data drive tokenizer: tokenizer dengan proses training

“Popular” Data-Driven Tokenizer

Alfan F. Wicaksono

Information Retrieval

Fakultas Ilmu Komputer, Universitas Indonesia

Byte-Pair Encoding (BPE) Tokenization Pasangan Byte

- Byte-Pair encoding was originally proposed by Philip Gage (1994) for **compression of strings of text**.

- This compression algorithm works by replacing the most frequent contiguous pairs of characters in a string with **unused placeholder bytes**.

- The associations between placeholder bytes and their original pairs are kept in a **lookup table**. This is useful for **decompression**.

Byte-Pair Encoding (BPE) Tokenization

xxxdqxxxdxo

"xx" is the most frequent pairs

"xx" is replaced by A

ganti XX jadi A

AxdqAxdxo

A -> xx

taro di lookup

"Ax" and "xd" are most frequent pairs;
But, we choose to replace "xd" with B.

xd jadiin B

ABqABxo

A -> xx

B -> xd

"AB" is the most frequent pairs; and is replaced by C

AB palign frequent terus di balikin lagi karena gak ada pasangan lagi

CqCxo

A -> xx

B -> xd

C -> AB

Stop!

The string cannot be compressed anymore.
No pairs of bytes occur more than once.

Byte-Pair Encoding (BPE) Tokenization

With a slight modification, BPE was used for tokenization when pretraining a “large language model”.

Training Steps:

- Compute the unique set of words used in the corpus (and their frequencies);
- Build the base vocabulary by taking all the single characters;
- Successively merge the most frequent pair of adjacent characters into a new, 2-character token and all instances of the pair are replaced by this new token. Don't forget to add this new token into the vocabulary as well; karekter yang paling sering muncul di merging
- Steps 2 and 3 are repeated until we get a desired size of vocabulary.

Byte-Pair Encoding (BPE) Tokenization

Corpus: “halo alo halo halo halo alo balon balon halo balon hakim baki hakim”

Pretokenization (Pretokenizer), kayak splitting whitespace

We need a normalization and pre-tokenization steps here

Corpus: [(halo, 5), (alo, 2), (balon, 3), (hakim, 2), (baki, 1)]

Corpus: [(“h” “a” “l” “o”, 5), (“a” “l” “o”, 2), (“b” “a” “l” “o” “n”, 3), (“h” “a” “k” “i” “m”, 2), (“b” “a” “k” “i”, 1)]

Vocab: [h, a, l, o, b, n, k, i, m]
Merge Rule: {}

Desired vocab size = **13**

Byte-Pair Encoding (BPE) Tokenization

Corpus: [("h" "a" "l" "o", **5**), ("a" "l" "o", **2**), ("b" "a" "l" "o" "n", **3**),
("h" "a" "k" "i" "m", **2**), ("b" "a" "k" "i", **1**)]

Vocab: [h, a, l, o, b, n, k, i, m]

Merge Rule: {}

Paling sering ketemu itu "l""o"

The pair **("l", "o")** is the most frequent adjacent pairs with 10 times of occurrence in the corpus. So, we merge them and them to the vocab!

Byte-Pair Encoding (BPE) Tokenization

Corpus: [("h" "a" "lo", **5**), ("a" "lo", **2**), ("b" "a" "lo" "n", **3**),
("h" "a" "k" "i" "m", **2**), ("b" "a" "k" "i", **1**)]

jangan lupa tambahin ke vocab.

Vocab: [h, a, l, o, b, n, k, i, m, **lo**]

Merge Rule: {(l, o): lo}

lo yang tadi di merge ke merge Rule

Byte-Pair Encoding (BPE) Tokenization

Corpus: [("h" "a" "lo", **5**), ("a" "lo", **2**), ("b" "a" "lo" "n", **3**),
("h" "a" "k" "i" "m", **2**), ("b" "a" "k" "i", **1**)]

Vocab: [h, a, l, o, b, n, k, i, m, **lo**]

Merge Rule: {(l, o): lo}

ekarang paling frequent alo

The pair **("a", "lo")** is now the most frequent adjacent pairs.

