$$

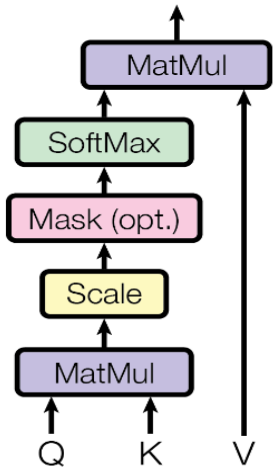
Có 2 phương pháp được sử dụng phổ biến nhất trong cơ chế attention:

- Additive attention: Hàm tính toán khả năng tương thích sử dụng feed-forward netwok với một layer ẩn.
- Dot-product attention: Được trình bày bên dưới

Trong khi cả hai phương pháp đều được xây dựng trên nền tảng lý thuyết phức tạp nhưng đối với dot-production attention khi ứng dụng trên thực tế cho ra hiệu quả tốt hơn về mặt thời gian và không gian.

i. Scaled Dot-product Attention

Từ các input đầu và là các vector q, k, v thể hiện tính đặc trưng cho token được tạo ra như trên, ta tiến hành nhân q với k, sau đó chia cho d_k, xử lý tiếp bằng hàm



Hình 2: Sacled Dot-product Attention [1]

SoftMax để thu được các trọng số gọi là attention-weights và cuối dùng nhân với giá trị v để thu được các kết quả attention của token input. Có thể xem việc này như việc lấy trung bình có trọng số cho các lận cận của một token.

Trên thực tế, ta có thể tận dụng các phép nhân ma trận bằng các ma trận Q, K, V được tạo với các vector q_i, k_i, v_i tương ứng để có thể đẩy nhanh quá trình huấn luyện model bằng cách tận dụng các cơ chế đa luồng và xử lý song song của các Graphic Processing Unit – GPU.

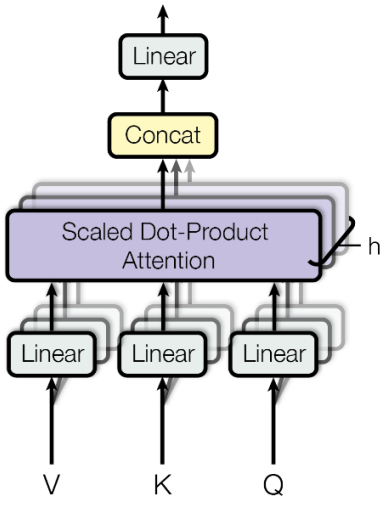
Đối với dot-product attention khi dk (dimention of the key vector) lớn có thể dẫn đến sự giãn nở nhanh về mạch kích cỡ trong quá trình thực hiện phép nhân. Ta có một bước scale các giá trị sau khi thực hiện phép nhân của QK^T cho $\sqrt{d_k}$ với mong muốn đưa các giá trị đó về phân phối chuẩn.

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

Với các giá trị của Q, K là biến độc lập - ngẫu nhiên có $\left(\mu = 0, \partial = 1\right)$. Kết quả của phép nhân $q.k = \sum_{i=1}^{d_k} q_i.k_i$ có $\left(\mu = 0, \partial = d_k\right)$

ii. Multi-Head Attentiond

Thay vì sử dụng chỉ một attention-layer thì ta thực hiện h-layers cho ba ma trận (Q, K, V).



