Bài 39 - Thực hành ứng dụng BERT

04 Jun 2020 - phamdinhkhanh

Menu

1. BERT trong Tiếng Việt

Ở bài 36 chúng ta đã tìm hiểu về các kiến trúc của model BERT gồm BERT Base, BERT Large và những ứng dụng trong các tác vụ NLP của nó. Sự ra đời của model BERT là một cột mốc rất quan trọng của ngành NLP mà có thể phân chia thành giai đoạn phát triển trước BERT và sau BERT. Các kết quả ứng dụng BERT đã phá vỡ các giới hạn trong NLP với rất nhiều các pretrain model xác lập kết quả SOTA trong các tác vụ. Để chứng minh cho những gì tôi nói là không nhảm nhí, bạn đọc có thể theo dõi tại leader board GLUE benchmark (https://gluebenchmark.com/leaderboard). Bên cạnh đó BERT giúp cho quá trình học chuyển giao trở nên khả thi hơn khi có thể can thiệp và fine tuning mô hình học sâu nhiều tầng ở mức độ sâu thay vì can thiệp nông như các mô hình trước.

Kể từ khi google public mã nguồn mở của BERT, đã có rất nhiều các dự án mã nguồn mở về BERT hỗ trợ huấn luyện và chia sẻ các mô hình pretrain BERT trên ngôn ngữ đơn phương và song ngữ. Đối với Tiếng Việt chúng ta có PhoBERT (https://github.com/VinAIResearch/PhoBERT) . Cá nhân mình sử dụng PhoBERT thì thấy các tác vụ NLP trong Tiếng Việt được cải thiện và đạt độ chính xác cao. Bạn đọc cũng có thể tự cảm nhận qua các phần thực hành ở bài hướng dẫn này. Trong Tiếng Việt thì chúng ta có thể ứng dụng BERT trong một số tác vụ như:

- Tìm từ đồng nghĩa, trái nghĩa, cùng nhóm dựa trên khoảng cách của từ trong không gian biểu diễn đa chiều.
- Xây dựng các véc tơ embedding cho các tác vụ NLP như sentiment analysis, phân loại văn bản, NER, POS, huấn luyện chatbot.
- Gợi ý từ khóa tìm kiếm trong các hệ thống search.
- Xây dựng các ứng dụng seq2seq như robot viết báo, tóm tắt văn bản, sinh câu ngẫu nhiên với ý nghĩa tương đồng.

Top

Và nhiều những ứng dụng khác mà mình có thể chưa liệt kê hết, rất mong bạn đọc bổ sung thêm. Mặc dù model BERT có rất nhiều các ứng dụng có thể fine tuning nhưng không thực sự nhiều bạn biết cách áp dụng. Một phần là bởi để fine tuning được BERT đòi hỏi bạn phải có kỹ năng lập trình với các deep learning framework như pytorch, tensorflow và thực sự hiểu sâu về kiến trúc và nguyên lý hoạt động của BERT. Gần đây mình nhận được một vài inbox hỏi về cách áp dụng BERT như thế nào trong các tác vụ NLP. Mình đã dành một thời gian để tìm hiểu và nghiên cứu sâu về mã nguồn và tham khảo các hướng dẫn. Chính vì vậy, bài viết này mình sẽ chia sẻ lại các ứng dụng của model BERT đối với Tiếng Việt mà mình đúc kết được. Nếu bạn đọc có thêm nhiều cách ứng dụng mới của BERT trong Tiếng Việt thì mình rất vui để đón nhận chia sẻ từ các bạn.

Trước khi tìm hiểu bài này mình khuyến nghị các bạn nên đọc qua Bài 36 - BERT model (https://phamdinhkhanh.github.io/2020/05/23/BERTModel.html) để hiểu về model BERT là gì và nguyên lý hoạt động của model BERT.

2. Kiến trúc RoBERTa

RoBERTa là một project của facebook kế thừa lại các kiến trúc và thuật toán của model BERT trên framework pytorch (pytorch cũng là một framework do facebook phát triển, rất được ưa chuộng bởi cộng đồng AI). Đây là một project hỗ trợ việc huấn luyện lại các model BERT trên những bộ dữ liệu mới cho các nguôn ngữ khác ngoài một số ngôn ngữ phổ biến. Kể từ khi ra đời, đã có rất nhiều các mô hình pretrain cho những ngôn ngữ khác nhau được huấn luyện trên RoBERTa.

Ở bài báo gốc cho biết mặc dù RoBERTa lặp lại các thủ tục huấn luyện từ model BERT, nhưng có một thay đổi đó là huấn luyện mô hình lâu hơn, với batch size lớn hơn và trên nhiều dữ liệu hơn. Ngoài ra để nâng cao độ chuẩn xác trong biểu diễn từ thì RoBERTa đã loại bỏ tác vụ dự đoán câu tiếp theo và huấn luyện trên các câu dài hơn. Đồng thời mô hình cũng thay đổi linh hoạt kiểu masking (tức ẩn đi một số từ ở câu output bằng token `<mask>`) áp dụng cho dữ liệu huấn luyện.

Bạn đọc có thể tìm hiểu thêm về kiến trúc này qua bài báo về RoBERTa (https://arxiv.org/abs/1907.11692).

Ở các mục tiếp theo mình sẽ hướng dẫn các bạn triển khai áp dụng model RoBERTa thông qua pretrain model PhoBERT cho Tiếng Việt.

Để bắt đầu bài thực hành, bạn đọc có thể mở file PhoBERT - tutorial Khanh Blog (https://colab.research.google.com/drive/16a4XFPioXYzQwyTusmzi1IiGP8kCHT9t?usp=sharing) và bắt đầu từ đây.

3. Load model BERT

Để áp dụng được model BERT thì trước tiên chúng ta cần phải load được model. Ví dụ này mình sẽ thực hành trên google colab. Bạn đọc cần mount google drive bằng câu lệnh bên dưới.

```
from google.colab import drive
import os

drive.mount('/content/gdrive')
path = "/content/gdrive/My Drive/Colab Notebooks/BERT"
os.chdir(path)
!ls
```

Chúng ta sẽ cần cài đặt các dependency packages sau đây: Top

- fairseq (https://github.com/pytorch/fairseq): Là project của facebook chuyên hỗ trợ các nghiên cứu và dự án liên quan đến model seq2seq.
- fastBPE: Là package hỗ trợ tokenize từ (word) thành các từ phụ (subwords) theo phương pháp mới nhất được áp dụng cho các pretrain model NLP hiện đại như BERT và các biến thể của BERT.
- vncorenlp: Là một package NLP trong Tiếng Việt, hỗ trợ tokenize và các tác vụ NLP khác.
- transformers (https://github.com/huggingface/transformers): Là một project của huggingface hỗ trợ huấn luyện các model dựa trên kiến trúc transformer như BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet, T5, CTRL,… phục vụ cho các tác vụ NLP trên cả nền tảng pytorch và tensorflow.

```
!pip3 install fairseq
!pip3 install fastbpe
!pip3 install vncorenlp
!pip3 install transformers
```

Tiếp theo là download các model pretrain từ list các pretrain models được liệt kê trong PhoBERT (https://github.com/VinAIResearch/PhoBERT).

Trong hướng dẫn này mình chỉ sử dụng pretrain model BERT base được huấn luyện từ package fairseq. Download và giải nén chúng bằng lần lượt các lệnh `wget` và `tar`.

```
!wget https://public.vinai.io/PhoBERT_base_fairseq.tar.gz
!tar -xzvf PhoBERT_base_fairseq.tar.gz
```

Sau khi download và giải nén pretrain model chúng ta sẽ kiểm tra thấy bên trong folder sẽ bao gồm 3 files đó là `bpe.codes, dict.txt, model.pt` có tác dụng như sau:

- bpe.codes: Là BPE token mà mô hình đã áp dụng để mã hóa văn bản sang index.
- dict.txt: Từ điển subword của bộ dữ liệu huấn luyện.
- model.pt: File lưu trữ của mô hình trên pytorch.

