به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

تمرین کامپیوتری شماره یک

نام و نام خانوادگی:

امیرحسین دبیری اقدم

شماره دانشجویی:

810197502

اسفند ماه 1400

فهرست

عنوان	شماره صفحه
چکیده	3
گام اول - الگوریتم‌های BPE و WordPiece + پیاده‌سازی BPE از پایه	4
گام دوم - الگوریتم‌های BPE و WordPiece با استفاده از کتاب‌خانه hugging face	9
گام سوم - توکنایز کردن کتاب گوتنبرگ با استفاده از هر دو الگوریتم	13

چکیده

توکنایز کردن[1] کلمات یکی از مهم‌ترین پیش‌پردازش‌ها در اکثر تسک‌های مرتبط با NLP است؛ در این تمرین کامپیوتری دو الگوریتم متداول انجام این کار پیاده‌سازی شد، با دیتاست‌های مختلف آموزش آن انجام شد و نتایج خروجی آن‌ها بررسی و تحلیل شد. الگوریتم اول Byte Pair Encoding (BPE) بود که صفر تا صد آن از پایه پیاده‌سازی و روی یک دیتاست ساده مراحل مختلف اجرای این الگوریتم بررسی شد و همچنین با استفاده از کتاب‌خانه پرقدرت Hugging face نیز پیاده سازی شد. الگوریتم دوم WordPiece بود که آن هم با کتاب‌خانه Hugging face پیاده سازی شد و خروجی آن‌ها روی یک نمونه متن بررسی و تحلیل شد.

[1] Tokenization

** تمام کدهای مربوط به این تمرین و توابع ذکر شده در این گزارش، در فایل نوت‌بوک ضمیمه شده آمده است. توضیحات و نتایج کدها در این گزارش ارائه شده است .*

** نوت‌بوک از کولب گوگل استخراج شده و ممکن است برخی دستورات آن (مثل دستورات دانلود با **wget** و...) روی ویندوز قابل اجرا نباشد و نیاز است که روی کولب اجرا شود. البته بقیه موارد قابلیت اجرای مجدد روی ویندوز را دارد به شرطی که به صورت دستی فایل‌های دیتاست‌ها دانلود شده و در کنار نوت‌بوک قرار بگیرد.*

گام اول – الگوریتم‌های BPE و WordPiece + پیاده‌سازی BPE از پایه

بررسی و مقایسه الگوریتم‌های BPE و WordPiece:

الگوریتم‌های BPE و WordPiece هر دو از الگوریتم‌های توکنایز کردن هستند که بر مبنای زیرکلمه (subword level) عملیات توکنایز کردن را انجام می‌دهند. این نوع توکنایزرها که به نوعی ترکیب توکنایزرهای بر مبنای کلمه (word level) و برمبنای کاراکتر (character level) هستند و مزیت هر دوی این توکنایزرها را با هم دارند و در عین اینکه عملکرد خوبی در بازنمایی مناسب کلمات (مستقل از کانتکست) دارند و خروجی آن‌ها برای استفاده به عنوان ورودی مدل‌های پیچیده‌تر مناسب است؛ اندازه Vocabulary معقولی هم دارند و بنابراین از نظر پیچیدگی حافظه و زمانی نیز مناسب هستند. این نوع توکنایزرها بر این قاعده استوارند که کلمات پرکاربرد و متداول نیازی نیست به زیر کلمه توکنایز شوند اما کلمات کمتر متداول باید به زیرکلمه‌های با معنی تجزیه شوند؛ این ویژگی به خصوص در توکنایز کردن زبان‌های مورفولوژیکال مثل ترکی استانبولی ممکن است مفید واقع شود.