Hình 3: Multi-Head Attention [1]

Mỗi kết quả một attention-layer có thể được xử lý song song. Sau đó thực hiện kết hợp lại đem làm input cho một linear layer nhằm giảm số chiều để thu được kết quả

cuối cùng. Khi sử dụng phương pháp này model có thể tập trung vào các thông tin đến từ nhiều dạng biểu diễn khác nhau.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

iii. Feed-forward Network

Trong mỗi layer ở encoder và decoder, ta có mạng lưới chuyển đổi nối tiếp feed-forwad. Hoạt động một cách độc lập và tương tự trên mỗi vị trí một cách riêng biệt và xác định. Bao gồm một fully-connected layer với hàm kích hoạt ReLU và kết thúc bằng một linear layer

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Số chiều của input x và output $\text{FFN}(x)$ là $d_{\text{model}} = 512$

Số chiều của inner-layers là $d_{ff} = 2048$

$$b_1 \in \mathbb{R}^{d_{ff}}, b_2 \in \mathbb{R}^{d_{\text{model}}}$$

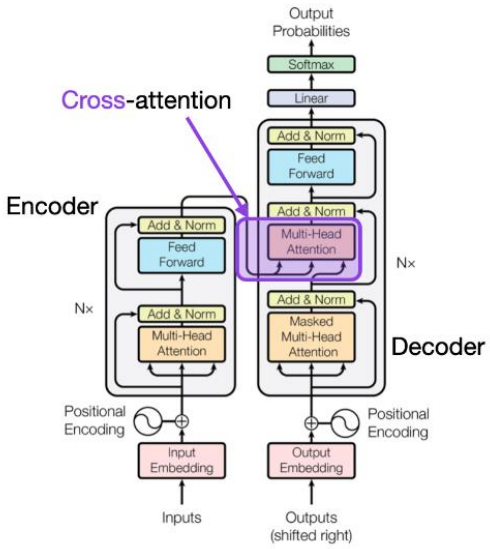
$$W_1 \in \mathbb{R}^{d_{\text{model}} \text{ x } d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \text{ x } d_{\text{model}}}$$

2.2.4. Layer Normalization and Residual Connections

- ***Add normalization*:** được áp dụng trên từng sub-layer, nhằm mục chuẩn hoác giá đầu ra của các layer đầu ra và ổn định quá tình thực hiện propagation từ output của layer này đến layer khác.
- ***Residual Connections:*** nhằm giải quyết vấn đề vanishing/exploding gradient đối với các mạng có độ sâu lớn như *transformer*; Ví dụ như hạn chế các giá trị gradient quá thấp (gradient tiến về 0) hoặc quá cao (gradient tiến ra vô cùng) gây ảnh hưởng đến quả trình huấn luyện model.

2.2.5. Decoder

Thành phần "Attention" trong Decoder không chỉ thực hiện Attention cho chính đầu vào từ Decoder, mà còn thực hiện Attention với các thông tin đến từ đầu ra của Encoder, cơ chế tương tác này còn được gọi là Cross Attention (khi ma trận Q đến từ một nguồn khác với ma trận K,V).



Hình 4: Cơ chế Cross-attention

Tuy nhiên, có một khía cạnh quan trọng cần được chú ý, đó là "Masked Attention". Khi xử lý từng từ trong câu đích Q, Decoder không nên xem trước các từ sau trong câu, vì chúng chưa được sinh ra. Vì vậy, "Masked Attention" được sử dụng để ngăn chặn việc "nhìn trước" thông tin chưa xuất hiện, bằng cách áp dụng một "mặt nạ" (mask) để che đi các từ ở tương lai trong câu. Điều này giúp mô hình tập trung vào các từ đã biết và không bị ảnh hưởng bởi các từ chưa được sinh ra, do vậy Decoder cần được thực hiện việc shift-right cho đầu vào bằng cách chèn một token đặc biệt như <start> vào phía trước của input. Bên cạnh là cơ chế autoregressive giúp cho Decoder thực hiện tác vụ sinh (generative task) với đầu vào là chính thông tin đầu ra của bước trước đó.

Mạng "Feed-Forward", giống như trong bộ mã hóa, là một mạng nơ-ron hoàn toàn kết nối có hai lớp tuyến tính.

Sau khi thông qua tất cả các lớp của Decoder, thông tin được truyền qua một lớp tuyến tính cuối cùng, sau đó là hàm softmax để tạo ra phân phối xác suất cho từng từ tiếp theo có thể xuất hiện trong câu đích Q.