Về BPE và subword là gì mình sẽ lý giải ở chương `4. Tìm hiểu về mã hóa BPE (Byte Pair Encoding)`.

```
!ls PhoBERT_base_fairseq
```

```
bpe.codes  dict.txt  model.pt
```

Load model pretrain PhoBERT

```
# Load the model in fairseq
from fairseq.models.roberta import RobertaModel
phoBERT = RobertaModel.from_pretrained('PhoBERT_base_fairseq', checkpo
phoBERT.eval()  # disable dropout (or leave in train mode to finetune
```

Top

```
loading archive file PhoBERT_base_fairseq
| dictionary: 64000 types



RobertaHubInterface(
  (model): RobertaModel(
    (decoder): RobertaEncoder(
      (sentence_encoder): TransformerSentenceEncoder(
        (embed_tokens): Embedding(64001, 768, padding_idx=1)
        (embed_positions): LearnedPositionalEmbedding(258, 768, paddi
        (layers): ModuleList(
          (0): TransformerSentenceEncoderLayer(
            (self_attn): MultiheadAttention(
              (k_proj): Linear(in_features=768, out_features=768, bia
              (v_proj): Linear(in_features=768, out_features=768, bia
              (q_proj): Linear(in_features=768, out_features=768, bia
              (out_proj): Linear(in_features=768, out_features=768, b
            )
            (self_attn_layer_norm): LayerNorm((768,), eps=1e-05, elem
            (fc1): Linear(in_features=768, out_features=3072, bias=Tr
            (fc2): Linear(in_features=3072, out_features=768, bias=Tr
            (final_layer_norm): LayerNorm((768,), eps=1e-05, elementw
          )
          ...
          (11): TransformerSentenceEncoderLayer(
            (self_attn): MultiheadAttention(
              (k_proj): Linear(in_features=768, out_features=768, bia
              (v_proj): Linear(in_features=768, out_features=768, bia
              (q_proj): Linear(in_features=768, out_features=768, bia
              (out_proj): Linear(in_features=768, out_features=768, b
            )
            (self_attn_layer_norm): LayerNorm((768,), eps=1e-05, elem
            (fc1): Linear(in_features=768, out_features=3072, bias=Tr
            (fc2): Linear(in_features=3072, out_features=768, bias=Tr
            (final_layer_norm): LayerNorm((768,), eps=1e-05, elementw
          )
        )
        (emb_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_af
      )
      (lm_head): RobertaLMHead(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine
      )
    )
    (classification_heads): ModuleDict()
  )
)
```

Ta có thể thấy kiến trúc RoBERTa theo BERT base đã giữa lại 12 block sub-layers là các multi-head attention ở phase Encoder và thêm một linear projection layer ở cuối để tạo ra véc tơ embedding cho từ. Ở mỗi multi-head attention, các output self-attention được khởi tạo từ quá trình nhân kết hợp giữa các ma trận chiếu Key, Value và Query. Các bạn có thể tìm hiểu về quá trình này ở Bài 4 - Attention is all you need (https://phamdinhkhanh.github.io/2019/06/18/AttentionLayer.html) của block.

Top

4. Tìm hiểu về mã hóa BPE (Byte Pair Encoding)

Toknenize là quá trình mã hóa các văn bản thành các index dạng số mang thông tin của văn bản để cho máy tính có thể huấn luyện được. Khi đó mỗi một từ hoặc ký tự sẽ được đại diện bởi một index.

Trong NLP có một số kiểu tokenize như sau:

Tokenize theo word level: Chúng ta phân tách câu thành các token được ngăn cách bởi khoảng trắng hoặc dấu câu. Khi đó mỗi token là một từ đơn âm tiết. Đây là phương pháp token được sử dụng trong các thuật toán nhúng từ truyền thống như GloVe, word2vec.

Tokenize theo multi-word level: Tiếng Việt và một số ngôn ngữ khác tồn tại từ đơn âm tiết (từ đơn) và từ đa âm tiết (từ ghép). Do đó nếu token theo từ đơn âm tiết sẽ làm nghĩa của từ bị sai khác. Ví dụ cụm từ `vô xác định` nếu được chia thành `vô`, `xác` và `định` sẽ làm cho từ bị mất đi nghĩa phủ định của nó. Do đó để tạo ra được các từ với nghĩa chính xác thì chúng ta sẽ sử dụng thêm từ điển bao gồm cả từ đa âm tiết và đơn âm để tokenize câu. Trong Tiếng Việt có khá nhiều các module hỗ trợ tokenize dựa trên từ điển như VnCoreNLP, pyvivn, underthesea.

Tokenize theo character level: Việc tokenize theo word level thường sinh ra một từ điển với kích thước rất lớn, điều này làm gia chi phí tính toán. Hơn nữa nếu tokenize theo word level thì đòi hỏi từ điển phải rất lớn thì mới hạn chế được những trường hợp từ nằm ngoài từ điển. Tuy nhiên nếu phân tích ta sẽ thấy hầu hết các từ đều có thể biểu thị dưới một nhóm các ký tự là chữ cái, con số, dấu xác định. Như vậy chỉ cần sử dụng một lượng các ký tự rất nhỏ có thể biểu diễn được mọi từ. Từ được token dựa trên level ký tự sẽ có tác dụng giảm kích thước từ điển mà vẫn biểu diễn được các trường hợp từ nằm ngoài từ điển. Đây là phương pháp được áp dụng trong mô hình fasttext.

Phương pháp mới BPE (SOTA): Nhược điểm của phương pháp tokenize theo character level đó là các token không có ý nghĩa nếu đứng độc lập. Do đó đối với các bài toán sentiment analysis, áp dụng tokenize theo character level sẽ mang lại kết quả kém hơn. Token theo word level cũng tồn tại hạn chế đó là không giải quyết được các trường hợp từ ngằm ngoài từ điển.

Một phương pháp mới đã được đề xuất trong bài báo Neural Machine Translation of Rare Words with Subword Units (https://arxiv.org/pdf/1508.07909.pdf) vào năm 2016, có khả năng tách từ theo level nhỏ hơn từ và lớn hơn ký tự được gọi là subword. Phương pháp đó chính là BPE (byte pair encoding). Theo phương pháp mới này, hầu hết các từ đều có thể biểu diễn bởi subword và chúng ta sẽ hạn chế được một số lượng đáng kể các token `<unk>` đại diện cho từ chưa từng xuất hiện trước đó. Rất nhanh chóng, Phương pháp mới đã được áp dụng ở hầu hết các phương pháp NLP hiện đại từ các lớp model BERT cho tới các biến thể của nó như OpenAI GPT, RoBERTa, DistilBERT, XLMNet. Kết quả áp dụng tokenize theo phương pháp mới đã cải thiện được độ chính xác trên nhiều tác vụ dịch máy, phân loại văn bản, dự báo câu tiếp theo, hỏi đáp, dự báo mối quan hệ văn bản.

Thuật toán BPE:

BPE (Byte Pair Encoding) là một kỹ thuật nén từ cơ bản giúp chúng ta index được toàn bộ các từ kể cả trường hợp từ mở (không xuất hiện trong từ điển) nhờ mã hóa các từ bằng chuỗi các từ phụ (subwords). Nguyên lý hoạt động của BPE dựa trên phân tích trực quan rằng hầu hết các từ đều có thể phân tích thành các thành phần con.

Chẳng hạn như từ: `low`, `lower`, `lowest` đều là hợp thành bởi `low` và những đuôi phụ `er`, `est`. Những đuôi này rất thường xuyên xuất hiện ở các từ. Như vậy khi biểu diễn từ `lower` chúng ta có thể mã hóa chúng thành hai thành phần từ phụ (subwords) tách biệt là `low` và `er`.

Top

Theo cách biểu diễn này sẽ không phát sinh thêm một index mới cho từ `lower` và đồng thời tìm được mối liên hệ giữa `lower` , `lowest` và `low` nhờ có chung thành phần từ phụ là `low` .