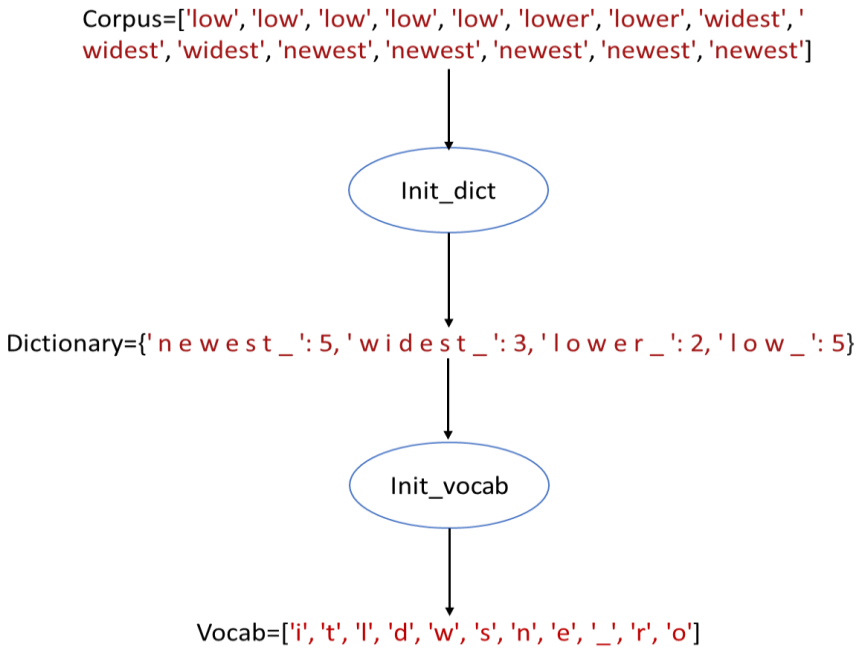
علاوه بر ویژگی‌های گفته شده که اکثر توکنایزرهای subword level دارای آن هستند، BPE و WordPiece به صورت کلی نیز الگوریتم‌های نسبتاً مشابهی هستند؛ در هر دو نیاز به پیش توکنایز کردن متن ورودی و تبدیل آن به کلمات (براساس whitespace یا punctuation یا هردو یا...) دارند. در ادامه هر دوی این الگوریتم‌ها از یک Vocabulary ساده که شامل تمام کاراکترهای موجود در متن داده آموزش است شروع می‌کنند. تفاوت عمده این دو الگوریتم در این است که الگوریتم BPE با توجه به فرکانس تعداد هر کلمه در داده آموزش، پر تکرارترین زوج سمبل در کنار هم (سمبل می‌تواند کاراکتر و زیرکلمه یا حتی کلمه باشد) را پیدا کرده و آن را به Vocabulary اضافه کرده و در تمام corpus آموزش این زوج سمبل را با هم ادغام کرده به صورت یکپارچه در نظر می‌گیرد و این قواعد ادغام کردن را نیز یاد می‌گیرد (ذخیره می‌کند) و این پروسه یافتن پرتکرارترین زوج سمبل و ادغام آن تا زمانی که اندازه Vocabulary به مقدار دلخواه برسد ادامه خواهد داشت (جزئیات بیشتر مربوط به پیاده سازی این الگوریتم در بخش‌های بعدی گزارش آمده است). الگوریتم WordPiece هم به طور مشابه جفت سمبل

انتخاب کرده، به Vocabulary افزوده، آن‌ها را ادغام کرده و قواعد ادغام را می‌آموزد اما تفاوت آن نحوه انتخاب جفت سمبل است. این الگوریتم جفت سمبل‌ها را براساس تعداد تکرارشان در corpus انتخاب نمی‌کند بلکه جفت سمبلی را انتخاب می‌کند که با اضافه کردن آن‌ها به Vocabulary، Likelihood داده آموزش ماکسیمم می‌شود. (Likelihood، مفهومی احتمالاتی و پرکاربرد در یادگیری ماشین است که بنا به قانون بیز و ... قابل محاسبه و تخمین است.) به صورت شهودی این الگوریتم یک جفت سمبل را وقتی ادغام می‌کند که ببیند واقعاً ارزش دارد و عملکرد توکنایزر روی داده آموزش بهبود می‌یابد. (حداکثر شدن احتمال یا Likelihood داده آموزش منجر به بهبود عملکرد توکنایزر می‌شود)؛ اما BPE بدون توجه به Likelihood و عمکلرد توکنایزر روی داده آموزش، تنها با توجه به فرکانس تکرار جفت سمبل در کنار هم آن‌ها را با هم ادغام می‌کند. بنا به کاربرد و تسکی که قرار است انجام شود ممکن است یکی از این دو الگوریتم عملکرد بهتری داشته باشد اما در حالت کلی با توجه به شباهت الگوریتمشان، خروجی آن‌ها نیز معمولاً به هم شبیه است و به صورت قاطع نمی‌توان ادعا کرد که یکی بر دیگری برتری کامل دارد.

