Homework 1

NLP

PREPARED BY

Mohsen Fayyaz - 810100524

Spring 2022

فهرست مطالب

گام اول

Byte Pair Encoding (BPE)

الگوریتم BPE[1] ابتدائا برای فشرده سازی طراحی شده است به این شکل که به صورت مکرر تعداد دوبایتی ها را میشمرد و سپس به جای پرتکرارترین جفت بایت یک بایت استفاده نشده جایگزین می‌کرده است. در پردازش زبان به جای واحد بایت از واحدهای زبانی (Sennrich et al., 2015) استفاده می‌کنیم. ابتدا کاراکترهای موجود در مجموعه متن را وارد vocabulary می‌کنیم. سپس جفت کاراکترها را می‌شماریم و آن جفتی که بیشتر از همه تکرار شده را به vocabulary اضافه کرده و آن دو را از این به بعد به عنوان یک کاراکتر در نظر میگیریم. همین کار را آنقدر تکرار می‌کنیم تا اندازه vocabulary به مقدار مشخصی که هایپر پارامتر مسئله است و ما آن را مشخص می‌کنیم برسیم.

WordPiece Algorithm

روش word piece که در مدل‌هایی مثل BERT استفاده می‌شود بسیار مشابه BPE است. ابتدا کاراکترها وارد vocabulary می‌شوند، , و به صورت پیش رونده قوانین ترکیب یاد گرفته می‌شوند. اما برخلاف BPE به جای در نظر گرفتن پرتکرارترین جفت، سعی می‌کند آن جفتی را انتخاب کند که likelihood داده آموزش را بیشینه کند.

پیاده‌سازی:

ابتدا corpus ورودی را آماده می‌کنیم.

```
def flatten(t):
    return [item for sublist in t for item in sublist]
corpus = flatten([["low"] * 5, ["lower"] * 2, ["widest"] * 3, ["newest"] * 5])
corpus = [list(s) for s in corpus]
print(corpus)

[['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'],
['l', 'o', 'w', 'e', 'r'], ['l', 'o', 'w', 'e', 'r'], ['w', 'i', 'd', 'e', 's', 't'],
['w', 'i', 'd', 'e', 's', 't'], ['w', 'i', 'd', 'e', 's', 't'], ['n', 'e', 'w', 'e', 's',
't'], ['n', 'e', 'w', 'e', 's', 't'], ['n', 'e', 'w', 'e', 's', 't'], ['n', 'e', 'w', 'e',
's', 't'], ['n', 'e', 'w', 'e', 's', 't']]
```

[1] (Gage, 1994)

برای راحتی مراحل بعد ساختار را به شکل بالا آورده‌ایم. این نیز اهمیت دارد که کلمات از فاصله انگار شکسته شده‌اند و روش مشابه sentencepeice نیست و هر کلمه جدا در نظر گرفته می‌شود و مثلا "it is" نمی‌تواند یک توکن شود. داخل هر کلمه هم لیستی از کاراکترها داریم.

```
class Tokenizer_BPE:
    def __init__(self, verbose=True):
        self.vocab = None
        self.verbose = verbose

    def __printv(self, string):
        if self.verbose:
            print(string)

    def __find_max_key(self, d: dict) -> str:
        max_value = -1
        for key, value in d.items():
            if value > max_value:
                max_key, max_value = key, value
        return max_key

    def __find_max_pair(self, corpus) -> str:
        pair_freq = dict()
        for word in corpus:
            for char_idx in range(len(word) - 1):
                pair = "".join(word[char_idx: char_idx + 2])
                pair_freq[pair] = pair_freq.get(pair, 0) + 1
        max_freq_pair = self.__find_max_key(pair_freq)
        self.__printv(f"{max_freq_pair} --> {pair_freq}")
        return max_freq_pair

    def __merge_pair(self, corpus, pair: str):
        for word_idx, word in enumerate(corpus):
            merged_word = []
            char_idx = 0
            while char_idx < len(word):
                if char_idx == len(word) - 1:
                    merged_word.append(word[char_idx])
                    break
                current_pair = "".join(word[char_idx: char_idx + 2])
                if current_pair == pair:
                    merged_word.append(pair)
                    char_idx += 2
                else:
                    merged_word.append(word[char_idx])
                    char_idx += 1
            corpus[word_idx] = merged_word
        return corpus

    def __get_unique_chars(self, corpus):
        return list(set(flatten(corpus)))
```

```
    def train(self, corpus: list, vocab_size: int) -> list:
        output = {"vocab": self.__get_unique_chars(corpus)}
        while len(output["vocab"]) < vocab_size:
            self.__printv(corpus)
            try:
                max_freq_pair = self.__find_max_pair(corpus)
                corpus = self.__merge_pair(corpus, max_freq_pair)
                output["vocab"].append(max_freq_pair)
            except Exception as e:
                break
        output["vocab"] = output["vocab"][:vocab_size]
        self.vocab = output["vocab"]
        return output

    def tokenize(self, word: str):
        word = list(word)
        for merge_pair in self.vocab:
            before_merge_len = len(word)
            word = self.__merge_pair([word], merge_pair)[0]
            if before_merge_len != len(word):
                self.__printv(f"Merged '{merge_pair}' --> {word}")
        return word
```

