COE548: LARGE LANGUAGE MODELS

Topic: Tokenization

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Outline

Tokenization

- Character vs word vs sub-word level tokenization
- WordPiece
- Byte Pair Encoding (BPE)

Tokenization

- The process of converting a sequence of text into smaller parts, known as tokens.
- These tokens can be as small as characters or as long as words or sentences.
- The primary reason this process matters is that it helps machines understand human language by breaking it down into bite-sized pieces (numeric), which are easier to analyze.

- Visualizing the token chatGPT considers per your prompt:
  - *platform.openai.com/tokenizer*

Tokenization: Character vs word vs sub-word level tokenizing

“This is tokenizing.”

- Character level:
  - *[T][h][i][s][i][s][t][o][k][e][n][i][z][i][n][g][.]*
- Word level:
  - *[This][is][tokenizing][.]*
- Sub-word level:
  - *[This][is][token][izing][.]*

Tokenization: Letter by letter?

- If we encode letter by letter, then it becomes hard to understand sentiment. For example:

SILENT ↔ LISTEN

same letters but ordered differently

- Thus, it may be better to encode in a word-by-word basis. There is character level tokenizing, word level, and subword level.

Tokenization: Word level

- Word-level tokenization splits a piece of text into words based on a delimiter.
- The most commonly used delimiter is space. You can also split your text using more than one delimiter, like space and punctuation marks.
- Depending on the delimiter you used, you will get different word-level tokens.

Tokenization: Word level

- Word-based tokenization can be easily done using Python's split() method. Apart from that, there are plenty of libraries in Python — NLTK, spaCy, Keras, Gensim, which can help you perform tokenization easily.

```
# Example sentence
sentence = "Natural language processing is fascinating."

# Word-level tokenization using split()
tokens = sentence.split()

# Display the tokens
print(tokens)
```

```
['Natural', 'language', 'processing', 'is', 'fascinating.']
```

Tokenization: Word level

```python
import nltk
from nltk.corpus import words

# Download the words corpus if not already downloaded
nltk.download('words')

# Get the list of English words
word_list = words.words()

# Assign numeric IDs to each word
word_to_id = {word: idx for idx, word in enumerate(word_list)}

# Display the first 10 words with their IDs
for word in list(word_to_id.items())[:10]:
    print(word)
```

```
('A', 0)
('a', 1)
('aa', 2)
('aal', 3)
('aalii', 4)
('aam', 5)
('Aani', 6)
('aardvark', 7)
('aardwolf', 8)
('Aaron', 9)
```

⋮

Tokenization: Word level

- Why we would (or wouldn't) want to take punctuation into account:

"Is this efficient?"

[Is][this][efficient?] → [efficient?]

Notice that the word "efficient" has punctuation attached to it.

"This is not efficient." → [efficient.]

We now have two tokens representing the same word.

This will lead the model to learn different representations of the word "efficient" and will make the representation of words (tokens) suboptimal. The number of representations a model will learn will explode (each word × number of punctuations used in a language).

Tokenization: Subword level

- The basic idea behind subword tokenization is to combine the best aspects of character and word tokenization.
- On the one hand, we want to split rare words into smaller units to allow the model to deal with complex words and misspellings.
- On the other hand, we want to keep frequent words as unique entities so that we can keep the length of our inputs to a manageable size.
- The main distinguishing feature of subword tokenization is that it is learned from the pre-training corpus using a mix of statistical rules and algorithms.

Tokenization: Subword level

- The subword-based tokenization algorithms generally use a special symbol to indicate which word is the start of the token and which word is the completion of the start of the token.
- For example, “tokenization” can be split into “token” and “##ization” which indicates that “token” is the start of the word and “##ization” is the completion of the word.
- Going back to first example:

“This is tokenizing.” ⟶ [This][is][token][##izing][.]