Phương pháp BPE sẽ thống kê tần suất xuất hiện của các từ phụ cùng nhau và tìm cách gộp chúng lại nếu tần suất xuất hiện của chúng là lớn nhất. Cứ tiếp tục quá trình gộp từ phụ cho tới khi không tồn tại các subword để gộp nữa, ta sẽ thu được tập subwords cho toàn bộ bộ văn bản mà mọi từ đều có thể biểu diễn được thông qua subwords.

Code của thuật toán BPE đã được tác giả chia sẻ tại subword-nmt (https://github.com/rsennrich/subword-nmt).

Qúa trình này gồm các bước như sau:

- Bước 1: Khởi tạo từ điển (vocabulary).
- Bước 2: Biểu diễn mỗi từ trong bộ văn bản bằng kết hợp của các ký tự với token `<\w>` ở cuối cùng đánh dấu kết thúc một từ (lý do thêm token sẽ được giải thích bên dưới).
- Bước 3: Thống kê tần suất xuất hiện theo cặp của toàn bộ token trong từ điển.
- Bước 4: Gộp các cặp có tần suất xuất hiện lớn nhất để tạo thành một n-gram theo level character mới cho từ điển.
- Bước 5: Lặp lại bước 3 và bước 4 cho tới khi số bước triển khai merge đạt đỉnh hoặc kích thước kỳ vọng của từ điển đạt được.

Bạn sẽ dễ hình dung hơn qua ví dụ bên dưới:

Gỉa sử từ điển của chúng ta gồm các từ với tần suất như sau: `vocab = {'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6, 'w i d e s t </w>': 3}` .

Coi mỗi ký tự là một token. Khi đó thống kê tần suất xuất hiện của các cặp ký tự như sau:

```
import collections

vocab = {'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6,

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

get_stats(vocab)
```

Top

```
defaultdict(int,
            {('d', 'e'): 3,
             ('e', 'r'): 2,
             ('e', 's'): 9,
             ('e', 'w'): 6,
             ('i', 'd'): 3,
             ('l', 'o'): 7,
             ('n', 'e'): 6,
             ('o', 'w'): 7,
             ('r', '</w>'): 2,
             ('s', 't'): 9,
             ('t', '</w>'): 9,
             ('w', '</w>'): 5,
             ('w', 'e'): 8,
             ('w', 'i'): 3})
```

Lựa chọn cặp từ phụ có tần suất xuất hiện nhỏ nhất và merge chúng thành một từ phụ mới.

```
import re, collections

pairs = get_stats(vocab)
best = max(pairs, key=pairs.get)
print('max pair frequency: ', best)

# Hàm merge byte max frequency

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    # Tìm kiếm các vị trí xuất hiện pair bytes
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        # Thay thế các cặp pair bytes bằng single byte gộp
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

merge_vocab(best, vocab)
```

```
max pair frequency:  ('e', 's')
```

```
{'l o w </w>': 5,
 'l o w e r </w>': 2,
 'n e w es t </w>': 6,
 'w i d es t </w>': 3}
```

Lặp lại quá trình thống kê tần suất cặp từ và gộp cặp từ với số lượt gộp là 1000

Top

```
vocab = {'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6,

def get_tokens(vocab):
    tokens = collections.defaultdict(int)
    for word, freq in vocab.items():
        word_tokens = word.split()
        for token in word_tokens:
            tokens[token] += freq
    return tokens

num_merges = 1000

for i in range(num_merges):
    pairs = get_stats(vocab)
    # max_freq = max(pairs.values())
    # if max_freq == 1:
    #   break

    if not pairs:
      break
    best = max(pairs, key=pairs.get)
    # print('best', best)
    vocab = merge_vocab(best, vocab)
    print('Iter: {}'.format(i))
    print('Best pair: {}'.format(best))
    tokens = get_tokens(vocab)
    print('Tokens: {}'.format(tokens))
    print('Number of tokens: {}'.format(len(tokens)))
    print('==========')
```

```
Iter: 10
Best pair: ('wi', 'd')
Tokens: defaultdict(<class 'int'>, {'low</w>': 5, 'low': 2, 'e': 2, '
Number of tokens: 8
==========
Iter: 14
Best pair: ('lower', '</w>')
Tokens: defaultdict(<class 'int'>, {'low</w>': 5, 'lower</w>': 2, 'ne
Number of tokens: 4
==========
```

Ta nhận thấy qua các lượt merge từ phụ, độ dài của các từ phụ trong từ điển tăng dần. Thuật toán hội tụ trước 1000 vòng lặp vì toàn bộ các từ phụ đã được merge và đạt ngưỡng của từng từ đơn.

Khi giới hạn kích thước của từ điển hoặc số lượng lượt merge ta sẽ thu được một từ điển từ phụ là thành phần của các từ trong từ điển. Khi đó mọi từ mới dường như sẽ có thể biểu diễn được theo từ phụ.

Ví dụ: Khi dừng số lượt merge tại bước 10 ta thu được từ điển: `{'low</w>': 5, 'low': 2, 'e': 2, 'r': 2, '</w>': 2, 'newest</w>': 6, 'wid': 3, 'est</w>': 3}`.

Khi đó ta có thể biểu diễn một token mới chưa từng xuất hiện trong từ điển là `wider` thành `wid e r`. Bạn đọc đã hình dung được tác dụng của từ phụ (subword) rồi chứ?

Top

Tác dụng của token </w>

Gỉa định khi tokenize câu `the highest mountain` theo từ phụ ta thu được biểu diễn `['the</w>', 'high', 'est</w>', 'moun', 'tain</w>']` . Khi đó để khôi phục được thành câu gốc ta chỉ cần nối các token lại theo thứ tự thành `the</w>highest</w>mountain</w>` . Chỉ cần thay `</w>` bằng khoảng trắng ta sẽ khôi phục được câu gốc: `the highest mountain` .

token `</w>` được thêm vào cuối mỗi từ để phân biệt các từ phụ nằm ở vị trí cuối câu với các vị trí khác để giúp cho việc giải mã token khả thi hơn.

Áp dụng BPE tokenize trong BERT:

Hầu hết các mô hình NLP hiện đại nhất đều đã chuyển sang tokenize theo BPE. Để sử dụng BPE tokenize từ các model pretrain của BERT ta thực hiện như sau:

Load model pretrain `RoBERTa`

```
# Load the model in fairseq
from fairseq.models.roberta import RobertaModel
phoBERT = RobertaModel.from_pretrained('PhoBERT_base_fairseq', checkpo
phoBERT.eval()  # disable dropout (or leave in train mode to finetune
```

Khai báo bpe tokenizer và thực hiện token.

```
from fairseq.data.encoders.fastbpe import fastBPE

# Khởi tạo Byte Pair Encoding cho PhoBERT
class BPE():
  bpe_codes = 'PhoBERT_base_fairseq/bpe.codes'

args = BPE()
phoBERT.bpe = fastBPE(args) #Incorporate the BPE encoder into PhoBERT
tokens = phoBERT.encode('Tôn Ngộ Không phò Đường Tăng đi Tây Trúc thỉ
print('tokens list : ', tokens)
# Decode ngược lại thành câu từ chuỗi index token
phoBERT.decode(tokens)  # 'Hello world!'
```

```
tokens list :  tensor([    0, 11623, 31433,   453, 44334,  2080,  5922
        31686,  3078,     2])


'Tôn Ngộ Không phò Đường Tăng đi Tây Trúc thỉnh kinh'
```

Số lượng các token là 13 lớn hơn số lượng các từ là 11 là vì BERT đã tự thêm các ký tự `<s>` và `</s>` đánh dầu từ bắt đầu và kết thúc câu.