Tóm lại, Decoder của Transformer không chỉ đảm bảo chất lượng cho tác vụ sinh mà còn tăng hiệu suất bằng cách xử lý song song và xem xét toàn bộ ngữ cảnh, với sự hỗ trợ quan trọng từ cơ chế "Masked Attention".

- ***Masking***

Masking được ứng dụng bên trong *scaled dot-product attention* bằng cách gán một giá trị vô cùng nhỏ vào các vị trí chưa được dự đoán trước khi bước vào quá trình *softmax*.

❖ ***Linear transformation and Softmax function***

Dùng để chuyển các "decoder" output thành những token có khả năng xảy ra tiếp theo.

Mô hình BERT

BERT, Bidirectional Encoder Representations from Transformers, là một machine learning (ML) model dùng để xử lý ngôn ngữ tự nhiên, được phát triển vào năm 2018 bởi các nhà nghiên cứu tại Google AI Language và hỗ trợ nhiều tác vụ hỗ trợ giải quyết nhiều bài toán về ngôn ngữ như phân tích cảm xúc (sentiment analysis) hay nhận dạng thực thể (named entity recognization). Đây được coi như là một sản phẩm mang tính đột phá trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP)

Có hai quá trình đối với framework này:

- ***Tiền huấn luyện (pre-training)***: Xuyên xuốt quá trình, model sẽ được train trên dữ liệu không có nhãn trên nhiều *tác vụ pre-training* khác nhau.
- ***fine-tuning***: Model được khởi tạo ban đầu với với các tham số đã được tiền huấn luyện sau đó dùng các dữ liệu được gắn nhãn ở tác vụ cuối để đi đến việc tạo dựng một model cho các bài toán cụ thể từ model BERT tổng quát hóa sẵn.

2.3. Kích thước mô hình & kiến trúc tổng quát

Kiến trúc mô hình BERT bao gồm multi-layer bidirectional Transformer encoder dựa trên kiến trúc transformer được miêu tả bên trên. Có hai kiến trúc gốc của BERT là $\text{BERT}_{\text{BASE}}$ và $\text{BERT}_{\text{LARGE}}$ với thông số được miêu tả như sau:

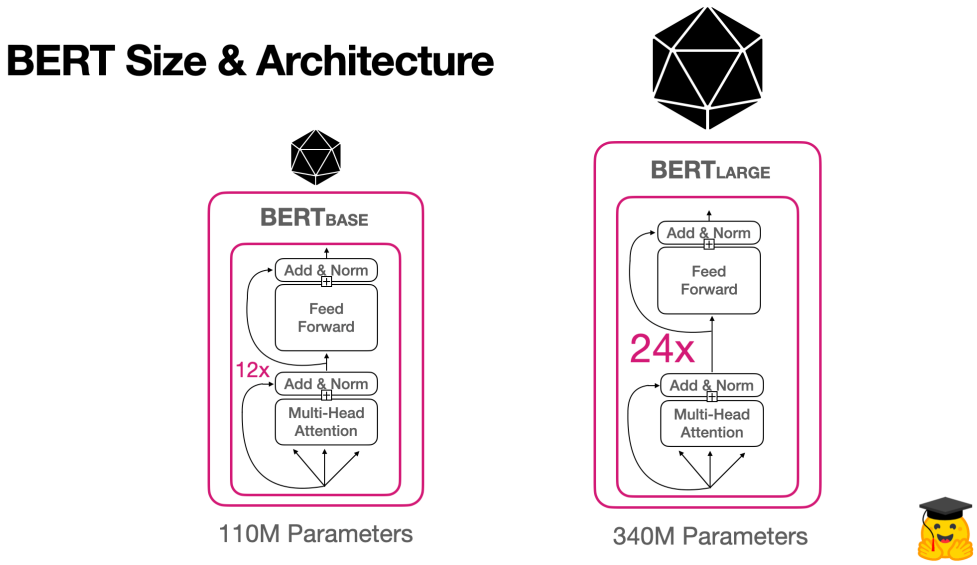
$\text{BERT}_{\text{BASE}}$ (L=12, H=768, A=12, Total Parameters=110M)