Tokenization: Subword level

- Many models which have obtained state-of-the-art results in the English language use some kind of subword-tokenization algorithms.
- A couple common subword-based tokenization algorithms are WordPiece used by BERT and Byte-Pair Encoding by GPT-2.
  - *They are particularly effective in handling out-of-vocabulary words and languages with rich morphology.*

WordPiece

- Initialize vocabulary
  - *Start with an initial vocabulary that typically includes all individual characters and some special tokens like [CLS], [SEP], [UNK], etc.*
  - *Divide each word in the training corpus into the sequence of letters that make it up (e.g., learn: l, ##e, ##a, ##r, ##n).*
  - *Store one occurrence per elementary unit.*
- Training Process
  - *The algorithm counts the frequency of all possible subwords (initially characters) in the training data.*
  - *Calculate a score for each adjacent pair of subwords (two consecutive characters or subwords) to decide which pair to merge.*
  - *The pair with the highest score is merged into a new subword. This process is repeated iteratively until a predefined vocabulary size is reached.*

WordPiece: Pair Scores

- The score $S(x, y)$ for a pair of subwords x and y is often calculated using the following equation:

$$S(x, y) = \frac{\text{Frequency}(xy)}{\text{Frequency}(x) \times \text{Frequency}(y)}$$

- Example:
  - *Corpus: “Natural language processing is fascinating”*
  - *Tokenized corpus = [["n”,”##a ”,” ##t ”,” ##u ”,” ##r ”,” ##a ”,” ##l“], ["l ”,” ##a ”,” ##n ”,” ##g ”,” ##u ”,”## a ”,” ##g ”,”## e“], ["p ”,” ##r ”,” ##o ”,” ##c ”,” ##e ”,” ##s ”,” ##s ”,” ##i ”,” ##n ”,” ##g“], ["i ”,” ##s“], ["f ”,”## a ”,”## s ”,” ##c ”,” ##i ”,” ##n ”,” ##a ”,” ##t ”,” ##i ”,” ##n ”,” ##g”]]*
  - *vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##o', ‘##g', ‘##s', ‘##c', ‘##a', ‘##r', ‘##i’, ‘i’, 'p’}*

WordPiece: Pair Scores

- Example:
  - *corpus = [["n”,”##a ”,” ##t ”,” ##u ”,” ##r ”,” ##a ”,” ##l“], ["l ”,” ##a ”,” ##n ”,” ##g ”,” ##u ”,”## a ”,” ##g ”,”## e“], ["p”,” ##r ”,” ##o ”,” ##c ”,” ##e ”,” ##s ”,” ##s ”,” ##i ”,” ##n ”,” ##g“], ["i ”,” ##s“], ["f ”,”## a ”,”## s ”,” ##c ”,” ##i ”,” ##n ”,” ##a ”,” ##t ”,” ##i ”,” ##n ”,” ##g”]]*
  - *vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##o', ‘##g', ‘##s', ‘##c', ‘##a', ‘##r', ‘##i’, ‘i’, 'p’}*

  - *Highest frequency pair: “ng” (3 counts, i.e., freq(ng) = 3)*
  - *However: freq(n)=5 and freq(g)=4*
  - *Thus score is 3/20 (maybe not the largest score?)*

WordPiece: Pair Scores

- Example:
  - *corpus = [["n”,”##a ”,” ##t ”,” ##u ”,” ##r ”,” ##a ”,” ##l“], ["l ”,” ##a ”,” ##n ”,” ##g ”,” ##u ”,”## a ”,” ##g ”,”## e“], ["p”,” ##r ”,” ##o ”,” ##c ”,” ##e ”,” ##s ”,” ##s ”,” ##i ”,” ##n ”,” ##g“], ["i ”,” ##s“], ["f ”,”## a ”,”## s ”,” ##c ”,” ##i ”,” ##n ”,” ##a ”,” ##t ”,” ##i ”,” ##n ”,” ##g”]]*
  - *vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##o', ‘##g', ‘##s', ‘##c', ‘##a', ‘##r', ‘##i’, ‘i’, 'p’}*

  - *Frequency pair of “pr” (1 counts, i.e., freq(pr) = 1)*
  - *Also, freq(p)=1 and freq(r)=2*
  - *Thus score is 1/2 (“pr” > “ng”)*

WordPiece: Pair Scores

- Example:
  - *corpus = [["n”,”##a ”,” ##t ”,” ##u ”,” ##r ”,” ##a ”,” ##l“], ["l ”,” ##a ”,” ##n ”,” ##g ”,” ##u ”,”## a ”,” ##g ”,”## e“], ["p”,” ##r ”,” ##o ”,” ##c ”,” ##e ”,” ##s ”,” ##s ”,” ##i ”,” ##n ”,” ##g“], ["i ”,” ##s“], ["f ”,”## a ”,”## s ”,” ##c ”,” ##i ”,” ##n ”,” ##a ”,” ##t ”,” ##i ”,” ##n ”,” ##g”]]*
  - *vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##o', ‘##g', ‘##s', ‘##c', ‘##a', ‘##r', ‘##i’, ‘i’, 'p’}*