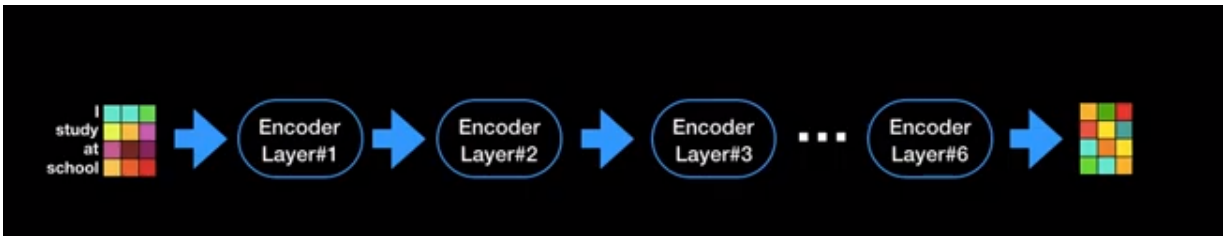
5. Extract features từ RoBERTa

Có 2 versions chính trả về 2 kích thước embedding khác nhau khi huấn luyện theo RoBERTa đó là:

Top

- `BERT base` : 12 sub-layers, kích thước embedding 768, số lượng head attention là 12.
- `BERT large` : 24 sub-layers, kích thước embedding 1024, số lượng head attention là 16.

Chúng ta có thể trích xuất được các đặc trưng được tạo ra từ BERT của phase Encoder tại layers cuối cùng hoặc toàn bộ các layers. Ngoài phương án trích suất véc tơ embedding tại layer cuối cùng, một số tác vụ classification trong NLP đã áp dụng trích suất đặc trưng từ từ những layers trước đó chẳng hạn như trong tác vụ PhoBERT Sentiment Classification (https://github.com/suicao/PhoBert-Sentiment-Classification) tác giả đã trích suất đặc trưng từ 4 layers cuối cùng thay vì chỉ trích suất từ một layer cuối.



Hình 1: Kiến trúc gồm nhiều layers tại encoder của model BERT. Mô hình huấn luyện từ RoBERTa cho phép ta trích suất các đặc trưng từ những layers của encoder. Có thể là layer cuối hoặc toàn bộ các layers.

Kích thước output của mỗi một layer sẽ là `batch_size x seq_len x d_model` . Phương pháp trích suất như sau:

```
# Extract the last layer's features
last_layer_features = phoBERT.extract_features(tokens)
# assert last_layer_features.size() == torch.Size([1, 5, 1024])
print('token size: ', tokens.size())
print('size of last layer: ', last_layer_features.size())

# Extract all layer's features (layer 0 is the embedding layer)
all_layers = phoBERT.extract_features(tokens, return_all_hiddens=True
print('number layer in all layers: ', len(all_layers))

# last_layer_features must equal to last layer in all_layers:
print('Last layer features: ', all_layers[-1] == last_layer_features)
```

```
token size:  torch.Size([13])
size of last layer:  torch.Size([1, 13, 768])
number layer in all layers:  13
Last layer features:  tensor([[[True, True, True,  ..., True, True, T
         [True, True, True,  ..., True, True, True],
         [True, True, True,  ..., True, True, True],
         ...,
         [True, True, True,  ..., True, True, True],
         [True, True, True,  ..., True, True, True],
         [True, True, True,  ..., True, True, True]]])
```

6. Điền từ (Filling mask)

Trong bài toán này chúng ta sẽ điền các từ hợp lý vào các vị trí còn trống của câu. Trên thực tế có rất nhiều ứng dụng của bài toàn filling mask như xây dựng hệ thống suggestion search, gợi ý gõ văn bản, tìm từ đồng nghĩa, tagging.

Top

Mô hình BERT tạo ra các biểu diễn từ từ quá trình ẩn các vị trí token một cách ngẫu nhiên trong câu input và dự báo chính chính từ đó ở output dựa trên bối cảnh là các từ xung quanh.

Như vậy khi đã biết các từ xung quanh, chúng ta hoàn toàn có thể dự báo được từ phù hợp nhất với vị trí đã được masking.

Down load package VnCoreNLP để tokenize các câu văn.

```
Download VnCoreNLP-1.1.1.jar & its word segmentation component (i.e. RD
!mkdir -p vncorenlp/models/wordsegmenter
!wget https://raw.githubusercontent.com/vncorenlp/VnCoreNLP/master/VnC
!wget https://raw.githubusercontent.com/vncorenlp/VnCoreNLP/master/mod
!wget https://raw.githubusercontent.com/vncorenlp/VnCoreNLP/master/mod
!mv VnCoreNLP-1.1.1.jar vncorenlp/
!mv vi-vocab vncorenlp/models/wordsegmenter/
!mv wordsegmenter.rdr vncorenlp/models/wordsegmenter/
```

Gỉa sử chúng ta có câu gốc là `Tôn Ngộ Không phò Đường Tăng đi thỉnh kinh tại Tây Trúc` . Từ được ẩn đi trong câu là `phò` sẽ được thay thế bằng token `<mask>` .

```
from vncorenlp import VnCoreNLP
rdrsegmenter = VnCoreNLP("vncorenlp/VnCoreNLP-1.1.1.jar", annotators=
text = 'Tôn Ngộ Không phò Đường Tăng đi thỉnh kinh tại Tây Trúc'
text_masked = 'Học sinh được  <mask> do dịch covid-19'
# Tokenize câu gốc và thay từ phò bằng <mask>
words = rdrsegmenter.tokenize(text)[0]
for i, token in enumerate(words):
  if token == 'phò':
    words[i] = ' <mask>'
text_masked_tok = ' '.join(words)
print('text_masked_tok: \n', text_masked_tok)
```

```
text_masked_tok:
 Tôn_Ngộ_Không  <mask> Đường Tăng đi thỉnh_kinh tại Tây_Trúc
```

Tìm ra top 10 từ thích hợp nhất cho vị trí `<mask>` tại câu trên.

Top

```
from fairseq.data.encoders.fastbpe import fastBPE
from fairseq import options
import numpy as np

# Khởi tạo Byte Pair Encoding cho PhoBERT
class BPE():
  bpe_codes = 'PhoBERT_base_fairseq/bpe.codes'
args = BPE()
phoBERT.bpe = fastBPE(args) #Incorporate the BPE encoder into PhoBERT

# Filling marks
topk_filled_outputs = phoBERT.fill_mask(text_masked_tok, topk=10)
topk_probs = [item[1] for item in topk_filled_outputs]
print('Total probability: ', np.sum(topk_probs))
print('Input sequence: ', text_masked_tok)
print('Top 10 in mask: ')
for i, output in enumerate(topk_filled_outputs):
  print(output[0])
```

```
Total probability:  0.8735223989933729
Input sequence:  Tôn_Ngộ_Không  <mask> Đường Tăng đi thỉnh_kinh tại Tâ
Top 10 in mask:
Tôn_Ngộ_Không và Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không đưa Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không cõng Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không hộ_tống Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không cùng Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không chở Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không theo Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không dẫn Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không , Đường Tăng đi thỉnh_kinh tại Tây_Trúc
Tôn_Ngộ_Không tháp_tùng Đường Tăng đi thỉnh_kinh tại Tây_Trúc
```

Ta thấy các từ ở vị trí `<mask>` tìm được khá tự nhiên và ngữ nghĩa của câu không khác mấy so với con người tạo ra. Sự chuẩn xác và tự nhiên có được dựa trên quá trình huấn luyện mô hình pretrain trên một bộ dữ liệu có kích thước rất lớn (khoảng 20GB).

7. Trích suất đặc trưng (Extract feature) cho từ

Sau khi load được model BERT, chúng ta hoàn toàn có thể trích suất đặc trưng cho một từ bất kỳ từ pretrain model. Từ các véc tơ nhúng được trích suất cho một từ hoặc một câu, chúng ta có thể đo lường similarity để tìm ra các câu tương đồng về nội dung hoặc các từ đồng nghĩa. Một ứng dụng khác đó là chúng ta có thể tận dụng các biểu diễn ngữ nghĩa của từ thông qua véc tơ embedding được huấn luyện từ BERT để chuyển giao sang các tác vụ phân loại văn bản, phân tích cảm xúc bình luận. Phương pháp tiếp cận sẽ tương tự như áp dụng các model GloVe, word2vec, fasttext trong học nông (shallow learning).

