20240404 Meeting

312707003黃鈺婷

Question-answering 管道

- 輸入的文本可能長於模型能接受的最大長度

  -> 使用truncation

```
inputs = tokenizer(question, long_context, max_length=384, truncation="only_second")
print(tokenizer.decode(inputs["input_ids"]))
```

- token最大長度為 384，若tokens總長度超過384，則會進行截斷
- 只有長篇文本會被截斷，問題不會被截斷
- 問題:若選擇答案在末尾的問題，當我們截斷它時，答案不存在

Question-answering 管道

- 將上下文分成更小的塊，指定最大長度，還包括塊之間的一些重疊

```
sentence = "This sentence is not too long but we are going to split it anyway."
inputs = tokenizer(
        sentence, truncation=True, return_overflowing_tokens=True, max_length=6, stride=2
)
```

```
'[CLS] This sentence is not [SEP]'
'[CLS] is not too long [SEP]'
'[CLS] too long but we [SEP]'
'[CLS] but we are going [SEP]'
'[CLS] are going to split [SEP]'
'[CLS] to split it anyway [SEP]'
'[CLS] it anyway. [SEP]'
```

- 每塊的token最大長度為6
- 每塊會有兩個token重疊

Question-answering 管道

```
dict_keys(['input_ids', 'attention_mask', 'overflow_to_sample_mapping'])
```

- 當將多個句子標記在一起時，可看出每個結果對應哪個句子

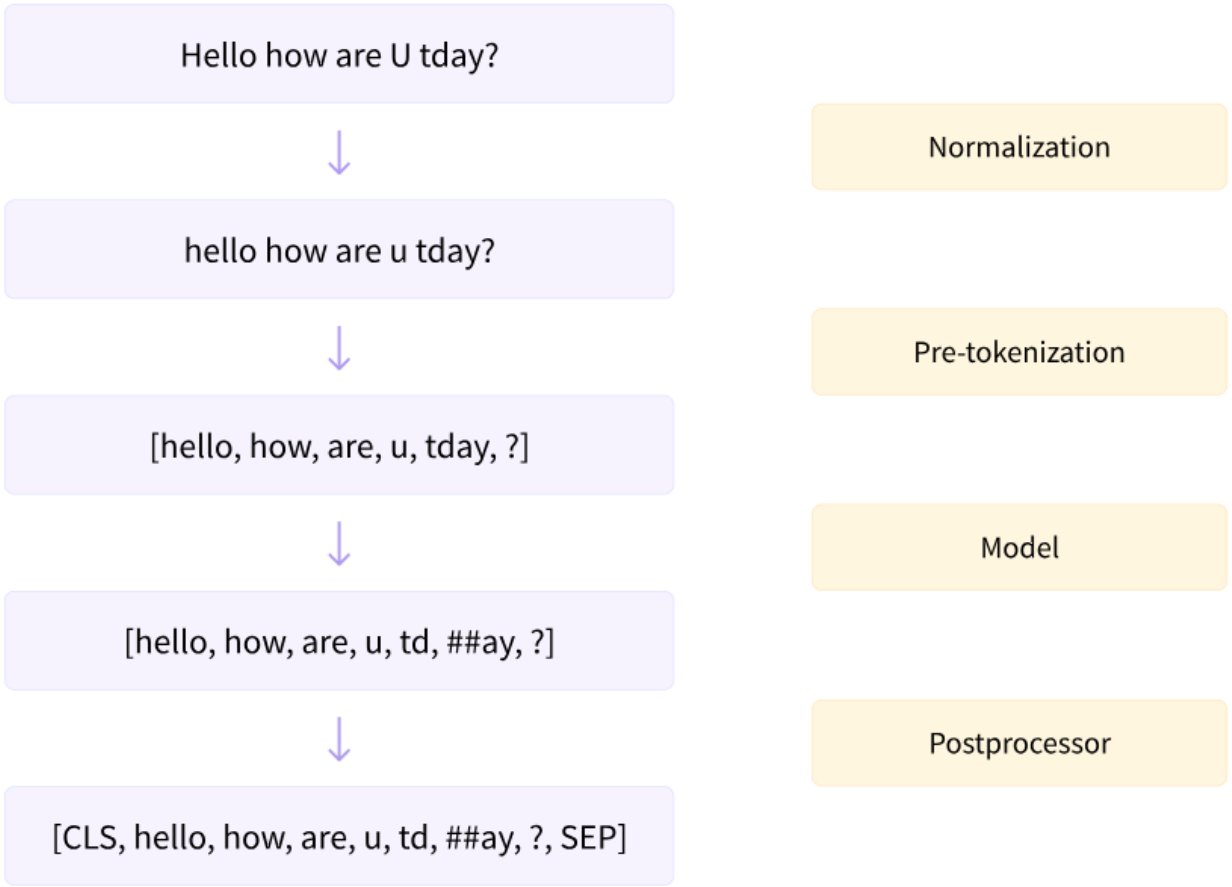
```
sentences = [
        "This sentence is not too long but we are going to split it anyway.",
        "This sentence is shorter but will still get split.",
]
inputs = tokenizer(
        sentences, truncation=True, return_overflowing_tokens=True, max_length=6, stride=2
)

print(inputs["overflow_to_sample_mapping"])
```

```
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
```

