Lecture 8: Byte Pair Encoding

Out-of-vocabulary (OOV) words are words that do **not exist in a model's vocabulary** and therefore cannot be recognized or processed directly by the model.

Word based tokenizer:

Splits word by word.

The problems of **word based tokenizer:**

- **Out-of-vocabulary (OOV) words** break the model because unseen words can't be represented.
- **Huge vocabulary size**, making models large, slow, and memory-heavy.
- **Poor handling of spelling mistakes**, treating even tiny typos as completely new words.
- **No understanding of subword structure**, so related words like *run*, *running*, *runner* become separate tokens.

Character based tokenizer:

Splits by character.

Advantages

- **No OOV problem** — every word can be built from characters.
- **Small vocabulary** — usually under 200 tokens.
- **Handles typos and variations well**.
- **Works across all languages without retraining vocab.**

Disadvantages

- **Harder to capture meaning**, as characters carry less semantic information than words/subwords.
- **Longer sequences**, since words break into many tokens.
- **Slower training and inference** because of longer sequences.

Subword based tokenizer:

A mix between character level and word level tokenization.

Rule 1: DO NOT split frequently used words into smaller subwords.

Rule 2: Split the rare words into smaller, meaningful subwords (even character level). Example: "boys" should be splitted into "boy" and "s"

Advantages of Subword-Based Tokenization

- **Captures shared roots and meanings across related words**

  Subword methods break words into meaningful pieces, so words like

  - *token*
  - *tokens*
  - *tokenizing*
  - *tokenization*

    all share the core subword **"token"**, helping the model understand they're related.

- **Handles different suffixes cleanly**

  For example, *tokenization* and *modernization* both contain **"ization"**,

  so the tokenizer may split them like:

  - *token + ization*
  - *modern + ization*

    The model learns that **"ization"** is a common suffix and carries similar meaning across words.
- **Reduces vocabulary size without losing expressiveness**

  Instead of having separate tokens for *happy*, *happier*, *happiest*, it can reuse:

  - *happi + er*
  - *happi + est*

    Fewer tokens, more flexibility.
- **Handles rare or new words gracefully**

  If the word *hyperquantization* appears, the tokenizer can break it into known subwords:

  - *hyper + quant + ization*

    No OOV issues, and the meaning is still partly preserved.
- **Supports multiple languages better**

  Shared subwords like **"tion"**, **"ing"**, **"un"**, **"re"** help the model generalize patterns across languages and word forms.
- **Balances meaning and sequence length**

  Words aren't too small (like character-based) or too large (like word-based).

  This "just-right size" captures semantics while keeping sequences shorter than character-level tokenization.

Byte Pair Encoding is a subword tokenization algorithm.

BPE Algoritm: Most common pair of consecutive bytes of data is replaced with a byte that does not occur in the data.

Example:

Original data: aaabdaaabac

a. The byte pair 'aa' occurs the most. We will replace it with z as z does not occur in the data.
b. Compressed data: ZabdZabac
c. The next common bbyte pair is 'ab'. We will replace this by Y.
d. Compressed data: ZYdZYac
e. Now, ac occurs only once. So, no need to encode it.
f. Repeat until no byte pair occurs more than once.

How is the BPE algorithm use for LLMs?

a. BPE ensures that most common words in the vocabulary are represented as a single token, while rare words are broken down into two or more subword tokens.
b. Let us consider the below dataset of words: {"old":7, "older":3, "finest":9, "lowest":4}
c. Preprocessing: We need to add end token **</w>** at the end of each word: {"old</w>":7, "older</w>":3, "finest</w>":9, "lowest</w>":4}
d. Let us now split words into characters and count their frequency:

Number	Token	Frequency
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13
11	t	13
12	w	4

e. Next step in the BPE algorithm is to look for the most frequent pairing.

f. Merge them and perform the same iteration again and again until we reach the token limit or iteration limit.

g. **Iteration 1:** Start with second most common token “e”.

Most common byte pair starting with e: “es”

Number	Token	Frequency
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13
12	w	4
13	es	9 + 4 = 13

h. It can be said from the table that ‘**e’ only comes with ‘s’ in this dataset. Otherwise, 0 times.**

Number	Token	Frequency
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13

i. **Iteration 2:** Merge the tokens “es” and “t” as they have appeared 13 times in our dataset.

j. By doing this, our model now knows that “est” is a common root word.

k. Again, “est” and “</w>” forms another byte pair. We see that, “est</w>” has appeared 13 times.

l. **Iteration 3:** Merge tokens “est” and “</w>”.