پیاده‌سازی BPE از پایه:

برای پیاده‌سازی BPE از پایه، فرض شده است که مانند corpus نمونه، کلمات ابتدا پیش توکنایز شده‌اند یعنی corpus به‌صورت لیستی از کلمات که (بر اساس whitespace یا ...) از هم جدا شده‌اند، موجود است. همچنین نیاز به چند تابع کمکی بود، یکی تابع add_spaces_EOW که کاراکترهای کلمه ورودی را با whitespace از هم جدا کرده و همچنین سمبل "_" که نمایانگر پایان کلمه است (end-of-word symbol) را به آن اضافه می‌کند. تابع init_dict نیز از روی لیست کلمات (corpus) یک دیکشنری تولید می‌کند که دفعات تکرار (فرکانس) هر کلمه در corpus را در خود نگه می‌دارد. این تابع روی کلمات corpus، تابع add_spaces_EOW را نیز اعمال می‌کند. در نهایت تابع کمکی init_vocab از روی دیکشنری به‌دست آمده از init_dict، Vocabulary اولیه شامل تمام سمبل‌های (کاراکترها، توکن EOW و ...) موجود در دیکشنری را تولید می‌کند.

به این ترتیب برای مثال داده شده خواهیم داشت:



شکل 1 – مراحل ساخت **Dictionary** و **Vocabulary** از **Corpus**

در نهایت تابع train_BPE پیاده‌سازی شده که با گرفتن corpus اولیه و فراخوانی توابع کمکی مذکور، Dictionary و Vocab را ساخته و الگوریتم BPE را اجرا می‌کند. یعنی با توجه به Vocab جفت سمبل‌های مجاور را در Dictionary جستجو کرده و پر تکرارترین جفت سمبل را پیدا کرده و با هم ادغام (merge) می‌کند و به Vocab اضافه می‌کند و همچنین در Dictionary نیز این سمبل‌ها را ادغام و یکپارچه می‌کند. این رویه پیدا کردن جفت سمبل پرتکرار و ادغام کردن آن و... تا زمانی ادامه پیدا می‌کند که تعداد سمبل‌های موجود در Vocab به مقدار vocab_size که به‌عنوان ورودی به تابع داده شده بود برسد (یا اینکه همه سمبل‌ها مرج شده باشند به‌طوری که جفت سمبلی پیدا نمی‌شود که با هم مرج شوند که در این حالت هم الگوریتم به پایان می‌رسد.).

به جهت سهولت در پیاده‌سازی الگوریتم فوق، برای یافتن جفت سمبل‌های مجاور در Dictionary و نیز ادغام آن‌ها از عبارات با قاعده (Regular Expression) و کتاب‌خانه پایتون آن استفاده شده است.