$\text{BERT}_{\text{LARGE}}$ (L=24, H=1024, A=16, Total Parameters=340M)

L : Số lượng layer – Transformer blocks

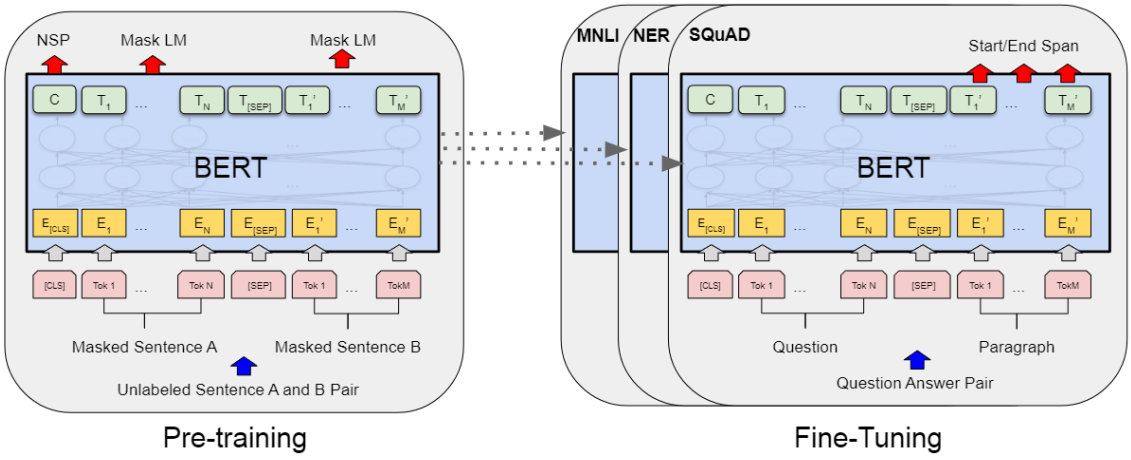
H : Số lượng hidden size

A: Số lượng self-attention heads



Hình 5: Bert model size & architecture [5]

BERT bao gồm hai quy trình là tiền huấn luyện (*pre-training*) và tinh chỉnh (*fine-*



Hình 6: Quy trình bên trong BERT

tuning).

Input: Một "sequence" đại diện cho một chuỗi token đầu vào có thể là một câu hoặc 2 câu đính kèm với nhau. Token đầu tiên của mỗi "sequence" luôn là một token đặc biệt (classification token – [CLS]) đại diện cho sequence từ lớp *hidden* cuối cùng cho tác vụ phân loại. Đối với đầu vào là một cặp câu thì sẽ được xem như là một câu "sequence" đơn thuần và giữa các câu được ngăn cách bởi token đặc biệt ([SEP]).

2.4. Tác vụ tiền huấn luyện (Pre-training)

2.4.1. Masked language modeling

Nhằn huấn luyện biểu diễn hai chiều một cách sâu sắc, một tỉ lệ token đầu vào đã được che (*mask*) và thực hiện dự đoán trên các token đã bị che đó.
* Tỉ lệ mask là 15% ngẫu nhiên của số lượng token trong câu [4].

2.4.2. Next sentence prediction (NSP)

NSP có thể được thực hiện trên bất kỳ bộ dữ liệu đơn ngữ (*monolingua corpus*) nào đó. Tác vụ NSP giúp cho mô hình ngôn ngữ có thể hiểu mối liên hệ tương quan giữa hai văn bản (hoặc câu). Trong đó, các tác giả sử dụng bộ dữ liệu với các cặp câu (A, B) với 50% B là câu văn nối tiếp của A, được gán nhãn là *IsNext*. Ngược lại 50% B được chọn ngẫu nhiên từ tập ngữ liệu (corpus), được hiểu là không phải câu văn nối tiếp của A, có nhãn là *NotNext*.