Byte-Pair Encoding (BPE) Tokenization

Corpus: [(“h” “alo”, **5**), (“alo”, **2**), (“b” “alo” “n”, **3**),
(“h” “a” “k” “i” “m”, **2**), (“b” “a” “k” “i”, **1**)]

Vocab: [h, a, l, o, b, n, k, i, m, **lo**, **alo**]
Merge Rule: {(l, o): lo, (a, lo): alo}

Byte-Pair Encoding (BPE) Tokenization

Corpus: [("h" "alo", **5**), ("alo", **2**), ("b" "alo" "n", **3**),
("h" "a" "k" "i" "m", **2**), ("b" "a" "k" "i", **1**)]

Vocab: [h, a, l, o, b, n, k, i, m, **lo**, **alo**]
Merge Rule: {(l, o): lo, (a, lo): alo}

2 + 1 = 3, karena ada 2 sebagai okurensi hakim dan 1 di baki.

Now the most frequent pairs are **("a", "k")** and **("k", "i")** with 3 occurrences. Suppose, we choose to merge **("a", "k")** first.

Byte-Pair Encoding (BPE) Tokenization

Corpus: [(“h” “alo”, **5**), (“alo”, **2**), (“b” “alo” “n”, **3**),
(“h” “ak” “i” “m”, **2**), (“b” “ak” “i”, **1**)]

Vocab: [h, a, l, o, b, n, k, i, m, **lo**, **alo**, **ak**]
Merge Rule: {(l, o): lo, (a, lo): alo, (a, k): ak}

Byte-Pair Encoding (BPE) Tokenization

Corpus: [(“h” “alo”, **5**), (“alo”, **2**), (“b” “alo” “n”, **3**),
(“h” “ak” “i” “m”, **2**), (“b” “ak” “i”, **1**)]

Vocab: [h, a, l, o, b, n, k, i, m, **lo**, **alo**, **ak**]
Merge Rule: {(l, o): lo, (a, lo): alo, (a, k): ak}

Now the most frequent pairs is **(“ak”, “i”)**

Byte-Pair Encoding (BPE) Tokenization

Corpus: [(“h” “alo”, **5**), (“alo”, **2**), (“b” “alo” “n”, **3**),
(“h” “aki” “m”, **2**), (“b” “aki”, **1**)]

Vocab: [h, a, l, o, b, n, k, i, m, **lo**, **alo**, **ak**, **aki**]
Merge Rule: {(l, o): lo, (a, lo): alo, (a, k): ak, (ak, i): aki}

It’s time to stop since the vocabulary size has reached **13**, as this is what we want.

Byte-Pair Encoding (BPE) Tokenization

After we train the BPE tokenizer, then what we can do if we want to tokenize a new text?

Tokenization Steps:

- Normalize and pre-tokenize the text;
- Split the words into lists of single characters;
- Apply the **merge rules** learned in the training phase.

Byte-Pair Encoding (BPE) Tokenization

Sentence: “halo kak hakim“

Sentence: [(“h“ “a“ “l“ “o“), (“k“ “a“ “k“), (“h“ “a“ “k“ “i“ “m“)]

Merge Rules:

(l, o): lo
(a, lo): alo
(a, k): ak
(ak, i): aki

Byte-Pair Encoding (BPE) Tokenization

Sentence: “halo kak hakim”

Sentence: [(“h” “a” “lo”), (“k” “a” “k”), (“h” “a” “k” “i” “m”)]

Merge Rules:

(l, o): lo
(a, lo): alo
(a, k): ak
(ak, i): aki

Byte-Pair Encoding (BPE) Tokenization

Sentence: “halo kak hakim”

Sentence: [(“h” “alo”), (“k” “a” “k”), (“h” “a” “k” “i” “m”)]

Merge Rules:

(l, o): lo

(a, lo): alo

(a, k): ak

(ak, i): aki

Byte-Pair Encoding (BPE) Tokenization

Sentence: “halo kak hakim“

Sentence: [(“h“ “alo“), (“k“ “ak“), (“h“ “ak“ “i“ “m“)]

Merge Rules:

(l, o): lo
(a, lo): alo
(a, k): ak
(ak, i): aki

Byte-Pair Encoding (BPE) Tokenization

Sentence: “halo kak hakim“

Sentence: [(“h“ “alo“), (“k“ “ak“), (“h“ “aki“ “m“)]

Tokenized Sentence: [“h“, “alo“, “p“, “ak“, “h“, “aki“, “m“]

Merge Rules:

(l, o): lo
(a, lo): alo
(a, k): ak
(ak, i): aki

WordPiece Tokenization

- It was developed by Google for training their language models.