با توجه به اینکه شرطی روی تعداد سمبل‌های موجود در Vocab در صورت پروژه گفته نشده، با فرض vocab_size=15، فرآیند اجرای الگوریتم فوق الذکر به صورت مرحله به مرحله، با استفاده از corpus داده شده در صورت پروژه، بعد از انجام مراحل اولیه ساخت Dictionary و Vocab که در شکل 1 توضیح داده شده، در زیر آمده است:

```
---------------------------------------------------------- i = 0 ----------------------------------------------------------
Dictionary {' n e w e s t _ ': 5, ' l o w _ ': 5, ' w i d e s t _ ': 3, ' l o w e r _ ': 2}
Vocab: ['w', 't', 'r', 'e', 's', 'i', 'd', 'o', 'n', 'l', '_']

---------------------------------------------------------- i = 1 ----------------------------------------------------------
Most frequent pair: " s t " , Count: 8
Updated dictionary after merging pairs: {' l o w _ ': 5, ' l o w e r _ ': 2, ' n e w e st _ ': 5, ' w i d e st _ ': 3}
Updated vocab after merging pairs: ['w', 't', 'r', 'e', 's', 'i', 'd', 'o', 'n', 'l', '_', 'st']

---------------------------------------------------------- i = 2 ----------------------------------------------------------
Most frequent pair: " st _ " , Count: 8
Updated dictionary after merging pairs: {' l o w _ ': 5, ' l o w e r _ ': 2, ' n e w e st_ ': 5, ' w i d e st_ ': 3}
Updated vocab after merging pairs: ['w', 't', 'r', 'e', 's', 'i', 'd', 'o', 'n', 'l', '_', 'st', 'st_']

---------------------------------------------------------- i = 3 ----------------------------------------------------------
Most frequent pair: " e st_ " , Count: 8
Updated dictionary after merging pairs: {' l o w _ ': 5, ' l o w e r _ ': 2, ' n e w est_ ': 5, ' w i d est_ ': 3}
Updated vocab after merging pairs: ['w', 't', 'r', 'e', 's', 'i', 'd', 'o', 'n', 'l', '_', 'st', 'st_', 'est_']

---------------------------------------------------------- i = 4 ----------------------------------------------------------
Most frequent pair: " l o " , Count: 7
Updated dictionary after merging pairs: {' n e w est_ ': 5, ' w i d est_ ': 3, ' lo w _ ': 5, ' lo w e r _ ': 2}
Updated vocab after merging pairs: ['w', 't', 'r', 'e', 's', 'i', 'd', 'o', 'n', 'l', '_', 'st', 'st_', 'est_', 'lo']

---------------------------------------------------------- i = 5 ----------------------------------------------------------
Most frequent pair: " lo w " , Count: 7
Updated dictionary after merging pairs: {' n e w est_ ': 5, ' w i d est_ ': 3, ' low _ ': 5, ' low e r _ ': 2}
Updated vocab after merging pairs: ['w', 't', 'r', 'e', 's', 'i', 'd', 'o', 'n', 'l', '_', 'st', 'st_', 'est_', 'lo', 'low']
```

شکل 2 – مراحل اجرای الگوریتم **BPE**

همانطور که در مراحل فوق مشاهده می‌شود، در ابتدا زوج سمبل "s" و "t" در کنار هم بیشترین تکرار را دارند (5 بار در کلمه "newest" و 3 بار در "widest" یعنی مجموعاً 8 بار - البته ممکن است زوج سمبل دیگری با 8 بار تعداد علاوه بر زوج سمبل "s" و "t" موجود باشد که در الگوریتم BPE تفاوتی نمی‌کند کدام انتخاب شود و در پیاده‌سازی انجام گرفته آخرین سمبل شناسایی شده با بیشترین تعداد تکرار انتخاب می‌شود که در اینجا همان زوج سمبل "s" و "t" است.) پس این دو سمبل با هم ادغام شده به صورت "st" درمی‌آیند و این ادغام در تمام کلمات Dictionary اعمال می‌شود و به Vocabulary هم اضافه می‌شود. به همین ترتیب زوج سمبل "st" (که در مرحله قبل تولید و به Vocabulary اضافه شده بود) و "_" (همان سمبل پایان کلمه EOW) بیشترین تکرار را دارند که با هم ادغام شده و ادامه الگوریتم مشابه قبل است...

در پایان، Vocabulary نهایی (شامل vocab_size=15 تا سمبل) و نیز زوج سمبل‌های ادغام شده (به ترتیب ادغام شدن در هنگام اجرای الگوریتم آموزش) به صورت زیر بدست می‌آید:

Final Vocab: ['w', 't', 'r', 'e', 's', 'i', 'd', 'o', 'n', 'l', '_', 'st', 'st_', 'est_', 'lo', 'low']
Merged pairs: [['s', 't'], ['st', '_'], ['e', 'st_'], ['l', 'o'], ['lo', 'w']]

تابع train_BPE در نهایت Vocab، Dictionary و نیز merged_pairs (جفت‌های ادغام شده) را خروجی می‌دهد که از این اطلاعات برای توکنایز کردن کلمه جدید در تابع tokenize_BPE استفاده می‌شود.