Các véc tơ embedding cho từng từ trong câu từ mô hình BERT được trích suất như sau:

Top

```
from fairseq.data.encoders.fastbpe import fastBPE

# Khởi tạo Byte Pair Encoding cho PhoBERT
class BPE():
  bpe_codes = 'PhoBERT_base_fairseq/bpe.codes'

args = BPE()
phoBERT.bpe = fastBPE(args) #Incorporate the BPE encoder into PhoBERT
doc = phoBERT.extract_features_aligned_to_words('học_sinh cấp 3 được
for tok in doc:
    print('{:10}{} (...) {}'.format(str(tok), tok.vector[:5], tok.vec
```

```
<s>       tensor([ 0.0534,  0.1301, -0.0475, -0.8371,  0.3862], grad_
học_sinh  tensor([ 0.1764,  0.1603,  0.0792, -0.6043, -0.3138], grad_
cấp       tensor([ 0.0679,  0.0194,  0.3450, -0.4951, -0.6394], grad_
3         tensor([-0.0465, -0.3846,  0.1337, -1.1276,  0.1910], grad_
được      tensor([ 0.1920, -0.0146,  0.2933,  0.0086,  0.0690], grad_
đến       tensor([-0.0108, -0.6463, -0.2906, -0.0317,  0.0561], grad_
trường    tensor([-0.0270,  0.2676,  0.3856,  0.3514,  0.1169], grad_
sau       tensor([-0.1175,  0.4808,  0.0772, -0.2991,  0.0147], grad_
nghỉ      tensor([ 0.4385,  0.4162,  0.1529, -0.1419, -0.1928], grad_
dịch      tensor([ 0.2958, -0.0976,  0.2024, -0.9278,  0.0270], grad_
covid     tensor([ 0.1539,  0.2343,  0.4054, -1.6919, -0.7180], grad_
</s>      tensor([ 0.0558,  0.0341, -0.0286, -0.6476,  0.4656], grad_
```

Khi đó mỗi từ sẽ được biểu diễn bằng 768 chiều là số chiều của hidden véc tơ trong mô hình BERT base.

8. Bài toán classification

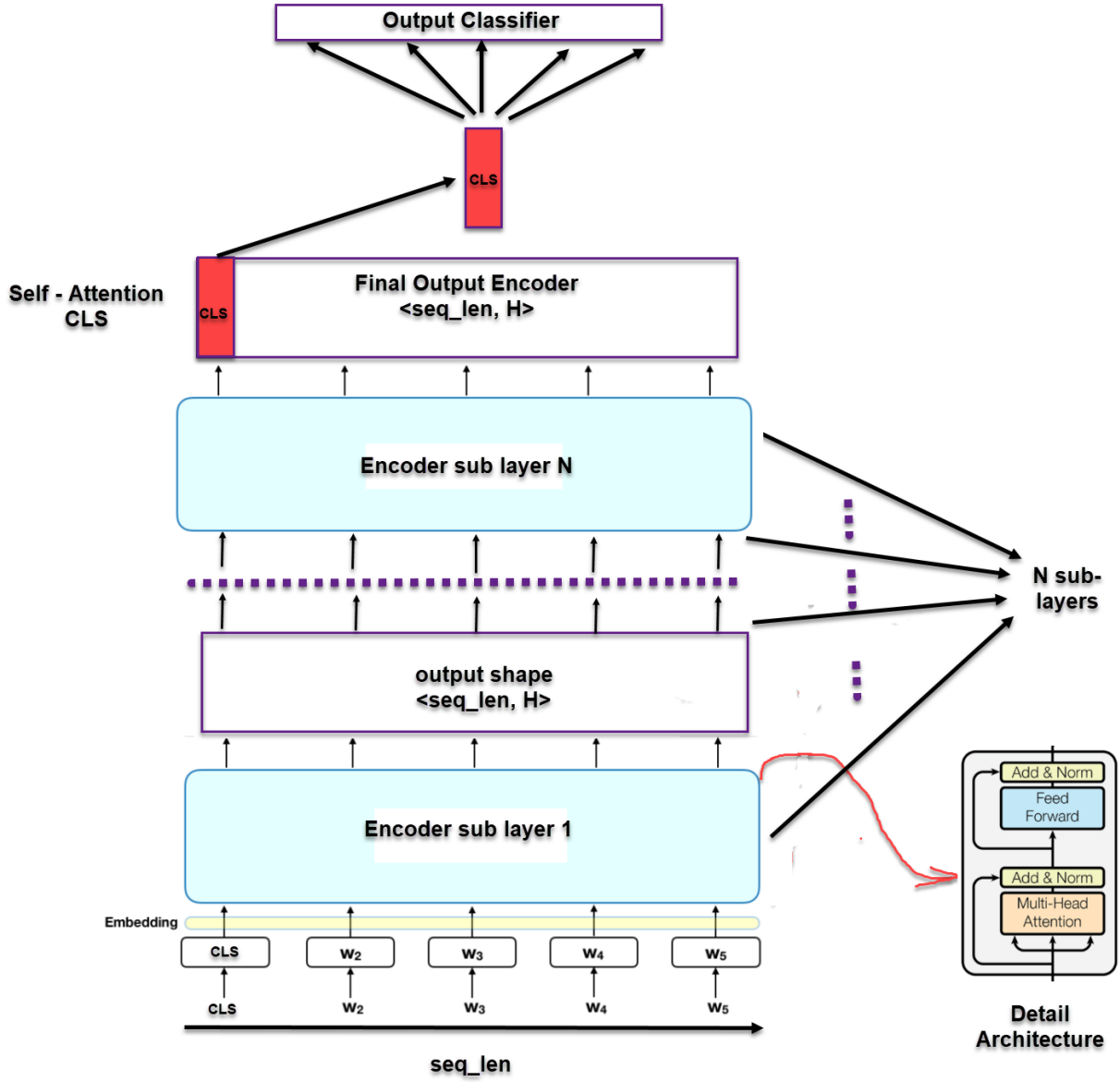
8.1. Kiến trúc mô hình

Ý tưởng fine-tuning được lấy từ bài báo How to Fine-Tune BERT for Text Classification? (https://arxiv.org/abs/1905.05583).

Model BERT base sẽ tạo ra một kiến trúc gồm 12 sub-layers ở encoder, 12 heads trong multi-head attention trên mỗi sub-layer. Output là tập hợp các véc tơ self-attention bằng chiều dài của input. Mỗi véc tơ có kích thước là 768.

Để fine-tuning lại kiến trúc của BERT cho tác vụ phân loại văn bản (text classification). Chúng ta truncate decoder của BERT, giữa nguyên kiến trúc encoder của transformer và sau đó trích suất ra biểu diễn véc tơ của token `CLS` đánh dấu vị trí đầu tiên. Véc tơ này sẽ được sử dụng làm đầu vào cho thuật toán classifier bằng cách thêm một linear projection layer (cũng chính là fully connected layer) ở cuối có kích thước bằng với số classes cần phân loại. Cụ thể hơn chúng ta cùng xem kiến trúc bên dưới.

Top



Hình 1: Kiến trúc fine-tuning classifier của BERT trong classification. Biểu diễn self-attention của token tại vị trí `CLS` được sử dụng làm input cho thuật toán phân loại. Chúng ta thêm một linear projection layer ở cuối cùng để tính toán phân phối xác suất.

Để lấy ví dụng cho quá trình fine-tuning lại PhoBERT cho tác vụ phân loại văn bản mình sẽ huấn luyện model phân loại topics báo chí. Chúng ta sẽ tìm hiểu về bộ dữ liệu cho mô hình.

