بنام خدا

دانشکده‌ی مهندسی برق و کامپیوتر

درس پردازش زبان‌های طبیعی

تمرین کامپیوتری 1

روز آپلود : 24 اسفند | **Tokenization** | استاد : دکتر فیلی

سرفصل مطالب

گام اول:

بررسی دو الگوریتم BPE و WPC :

شباهت و تفاوت دو الگوریتم را در زیر شرح می‌دهیم :

:BPE

- BPE از یک Vocabulary خالی شروع کرده و با توجه به تعداد تکرار Sub-word ها (frequency)، اقدام به merge آنها می‌کند. pair های ادغام شده به Vocabulary اضافه می‌شوند و Sub-word های تشکیل دهنده pair جدید، از Corpus حذف می‌شود. افزودن به Vocabulary می‌تواند با یک Hyper-parameter بسته به کاربرد کنترل شود.
- pair های ایجاد شده در Vocabulary نهایی ممکن است بی‌معنا باشند و برای ورودی متن جدید امکان دارد کاربردی نباشند.

:WPC

- WordPiece به BPE از این نظر شباهت دارد که در ابتدا Vocabulary از Character ها و Symbol ها تشکیل می‌دهد و شروع به اضافه کردن به Vocabulary می‌کند تا به حد معینی برسد.
- تفاوت اصلی الگوریتم WPC با BPE در انتخاب Sub-word ها برای merge شدن می‌باشد. در BPE این انتخاب بر اساس تعداد(frequency) می‌باشد ولی در WPC روش انتخاب بر مبنای بیشینه کردن Likelihood در Corpus می‌باشد. در واقع در هر مرحله بروز رسانی Vocabulary، یک Language model بر روی آن آموزش داده می‌شود و بهترین pair که احتمال مشاهده را بیشینه می‌کند، انتخاب می‌شود و به Vocabulary اضافه می‌شود.

پیاده‌سازی الگوریتم BPE قدم به قدم :

در اولین مرحله، نیاز به تابعی داریم که با بروزرسانی Vocabulary، تعداد تکرار حروف مجاور(Adjacent) در قالب یک dictionary برگرداند. اسم تابع نوشته شده، update_freq می‌باشد:

```
import re, collections
from collections import defaultdict

def update_freq(vocab):
    pairs = defaultdict(int)
    for word, freq in vocab.items():
        #symbols = [w for w in word]
```

```
        symbols = word.split()
        #print(symbols)
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs
```

در ادامه، خروجی این تابع برای اولین مرحله که هنوز Vocabulary خالی می‌باشد را می‌بینیم :

```
train_data =
{'l o w -': 5, 'l o w e r -': 2, 'n e w e s t -': 6, 'w i d e s t -': 3}
update_freq(train_data)
```

```
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', '-'): 2,
             ('s', 't'): 9,
             ('t', '-'): 9,
             ('w', '-'): 5,
             ('w', 'e'): 8,
             ('w', 'i'): 3})
```

حال نیاز داریم با توجه به خروجی بدست آمده در بالا، بهترین pair را در هر مرحله انتخاب کنیم :

```
def detect_best(MyDict):
  ref=0
  sel=(",")
  for tup,num in MyDict.items():
    if num > ref :
      sel=tup
      ref=num
  return sel

sel=detect_best(update_freq(train_data))
sel
```

که خروجی ('e', 's') می‌باشد.

در ادامه تابعی می‌نویسیم که با تشخیص بهترین pair در بالا، Corpus را بروزرسانی کند :

```
def reconst_corpus(pair,data):
  for string,num in data.items():
```

```
    if pair[0]+' '+pair[1] in string:
      data[string.replace(pair[0]+' '+pair[1],pair[0]+pair[1])] = data.pop(string)
  return data


reconst_corpus(('t', '-'),train_data)
```

```
{'l o w -': 5, 'l o w e r -': 2, 'n e w e s t-': 6, 'w i d e s t-': 3}
```

نکته مهم این است که، Corpus به روز شده در این مرحله، ورودی تابع update_freq تعریف شده در ابتدا می‌باشد. در نهایت برای شرط توقف الگوریتم ، تعداد pair های ایجاد شده در Vocabulary را برابر 10 قرار می‌دهیم .