1	</w>	23 - 13 = 10
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13 - 13 = 0
15	est</w>	13

Note: If we had left it at only ‘est’, there would’ve been no difference between words like **‘estimate’ and ‘highest’.** So, doing this helps the algorithm understand difference of when words are ending or not ending.

m. **Iteration 4:** Merge ‘o’ and ‘l’, has appeared 10 times.

Number	Token	Frequency
1	</w>	23
2	o	14 - 10 = 4
3	l	14 - 10 = 4
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13
15	ol	7 + 3 = 10

n. **Iteration 5:** Merge ‘ol’ and ‘d’, has appeared 10 times.

Number	Token	Frequency
1	</w>	23 - 13 = 10
2	o	14 - 10 = 4
3	l	14 - 10 = 4
4	d	10 - 10 = 0
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13 - 13 = 0
15	est</w>	13
16	ol	7 + 3 = 10 - 10 = 0
17	old	7 + 3 = 10

So, the root words are getting captured this way.

o. ‘f’, ‘i’, ‘n’ appear 9 times. But we just have one word with these characters. So, we are not merging them.

p. Let us remove the tokens with 0 count.

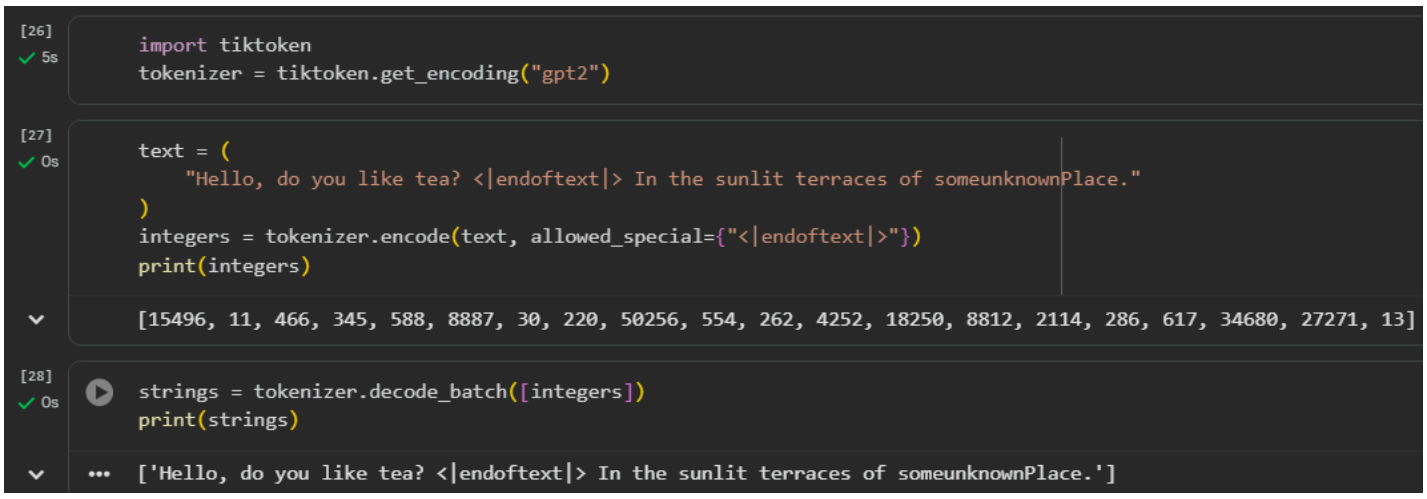
Number	Token	Frequency
1	</w>	10
2	o	4
3	l	4
4	e	3
5	r	3
6	f	9
7	i	9
8	n	9
9	w	4
10	est</w>	13
11	old	10

These are the final tokens that will be used for the further process of the LLM. This list of tokens will serve as the vocabulary.

The stopping criteria can either be the **token count or the number of iterations.**

Open AI uses tiktokens for tokenizing.

Code:



The token is assigned a relatively large token ID, namely, 50256.

In fact, the BPE tokenizer, which was used to train models such as GPT-2 and 3, and the original GPT model used in ChatGPT, has a total vocabulary size of 50,257, with being assigned the larget token ID.

The BPE tokenizer above encodes and decoeds unknown words, such as "someunknownPlace" correctly. The BPE tokenizer can handle any unkown word. How does it achieve this without using <|unk|> tokens?

The algoritm underlyinng BPE breaks down words that aren't in its predefined vocabulary into smaller sub word units or even individual characters. This enables it to handle out of vocabulary words.