8.2. Dữ liệu

Dữ liệu mà mình sử dụng là VNTC (https://github.com/duyvuleo/VNTC.git) với các bài báo đã được sắp xếp theo 10 topics. Bộ dữ liệu bao gồm 33 nghìn bài báo trên tập train và 50 nghìn bài báo trên tập test có phân bố số lượng theo topics như sau:

Top

```
***Train***
Topic   Topic ID        #files
********************************************************************************
Chinh tri Xa hoi        XH      5219
Doi song        DS              3159
Khoa hoc        KH              1820
Kinh doanh      KD              2552
Phap luat       PL              3868
Suc khoe        SK              3384
The gioi        TG              2898
The thao        TT              5298
Van hoa         VH              3080
Vi tinh         VT              2481

Total                           33759

***Test***
Chinh tri Xa hoi        XH      7567
Doi song        DS              2036
Khoa hoc        KH              2096
Kinh doanh      KD              5276
Phap luat       PL              3788
Suc khoe        SK              5417
The gioi        TG              6716
The thao        TT              6667
Van hoa         VH              6250
Vi tinh         VT              4560

Total                           50373

*** Notes:
        * For Train & Test
                + DS_VNE_(...) : VnExpress news agency (http://vnexpress.net/)
                + DS_TT_(...):  Youth news agency (http://tuoitre.vn/)
                + DS_TN_(...): Thanh Nien news agency (http://thanhnien.vn/)
                + DS_NLD_(...): Nguoi Lao Dong news agency (http://nld.com.vn/)
```

Dữ liệu sau xử lý được mình chia sẻ. Nếu không muốn tìm hiểu quá trình tạo dữ liệu, bạn đọc có thể chuyển qua mục `Tokenize Input và output` và bỏ qua bước này.

8.2.1. Đọc và lưu dữ liệu

```
1     !git clone https://github.com/duyvuleo/VNTC.git
2     !ls VNTC/Data/10Topics/Ver1.1
```

Sau khi đã download dữ liệu về, chúng ta sẽ đọc và lưu các bài báo vào những list chứa nội dung và nhãn tương ứng theo 2 folders train và test.

Top

```
import glob2
from tqdm import tqdm

train_path = 'Train_Full/*/*.txt'
test_path = 'Test_Full/*/*.txt'

# Hàm đọc file txt
def read_txt(path):
  with open(path, 'r', encoding='utf-16') as f:
    data = f.read()
  return data

# Hàm tạo dữ liệu huấn luyện cho tập train và test
def make_data(path):
  texts = []
  labels = []
  for file_path in tqdm(glob2.glob(train_path)):
    try:
      content = read_txt(file_path)
      label = file_path.split('/')[1]
      texts.append(content)
      labels.append(label)
    except:
      next
  return texts, labels

text_train, label_train = make_data(train_path)
text_test, label_test = make_data(test_path)
```

Quá trình đọc files sẽ tốn khá nhiều thời gian. Do đó các bạn có thể tạo các hàm lưu trữ lại các list nội dung và nhãn và load lại cho lượt huấn luyện sau.

```
import pickle

def _save_pkl(path, obj):
  with open(path, 'wb') as f:
    pickle.dump(obj, f)

def _load_pkl(path):
  with open(path, 'rb') as f:
    obj = pickle.load(f)
  return obj

# Lưu lại các files
_save_pkl('text_train.pkl', text_train)
_save_pkl('label_train.pkl', label_train)
_save_pkl('text_test.pkl', text_test)
_save_pkl('label_test.pkl', label_test)
```

```
print('text content:\n', text_train[0])
print('label:\n', label_train[0])
```

Top

```
text content:
 Tấm hít nhỏ xinh
Tủ lạnh hay phía tường trước bàn làm việc của bạn sẽ đẹp hơn nếu có nl
Chuẩn bị: nam châm dày 3 mm; gỗ mỏng; sơn; keo dán gỗ; cọ, cưa.
Thực hiện:
Bước 1: Cưa 3 mảnh gỗ vuông làm nền, diện tích 4 cm2. Bạn có thể thay
Bước 2: Dùng keo dán những mảnh gỗ nhỏ vào mảnh gỗ nền. Cần chú ý phố
Chú ý: Chọn loại gỗ thật mỏng, nếu không sản phẩm trông rất thô. Khôn



label:
 Doi song
```

8.2.2. Tokenize nội dung

Tiếp theo ta sẽ tokenize các câu văn sang chuỗi index và padding câu văn về cũng một độ dài.

```
max_sequence_length = 500

def convert_lines(lines, vocab, bpe):
  '''
  lines: list các văn bản input
  vocab: từ điển dùng để encoding subwords
  bpe:
  '''
  # Khởi tạo ma trận output
  outputs = np.zeros((len(lines), max_sequence_length)) # --> shape (
  # Index của các token cls (đầu câu), eos (cuối câu), padding (paddi
  cls_id = 0
  eos_id = 2
  pad_id = 1

  for idx, row in tqdm(enumerate(lines), total=len(lines)):
    # Mã hóa subwords theo byte pair encoding(bpe)
    subwords = bpe.encode('<s> '+ row +' </s>')
    input_ids = vocab.encode_line(subwords, append_eos=False, add_if_
    # Truncate input nếu độ dài vượt quá max_seq_len
    if len(input_ids) > max_sequence_length:
      input_ids = input_ids[:max_sequence_length]
      input_ids[-1] = eos_id
    else:
      # Padding nếu độ dài câu chưa bằng max_seq_len
      input_ids = input_ids + [pad_id, ]*(max_sequence_length - len(i

    outputs[idx,:] = np.array(input_ids)
  return outputs
```

8.3. Tokenize Input và output

Các bạn có thể download lại dữ liệu $\mathbf{X}, \mathbf{y}$ mà tôi đã chuẩn bị cho huấn luyện tại Dữ liệu Tokenize (https://drive.google.com/drive/folders/1stRredI0fZ2vE5_SKGggrgDxnV1bxhr1?usp=sharing) và bỏ qua bước này. Thực hiện luôn bước tiếp theo `Load model BERT`. Top

- Chuẩn bị X input: Tokenize nội dung các văn bản sang chuỗi indices.

- Chuẩn bị y output: Encoding các label output thành indices đánh dấu số thứ tự của văn bản.

```
from tqdm import tqdm
import torch

max_sequence_length = 256
def convert_lines(lines, vocab, bpe):
  '''
  lines: list các văn bản input
  vocab: từ điển dùng để encoding subwords
  bpe:
  '''
  # Khởi tạo ma trận output
  outputs = np.zeros((len(lines), max_sequence_length), dtype=np.int3
  # Index của các token cls (đầu câu), eos (cuối câu), padding (paddi
  cls_id = 0
  eos_id = 2
  pad_id = 1

  for idx, row in tqdm(enumerate(lines), total=len(lines)):
    # Mã hóa subwords theo byte pair encoding(bpe)
    subwords = bpe.encode('<s> '+ row +' </s>')
    input_ids = vocab.encode_line(subwords, append_eos=False, add_if_
    # Truncate input nếu độ dài vượt quá max_seq_len
    if len(input_ids) > max_sequence_length:
      input_ids = input_ids[:max_sequence_length]
      input_ids[-1] = eos_id
    else:
      # Padding nếu độ dài câu chưa bằng max_seq_len
      input_ids = input_ids + [pad_id, ]*(max_sequence_length - len(i

    outputs[idx,:] = np.array(input_ids)
  return outputs

# Load the dictionary
vocab = Dictionary()
vocab.add_from_file("PhoBERT_base_transformers/dict.txt")


# Test encode lines
lines = ['Học_sinh được nghỉ học bắt dầu từ tháng 3 để tránh dịch cov
[x1, x2] = convert_lines(lines, vocab, phoBERT.bpe)
print('x1 tensor encode: {}, shape: {}'.format(x1[:10], x1.size))
print('x1 tensor decode: ', phoBERT_cls.decode(torch.tensor(x1))[:103
```

Top

```
from tqdm import tqdm
import torch

max_sequence_length = 256
def convert_lines(lines, vocab, bpe):
  '''
  lines: list các văn bản input
  vocab: từ điển dùng để encoding subwords
  bpe:
  '''
  # Khởi tạo ma trận output
  outputs = np.zeros((len(lines), max_sequence_length), dtype=np.int3
  # Index của các token cls (đầu câu), eos (cuối câu), padding (paddi
  cls_id = 0
  eos_id = 2
  pad_id = 1

  for idx, row in tqdm(enumerate(lines), total=len(lines)):
    # Mã hóa subwords theo byte pair encoding(bpe)
    subwords = bpe.encode('<s> '+ row +' </s>')
    input_ids = vocab.encode_line(subwords, append_eos=False, add_if_
    # Truncate input nếu độ dài vượt quá max_seq_len
    if len(input_ids) > max_sequence_length:
      input_ids = input_ids[:max_sequence_length]
      input_ids[-1] = eos_id
    else:
      # Padding nếu độ dài câu chưa bằng max_seq_len
      input_ids = input_ids + [pad_id, ]*(max_sequence_length - len(i

    outputs[idx,:] = np.array(input_ids)
  return outputs

# Load the dictionary
vocab = Dictionary()
vocab.add_from_file("PhoBERT_base_transformers/dict.txt")


# Test encode lines
lines = ['Học_sinh được nghỉ học bắt dầu từ tháng 3 để tránh dịch cov
[x1, x2] = convert_lines(lines, vocab, phoBERT_cls.bpe)
print('x1 tensor encode: {}, shape: {}'.format(x1[:10], x1.size))
print('x1 tensor decode: ', phoBERT_cls.decode(torch.tensor(x1))[:103
```