تابع tokenize_BPE در آغاز، تابع add_spaces_EOW را روی کلمه ورودی‌اش اعمال می‌کند و بعد با توجه به merged_pairs به‌دست آمده از train_BPE و به همان ترتیبی که جفت‌ها در تابع train_BPE انتخاب شده بودند، در کلمه ورودی نیز به دنبال این جفت‌ها گشته و در صورت وجود، آنها را با هم ادغام می‌کند و به این ترتیب عملیات توکنایز کردن انجام می‌شود.

برای مثال کلمه"lowest" ابتدا به " l o w e s t _ " تبدیل شده و با توجه به اینکه جفت "s t" را داراست به " l o w e st _ " تبدیل و بعد چون جفت "st _" را دارد با ادغام آنها " l o w e st_ " حاصل می‌شود و به همین ترتیب زوج "e st_ " ادغام شده و بعد زوج "l o" و در نهایت زوج "lo w" و بنابراین نتیجه نهایی توکنایز شده کلمه"lowest" به‌صورت "low est" به‌دست می‌آید. به بیان دیگر در مرحله قبل، الگوریتم توکن‌های "low" و "est_" را یاد گرفته بود (این دو سمبل در Vocab نهایی موجود هستند)؛ بنابراین علیرغم اینکه در داده آموزش کلمه"lowest" وجود نداشت (اصطلاحاً OOV بود یعنی کلمه جدید که قبلاً دیده نشده بود) اما الگوریتم با توجه به کلمات "widest" و "newest" الگوی noun + est در انگلیسی به نوعی کشف کرده بود، همچنین وجود کلمات "low" و "lower" در داده آموزش نیز باعث شد تا الگوریتم"low" را به عنوان یک توکن مجزا شناسایی کند.

بنابراین با اجرای الگوریتم مذکور (تابع tokenize_BPE) کلمه"lowest" به درستی به دو زیرکلمه "low" + "est" توکنایز می‌شود. (شکل 3)

```
i = 0 :  l o w e s t _
i = 1 :  l o w e s t_
i = 2 :  l o w e st_
i = 3 :  l o w est_
i = 4 :  l ow est_
i = 5 :  low est_

tokenization final result:  low est
```

شکل 3 – مراحل توکنایز شدن کلمه **"lowest"** که یک کلمه **OOV** است.

گام دوم – الگوریتم‌های BPE و WordPiece با استفاده از کتابخانه hugging face

در این مرحله با استفاده از کتابخانه hugging face مدل‌هایی براساس الگوریتم‌های BPE و WordPiece را روی دو دیتاست معرفی شده در صورت پروژه یعنی یک کتاب از گوتنبرگ و متن ویکی پدیای انگلیسی (تمام آن یعنی هر سه فایل test, train, valid) آموزش داده و در پایان متن زیر را با مدل‌های بدست آمده توکنایز می‌کنیم.

"This is a deep learning tokenization tutorial. Tokenization is the first step in a deep learni ng NLP pipeline. We will be comparing the tokens generated by each tokenization model. Excited much?!😍"

برای این کار تابع create_tokenizer پیاده‌سازی شده که دو ورودی tokenizer_type و corpus_name را گرفته و با توجه به مقادیر آن‌ها مدل‌هایی را بر اساس الگوریتم‌های BPE یا WordPiece و با استفاده از یکی از دیتاست‌های گوتنبرگ یا ویکی پدیا آموزش می‌دهد (پارامترهای آپشنال مدل‌ها نظیر vocab_size و min_frequency همان مقادیر پیشفرض یعنی به ترتیب 30 هزار و 0 در نظر گرفته شده‌اند). در این تابع به همه مدل‌ها توکن <UNK> را هم اضافه می‌کنیم تا اموجی‌ها و... بعد از توکنایز کردن از متن حذف نشوند زیرا در برخی کاربردها حذف اموجی‌ها و... از متن ممکن است باعث از دست رفتن اطلاعات ارزشمند شود، بنابراین توکن <UNK> را برای هندل کردن این قبیل موضوعات در نظر می‌گیریم تا به جای حذف شدن، صرفاً به <UNK> تبدیل شوند و در آینده در صورت نیاز به بررسی و در نظر گرفتن اموجی‌ها بتوان به آن‌ها دسترسی پیدا کرد. همچین برای پیش توکنایزر، کلمات corpus براساس whitespace جداسازی می‌شوند. این تابع در نهایت توکنایزر ترین شده را خروجی می‌دهد و همچنین آن ها را در فایل json با نام متناظر ذخیره می‌کند. (فایل‌های مربوط به هر توکنایزر آموزش داده شده در کنار این گزارش ضمیمه شده است.)

