* when working w/ multiple text sources, we add <|endoftext|> tokens between these texts
* these <|endoftext|> tokens act as markers, signaling the start of end of a particular segment
* This leads to more effective processing and understanding by the LLM

Byte pair encoding

* words themselves are broken into subwords, and these are the tokens

(ch) (as) (ed); one possible subword breakdown

used to train GPT-2, GPT 3, and OG Model for ChatGPT

Tokenization Algorithms

① word based ② sub-word based ③ character based

① word based

My hobby is playing cricket

↓

['My', 'hobby', 'is', 'playing', 'cricket']

o Problem: what to do w/ Out of vocab (OOV) words, diff meaning of similar words [illegible] → similar, not

③ character based

↓

My hobby is playing cricket

['M', 'y', 'h', ...]

↓

very small vocabulary, Every language has fixed # of characters (English ≈ 256)

↳ solves OOV

problem

o meaning associated w/ words is completely lost. tokenized sequence is much longer than initial raw

'Dinosaur' (Character based tokenization)
8 tokens

② Sub-word based tokenization
Rule 1: Do not split frequently used words into smaller subwords
Rule 2: Split rare words into smaller, meaningful subwords

ex:
"boy" should not be split
"boys" should be split into "boy" and "s"

① The subword splitting helps model learn that different words w/ same root word as "token"
② Helps model learn that like "tokens" and "tokenizing" are similar in meaning

"tokenization" and "modernization" are made up of diff. root words but have same suffix "ization" and are used in syntatic situations.

* Byte Pair Encoding (BPE) is a subword tokenization algorithm *
* BPE also: Most common pair of consecutive bytes of data is replaced w/ a byte that does not occur in data.

Ex: original data = 'aaabdaaabac'
ⓐ byte pair 'aa' occurs the most, we will replace w/ Z as Z does not occur in the data (aa is a PAIR)
ⓑ compressed data: ZabdZabac
ⓒ Next common byte pair is 'ab', we will replace this by Y
ⓓ ZYdZYac → only byte pair left. Appears only once, so we do not encode it.

WdWac if you want

④ How is BPE used for LLMs?
(a) BPE ensures that most common words in vocab are represented as a single token, while rare words are broken down into two or more subword tokens
(b) Let us take this example!
* {"old": 7, "older": 3, "finest": 9, "lowest": 4}
* Preprocessing: Need to add end token "</w>" at the end of each word.
{"old</w>": 7, "older</w>": 3, ...}

* Now split words into characters and count their frequency

#	token	freq
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

→ NEXT PAGE

Next step! BPE algorithm is to look for most frequent pairing
(e) ↳ Merge them and perform the same iteration again and again until we reach the token limit or iteration limit

[Iteration 1]: Start w/ second most common token 'e'; most common byte pair starting w/ "e" is "es".

In table, subtract the 'e' count and the 's' count by 13, Add 'es' to token list and assign it 13 frequency. 'es'->13, 'e'->3, 's'->0

[Iteration 2]: Merge tokens "es" and "t" as they have appeared 13 times in our dataset

In table, subtract the 'es' and 't' count by 13, Add 'est' to token list and assign it 13 frequency. 'est'->13, 'es'->0, 't'->0

[Iteration 3]
(8) We see that "est</w>" has appeared 13 times,
so in table, subtract the 'est' and '</w>' by 13, Add 'est</w>' to token list and assign it 13 frequency. 'est'->0, '</w>'=10, 'est</w>'=13
↳ Helps also understand difference between (est)imate and highEST

[Iteration 4] "o" and "l" has appeared 10 times
so in table, subtract the 'o' and 'l' by 10, Add 'ol' to token list and assign it 10 frequency, 'o'->4, 'l'->4, 'ol'->10

[Iteration 5]! 'ol' and 'd' has appeared 10 times
so in table, subtract 'ol' and 'd' by 10...

"Does not make sense to merge 'f' 'i' 'n' to 'fin' because does not appear much, we merged together b/c 'old' and 'est' were present in lots of words.

Final table:

#	token	freq
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

List of 11 tokens will serve as our vocabulary.

Stopping criteria can either be token count, or # of iterations.