برای تعریف ووکب اولیه از روی کاراکترهای داده `self.__get_unique_charss` استفاده شده است که در آن با گرفت set از لیست تمام کاراکترهای داده، کاراکترهای یکتا را به ووکب اضافه می‌کنیم.

فرکانس رخداد جفت‌ها در تابع `self.__find_max_pair` انجام می‌شود. در این تابع دیکشنری pair_freq ساخته می‌شود که با حرکت روی داده هر ترکیب دوتایی به عنوان کلید استفاده می‌شود و تعداد آن کلید را یکی اضافه می‌کند و در انتها آن کلیدی (جفتی) که بیشترین تکرار را داشته بازگردانده می‌شود.

ادغام پرتکرارترین جفت در `self.__merge_pair` انجام می‌شود. این تابع corpus و آن جفتی که می‌خواهیم یکی شود را گرفته و corpus جدید که در آن این اتفاق افتاده را می‌دهد. برای این کار روی corpus حرکت کرده و هر جا جفت مورد نظر دیده شود جایگزین می‌شود و اگر دیده نشود خود توکن گذاشته می‌شود.

```
pe_tokenizer = Tokenizer_BPE(verbose=True)
print("### Training ###")
output = bpe_tokenizer.train(corpus.copy(), vocab_size=100)
print(output)
```

