به نام خدا

پدرام رستمی – ۸۱۰۱۰۰۳۵۳

تمرین اول پردازش زبان طبیعی

گام اول – تفاوت الگوریتم‌های BPE و WordPiece

الگوریتم‌های BPE و WordPiece هر دو از الگوریتم‌های tokenization برای زیر کلمات (subwords) هستند. در این دسته از الگوریتم‌ها بخش‌های جزئی‌تر و پرتکرار کلمات نیز به عنوان token ایجاد می‌شوند. برای مثال کلمه‌ی "following" می‌تواند به token های "follow" و "ing" تبدیل شود.

در الگوریتم BPE ابتدا کلمات متن به همراه تعداد تکرار آنان به کمک یک الگوریتم ساده مانند جداسازی با space به دست می‌آیند. سپس همه‌ی کاراکترهای استفاده شده در کلمات به عنوان vocabulary انتخاب می‌شوند. سپس به صورت مکرر پرتکرارترین زوج‌های کلمات در vocabulary را با هم ادغام کرده و به vocabulary اضافه می‌کنیم. شرط توقف این الگوریتم می‌تواند تعداد کلمات vocabulary باشد.

الگوریتم WordPiece بسیار شبیه به الگوریتم BPE است. تنها تفاوت این الگوریتم با الگوریتم BPE در انتخاب زوج‌های کلمات در vocabulary است. در الگوریتم WordPiece به جای انتخاب زوج کلماتی که بیشترین تکرار را دارند، زوج کلماتی انتخاب می‌شوند که likelihood را در داده‌های آموزش بیشینه می‌کنند. شرط توقف این الگوریتم هم محدودیت برای حداکثر سایز vocabulary است.

برای پیاده سازی الگوریتم BPE ابتدا لازم است تا کلمات corpus به کمک جداسازی به وسیله‌ی space به دست آیند و حروف استفاده شده در آن‌ها و فرکانس تکرارشان به دست آید. این عملیات‌ها در بخش زیر از کد پیاده سازی BPE انجام می‌شود.

```
words = corpus.split()
word_freq_dict = defaultdict(int)
for word in words:
    chars = list(word)
    chars.append('__')
    key = tuple(chars)
    word_freq_dict[key] += 1
vocabulary = defaultdict(int)
for word, freq in word_freq_dict.items():
    for char in word:
        vocabulary[char] += freq
```

سپس در یک حلقه که محدود کننده تعداد توکن‌های تولید شده در vocabulary است، شروع به تولید توکن‌های جدید می‌کنیم. ابتدا تمام pair هایی از توکن‌های موجود در vocabulary که در کلمات موجود است را می‌شماریم. در صورتی که هیچ pair ای برای شمردن وجود نداشته باشد، به این معناست که کلمات به بزرگ‌ترین توکن‌های ممکن تبدیل شده‌اند و دیگر کلمه‌ای وجود ندارد که از توکن‌های کوچک‌تر به وجود آید. پس از تمام شدن این بخش، pair توکنی که بیشترین فرکانس را دارد به عنوان بهترین pair برای ترکیب انتخاب می‌شود و به vocabulary اضافه می‌شود. کد مربوط به این عملیات‌ها در تصویر زیر قابل مشاهده است.

```
while len(vocabulary.keys()) < 30:
    pairs_freq_dict = defaultdict(int)
    for word, freq in word_freq_dict.items():
        for i, next in zip(word, word[1:]):
            pairs_freq_dict[i + next] += freq
    if not pairs_freq_dict:
        break
    best_pair = max(pairs_freq_dict, key=pairs_freq_dict.get)
    vocabulary[best_pair] = pairs_freq_dict[best_pair]
```

پس از به دست آوردن پر تکرارترین pair از توکن‌ها، باید آن‌ها را در کلمات تبدیل به یک توکن کنیم. برای این کار، بر روی تمام pair توکن‌های کنار هم کلمات حلقه زده و در صورتی که برابر با بهترین pair توکن‌ها بودند، آن‌ها را با همدیگر تکرار می‌کنیم و همچنین از تعداد فرکانس‌های توکن‌های تشکیل دهنده‌ی بهترین pair توکن‌ها، فرکانس کلماتی که شامل آن‌ها است را کم می‌کنیم. کد مربوط به این بخش در تصویر زیر قابل مشاهده است.

```
    word_freq_dict_temp = word_freq_dict.copy()
    for word, freq in word_freq_dict_temp.items():
        new_word = []
        skip = False
        for i, next in zip(word, word[1:]):
            if not skip:
                if i + next == best_pair:
                    new_word.append(i + next)
                    vocabulary[i] -= freq
                    vocabulary[next] -= freq
                    skip = True
                else:
                    new_word.append(i)
            else:
                skip = False
        if not skip:
            new_word.append(word[-1])
        new_word = tuple(new_word)
        if word != new_word:
            del word_freq_dict[word]
            word_freq_dict[new_word] = freq
```

در انتها توکن‌هایی که بر اثر ترکیب با توکن‌های دیگر، فرکانسشان صفر شده است را از vocabulary حذف کرده و بقیه‌ی کلمات را به عنوان vocabulary برمی‌گردانیم. در کد زیر این بخش آورده شده است. همچنین نشان داده شده است که کلمه‌ی lowest در vocabulary وجود ندارد.

```
            temp_vocabulary = vocabulary.copy()
            for word, freq in temp_vocabulary.items():
                if freq == 0:
                    del vocabulary[word]
            return list(vocabulary.keys())

        vocabulary = BPE(corpus)
        if "lowest__" in vocabulary:
            print("'lowest' is in vocabulary")
        else:
            print("'lowest' is not in vocabulary")
[3]  ✓ 0.9s
...  'lowest' is not in vocabulary
```

گام دوم

در حالت کلی، مدل‌هایی که با داده‌های wikitext آموزش داده شدند، عملکرد بهتری داشتند. زیرا مجموعه داده‌ی wikitext بسیار بزرگتر از مجموعه داده‌ی یکی از کتاب‌های gutenberg است. در نتیجه کلمات بیشتری مشاهده می‌شوند و مدلی با کارایی بهتر می‌سازند.

همچنین در مجموع عملکرد دو مدل BPE و WordPiece تفاوت بسیار عمده‌ای با یکدیگر نداشتند و هر دو عملکرد تقریبا مناسبی داشتند. هر دو مدل اموجی را به عنوان توکن UNK تشخیص دادند.

گام سوم

نام الگوریتم استفاده شده برای توکنایز	تعداد توکن‌های خروجی الگوریتم برای کتاب گوتنبرگ	
	توکنایزر آموزش داده شده بر روی کتاب گوتنبرگ	توکنایزر آموزش داده شده بر روی کل داده‌های ویکی‌پدیا
Byte Pair Encoding(BPE)	۱۲۲۸۸۵	۱۴۰۹۵۸
WordPiece	۱۲۲۸۰۴	۱۴۰۸۳۱

** برای اجرای تمام کدهای مرتبط با دیتاست‌های wikitext و Gutenberg باید فایل‌های داده را در فولدری به نام data و در کنار پروژه‌ی اصلی قرار دهید.