Trong hình 6. Tác vụ NSP được đại diện bởi kí hiệu C và nhận một trong hai giá trị đầu ra (isNext hoặc NotNext tương ứng với một giá trị nhị phân do đó quá trình này còn được gọi là tác vụ *binarized next sentence prediction*)

2.5. Tinh chỉnh (Fine-tuning)

Được nêu trong phần thiết lập thực nghiệm (ứng với bài toán cụ thể)

Từ quá trình tiền huấn luyện (pre-training), mô hình BERT có thể được áp dụng cho các tác vụ cuối (downstream tasks) tùy theo từng ngữ cảnh và bài toán. Các tác giả bài báo đã sử dụng 11 tác vụ NLP khác nhau [4] để đánh giá sự hiệu quả của mô hình và cho thấy sự mạnh mẽ trong các tác vụ học chuyển tiếp (transfer-learning) của mô hình BERT.

III. Bộ dữ liệu

Mô tả bộ dữ liệu

Bộ dữ liệu được dùng cho bài lab này là ***Amazon Product Reviews [6]***, gồm các đánh giá sản phẩm từ *Amazon*. Đây là một tập dữ liệu lớn bao gồm 142,8 triệu đánh giá từ tháng 5 năm 1996 đến tháng 7 năm 2014.

Tuy nhiên trong phạm vi bài lab, chỉ sử dụng một phần dữ liệu được public, cụ thể là khoảng 15000 mẫu cho mỗi label cần dùng để sử dụng trong quá trình xây dựng mô hình.

Một object cho mỗi label chứa các trường đã được xử lý được mình họa như sau:

{"*reviewerID*":"*A2VNYWOPJ13AFP*",
"*asin*":"0981850006",
"*reviewerName*":"...",
"*helpful*": [6,7],
"*reviewText*":"...",
"*overall*":5.0,
"*summary*":"*Delish*",
"*unixReviewTime*":1259798400,
"*reviewTime*":"12 3, 2009",
"*label*":18}

Trong ngữ cảnh của bài báo cáo, ta chỉ tập trung sử dụng trường dữ liệu "reviewText" và "label" cho quá trình huấn luyện.

Data link here

IV. Thiết lập thực nghiệm

Phương pháp cho bài toán cụ thể

3.1. Mô hình sử dụng

Xem xét về mặt kích thước bộ dữ liệu để train đi kèm với nguồn tài nguyên hạn chế mà nhóm có thể sử dụng thì nhóm quyết định dùng mô hình DistilBERT [], một phiên bản nhỏ hơn (distilation version) của mô hình BERT-base nhưng vẫn đảm bảo xấp xỉ về tính hiệu quả.

Bảng so sánh về kích thước tham số và thời gian xử lý giữa 3 mô hình ELMo, BERT-base, DistilBERT [7]:

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

Hình 7: Thời gian xử lý trên cùng một tác vụ với các điều kiện thiết lập so sánh tương đương [7]

3.2. Quy trình huấn luyện (Fine-tuning)

Đối với dữ liệu đầu vào, vì gồm nhiều object với nhiều dữ liệu khác nhau nên ta chỉ mong muốn sử dụng 2 trường trong 1 object là "reviewText" và "label" nên cần một thao tác lấy data đúng vị trí trước khi bước vào quá trình train.

