Tokenizers

The first step in the NLP pipeline is converting sequences of text into sequences of tokens.

This process is called *Tokenization*.

Fortunately, many toolkits provide implementations of Tokenizers so that you don't have to do this yourself.

Our concern is to discuss the concepts and limitations.

We refer the interested read to some good *practical* resources on Tokenization

- Hugging Face course section on Tokenization (https://huggingface.co/course/chapter2/4?fw=pt)
- TensorFlow `tensorflow_text` API (https://www.tensorflow.org/text)
- Keras Project NLP API (https://keras.io/api/keras_nlp/tokenizers/)
    - Note: this is the Keras project **not** the implementation of Keras in TensorFlow

Word based tokenization

There are several varieties of Tokenization.

The first is Word Based

- convert words into tokens

It would be naive to imagine that each word is a token

- The size of the vocabulary $\mathbf{V}$ can easily be tens of thousands.
- Not only are there many words, there are *variations* of each word
    - verbs: past, present and future tense
    - nouns:
        - singular, plural
        - gender agreement with subject

"Classical" NLP has developed techniques to eliminate word variation by conversion to "canonical" form

- stemming: convert word to its root form (root may not be in dictionary)
    - drive, driver, driving ↦ driv
- lemmatization: convert word to a root that is in dictionary
    - him: ↦ he
    - took ↦ take
    - earlier ↦ early

So one way to reduce the vocabulary size is by using these techniques.

There is still a place for this techniques in Deep Learning.

- spaCy (https://spacy.io/) is a very popular toolkit for dealing with text.

It is also possible that Embeddings will create a small number of "dimension" (e.g., variations) that are common to many words

- the "plural" dimension

Issues with Word based tokenization

Most word based tokenizers assumed that words are delimited by the space and a fixed number of punctuation characters.

A big unresolved issue with word based tokenization:

- No matter how large the vocabulary $\mathbf{V}$
- There will still be words that are Out of Vocabulary (OOV)

One way to address this is with Character based tokenization

- each character is a token

But this may inhibit learning

- Characters are too low level
- Compared to words

Sub-word Tokenization

An elegant solution to the OOV issue is *sub-word tokenization*

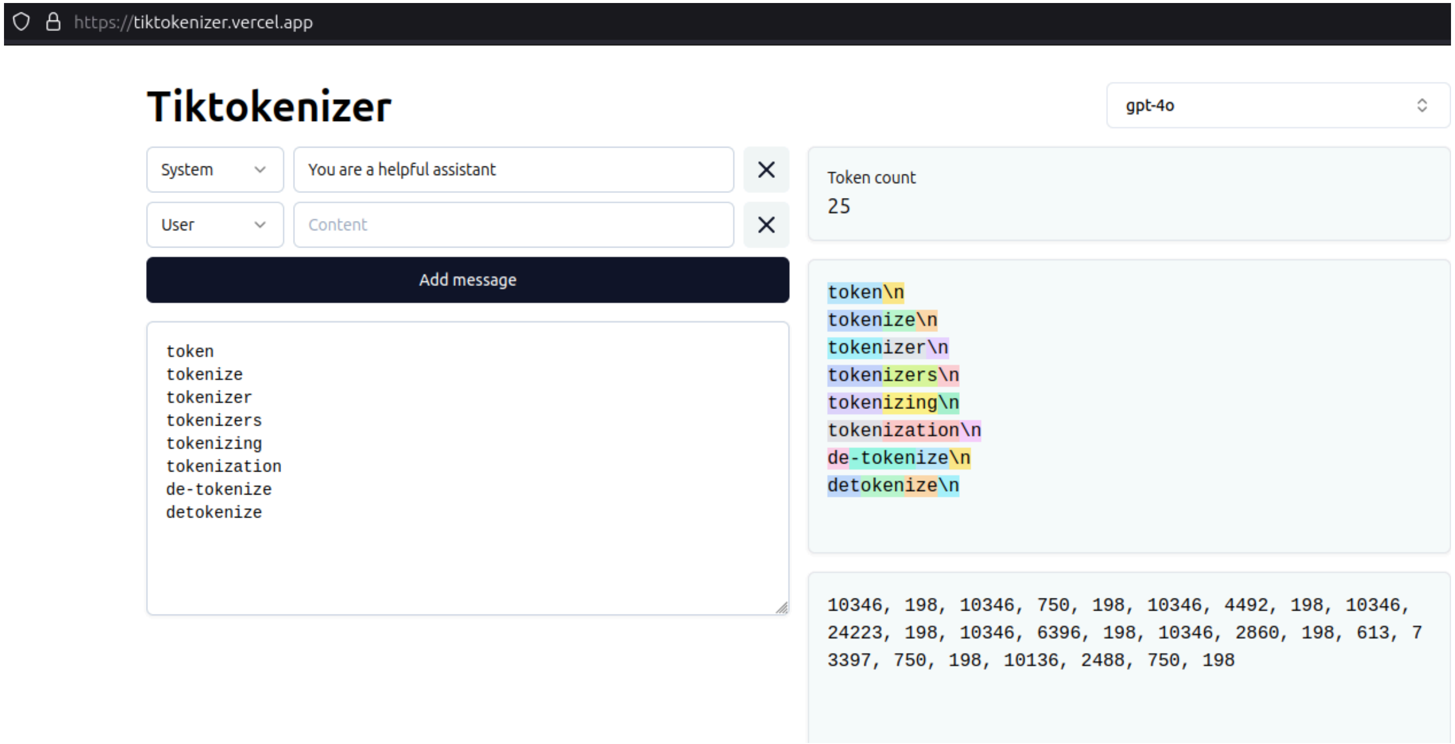
- break an OOV token into pieces, each of which is in-vocabulary
  - here's $\mapsto$ here, \', s
  - tokenizer $\mapsto$ token, ##izer