خروجی هر کدام از مدل‌ها به ورودی فوق در ادامه آمده است.

- مدل با الگوریتم BPE و دیتاست گوتنبرگ:

['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]']

- مدل با الگوریتم BPE و دیتاست ویکی پدیا:

['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]']

- مدل با الگوریتم WordPiece و دیتاست کوتنبرگ:

['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]']

- مدل با الگوریتم WordPiece و دیتاست ویکی پدیا:

['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]']

تعداد توکن‌های هر کدام از 4 حالت فوق، در جدول ذیل آمده است:

ردیف	نام الگوریتم استفاده شده برای توکنایز	تعداد توکن‌های خروجی الگوریتم برای متن مذکور	
		توکنایزر آموزش داده شده بر روی کتاب گوتنبرگ	توکنایزر آموزش داده شده بر روی کل داده‌های ویکی پدیا
1	Byte Pair Encoding (BPE)	55	47
2	WordPiece	52	48

اولین نکته ای که مشاهده می‌شود، هر دو الگوریتم وقتی روی داده با حجم بزرگتر (ویکی پدیا) آموزش دیده‌اند نسبت به حالتی که روی دیتاست با حجم کمتر (گوتنبرگ) آموزش دیده‌اند، متن نمونه داده شده را به تعداد توکن‌های کمتری تجزیه کرده‌اند زیرا که دیتاست بزرگتر ظاهراً تا حدی باعث بهبود عملکرد مدل شده و هدف اصلی این نوع توکنایزر ها یعنی تبدیل نکردن کلمات متداول به زیر کلمات و برعکس تبدیل کلمات کمتر متداول به زیر کلمات با معنی را بهتر برآورده می‌کند و در کل کیفیت توکن‌های مدل‌های آموزش داده شده با حجم داده بیشتر، بهتر است. برای مثال هر دو الگوریتم WordPiece و BPE وقتی روی دیتاست گوتنبرگ آموزش دیده‌اند، کلمه نسبتاً متداول "pipeline" را به صورت 'pip', '##el', '##ine' و 'pi', 'pe', 'line' تجزیه کرده‌اند که خیلی مناسب نیست اما وقتی با دیتاست بزرگتر ویکی پدیا آموزش دیده‌اند توانسته‌اند این کلمه را به همان صورت "pipeline" توکنایز کنند. یا مثال دیگر کلمه "excited" است که هر دو الگوریتم وقتی که با دیتاست بزرگتر (ویکی پدیا) آموزش دیده‌اند بهتر توانسته‌اند زیر کلمه‌های آن را شناسایی کنند. پس همانطور که پیش تر هم بیان شد، به طور کلی مشاهده می‌شود که هرقدر مدل‌ها روی دیتاست بزرگتری آموزش ببینند، عملکردی بهتری علی الخصوص در مواجه با کلمات OOV از خود نشان می‌دهند و توکن‌های بهتر و

نزدیک به الگوهای واقعی زبان طبیعی تولید می‌کنند. البته از روی عملکرد مدل‌ها تنها روی یک متن نسبتاً کوتاه حکم کلی نمی‌توان داد و ممکن است تحت شرایطی بسته به کاربرد و تسکی که انجام می‌شود، مدل آموزش دیده روی یک دیتاست کوچکتر عملکرد بهتری نسبت به مدل آموزش دیده با یک دیتاست بزرگتر داشته باشد.

مورد دیگر در مورد مواجه با کلمات OOV می‌توان به کلمه "NLP" اشاره کرد که مشاهده می‌شود هر دو الگوریتم که روی دیتاست گوتنبرگ (دیتاست کوچک) آموزش دیده‌اند به ابتدایی ترین روش ممکن یعنی تجزیه به کاراکترهای سازنده این کلمه را تجزیه کرده‌اند. البته هر دو الگوریتم حتی وقتی که روی دیتاست ویکی پدیا (دیتاست بزرگ) هم آموزش دیده‌اند باز هم نتوانسته‌اند این acronym را به درستی و به صورت یک توکن در نظر بگیرند بلکه به صورت 'NL', 'P' تجزیه کرده‌اند.