Ta cần một số giá trị thuộc về từng index được chỉ định đối với mỗi lần gọi bao gồm các trường sau (kỹ thuật xây dựng được nêu trong hình 7):

- "*input_ids*" và "*attention_mask*": giá trị được giả về sau khi embedding một sequence với độ xài xác định (length = 512).
- "*label*": label của từng sequence

```
class AmazonProductDataset(Dataset):
    def __init__(self, all_data: list, tokenizer: PreTrainedTokenizerBase, max_length: int = 512):
        super(AmazonProductDataset, self).__init__()

        self.all_data = all_data
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __getitem__(self, idx):
        data_item = self.all_data[idx]
        data_encoding = self.tokenizer(data_item["reviewText"],
                                       truncation=True,
                                       max_length=self.max_length,
                                       padding="max_length",
                                       return_tensors="pt")

        return {
            "input_ids": data_encoding["input_ids"].flatten(),
            "attention_mask": data_encoding["attention_mask"].flatten(),
            "labels": torch.tensor(data_item["label"]).long(),
        }

    def __len__(self):
        return len(self.all_data)
```

Hình 8: Dataset Class

Các thông số của quá trình train:

- Chọn learning_rate đủ nhỏ cho mô hình đã được pretrain - *DistilBERT*: $5e^{-5}$
- Số lượng epoch: 4
- Train batch size: 64 (đối với RTX-3090) hoặc 16 (đối với bản free GPU của GG Colab và Kaggle).
- Evaluate batch size: 64 (đối với RTX-3090) hoặc 64 (đối với bản free GPU của GG Colab và Kaggle).
- Tỉ lệ của bộ data dùng trong quá trình huấn luyện mô hình: train (75%) evaluate (15%) test (10%).
- Optimizer: Dùng AdamW làm optimizer cho mô hình.

Số token trung bình cho các tập dữ liệu là:

train = 102.53506666666667,

evaluate = 103.54137037037037,

test = 103.62158333333333

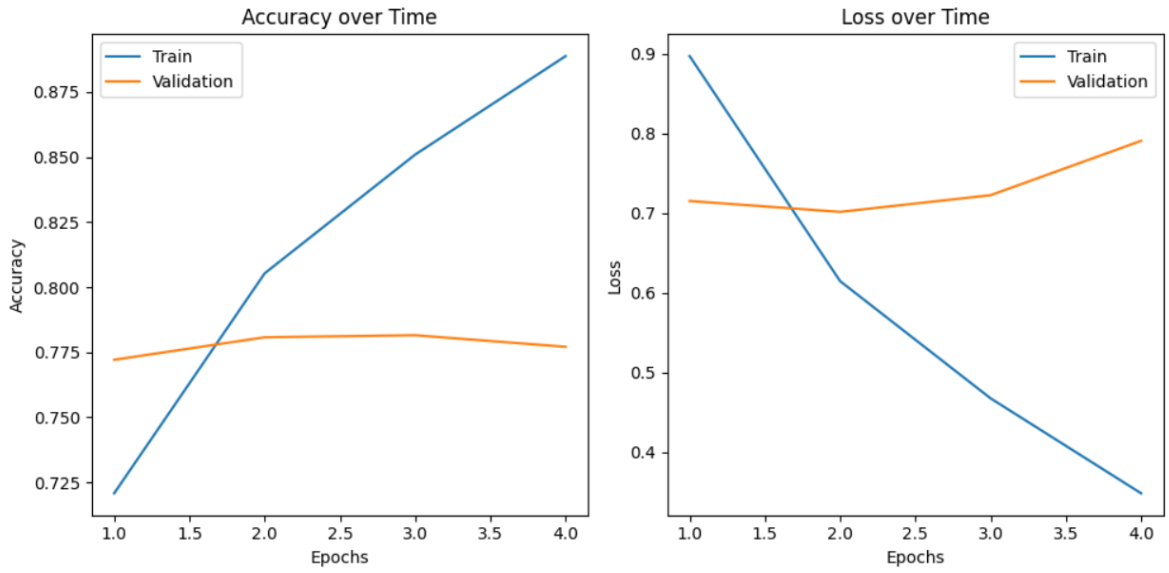
3.3. Kết quả và nhận xét

Accuracy và loss của từng epoch trong quá trình huấn luyện mô hình:

```
history = {'acc': {'train': [0.7208444444444444,
                             0.8052962962962963,
                             0.8508925925925926,
                             0.8887259259259259],
                   'val': [0.7721111111111111,
                           0.7807592592592593,
                           0.781537037037037,
                           0.7770925925925926]},
           'loss': {'train': [0.8969460922724038,
                              0.6145736904654578,
                              0.4674798861485037,
                              0.3483439381546702],
                    'val': [0.7151179275170887,
                            0.7014556299383041,
                            0.7224181491828643,
                            0.7905907433886099]}}
```

Hình 10: Có tất cả 4 epoch tương ứng, vị trí của các phần tử trong mỗi mảng thuộc về epoch tương ứng.

Trực quan hóa các thông số thu được trong quá trình huấn luyện mô hình:



Hình 9: Mô hình hóa kết quả bằng matplotlib.

Nhận xét: Ta có các giá trị trả về sau khi áp dụng mô hình đã được huấn luyện trên tập test như sau:

test_loss = 0.697785903621948

test_acc = 0.7820277777777778

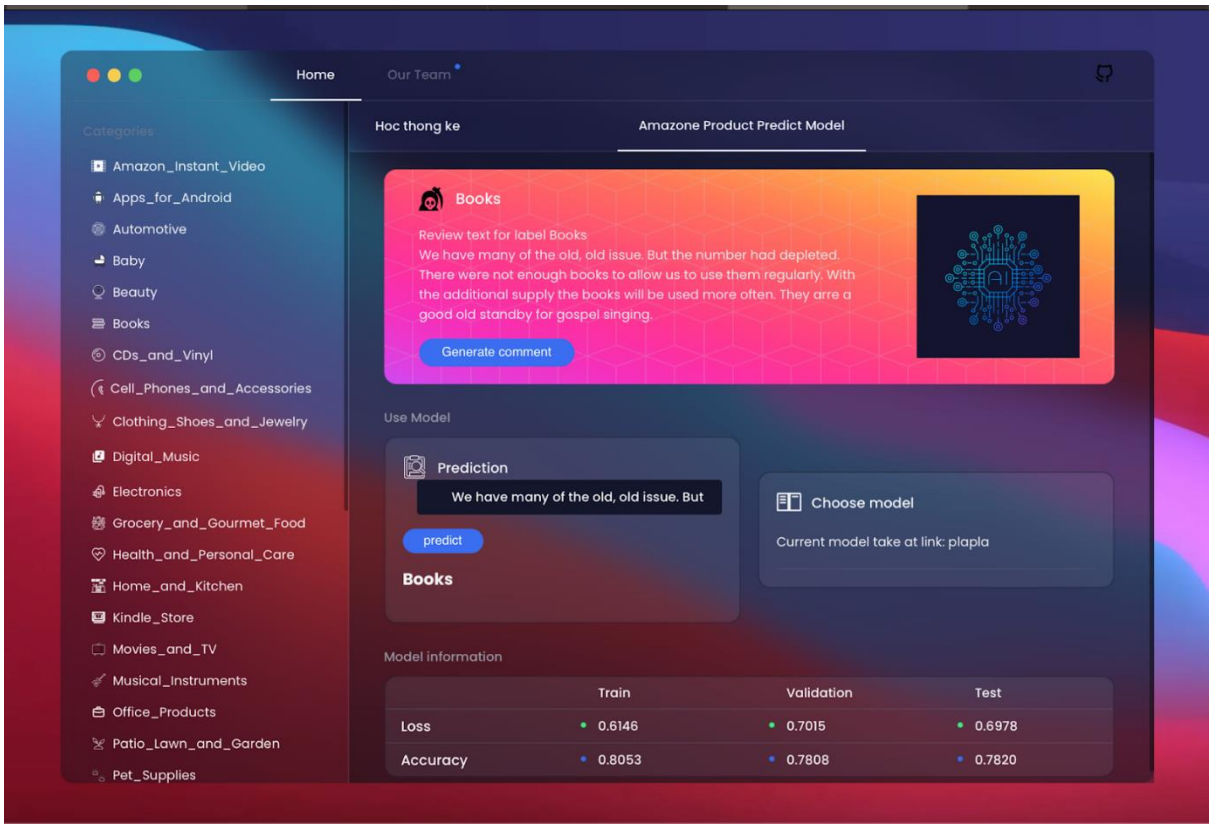
Accuracy của model hiện tại có thể được xem là không cao và khó có thể chấp nhận được với mong muốn của một bài toán phân lớp trên thực tế. Mô hình hiện tại đang có xu hướng overfit với dữ liệu được train. Hiện tượng overfit xảy ra một phần là do lượng dữ liệu đầu vào chưa đủ lớn cũng như việc sử dụng mô hình có tính hiệu quả không quá cao đối với tác vụ phân lớp.