- WordPiece is similar to BPE with the difference lying on two things:
  - The scoring function used to merge two adjacent pairs;
  - The way they tokenize a new string.

dari fase training cara tokenisasi nya agak berbeda.

awalan dan tengah2 berbeda, misalnya

high, h awal di "high" beda dengan h diakhir "high", dimana awal itu h, diakhir itu ##h

- Like BPE, WordPiece starts from a base vocabulary containing single characters, but with prefix **“##”** for characters **inside the words**.

yang pertama gak pake pager, cuman kedua, dst.

pergi → **p ##e ##r ##g ##i**

WordPiece Tokenization

Corpus: [(hai, 5), (lai, 2), (hau, 6), (kau, 3), (haus, 10)]

Corpus: [(h ##a ##i, 5), (l ##a ##i, 2), (h ##a ##u, 6),
(k ##a ##u, 3), (h ##a ##u ##s, 10)]

Vocab: [h, l, k, ##a, ##i, ##u, ##s]

Unlike BPE, WordPiece does not need to keep merge rules; what WordPiece needs to tokenize a new text is just a learned vocabulary.

WordPiece Tokenization

- How to merge an adjacent pair?
- Instead of selecting the most frequent pair, WordPiece computes a score for each pair $(\boldsymbol{X}, \boldsymbol{Y})$, using:

sama caranya dengan menghitung collocation (kata-kata yang sering muncul bersama)

$$score(X, Y) = \frac{freq(X, Y)}{freq(X) \cdot freq(Y)}$$

karena kalo rumah sakit kan gabungan dari rumah dan sakit bisa aja tetap gede

- This scoring function favors a pair (X,Y) that **tends to occur together more frequently** than each of its components individually.

WordPiece Tokenization

Corpus: [(hai, 5), (lai, 2), (hau, 6), (kau, 3), (haus, 10)]

Corpus: [(h ##a ##i, 5), (l ##a ##i, 2), (h ##a ##u, 6),
(k ##a ##u, 3), (h ##a ##u ##s, 10)]

Vocab: [h, l, k, ##a, ##i, ##u, ##s]

(h, ##a) is the most frequent pair (21 times).
h occurs 21 times, and **##a** appears 26 times
Score(h, ##a) = 21 / (21 * 26) = **1 / 26**

(##u, ##s) appears 10 times.
##u occurs 19 times, and **##s** appears 10 times
Score(h, ##a) = 10 / (19 * 10) = **1 / 19**

Merge adjacent pair with the highest score!

WordPiece Tokenization

Modification from:
https://huggingface.co/learn/nlp-course/chapter6/6

```
corpus = [("hai", 5), ("lai", 2), ("hau", 6), ("kau", 3), ("haus", 10)]

def base_vocab(corpus):
    vocab = []
    for word, _ in corpus:
        first_char = word[0]
        tail = word[1:]
        if first_char not in vocab:
            vocab.append(word[0])
        for letter in tail:
            if f"##{letter}" not in vocab:
                vocab.append(f"##{letter}")
    return vocab

vocab = sorted(base_vocab(corpus))
print(vocab)  #['##a', '##i', '##s', '##u', 'h', 'k', 'l']
```

WordPiece Tokenization

Modification from:
https://huggingface.co/learn/nlp-course/chapter6/6

```
def create_word_splits(corpus):
    splits = {}
    for word, freq in corpus:
        split = []
        for i, char in enumerate(word):
            if i == 0:
                split.append(char)
            else:
                split.append(f"##{char}")
        splits[word] = split
    return splits

initial_word_splits = create_word_splits(corpus)
print(initial_word_splits)
```

```
{'hai': ['h', '##a', '##i’],
 'lai': ['l', '##a', '##i’],
 'hau': ['h', '##a', '##u’],
 'kau': ['k', '##a', '##u’],
 'haus': ['h', '##a', '##u', '##s']}
```

WordPiece Tokenization

Modification from:
https://huggingface.co/learn/nlp-course/chapter6/6

```
from collections import defaultdict

def pair_scores(corpus, word_splits):
    individual_freqs = defaultdict(int)
    pair_freqs = defaultdict(int)
    for word, freq in corpus:
        split = word_splits[word]
        if len(split) == 1:
            individual_freqs[split[0]] += freq
            continue
        for i in range(len(split) - 1):
            pair = (split[i], split[i + 1])
            individual_freqs[split[i]] += freq
            pair_freqs[pair] += freq
        individual_freqs[split[-1]] += freq

    scores = {pair: freq / (individual_freqs[pair[0]] *
                    individual_freqs[pair[1]])
                        for pair, freq in pair_freqs.items()}
    return scores
```