خروجی:

```
### Training ###
[['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w', 'e', 'r'], ['l', 'o', 'w', 'e', 'r'], ['w', 'i', 'd', 'e', 's', 't'], ['w', 'i', 'd', 'e', 's', 't'], ['w', 'i', 'd', 'e', 's', 't'], ['n', 'e', 'w', 'e', 's', 't'], ['n', 'e', 'w', 'e', 's', 't'], ['n', 'e', 'w', 'e', 's', 't'], ['n', 'e', 'w', 'e', 's', 't']]
es --> {'lo': 7, 'ow': 7, 'we': 7, 'er': 2, 'wi': 3, 'id': 3, 'de': 3, 'es': 8, 'st': 8, 'ne': 5, 'ew': 5}
[['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w', 'e', 'r'], ['l', 'o', 'w', 'e', 'r'], ['w', 'i', 'd', 'es', 't'], ['w', 'i', 'd', 'es', 't'], ['w', 'i', 'd', 'es', 't'], ['n', 'e', 'w', 'es', 't'], ['n', 'e', 'w', 'es', 't'], ['n', 'e', 'w', 'es', 't'], ['n', 'e', 'w', 'es', 't'], ['n', 'e', 'w', 'es', 't']]
est --> {'lo': 7, 'ow': 7, 'we': 2, 'er': 2, 'wi': 3, 'id': 3, 'des': 3, 'est': 8, 'ne': 5, 'ew': 5, 'wes': 5}
[['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w'], ['l', 'o', 'w', 'e', 'r'], ['l', 'o', 'w', 'e', 'r'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est']]
lo --> {'lo': 7, 'ow': 7, 'we': 2, 'er': 2, 'wi': 3, 'id': 3, 'dest': 3, 'ne': 5, 'ew': 5, 'west': 5}
[['lo', 'w'], ['lo', 'w'], ['lo', 'w'], ['lo', 'w'], ['lo', 'w'], ['lo', 'w'], ['lo', 'w', 'e', 'r'], ['lo', 'w', 'e', 'r'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est']]
low --> {'low': 7, 'we': 2, 'er': 2, 'wi': 3, 'id': 3, 'dest': 3, 'ne': 5, 'ew': 5, 'west': 5}
[['low'], ['low'], ['low'], ['low'], ['low'], ['low', 'e', 'r'], ['low', 'e', 'r'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est'], ['n', 'e', 'w', 'est']]
ne --> {'lowe': 2, 'er': 2, 'wi': 3, 'id': 3, 'dest': 3, 'ne': 5, 'ew': 5, 'west': 5}
[['low'], ['low'], ['low'], ['low'], ['low'], ['low', 'e', 'r'], ['low', 'e', 'r'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['ne', 'w', 'est'], ['ne', 'w', 'est'], ['ne', 'w', 'est'], ['ne', 'w', 'est'], ['ne', 'w', 'est']]
new --> {'lowe': 2, 'er': 2, 'wi': 3, 'id': 3, 'dest': 3, 'new': 5, 'west': 5}
[['low'], ['low'], ['low'], ['low'], ['low'], ['low', 'e', 'r'], ['low', 'e', 'r'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['new', 'est'], ['new', 'est'], ['new', 'est'], ['new', 'est'], ['new', 'est']]
newest --> {'lowe': 2, 'er': 2, 'wi': 3, 'id': 3, 'dest': 3, 'newest': 5}
[['low'], ['low'], ['low'], ['low'], ['low'], ['low', 'e', 'r'], ['low', 'e', 'r'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['w', 'i', 'd', 'est'], ['newest'], ['newest'], ['newest'], ['newest'], ['newest']]
wi --> {'lowe': 2, 'er': 2, 'wi': 3, 'id': 3, 'dest': 3}
[['low'], ['low'], ['low'], ['low'], ['low'], ['low', 'e', 'r'], ['low', 'e', 'r'], ['wi', 'd', 'est'], ['wi', 'd', 'est'], ['wi', 'd', 'est'], ['newest'], ['newest'], ['newest'], ['newest'], ['newest']]
wid --> {'lowe': 2, 'er': 2, 'wid': 3, 'dest': 3}
[['low'], ['low'], ['low'], ['low'], ['low'], ['low', 'e', 'r'], ['low', 'e', 'r'], ['wid', 'est'], ['wid', 'est'], ['wid', 'est'], ['newest'], ['newest'], ['newest'], ['newest'], ['newest']]
widest --> {'lowe': 2, 'er': 2, 'widest': 3}
[['low'], ['low'], ['low'], ['low'], ['low'], ['low', 'e', 'r'], ['low', 'e', 'r'], ['widest'], ['widest'], ['widest'], ['newest'], ['newest'], ['newest'], ['newest'], ['newest']]
lowe --> {'lowe': 2, 'er': 2}
[['low'], ['low'], ['low'], ['low'], ['low'], ['lowe', 'r'], ['lowe', 'r'], ['widest'], ['widest'], ['widest'], ['newest'], ['newest'], ['newest'], ['newest'], ['newest']]
lower --> {'lower': 2}
[['low'], ['low'], ['low'], ['low'], ['low'], ['lower'], ['lower'], ['widest'], ['widest'], ['widest'], ['newest'], ['newest'], ['newest'], ['newest'], ['newest']]
{'vocab': ['d', 'o', 'l', 'w', 'n', 'i', 's', 't', 'r', 'e', 'es', 'est', 'lo', 'low', 'ne', 'new', 'newest', 'wi', 'wid', 'widest', 'lowe', 'lower']}
```

در تصویر خروجی چون verbose فعال بوده جزئیات هر مرحله شامل تعداد جفتها، جفتی که بیشترین تکرار را داشته، و corpus پس از merge کردن جفت دیده می‌شود.
خط آخر نیز خروجی نهایی است که همان ترتیب اضافه شدن جفت‌ها به ووکب را دارد.