Ý tưởng về việc cải thiện: Có thể tham khảo các mô hình chuyên dụng cho bài toán phân lớp với các kỹ thuật phù hợp hơn như: Roberta, XLM-RoBERTa, XLNet, …

Demo sản phẩm

Web được tạo ra để thử nghiệm kết quả dự đoán trên mô hình đã được huấn luyện, người dùng có thể chọn ngẫu nhiên một trong những "reviewText" được lưu trữ sẵn trong database của web (cụ thể là 10 mẫu cho mỗi label). Từ đó thực hiện dự đoán và xem xét và đánh giá kết quả đầu ra.

- Bước 1: Chọn một trong các loại thuộc tập các catagories được thể hiện bên trái.
- Bước 2: Nhấn ***generate comment*** để tự động điền một sequence "reviewText vào ô predict" hoặc dán một sequence khác vào textbox.
- Bước 3: Nhấn predict để tiến hành dự đoán.



Hình 11: giao diện web demo

*Mặt hạn chế: do giới hạn về mặt phần cứng nên web có thể response với tốc độ chậm.

Live site here

V. Đánh giá nhóm

Bảng phân công và đánh giá mức độ hoàn thành công việc

STT	MSSV	Họ và tên	Công việc	Mức độ hoàn thành
1	20120083	Nguyễn Trọng Hiếu	Tìm hiểu về BERT, xây dựng mô hình phân lớp	100%
2	20120105	Lê Hoàng Huy	Tìm bộ data phù hợp cho bài toán	100%
3	20120149	Phạm Sỹ Nguyên	Xây dựng web để demo ứng dụng của mô hình	100%
4	20120515	Luân Mã Khương	Tìm hiểu về kiến trúc Transformer	100%

VI. Tài liệu tham khảo

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (Vol. 2017-December, pp. 5999–6009). Neural information processing systems foundation.

[2] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In 1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings. International Conference on Learning Representations, ICLR.

[3] Alammar, J. (2019). The Illustrated Transformer. Blog, 1–21. Retrieved from http://jalammar.github.io/illustrated-transformer/%0Ahttps://jalammar.github.io/illustrated-transformer/

[4] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference (Vol. 1, pp. 4171–4186). Association for Computational Linguistics (ACL).

[5] Britney M. (2022, March). BERT 101: State Of The Art NLP Model Explained. HuggingFace Blog. https://huggingface.co/blog/bert-101

[6] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc VLe, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.

[7] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108

[8] Hugging Face. Tasks: Sequence Classification. Retrieved from https://huggingface.co/docs/transformers/tasks/sequence_classification

[6] McAuley, J. Amazon Product Data. Retrieved from https://cseweb.ucsd.edu/~jmcauley/datasets/amazon/links.html