Sub-word tokenization has become fairly common in modern NLP systems

- Balances brevity of character encoding
- With expressiveness of word encoding

Here is an illustration of how variants of a word are tokenized.

Sub-word tokenization



Tiktokenizer

gpt-4o

System
You are a helpful assistant

User
Content

```
token
tokenize
tokenizer
tokenizers
tokenizing
tokenization
de-tokenize
detokenize
```

Token count
25

```
token\n
tokenize\n
tokenizer\n
tokenizers\n
tokenizing\n
tokenization\n
de-tokenize\n
detokenize\n
```

```
10346, 198, 10346, 750, 198, 10346, 4492, 198, 10346,
24223, 198, 10346, 6396, 198, 10346, 2860, 198, 613, 7
3397, 750, 198, 10136, 2488, 750, 198
```

Attribution: https://tiktokenizer.vercel.app/

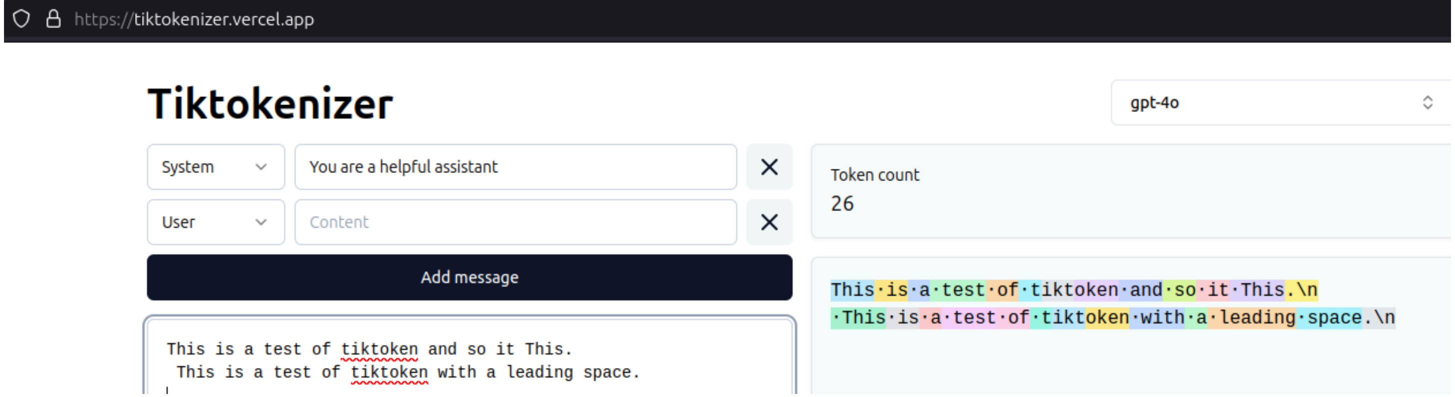
Some subtleties to observe:

- the tokenization of the *same word*
- is different when the word appears
    - at the start of the sentence
    - rather than the middle

See the token for the word "This" that appears at the start and end of the first sentence

- most tokens begin with a "blank": middle-of-sentence word

Sub-word tokenization: word at start vs middle of sentence



gpt-4o

System

You are a helpful assistant

User

```
This is a test of tiktoken and so it This.
 This is a test of tiktoken with a leading space.
```

Token count
26

```
This·is·a·test·of·tiktoken·and·so·it·This.\n
·This·is·a·test·of·tiktoken·with·a·leading·space.\n
```

Large Language Model's can solve complex problems yet fail on grade-school arithmetic.