Như vậy ta thấy rằng các câu văn đã được encode về token index. Từ token index có thể decode ngược trở lại thành câu input sau khi đã thêm các token đặc biệt đánh dấu vị trí bắt dầu: `<s>` , kết thúc: `</s>` câu và các vị trí nằm ngoài câu: `<pad>` . Ta sẽ token toàn bộ câu input sang index như sau:

```
X = convert_lines(text_train, vocab, phoBERT_cls.bpe)
print('X shape: ', X.shape)
```

Sau cùng ta thu được các chuỗi index có kích thước là 256, bằng với kích thước của các câu sau khi đã padding. Tiếp theo ta tạo output `y` bằng index cho các nhãn của câu.

Top

```
from sklearn.preprocessing import LabelEncoder
lb = LabelEncoder()
lb.fit(label_train)
y = lb.fit_transform(label_train)
print(lb.classes_)
print('Top 5 classes indices: ', y[:5])
```

Lưu lại dữ liệu $\mathbf{X}$ và $\mathbf{y}$

```
# Save dữ liệu
_save_pkl('PhoBERT_pretrain/X1.pkl', X)
_save_pkl('PhoBERT_pretrain/y1.pkl', y)
_save_pkl('PhoBERT_pretrain/labelEncoder1.pkl', lb)

# Load lại dữ liệu
X = _load_pkl('PhoBERT_pretrain/X1.pkl')
y = _load_pkl('PhoBERT_pretrain/y1.pkl')

print('length of X: ', len(X))
print('length of y: ', len(y))
```

8.4. Load model BERT

Tiếp theo ta sẽ load pretrain model BERT từ file weight mà chúng ta đã download trước đó. Để tùy chỉnh mô hình cho phù hợp với tác vụ phân loại văn bản. Chúng ta sẽ thêm vào sau cùng pretrain model một head layer là linear projection có số units ở output bằng với số lượng classes cần phân loại và bằng 10 thông qua hàm `phoBERT_cls.register_classification_head('new_task', num_classes=10)` . Cụ thể như sau:

```
# Load the model in fairseq
from fairseq.models.roberta import RobertaModel
from fairseq.data.encoders.fastbpe import fastBPE
from fairseq.data import Dictionary

phoBERT_cls = RobertaModel.from_pretrained('PhoBERT_base_fairseq', ch
phoBERT_cls.eval()  # disable dropout (or leave in train mode to fine

# Load BPE
class BPE():
  bpe_codes = 'PhoBERT_base_fairseq/bpe.codes'

args = BPE()
phoBERT_cls.bpe = fastBPE(args) #Incorporate the BPE encoder into Phol

# Add header cho classification với số lượng classes = 10
phoBERT_cls.register_classification_head('new_task', num_classes=10)
tokens = 'Học_sinh được nghỉ học bắt đầu từ tháng 3 do ảnh hưởng của
token_idxs = phoBERT_cls.encode(tokens)
logprobs = phoBERT_cls.predict('new_task', token_idxs)  # tensor([[-1
logprobs
```

Top

```
loading archive file PhoBERT_base_fairseq
| dictionary: 64000 types




tensor([[-2.3722, -2.1128, -2.2945, -2.3484, -2.2294, -2.1600, -2.5104
         -2.4144, -2.2299]], grad_fn=<LogSoftmaxBackward>)
```

8.5. Huấn luyện model

Trước khi huấn luyện mô hình chúng ta sẽ xây dựng các hàm đánh giá mô hình theo 2 metric là `accuracy` và `f1_score`.

```
import torch
import numpy as np
from sklearn.metrics import accuracy_score, f1_score

def evaluate(logits, targets):
    """
    Đánh giá model sử dụng accuracy và f1 scores.
    Args:
        logits (B,C): torch.LongTensor. giá trị predicted logit cho c
        targets (B): torch.LongTensor. actual target indices.
    Returns:
        acc (float): the accuracy score
        f1 (float): the f1 score
    """
    # Tính accuracy score và f1_score
    logits = logits.detach().cpu().numpy()
    y_pred = np.argmax(logits, axis = 1)
    targets = targets.detach().cpu().numpy()
    f1 = f1_score(targets, y_pred, average='weighted')
    acc = accuracy_score(targets, y_pred)
    return acc, f1

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
logits = torch.tensor([[0.1, 0.2, 0.7],
                       [0.4, 0.1, 0.5],
                       [0.1, 0.2, 0.7]]).to(device)
targets = torch.tensor([1, 2, 2]).to(device)
evaluate(logits, targets)
```

```
(0.6666666666666666, 0.5333333333333333)
```

Top

```
def validate(valid_loader, model, device):
    model.eval()
    accs = []
    f1s = []
    with torch.no_grad():
        for x_batch, y_batch in valid_loader:
            x_batch = x_batch.to(device)
            y_batch = y_batch.to(device)
            outputs = model.predict('new_task', x_batch)
            logits = torch.exp(outputs)
            acc, f1 = evaluate(logits, y_batch)
            accs.append(acc)
            f1s.append(f1)

    mean_acc = np.mean(accs)
    mean_f1 = np.mean(f1s)
    return mean_acc, mean_f1
```

Hàm huấn luyện mô hình trên từng epoch.

```
def trainOnEpoch(train_loader, model, optimizer, epoch, num_epochs, c
    model.train()
    sum_epoch_loss = 0
    sum_acc = 0
    sum_f1 = 0
    start = time.time()
    for i, (x_batch, y_batch) in enumerate(train_loader):
      x_batch = x_batch.to(device)
      y_batch = y_batch.to(device)
      optimizer.zero_grad()
      y_pred = model.predict('new_task', x_batch)
      logits = torch.exp(y_pred)
      acc, f1 = evaluate(logits, y_batch)
      loss = criteria(y_pred, y_batch)
      loss.backward()
      optimizer.step()

      loss_val = loss.item()
      sum_epoch_loss += loss_val
      sum_acc += acc
      sum_f1 += f1
      iter_num = epoch * len(train_loader) + i + 1

      if i % log_aggr == 0:
            print('[TRAIN] epoch %d/%d  observation %d/%d batch loss:
                % (epoch + 1, num_epochs, i, len(train_loader), loss_
                  len(x_batch) / (time.time() - start)))
      start = time.time()
```

Quá trình huấn luyện một model classification trên pytorch sẽ bao gồm những bước chính sau đây:

- Khởi tạo DataLoader để quản lý dữ liệu đưa vào huấn luyện và thẩm định.
- Thiết lập kiến trúc mô hình.

Top

- Khai báo hàm loss function.
- Phương pháp optimization giúp tối ưu loss function.
- Huấn luyện mô hình qua các epochs.

Bên dưới chúng ta sẽ lần lượt thực hiện các bước trên.