برای اعمال این توکنایزر آموزش دیده روی کلمات تابع `def tokenize` نوشته شده است. این تابع بر اساس توضیح صفحه 19 فصل دوم کتاب Jurafsky روی ووکب آموزش دیده به ترتیب حرکت می‌کند و سعی می‌کند آن جفت را اعمال کند.

```
print("### Tokenizing ###")
print(bpe_tokenizer.tokenize("lowest"))
```

خروجی

```
### Tokenizing ###
Merged 'es' --> ['l', 'o', 'w', 'es', 't']
Merged 'est' --> ['l', 'o', 'w', 'est']
Merged 'lo' --> ['lo', 'w', 'est']
Merged 'low' --> ['low', 'est']
['low', 'est']
```

باز هم چون verbose روشن بود روال کار نیز چاپ شده است.

همانطور که دیده می‌شود ترکیب‌هایی که انجام شده آورده شده است. ترتیب به همان ترتیب ووکب است و نتیجه ترکیب هم آورده شده است. قابل توجه است که با این که این کلمه out of vocabulary بود اما BPE می‌تواند به شکل مناسبی آن را به low و est که در ووکب بود بشکند و این قطعا در ادامه کار تاثیر خوبی خواهد داشت زیرا انتظار داریم مدل زبانی این دو را بشناسد و ترکیب آن‌ها را با اینکه قبلا ندیده بتواند حدس بزند.

گام دوم

ابتدا داده‌های مورد استفاده را دانلود و آماده می‌کنیم.

```
! pip install tokenizers
! wget https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-103-raw-v1.zip
! wget http://www.gutenberg.org/cache/epub/16457/pg16457.txt
! unzip wikitext-103-raw-v1.zip
```

و از تکه کد زیر استفاده می‌کنیم تا BPE و WP را آموزش دهیم.

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

files = [f"wikitext-103-raw/wiki.{split}.raw" for split in ["test",
"train", "valid"]]
# files = ["pg16457.txt"]

# BPE
tokenizer_bpe = Tokenizer(BPE(unk_token="[UNK]"))
trainer = BpeTrainer(special_tokens=["[UNK]", "[CLS]", "[SEP]", "[PAD]",
"[MASK]"], vocab_size=3*10**6)
tokenizer_bpe.pre_tokenizer = Whitespace()  # Avoid "it is" as a token
tokenizer_bpe.train(files, trainer)

# WordPiece
tokenizer_wp = Tokenizer(WordPiece(unk_token="[UNK]"))
trainer = WordPieceTrainer(special_tokens=["[UNK]", "[CLS]", "[SEP]",
"[PAD]", "[MASK]"], vocab_size=3*10**6)
tokenizer_wp.pre_tokenizer = Whitespace()  # Avoid "it is" as a token
tokenizer_wp.train(files, trainer)
```

اولین چیزی که دیده می‌شود این است که باید توکن ناشناخته یا UNK مشخص شود که چه باشد و پس از این کار توکنایزر دیگر توکنی به آن شکل را رزرو خواهد کرد. در این دو روش چون کاراکترهای زبان هم وجود دارند کلماتی اگر نباشند هم شکسته می‌شوند، اما وقتی که کاراکتری خارج از کاراکترهای آموزش باشد مثل یک emoji توکن ناشناخته انتخاب می‌شود.

در ادامه توکن‌های خاص هم تعریف شده‌اند که مثلا CLS همان توکن اول BERT است که classification با استفاده از آن انجام می‌شود یا SEP توکنی است که وقتی دو جمله بخواهیم بدهیم مثل

داده‌های MNLI این توکن مرز بین دو جمله را مشخص می‌کند.
نکته مهم بعدی این است که با استفاده از Whitespace یک pre_tokenizer اضافه می‌کنیم تا کلمات را از فاصله‌ها بشکند تا چیزی مثل It is یک توکن نشود. این موضوع در sentencepiece که موضوع این تمرین نیست متفاوت می‌شود.
در آخر سایز ووکب را نیز مشخص می‌کنیم که به اندازه کافی بزرگ گذاشتیم تا محدود نشود و بتوانیم تعداد را متوجه شویم. و سپس train را روی فایل‌های مورد نظر اجرا می‌کنیم.

نتیجه BPE آموزش روی ویکیپدیا:

```
output = tokenizer_bpe.encode("This is a deep learning tokenization
tutorial. Tokenization is the first step in a deep learning NLP
pipeline. We will be comparing the tokens generated by each tokenization
model. Excited much?!😁 ?")
print(output.tokens)

['This', 'is', 'a', 'deep', 'learning', 'token', 'ization', 'tutorial',
'.', 'Token', 'ization', 'is', 'the', 'first', 'step', 'in', 'a',
'deep', 'learning', 'NL', 'P', 'pipeline', '.', 'We', 'will', 'be',
'comparing', 'the', 'tokens', 'generated', 'by', 'each', 'token',
'ization', 'model', '.', 'Excited', 'much', '?', '!', '[UNK]', '?']
```

نتیجه WP آموزش روی ویکیپدیا:

```
output = tokenizer_wp.encode("This is a deep learning tokenization
tutorial. Tokenization is the first step in a deep learning NLP
pipeline. We will be comparing the tokens generated by each tokenization
model. Excited much?!😁 ?")
print(output.tokens)

['This', 'is', 'a', 'deep', 'learning', 'token', '##ization',
'tutorial', '.', 'Token', '##ization', 'is', 'the', 'first', 'step',
'in', 'a', 'deep', 'learning', 'NL', '##P', 'pipeline', '.', 'We',
'will', 'be', 'comparing', 'the', 'tokens', 'generated', 'by', 'each',
'token', '##ization', 'model', '.', 'Excited', 'much', '[UNK]', '?']
```

یکی از تفاوت‌هایی که بین خروجی BPE و WP است این است که توکن‌های WP دارای ## هستند مانند '##atorial' که نشان می‌دهد این توکن باید بدون فاصله به کلمه کنارش بچسبد. این مورد برای decoding مفید است که برعکس مسیر tokenization را طی کنیم و از روی توکن‌ها بتوانیم به متن اصلی برسیم ولی این نکته در توکنهای BPE نیست.

در این مثال مشخصا Tokenization و NLP به صورت کامل توکن نبودند و شکسته شده‌اند. همچنین همانطور که گفته شد، کاراکتری که در کاراکترهای آموزشی اصلا نبوده باشد مثل emoji 😁 باعث می‌شود توکنایزر نتواند آن را به جزایی بشکند و [UNK] استفاده می‌شود. البته تعداد توکنها 42 و 40 است که علت ?! قبل از این ایموجی است که در BPE دو کاراکتر جدا شده است ولی ظاهرا در WP با ایموجی همه با هم یک unk در نظر گرفته شده‌اند. این می‌تواند به علت نوع توکنایز کردن دو الگوریتم و تفاوت این دو در استفاده از فرکانس و likelihood باشد. البته با توجه به اینکه ?! در ووکب WP هم وجود دارند احتمال بیشتر این است که کد مرحله encode در WP متفاوت از BPE نوشته شده باشد.

نتیجه BPE آموزش روی 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]', '?']
```

نتیجه WP آموزش روی گوتنبرگ:

```
['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]', '?']
```

همانطور که دیده می‌شود چون گوتنبرگ خیلی دیتای کمتری داشت، در تست هم بد عمل کرده و مجبور به شکست خیلی از کلمات شده که در آموزشش ندیده بوده است. به عنوان مثال tutorial به 't', '##ut','##oria', '##l" شکسته شده است که اصلا مناسب نیست یک کلمه با معنی به 4 توکن نسبتا بی معنی شکسته شود. بنابراین این توکنایزر نسبت به حالتی که روی ویکیپدیا آموزش دیده شده بود خیلی بدتر عمل کرده است. این را می‌توانستیم از روی تعداد اندک توکن‌ها (حدود 15k) نسبت به ویکیپدیا (حدود 800k) هم بفهمیم.