標計化管道中的步驟

- 在將文本拆分為子標記之前，分詞器執行兩個步驟：normalization和pre-tokenization.



Normalization

- 涉及一些常規清理，例如刪除不必要的空格、小寫和/或刪除重音符號

```
from  transformers  import  AutoTokenizer

tokenizer  =  AutoTokenizer.from_pretrained("bert-base-uncased")
```

- 選擇bert-base-uncased檢查點，標準化會用小寫並刪除重音

```
print(tokenizer.backend_tokenizer.normalizer.normalize_str("Héllò  hôw  are  ü?"))
```

```
'hello how are u?'
```

Pre-tokenization

- 因為分詞器不能單獨在原始文本上進行訓練，所以需要將文本拆分為小實體，例如單詞

  -> 預標記化

- 三種主要的子詞標記化算法

1. BPE(GPT、GPT-2、RoBERTa、DeBERTa)
2. WordPiece(BERT)
3. Unigram(T5)

字節對編碼(BPE)標記化

- 在完成標準化和預標記化步驟之後，先計算語料庫中的**單詞集**，然後通過獲取用於編寫這些單詞的所有符號來**構建詞彙表**
- 語料庫的五個詞 `"hug", "pug", "pun", "bun", "hugs"`
- 基礎詞彙將是 ["b", "g", "h", "n", "p", "s", "u"]
- 獲得這個基本詞彙後，透過學習**合併**達到所需的詞彙量

字節對編碼(BPE)標記化

- 合併規則：最頻繁的一對將被合併

```
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

- ("u", "g"), 出現在"hug","pug","hugs"中, 在詞彙表中總共 20 次
- 第一個合併規則是 (“u”, “g”) -> “ug”, “ug”將會被添加到詞彙表中, 並且這對應該合併到語料庫的所有單詞中

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

字節對編碼(BPE)標記化

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

- 現在最頻繁的一對是("h", "ug"),所以我們學習了合併規則("h", "ug") -> "hug"

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]
Corpus: ("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)
```

- 繼續合併, 直到達到所需的詞彙量

字節對編碼(BPE)標記化

- 標記化緊跟訓練過程, 透過應用以下步驟對新輸入進行標記:

1. 正規化
2. 預標記化
3. 將單詞拆分為單個字符
4. 將學習的合併規則按順序應用於這些拆分

- 假設有以下三個合併規則，基礎詞彙 ["b", "g", "h", "n", "p", "s", "u"]

```
("u", "g") -> "ug"
("u", "n") -> "un"
("h", "ug") -> "hug"
```

"bug"將被標記為 ["b", "ug"]

"mug",將被標記為 ["[UNK]", "ug"]

"thug" 會被標記為 ["[UNK]", "hug"]

BPE 算法的實現

```
corpus = [
        "This is the Hugging Face course.",
        "This chapter is about tokenization.",
        "This section shows several tokenizer algorithms.",
        "Hopefully, you will be able to understand how they are trained and generate tokens.",
]
```