خروجی نهایی به همراه تغییرات ایجاد شده در Vocabulary و Corpus به صورت زیر می‌باشد :

```
Iteration 1:

new merge: ('e', 's')
train data: {'l o w -': 5, 'l o w e r -': 2, 'n e w es t -': 6, 'w i d es t -': 3}

Iteration 2:

new merge: ('es', 't')
train data: {'l o w -': 5, 'l o w e r -': 2, 'n e w est -': 6, 'w i d est -': 3}

Iteration 3:

new merge: ('est', '-')
train data: {'l o w -': 5, 'l o w e r -': 2, 'n e w est-': 6, 'w i d est-': 3}

Iteration 4:

new merge: ('l', 'o')
train data: {'n e w est-': 6, 'w i d est-': 3, 'lo w -': 5, 'lo w e r -': 2}

Iteration 5:

new merge: ('lo', 'w')
train data: {'n e w est-': 6, 'w i d est-': 3, 'lo w e r -': 2, 'low -': 5}

Iteration 6:

new merge: ('n', 'e')
train data: {'w i d est-': 3, 'lo w e r -': 2, 'low -': 5, 'ne w est-': 6}

Iteration 7:

new merge: ('ne', 'w')
```

```
train data: {'w i d est-': 3, 'lo w e r -': 2, 'low -': 5, 'new est-': 6}

Iteration 8:

new merge: ('new', 'est-')
train data: {'w i d est-': 3, 'lo w e r -': 2, 'low -': 5, 'newest-': 6}

Iteration 9:

new merge: ('low', '-')
train data: {'w i d est-': 3, 'lo w e r -': 2, 'newest-': 6, 'low-': 5}

Iteration 10:

new merge: ('w', 'i')
train data: {'lo w e r -': 2, 'newest-': 6, 'low-': 5, 'wi d est-': 3}
```

نحوه ادغام Sub-word ها در زیر به صورت خلاصه آورده شده است :

```
{'es': ('e', 's'),
 'est': ('es', 't'),
 'est-': ('est', '-'),
 'lo': ('l', 'o'),
 'low': ('lo', 'w'),
 'low-': ('low', '-'),
 'ne': ('n', 'e'),
 'new': ('ne', 'w'),
 'newest-': ('new', 'est-'),
 'wi': ('w', 'i')}
```

برای اینکه پیاده سازی کلمه ”Lowest“ را نشان دهیم، ابتدا باید ترتیب pair شدن Sub-word ها را در یک dictionary ذخیره کنیم :

```
{('e', 's'): 0,
 ('es', 't'): 1,
 ('est', '-'): 2,
 ('l', 'o'): 3,
 ('lo', 'w'): 4,
 ('low', '-'): 8,
 ('n', 'e'): 5,
 ('ne', 'w'): 6,
 ('new', 'est-'): 7,
 ('w', 'i'): 9}
```

حال کافی است تابعی بنویسیم که در هر مرحله حروف مجاور کلمه بروز شده OOV را پیدا کنید و بهترین pair را طبق dictionary بالا پیدا کنید. در صورتی که هیچ دو Sub-word مجاوری در dictionary بالا یافت نشد، الگوریتم را خاتمه می‌دهیم.

مراحل مورد نظر برای کلمه ”Lowest“ به صورت زیر خواهد بود :

word split into characters:('l', 'o', 'w', 'e', 's', 't', '-')

Iteration:1

Here are the potential pairs to be merged: {('o', 'w'), ('s', 't'), ('t', '-'), ('l', 'o'), ('e', 's'), ('w', 'e')}
candidate for merging: ('e', 's')
word after merging: ('l', 'o', 'w', 'es', 't', '-')

Iteration:2

Here are the potential pairs to be merged: {('o', 'w'), ('t', '-'), ('es', 't'), ('w', 'es'), ('l', 'o')}
candidate for merging: ('es', 't')
word after merging: ('l', 'o', 'w', 'est', '-')

Iteration:3

Here are the potential pairs to be merged: {('o', 'w'), ('w', 'est'), ('est', '-'), ('l', 'o')}
candidate for merging: ('est', '-')
word after merging: ('l', 'o', 'w', 'est-')

Iteration:4

Here are the potential pairs to be merged: {('o', 'w'), ('w', 'est-'), ('l', 'o')}
candidate for merging: ('l', 'o')
word after merging: ('lo', 'w', 'est-')

Iteration:5

Here are the potential pairs to be merged: {('w', 'est-'), ('lo', 'w')}
candidate for merging: ('lo', 'w')
word after merging: ('low', 'est-')

Iteration:6

Here are the potential pairs to be merged: {('low', 'est-')}
candidate for merging: None

Candidate not in BPE merges, the algorithm stops.