نکته دیگر قابل مشاهده، تفاوت WordPiece و BPE در نمایش توکن‌هایی که در وسط کلمه قرار گرفته‌اند است؛ به عبارت دیگر WordPiece برای نمایش توکن‌هایی که نقش پسوند را دارند از علامت ## استفاده می‌کند. مثلاً همان کلمه "excited" را اگر در نظر بگیریم، الگوریتم BPE (آموزش دیده روی دیتاست ویکی پدیا) آن را به صورت 'Ex', 'cited' توکنایز کرده و الگوریتم WordPiece (آموزش دیده روی دیتاست ویکی پدیا) آن را به صورت 'Exc', '##ited' توکنایز کرده (اندک تفاوت در نحوه توکنایز کردن این کلمه و برخی دیگر از کلمات در متن نمونه، ناشی از تفاوت روش انتخاب زوج سمبل‌ها در دو الگوریتم است که پیش تر راجع به آن توضیح داده شده). در این مثال مشاهده می‌شود که هر دو الگوریتم پیشوندها را به یک شکل مشخص کرده‌اند اما الگوریتم WordPiece در نمایش پسوندها در ابتدای آن‌ها از علامت ## استفاده کرده. این ویژگی برای بازسازی متن اولیه از روی توکن‌ها می‌تواند کمک کننده باشه.

همچنین در مورد اموجی 😍 که در متن نمونه آمده است همانطور که قبلاً هم گفته شد توکن خاص <UNK> را به مدل‌ها اضافه کرده بودیم تا این قبیل موارد را مدل‌ها بتوانند به درستی هندل کنند و اموجی‌ها و ... بی دلیل حذف نشوند.

سخن پایانی مقایسه این دو الگوریتم: در حالت کلی به نظر می‌رسد WordPiece حداقل برای این مثال خاص اندکی بهتر عمل کرده. مثلاً کلمه "tokens" را به صورت 'to', '##ken', '##s' توکنایز کرده یعنی توانسته s جمع را به خوبی از بقیه کلمه جدا کند اما BPE به صورت 'tok', 'ens' توکنایز کرده و نتوانسته s جمع را از بقیه کلمه جدا کند؛ و به طور کلی با بررسی خروجی این دو الگوریتم به نظر می‌رسد توکن‌های WordPiece به ساختار واقعی زبان انگلیسی بیشتر شبیه است البته تفاوت فاحشی هم بین خروجی این دو الگوریتم نیست و هر دو وقتی روی دیتاست به اندازه کافی بزرگ مثل ویکی پدیا آموزش داده شوند عملکرد قابل قبولی دارند و متناسب به تسکی که قرار است بعد از عملیات توکنایز

کردن انجام شود ممکن است هر کدام از این دو الگوریتم عملکرد بهتری نسبت به دیگری نشان دهد اما به نظر من رویکرد WordPiece که مبنی بر Likelihood است رویکرد هوشمندانه‌تری است و احتمال اینکه نتیجه مطلوب‌تری با این مدل حاصل شود بیشتر است نسبت به BPE؛ هرچند مدل‌های دیگری نظیر SentencePiece و... نیز وجود دارد که ممکن است از دو الگوریتمی که در این پروژه بررسی شدند عملکرد بهتری (بسته به تسک یا...) داشته باشد.

گام سوم – توکنایز کردن کتاب گوتنبرگ با استفاده از هر دو الگوریتم

در پایان یک تابع کمکی read_doc می‌نویسیم که کتاب گوتنبرگ یا هر فایل متنی دیگر را که دارای انکدینگ utf8 هستند را با utf-8-sig خوانده (تا BOM را به عنوان متن در نظر نگیرد) و به صورت یک متن یکپارچه در یک متغیر string خروجی می‌دهد و بعد این متن را به توکنایزرهای مختلفی که در گام قبلی آموزش داده بودیم، توکنایز می‌کنیم که نتایج آن در جدول زیر آمده است:

ردیف	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
1	Byte Pair Encoding (BPE)	122904	141460
2	WordPiece	130221	144716