- 需要將該語料庫預先標記為單詞，因複製BPE標記器，我們將使用gpt2標記器作為預標記化的標記器

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

BPE 算法的實現

- 進行預標記化時計算語料庫中每個單詞的頻率

```
from collections import defaultdict
word_freqs = defaultdict(int)
for text in corpus:
        words_with_offsets = tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str(text)
        new_words = [word for word, offset in words_with_offsets]
        for word in new_words:
                word_freqs[word] += 1
print(word_freqs)
```

```
defaultdict(int, {'This': 3, 'Ġis': 2, 'Ġthe': 1, 'ĠHugging': 1, 'ĠFace': 1, 'ĠCourse': 1, '.': 4, 'Ġchapter': 1,
        'Ġabout': 1, 'Ġtokenization': 1, 'Ġsection': 1, 'Ġshows': 1, 'Ġseveral': 1, 'Ġtokenizer': 1, 'ĠalgorithmsĠ': 1,
        'Hopefully': 1, ',': 1, 'Ġyou': 1, 'Ġwill': 1, 'Ġbe': 1, 'Ġable': 1, 'Ġto': 1, 'Ġunderstand': 1, 'Ġhow': 1,
        'Ġthey': 1, 'Ġare': 1, 'Ġtrained': 1, 'Ġand': 1, 'Ġgenerate': 1, 'Ġtokens': 1})
```

BPE 算法的實現

- 計算基本詞彙, 由語料庫中使用的所有字符組成

```
[',', '.', 'C', 'F', 'H', 'T', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'k', 'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'y', 'z', 'Ġ']
```

- 計算每對的頻率

```python
def compute_pair_freqs(splits):
    pair_freqs = defaultdict(int)
    for word, freq in word_freqs.items():
        split = splits[word]
        if len(split) == 1:
            continue
        for i in range(len(split) - 1):
            pair = (split[i], split[i + 1])
            pair_freqs[pair] += freq
    return pair_freqs
```

```
('T', 'h'): 3
('h', 'i'): 3
('i', 's'): 5
('Ġ', 'i'): 2
('Ġ', 't'): 7
('t', 'h'): 3
```

BPE 算法的實現

- 找到最頻繁的對

```
best_pair = ""
max_freq = None

for pair, freq in pair_freqs.items():
    if max_freq is None or max_freq < freq:
        best_pair = pair
        max_freq = freq

print(best_pair, max_freq)
```

```
('Ġ', 't') 7
```

BPE 算法的實現

- 合併

```
def merge_pair(a, b, splits):
    for word in word_freqs:
        split = splits[word]
        if len(split) == 1:
            continue
        i = 0
        while i < len(split) - 1:
            if split[i] == a and split[i + 1] == b:
                split = split[:i] + [a + b] + split[i + 2 :]
            else:
                i += 1
        splits[word] = split
    return splits
```

- ('Ġ', 't') -> 'Ġt'

BPE 算法的實現

- 重複前面的步驟，直到達到目標詞彙量

```
vocab_size = 50

while len(vocab) < vocab_size:
    pair_freqs = compute_pair_freqs(splits)
    best_pair = ""
    max_freq = None
    for pair, freq in pair_freqs.items():
        if max_freq is None or max_freq < freq:
            best_pair = pair
            max_freq = freq
    splits = merge_pair(*best_pair, splits)
    merges[best_pair] = best_pair[0] + best_pair[1]
    vocab.append(best_pair[0] + best_pair[1])
```

BPE 算法的實現

- 新的詞彙表(由特殊標記、初始字母和所有合併結果組成)

```
['<|endoftext|>', ',', '.', 'C', 'F', 'H', 'T', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm', 'n', 'o′',
 'p', 'r', 's', 't', 'u', 'v', 'w', 'y', 'z', 'Ġ', 'Ġt', 'is', 'er', 'Ġa', 'Ġto', 'en', 'Th', 'This', 'ou', 'se′',
'Ġtok', 'Ġtoken', 'nd', 'Ġis', 'Ġth', 'Ġthe', 'in', 'Ġab', 'Ġtokeni']
```

```
tokenize("This is not a token.")
```

```
['This', 'Ġis', 'Ġ', 'n', 'o', 't', 'Ġa', 'Ġtoken', '.']
```