گام دوم:

در ابتدا دو دیتاست داده شده توسط سوال را در محیط Google Colab دانلود و unzip می‌کنیم :

```
!wget http://www.gutenberg.org/cache/epub/16457/pg16457.txt

!wget https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-103-raw-v1.zip

!unzip wikitext-103-raw-v1.zip
```

برای پیاده سازی به کمک Hugging face، نیاز به نصب کتابخانه Tokenizer می‌باشد :

```
!pip install tokenizers
```

بعد از نصب، نوبت به Import زیرکتابخانه های مدل های BPE,WPC و Trainer آنها می‌باشد :

```
from tokenizers import Tokenizer
from tokenizers.models import BPE, WordPiece
from tokenizers.trainers import BpeTrainer, WordPieceTrainer
```

*قبل از اعمال تابع Tokenizer نیاز است که ورودی ما ابتدا به کلمات تبدیل شود که به کمک Pre-Tokenizer انجام می‌شود. در اینجا معیار تفکیک لغات را فاصله(Space) در نظر می‌گیریم :

```
from tokenizers.pre_tokenizers import Whitespace
```

از آنجا که برای مدل BERT، Tokens های مشخص و خاصی تعریف شده است(مثل "<UNK>" که در مواقعی که BERT نتواند یک کلمه را به Sub-word ها تجزیه کند، به جای آن "<UNK>" قرار می‌دهد.)، باید آنها را معرفی کنیم :

```
Unknown_token = "<UNK>" # Unknown words
Special_tokens = ["<UNK>", "<SEP>", "<MASK>", "<CLS>"]  # Special tokens used in BERT model
```

با تعریف بالا، حال تابعی می‌نویسم که Tokenizer و Trainer یک الگوریتم را برگرداند :

```
def prepare_tokenizer_trainer(algorithm):
    if algorithm == 'BPE': # Stands for Binary Pair Encoding.
        tokenizer = Tokenizer(BPE(unk_token = Unknown_token))
        trainer = BpeTrainer(special_tokens = Special_tokens)

    elif algorithm == 'WPC': # Stands for Word Piece.
        tokenizer = Tokenizer(WordPiece(unk_token = Unknown_token))
        trainer = WordPieceTrainer(special_tokens = Special_tokens)
    else:
      print("Invalid Algorithm, Try again !")

    tokenizer.pre_tokenizer = Whitespace()
    return tokenizer, trainer
```

حال که کلاس های Tokenizer و Trainer آماده هستند، تابعی می‌نویسم که با گرفتن دادگان آموزش، مدل را آموزش دهید (در اینجا دو دیتاست نصب شده در ابتدا سوال، به عنوان داددگان آموزش هستند):

```
def train_tokenizer(input_files, algorithm='BPE'):

    tokenizer, trainer = prepare_tokenizer_trainer(algorithm)
    tokenizer.train(input_files, trainer) # training the tokenzier
```

```
    #tokenizer.save("./tokenizer-trained.json")
    #tokenizer = Tokenizer.from_file("./tokenizer-trained.json")
    return tokenizer
```

حال که مدل آموزش دیده است، کافی است با گرفتن یک متن دلخواه ورودی، آنرا Encode کند :

```
def tokenize(input_string, tokenizer):
    output = tokenizer.encode(input_string)
    return output
```

در نهایت تابعی به کمک مجموع توابع بالا می‌نویسم تا برای هر دیتاست و هر الگوریتم، Tokens ها را برگرداند :

```
def return_tokenized(input_string ,input_files):
  tokens_dict = {}
  len_tokens=[]
  for file in input_files:
      print(f"========Using vocabulary from {file}=======")
      for algorithm in ['BPE','WPC']:
          trained_tokenizer = train_tokenizer(file, algorithm)
          output = tokenize(input_string, trained_tokenizer)
          tokens_dict[algorithm] = output.tokens
          len_tokens.append(len(output.tokens))
          print("----", algorithm, "----")
          print(output.tokens, "->", len(output.tokens))
  return len_tokens
```

که 4 خروجی خواهیم داشت :

```
len_tokens=return_tokenized(input_string ,[small_file, large_files])
```