If my LLM is so smart, why can't it add small integers ?

The answer is that it is not "seeing" (the tokenization) what the human is seeing.

Note

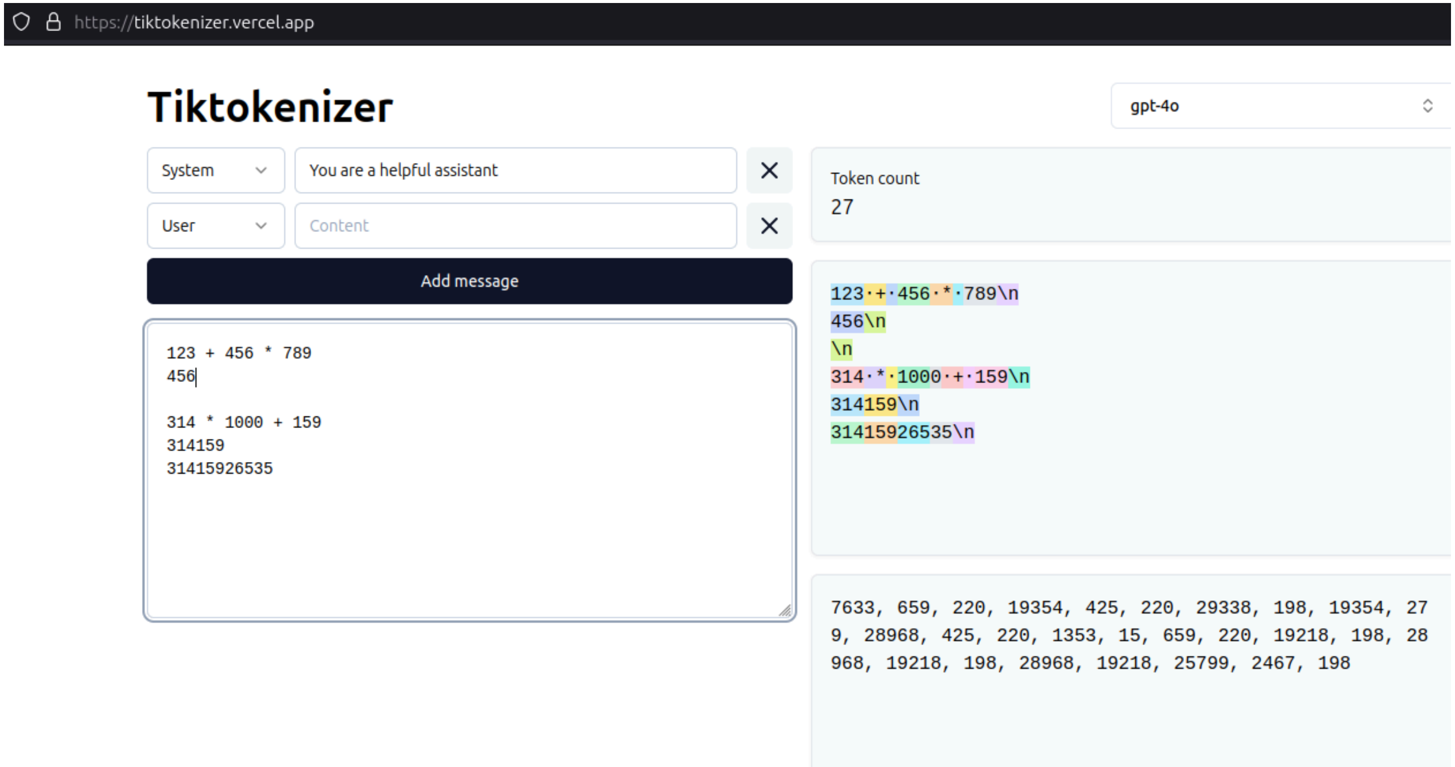
Most LLM's circumvent this issue my using "tools" to perform math

- typically: generates Python code and runs it !

Notice that

- three digit sequences are unique tokens
- but anything longer than 3 digits spans multiple tokens

Sub-word tokenization: Math



```
123 + 456 * 789
456

314 * 1000 + 159
314159
31415926535
```

Token count
27

```
123·+·456·*·789\n
456\n
\n
314·*·1000·+·159\n
314159\n
31415926535\n
```

```
7633, 659, 220, 19354, 425, 220, 29338, 198, 19354, 27
9, 28968, 425, 220, 1353, 15, 659, 220, 19218, 198, 28
968, 19218, 198, 28968, 19218, 25799, 2467, 198
```

Attribution: https://tiktokenizer.vercel.app/

Sub-word tokenization: Byte Pair Encoding (BPE)

There are several methods to achieve sub-word Tokenization.