Top

```
import os
import time
import random
import argparse
import pickle
import numpy as np
from tqdm import tqdm
from os.path import join

import torch
from torch import nn
from torch.utils.data import DataLoader
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
from torch.autograd import Variable
from torch.backends import cudnn
from sklearn.model_selection import StratifiedKFold

# Load the model in fairseq
from fairseq.models.roberta import RobertaModel
from fairseq.data.encoders.fastbpe import fastBPE
from fairseq.data import Dictionary
from transformers.modeling_utils import *
from transformers import *

# Khởi tạo argument
EPOCHS = 20
BATCH_SIZE = 6
ACCUMULATION_STEPS = 5
FOLD = 4
LR = 0.0001
LR_DC_STEP = 80
LR_DC = 0.1
CUR_DIR = os.path.dirname(os.getcwd())
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
FOLD = 4
CKPT_PATH2 = 'model_ckpt2'

if not os.path.exists(CKPT_PATH2):
    os.mkdir(CKPT_PATH2)

# Khởi tạo DataLoader
splits = list(StratifiedKFold(n_splits=5, shuffle=True, random_state

for fold, (train_idx, val_idx) in enumerate(splits):
    best_score = 0
    if fold != FOLD:
        continue
    print("Training for fold {}".format(fold))

    # Create dataset
    train_dataset = torch.utils.data.TensorDataset(torch.tensor(X[tr
    valid_dataset = torch.utils.data.TensorDataset(torch.tensor(X[va

    # Create DataLoader
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_
```

Top

```
58          valid_loader = torch.utils.data.DataLoader(valid_dataset, batch_
59
60          # Khởi tạo model:
61          MODEL_LAST_CKPT = os.path.join(CKPT_PATH2, 'latest_checkpoint.pt
62          if os.path.exists(MODEL_LAST_CKPT):
63            print('Load checkpoint model!')
64            phoBERT_cls = torch.load(MODEL_LAST_CKPT)
65          else:
66            print('Load model pretrained!')
67            # Load the model in fairseq
68            from fairseq.models.roberta import RobertaModel
69            from fairseq.data.encoders.fastbpe import fastBPE
70            from fairseq.data import Dictionary
71
72            phoBERT_cls = RobertaModel.from_pretrained('PhoBERT_base_fairs
73            phoBERT_cls.eval()  # disable dropout (or leave in train mode
74
75            # # Load BPE
76            # class BPE():
77            #   bpe_codes = 'PhoBERT_base_fairseq/bpe.codes'
78
79            # args = BPE()
80            # phoBERT_cls.bpe = fastBPE(args) #Incorporate the BPE encoder
81
82            # Add header cho classification với số lượng classes = 10
83            phoBERT_cls.register_classification_head('new_task', num_class
84
85          ## Load BPE
86          print('Load BPE')
87          class BPE():
88            bpe_codes = 'PhoBERT_base_fairseq/bpe.codes'
89
90          args = BPE()
91          phoBERT_cls.bpe = fastBPE(args) #Incorporate the BPE encoder int
92          phoBERT_cls.to(DEVICE)
93
94          # Khởi tạo optimizer và scheduler, criteria
95          print('Init Optimizer, scheduler, criteria')
96          param_optimizer = list(phoBERT_cls.named_parameters())
97          no_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']
98          optimizer_grouped_parameters = [
99              {'params': [p for n, p in param_optimizer if not any(nd in n
100             {'params': [p for n, p in param_optimizer if any(nd in n for
101         ]
102
103         num_train_optimization_steps = int(EPOCHS*len(train_dataset)/BAT
104         optimizer = AdamW(optimizer_grouped_parameters, lr=LR, correct_b
105         scheduler = get_linear_schedule_with_warmup(optimizer, num_warmu
106         scheduler0 = get_constant_schedule(optimizer)  # scheduler với h
107         # optimizer = optim.Adam(phoBERT_cls.parameters(), LR)
108         criteria = nn.NLLLoss()
109         # scheduler = StepLR(optimizer, step_size = LR_DC_STEP, gamma =
110         avg_loss = 0.
111         avg_accuracy = 0.
112         frozen = True
113         for epoch in tqdm(range(EPOCHS)):
114             # warm up tại epoch đầu tiên, sau epoch đầu sẽ phá băng các
115             if epoch > 0 and frozen:
```

Top

```
                for child in phoBERT_cls.children():
                    for param in child.parameters():
                        param.requires_grad = True
                frozen = False
                del scheduler0
                torch.cuda.empty_cache()
            # Train model on EPOCH
            print('Epoch: ', epoch)
            trainOnEpoch(train_loader=train_loader, model=phoBERT_cls, o
            # scheduler.step(epoch = epoch)
            # Phá băng layers sau epoch đầu tiên
            if not frozen:
                scheduler.step()
            else:
                scheduler0.step()
            optimizer.zero_grad()
            # Validate on validation set
            acc, f1 = validate(valid_loader, phoBERT_cls, device=DEVICE)
            print('Epoch {} validation: acc: {:.4f}, f1: {:.4f} \n'.form
            # Store best model checkpoint
            ckpt_dict = {
                'epoch': epoch + 1,
                'state_dict': phoBERT_cls.state_dict(),
                'optimizer': optimizer.state_dict()
            }
            # Save model checkpoint into 'latest_checkpoint.pth.tar'
            torch.save(ckpt_dict, MODEL_LAST_CKPT)
```

```
  0%|          | 0/20 [00:00<?, ?it/s] [A

Init Optimizer, scheduler, criteria
...
[TRAIN] epoch 4/20  observation 2700/4502 batch loss: 2.2600 (avg 2.25
```

Thời gian huấn luyện sẽ khá lâu, các bạn nên kiên nhẫn chờ đợi. Mình chỉ dừng ở epochs số 4 cho mục đích demo. Mặc dù kết quả chỉ đạt 16% accuracy nhưng accuracy luôn tăng. Learning rate của mô hình cũng được thiết lập khá nhỏ để tránh nhảy khỏi điểm tối ưu toàn cục (global optimal value). Ngoài cách fine tuning model từ fairseq như trên, các bạn có thể tham khảo thêm một cách khác của PhoBERT-Sentiment-Classification Khoi Nguyen (https://github.com/suicao/PhoBert-Sentiment-Classification/) thực hiện fine tuning dựa trên pretrain model huấn luyện từ transformers.

9. Huấn luyện RoBERTa trên dữ liệu của bạn

Ngoài ra chúng ta có thể tự huấn luyện pretrain model BERT dựa trên kiến trúc RoBERTa theo hướng dẫn tại RoBERTa - README (https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.pretraining.md). Việc thực hiện khá đơn giản, bài đã khá dài nên mình gửi link cho bạn đọc tự nghiên cứu.

10. Tổng kết

Top

Như vậy mình đã giới thiệu với các bạn rất nhiều các ứng dụng khác nhau trong việc áp dụng các model pretrain RoBERTa. Trong đó có các tác vụ chính như: điền từ (filling mask), suggest search, phân loại văn bản, phân loại cảm xúc và tìm từ đồng nghĩa, trái nghĩa.

Một số tác vụ khác của NLP có thể fine tuning được từ mô hình RoBERTa như ứng dụng hỏi đáp, sinh văn bản ngẫu nhiên, sinh văn bản có nội dung tương tự nhưng đòi hỏi phải có những bộ dữ liệu chuyên biệt cho các tác vụ này. Nhưng dữ liệu như vậy cho Tiếng Việt đang rất thiếu và hiếm. Bạn đọc cũng có thể thực hành trên các bộ dữ liệu của Tiếng Anh theo hướng dẫn tại mục fine tuning - RoBERTa (https://github.com/pytorch/fairseq/tree/master/examples/roberta).

Như vậy qua bài viết này các bạn đã nắm bắt được các phương tiện và công cụ mới trong việc tiếp cận các bài toán của NLP. Đây là những phương pháp rất mạnh và hứa hẹn sẽ mang lại nhiều cải thiện đáng kể cho các tác vụ NLP của bạn. Đừng quên like và share bài viết này nếu bạn cảm thấy kiến thức mình chia sẻ là hữu ích với bạn.

11. Tài liệu

1. faiseq (https://github.com/pytorch/fairseq)
2. RoBERTa (https://github.com/pytorch/fairseq/tree/master/examples/roberta)
3. transformers (https://github.com/huggingface/transformers)
4. PhoBERT-Sentiment-Classification (https://github.com/suicao/PhoBert-Sentiment-Classification/)
5. PhoBERT (https://github.com/VinAIResearch/PhoBERT)

Top