WordPiece Tokenization

Modification from:
https://huggingface.co/learn/nlp-course/chapter6/6

```
pair_scores = pair_scores(corpus, initial_word_splits)
for pair, score in enumerate(pair_scores.items()):
    print(f"{pair}: {score}")
```

```
0: (('h', '##a'), 0.038461538461538464)
1: (('##a', '##i'), 0.038461538461538464)
2: (('l', '##a'), 0.038461538461538464)
3: (('##a', '##u'), 0.038461538461538464)
4: (('k', '##a'), 0.038461538461538464)
5: (('##u', '##s'), 0.05263157894736842)
```

WordPiece Tokenization

Modification from:
https://huggingface.co/learn/nlp-course/chapter6/6

```
desired_vocab_size = 30
splits = initial_word_splits

while len(vocab) < desired_vocab_size:

    scores = pair_scores(corpus, splits)

    best_pair, max_score = "", None
    for pair, score in scores.items():
        if max_score is None or max_score < score:
            best_pair = pair
            max_score = score

    splits = merge_pair(*best_pair, splits)

    new_token = ( best_pair[0] + best_pair[1][2:]
                  if best_pair[1].startswith("##")
                  else best_pair[0] + best_pair[1])
    vocab.append(new_token)
```

The Whole Training Process

Like BPE, merge adjacent pairs until we get a desired vocab size

Suppose we have a procedure for merging pairs on splits

WordPiece Tokenization

After we train a WordPiece tokenizer, how to tokenize a new string?

- What we need is just the trained vocab;
- First, we pre-tokenize the string;
- Second, we find the longest prefix and split it, then we repeat the process on the rest of the string, and so on.

WordPiece Tokenization

```
Vocab =

['##a', '##h', '##i', '##n', '##s', '##t', '##u', 'H', 'T',
'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab',
'##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'Hu', 'Hug', 'Hugg']
```

Hugging

WordPiece Tokenization

```
Vocab =

['##a', '##h', '##i', '##n', '##s', '##t', '##u', 'H', 'T',
'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab',
'##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'Hu', 'Hug', 'Hugg']
```

Hugging **Hugging**

WordPiece Tokenization

```
Vocab =

['##a', '##h', '##i', '##n', '##s', '##t', '##u', 'H', 'T',
'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab',
'##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'Hu', 'Hug', 'Hugg']
```

Hugging   Hugging   Hugg ##ing

WordPiece Tokenization

```
Vocab =

['##a', '##h', '##i', '##n', '##s', '##t', '##u', 'H', 'T',
'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab',
'##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'Hu', 'Hug', 'Hugg']
```

Hugging   **Hugging**   **Hugg ##ing**   **Hugg ##ing**

WordPiece Tokenization

```
Vocab =

['##a', '##h', '##i', '##n', '##s', '##t', '##u', 'H', 'T',
'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab',
'##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'Hu', 'Hug', 'Hugg']
```

Hugging   Hugging   Hugg ##ing   Hugg ##ing

Hugg ##i ##ng

WordPiece Tokenization

```
Vocab =

['##a', '##h', '##i', '##n', '##s', '##t', '##u', 'H', 'T',
'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab',
'##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'Hu', 'Hug', 'Hugg']
```

Hugging   **Hugging**   **Hugg ##ing**   **Hugg ##ing**

Hugg ##i ##n ##g

WordPiece Tokenization

```
Vocab =

['##a', '##h', '##i', '##n', '##s', '##t', '##u', 'H', 'T',
'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab',
'##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'Hu', 'Hug', 'Hugg']
```

Hugging   Hugging   Hugg ##ing   Hugg ##ing

Hugg ##i ##n ##g

Not Found!

WordPiece Tokenization

```
Vocab =

['##a', '##h', '##i', '##n', '##s', '##t', '##u', 'H', 'T',
'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab',
'##fu', 'Fa', 'Fac', '##ct', '##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'Hu', 'Hug', 'Hugg']
```

Hugging   Hugging   Hugg ##ing   Hugg ##ing

Hugg ##i ##n [UNK]

If it's not found in the vocab, we replace it with the special token **[UNK]**