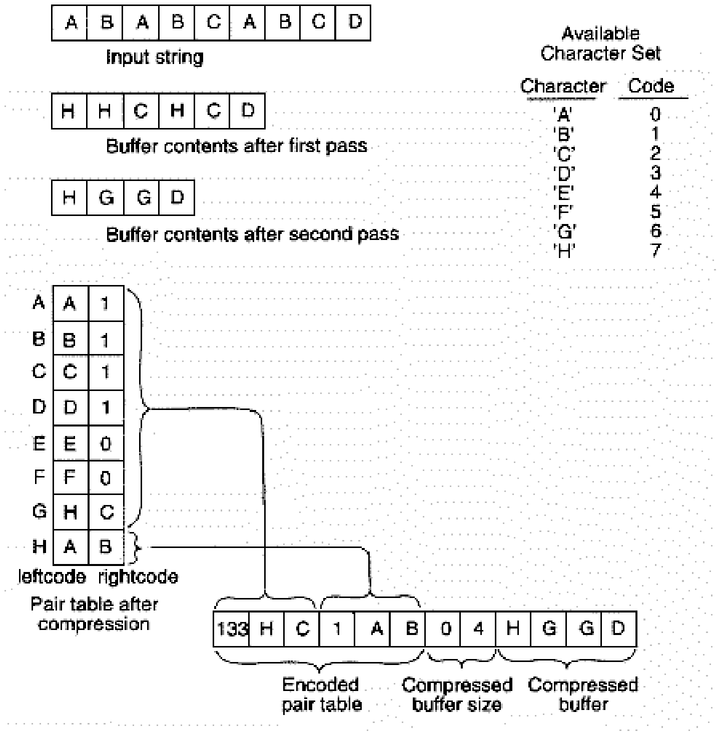
Some of the objectives are

- to **not** split frequently occurring words into pieces
    - So that the model consuming the tokens has access to meaningful single word pieces
- to recognize pieces that are common prefix and suffix forms
    - So that the model consuming the tokens can derive the commonality indicated by the suffix
        - e.g. "izer"

A simple version is derived from *Byte Pair Encoding* (BPE) (https://arxiv.org/pdf/1508.07909.pdf)

- BPE is a way to replace common sequences of bytes with a short symbol
- Word segmentation: the analog of BPE, where we replace common sequences of tokens with a short symbol

Universal model: adapting task-specific inputs



Picture from: http://twimgs.com/ddj/cuj/images/cuj9402gage/fig1.gif"

- Pass 1: AB ↦ H
  - byte sequence "A", "B" occurs 3 times in input
  - replaced with symbol "H" (recorded dictionary entry "H"_
- Pass 2: HC ↦ G
  - symbol sequence "H", "C" (occurring twice) replaced with symbol "G"

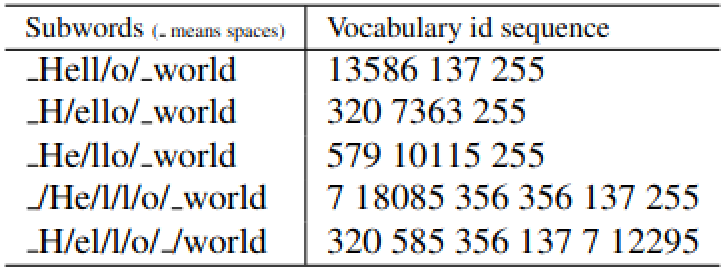
WordPiece sub-word tokenization

WordPiece (https://arxiv.org/pdf/1609.08144.pdf) is a variant of BPE.

Sentence Piece sub-word tokenization

BPE provides a unique encoding (*segmentation*) of a sentence of text.

But there are multiple ways of dividing words into segments.



Subwords (_ means spaces)	Vocabulary id sequence
_Hell/o/_world	13586 137 255
_H/ello/_world	320 7363 255
_He/llo/_world	579 10115 255
_/He/l/l/o/_world	7 18085 356 356 137 255
_H/el/l/o/_/world	320 585 356 137 7 12295

Table 1: Multiple subword sequences encoding the same sentence “Hello World”

Attribution: Table 1 in https://arxiv.org/pdf/1804.10959.pdf

The paper Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates (https://arxiv.org/pdf/1804.10959.pdf) proposed

- using multiple segmentations
- associating a probability with each

The advantages include

- accommodation of multiple languages
- increased robustness of training

SentencePiece (https://arxiv.org/pdf/1808.06226.pdf) is an Open Source implementation of the subword regularization technique.

In [1]:

```
print("Done")
```

```
Done
```