========Using vocabulary from ['pg16457.txt']=======
---- BPE ----
['This', 'is', 'a', 'deep', 'learning', 'to', 'ken', 'ization', 't', 'ut', 'or', 'ial', '.', 'T', 'ok', 'en', 'ization', 'is', 'the', 'first', 'step', 'in', 'a', 'deep', 'learning', 'N', 'L', 'P', 'pi', 'pe', 'line', '.', 'We', 'will', 'be', 'comparing', 'the', 'to', 'k', 'ens', 'generated', 'by', 'each', 'to', 'ken', 'ization', 'model', '.', 'Ex', 'c', 'ited', 'much', '?', '!', '<UNK>'] => 55
---- WPC ----
['This', 'is', 'a', 'deep', 'learning', 'to', '##ken', '##ization', 't', '##ut', '##oria', '##l', '.', 'To', '##ken', '##ization', 'is', 'the', 'first', 'step', 'in', 'a', 'deep', 'learning', 'N', '##L', '##P', 'pip', '##el', '##ine', '.', 'We', 'will', 'be', 'comparing', 'the', 'to', '##ken', '##s', 'generated', 'by', 'each', 'to', '##ken', '##ization', 'model', '.', 'Ex', '##ci', '##ted', 'much', '<UNK>'] => 52
========Using vocabulary from ['./wikitext-103-raw/wiki.test.raw', './wikitext-103-raw/wiki.train.raw', './wikitext-103-raw/wiki.valid.raw']=======
---- BPE ----
['This', 'is', 'a', 'deep', 'learning', 'to', 'ken', 'ization', 'tut', 'orial', '.', 'Tok', 'en', 'ization', 'is', 'the', 'first', 'step', 'in', 'a', 'deep', 'learning', 'NL', 'P', 'pipeline', '.', 'We', 'will', 'be', 'comparing', 'the', 'tok', 'ens', 'generated', 'by', 'each', 'to', 'ken', 'ization', 'model', '.', 'Ex', 'cited', 'much', '?', '!', '<UNK>'] => 47
---- WPC ----

['This', 'is', 'a', 'deep', 'learning', 'to', '##ken', '##ization', 'tut', '##orial', '.', 'Tok', '##eni', '##za', '##ti', '##on', 'is', 'the', 'first', 'step', 'in', 'a', 'deep', 'learning', 'NL', '##P', 'pipeline', '.', 'We', 'will', 'be', 'comparing', 'the', 'to', '##ken', '##s', 'generated', 'by', 'each', 'to', '##ken', '##ization', 'model', '.', 'Exc', '##ited', 'much', '<UNK>'] => 48

خلاصه تعداد Token ها در جدول زیر آمده است :

جدول 1.2 : خلاصه Token های بدست آمده برای هر روش و هر داده آموزش

	BPE	WPC
pg16457.txt	55	47
wiki.raw	52	48

تحلیل :

- طبق جدول بالا، الگوریتم BEP برای دیتاست بزرگتر، تعداد Token کمتری ایجاد کرده است که به معنای این است که توانسته با دیدن دیتای بیشتر، تعداد Sub-word های بیشتری را merge کند.
- تعداد Token های الگوریتم WPC نسبت به BPE کمتر است که به تفاوت الگوریتم انتخاب pair بین این دو و نوع داده ورودی بستگی دارد.
- در هر دو الگوریتم، با دیدن دیتاست بزرگتر، تعداد Token ها کمتر شده است.
- در الگوریتم WPC، Punctuation های "?" , "!" در Token ها نادیده گرفته شده‌اند، در حالیکه در BPE، طبیعتا لحاظ شده‌اند.
- در الگوریتم WPC، Suffix ها و prefix ها به صورت "##" در انتها یا ابتدای Sub-word مشخص شده‌اند در حالیکه BPE از این قاعده تبعیت نمی‌کند.

گام سوم:

در این مرحله به جای متن ورودی کوتاه گام دوم، از متن کتاب یوتنبرگ استفاده می‌شود :

```
with open('pg16457.txt') as f:
   lines = f.readlines()
new_input=" ".join(str(x) for x in lines)
```

حال تابع نهایی گام دوم را اینبار برای متن ورودی جدید اجرا می‌کنیم :

جدول 1.3 : خلاصه Token های بدست آمده برای هر روش و هر داده آموزش

	BPE	WPC
pg16457.txt	122739	140872
wiki.raw	122739	140735