  - *Iteration #1: Highest score: “p” and “##r” ⟶ “pr”*
  - *New vocab = vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##o', ‘##g', ‘##s', ‘##c', ‘##a', ‘##r', ‘##i’, ‘i’, ‘pr’}*

WordPiece: Pair Scores

- Example:
  - *corpus = [["n”,”##a ”,” ##t ”,” ##u ”,” ##r ”,” ##a ”,” ##l“], ["l ”,” ##a ”,” ##n ”,” ##g ”,” ##u ”,”## a ”,” ##g ”,”## e“], ["pr”,” ##o ”,” ##c ”,” ##e ”,” ##s ”,” ##s ”,” ##i ”,” ##n ”,” ##g“], ["i ”,” ##s“], ["f ”,”## a ”,”## s ”,” ##c ”,” ##i ”,” ##n ”,” ##a ”,” ##t ”,” ##i ”,” ##n ”,” ##g”]]*
  - *vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##o', ‘##g', ‘##s', ‘##c', ‘##a', ‘##r', ‘##i’, ‘i’, ‘pr’}*

  - *Iteration #2: Highest score: “pr” and “##o” ⟶ “pro”*
  - *New vocab = vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##g', ‘##s', ‘##c', ‘##a', ‘##r', ‘##i’, ‘i’, ‘pro’}*

WordPiece: Pair Scores

- Example:
  - *corpus = [["n”,”##a ”,” ##t ”,” ##u ”,” ##r ”,” ##a ”,” ##l“], ["l ”,” ##a ”,” ##n ”,” ##g ”,” ##u ”,”## a ”,” ##g ”,”## e“], ["pro”,” ##c ”,” ##e ”,” ##s ”,” ##s ”,” ##i ”,” ##n ”,” ##g“], ["i ”,” ##s“], ["f ”,”## a ”,”## s ”,” ##c ”,” ##i ”,” ##n ”,” ##a ”,” ##t ”,” ##i ”,” ##n ”,” ##g”]]*
  - *vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##o', ‘##g', ‘##s', ‘##c', ‘##a', ‘##r', ‘##i’, ‘i’, ‘pro’}*

  - *Iteration #3: Highest score: “u” and “##r” ⟶ “ur”*
  - *New vocab = vocab = {‘##u', 'f', ‘##t', ‘##l’, ‘l’, ‘##e', ‘##n’, ‘n’, ‘##g', ‘##s', ‘##c', ‘##a’, ‘ur', ‘##i’, ‘i’, ‘pro’}*

WordPiece: Pair Scores

- The “likelihood” score $S(x, y)$ for a pair of subwords x and y is often calculated using the following equation:

$$S(x, y) = \frac{\text{Frequency}(xy)}{\text{Frequency}(x) \times \text{Frequency}(y)}$$

- Example:
  - *corpus = ["natural", "language", "processing", "is", "fascinating.”]*

- Thought experiment: As is - what would you do about the “.”?

WordPiece: Thought Experiment

- Would the rarer words always be merged first?

WordPiece: Tokenization Process

- During tokenization, the text is split into the largest possible subwords that are present in the vocabulary. If a word is not found in the vocabulary, it's split into smaller subwords or characters until it matches something in the vocabulary.

- **Longest Match First**: The algorithm looks for the longest sequence of characters at the beginning of the word that exists in the vocabulary.
- **Subword Formation**: Once the longest match is found, that subword is added to the token list, and the remaining part of the word is processed similarly.
- **Continuation Tokens**: If the match is not at the start of the word (but rather a continuation), it is prefixed with ## to indicate that it is a subword continuation.

Byte-Pair Encoding (BPE)

Very similar process to WordPiece.

- **Initialize Vocabulary**: Start with a vocabulary containing all characters.
- **Count Pair Frequencies**: Count the frequencies of all adjacent character pairs.
- **Merge Pairs**: Identify the most frequent pair and merge it into a new token. Add this new token to the vocabulary.
- **Repeat**: Continue the process until a predefined vocabulary size is reached.

Byte-Pair Encoding (BPE)

- Key differences:
  - ***Frequency-Based Merging**: BPE merges the most frequent pairs based purely on frequency.*
  - ***No Special Handling of Continuation**: BPE doesn't typically mark subwords with prefixes like ##, so the output tokens can be any combination of characters or subwords.*
    - **Example**: For the word "huggingface", BPE might produce tokens like
    - ['hug', 'ging', 'face']