در این مثال کمی بهتر می‌تواند تفاوت و بهبود WP نسبت به BPE را دید چون تعداد توکن 56 و 53 است و مثلا یکی از نمونه ها 'To', '##ken', '##ization' در WP و 'T', 'ok', 'en', 'ization' در BPE

است که نشان می‌دهد WP توانسته توکن‌های بهتری را آموزش ببیند و به توکن‌های کمتری شکست داشته باشد.

ردیف	نام الگوریتم استفاده شده برای توکنایز	تعداد توکن های خروجی الگوریتم برای متن نمونه	
1		توکنایزر آموزش داده شده بر روی کتاب گوتنبرگ	توکنایزر آموزش داده شده بر روی کل داده های ویکی پدیا
2	Byte Pair Encoding (BPE)	56	42
	WordPiece	53	40

گام سوم

```
with open("pg16457.txt", "r") as f:
    g = f.read()

output = tokenizer_bpe.encode(g)
print("BPE", len(output.tokens))
output = tokenizer_wp.encode(g)
print("WP ", len(output.tokens))
```

ردیف	نام الگوریتم استفاده شده برای توکنایز	تعداد توکن های خروجی الگوریتم برای کتاب گوتنبرگ	
1		توکنایزر آموزش داده شده بر روی کتاب گوتنبرگ	توکنایزر آموزش داده شده بر روی کل داده های ویکی پدیا
2	Byte Pair Encoding (BPE)	122,739	127,262
	WordPiece	122,739	124,054

ابتدائا باید اشاره کرد که وقتی یک توکنایزر روی گوتنبرگ آموزش دیده و همان را هم توکنایز می‌کند طبیعتا ووکب آن را شناخته است و همه کلمات را به طور کامل توکنایز می‌کند. اما توکنایزری که روی ویکیپدیا آموزش دیده است و حالا باید گوتنبرگ را توکنایز کند، گوتنبرگ برای out of distribution است و قطعا چیزهایی هست که در گوتنبرگ هست و در ویکیپدیا نبوده است. به همین دلیل است که توکنایزرهای گوتنبرگ توانستند تعداد توکن کمتری نسبت به ویکیپدیا استفاده کنند و البته که اختلاف خیلی هم زیاد نیست و ویکیپدیا را هم می‌توان گفت که خوب عمل کرده است. (اگر برعکس این آزمایش انجام می‌شد قطعا آموزش روی گوتنبرگ نتیجه خیلی بدی روی ویکیپدیا داشت).

ضمنا باید اشاره کرد که در این مورد می‌بینیم که توجه به likelihood به جای تعداد تکرار خالی توانسته باعث شود که WP با تعداد توکن کمتری متن را نسبت به BPE توکنایز کند و برتری نسبی خود را نشان داده است.

همچنین تعداد کل ووکب استخراج شده از خود داده آموزش نیز در ادامه آمده است.

ردیف	نام الگوریتم استفاده شده برای توکنایز	تعداد ووکب خروجی الگوریتم	
1		توکنایزر آموزش داده شده بر روی کتاب گوتنبرگ	توکنایزر آموزش داده شده بر روی کل داده های ویکی پدیا
2	Byte Pair Encoding (BPE)	16,537	777,366
	WordPiece	17,567	812,973

طبیعتا چون دیتای ویکیپدیا خیلی بزرگ‌تر و متنوع‌تر از گوتنبرگ است ووکب بزرگتری نیز استخراج می‌شود. همچنین ویکیپدیا کاراکترهای غیر انگلیسی هم مانند 著 زیاد دارد و به همین دلیل این تایپها لزوما انگلیسی نیستند و تایپهای زبان‌های دیگر نیز وجود دارند.