TokenizationExamples

September 3, 2024

1 Tokenization

In this exercise, we will see examples of two types of tokenization: - **Rule based** tokenization: this is done using a combination of code and regular expressions that encode tokenization rules developed by experts. Here we use the `SpaCy` tokenizer as an example. - **BPE** tokenization: this is done to using statistics over a large corpus of text in order to determine how strings should be segmented into *subword* tokens. Here we use the `tiktoken` tokenizer from OpenAI.

1.1 SpaCy tokenization and other examples

[1]:
```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("MS is looking at buying U.K. startup for $11.1 million.")
for token in doc:
  print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
        token.shape_, token.is_alpha, token.is_stop)
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R)
SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel
Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX)
instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R)
SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel
Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX)
instructions.
MS MS PROPN NNP nsubj XX True False
is be AUX VBZ aux xx True True
looking look VERB VBG ROOT xxxx True False
at at ADP IN prep xx True True
buying buy VERB VBG pcomp xxxx True False
U.K. U.K. PROPN NNP dobj X.X. False False
startup startup NOUN NN dep xxxx True False
for for ADP IN prep xxx True True
$ $ SYM $ quantmod $ False False
11.1 11.1 NUM CD compound dd.d False False
million million NUM CD pobj xxxx True False
. . PUNCT . punct . False False
```

1.1.1 Display syntactic dependencies

[2]:
```
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Microsoft plans to buy Activision for $69 billion.")
displacy.render(doc, style = "dep")
```

```
<IPython.core.display.HTML object>
```

[3]:
```
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Microsoft plans to buy Activision for $69 billion.")
displacy.render(doc, style = "ent")
```

```
<IPython.core.display.HTML object>
```

1.1.2 Use directly the tokenizer component

[4]:
```
from spacy.lang.en import English

nlp = English()
tokenizer = nlp.tokenizer
tokens = tokenizer("U.S. economy is healing, but there's a long way to go. "
                   "The spread of Covid-19 led to surge in orders for factory␣
 ↪robots."
                  "Fine-tuning models is time-consuming.")

for token in tokens:
  print(token.text, end = ' ')
print()
```

```
U.S. economy is healing , but there 's a long way to go . The spread of Covid-19
led to surge in orders for factory robots . Fine - tuning models is time -
consuming .
```

[5]:
```
from spacy.lang.en import English

nlp = English()
tokenizer = nlp.tokenizer
tokens = tokenizer("I think what she said is soooo craaaazy!")
for token in tokens:
  print(token.text, end = ' ')
print()
```

```
I think what she said is soooo craaaazy !
```

1.1.3 Use only the tokenizer component by disabling other pipeline modules

[6]:
```
import spacy
nlp = spacy.load("en_core_web_sm", exclude = ['tagger, ner, parser'])

doc = nlp("U.S. economy is healing, but there's a long way to go. "
          "The spread of Covid-19 led to surge in orders for factory robots.")

for token in doc:
  print(token.text, end = ' ')
print()
print()

for sent in doc.sents:
    for token in sent:
        print(token.text, end = ' ')
    print()
```

```
U.S. economy is healing , but there 's a long way to go . The spread of Covid-19
led to surge in orders for factory robots .

U.S. economy is healing , but there 's a long way to go .
The spread of Covid-19 led to surge in orders for factory robots .
```

1.1.4 Use a special sentencizer that does not require syntactic parsing, for efficiency

[7]:
```
from spacy.lang.en import English

nlp = English()
nlp.add_pipe("sentencizer")

doc = nlp("U.S. economy is healing, but there's a long way to go. "
          "The spread of Covid-19 led to surge in orders for factory robots.")

# Print tokens, one sentence per line.
for sent in doc.sents:
  for token in sent:
    print (token, end = ' ')
  print()
```

```
U.S. economy is healing , but there 's a long way to go .
The spread of Covid-19 led to surge in orders for factory robots .
```

1.1.5 By default, it seems spaCy creates tokens for newline characters

[8]:
```
nlp = spacy.load("en_core_web_sm")

stanza = "I am a contract-drafting em,\n" \
```

```
  "The loyalest of lawyers!\n" \
  "I draw up terms for deals 'twixt firms\n" \
  "To service my employers!"
print(stanza, '\n')

doc = nlp(stanza)

print(f"Stanza has {len(list(doc.sents))} sentences.\n")

# Print tokens, one sentence per line.
for sent in doc.sents:
    for token in sent:
        if token.text == '\n':
            print('<NL>', end = ' ')
        else:
            print(token, end = ' ')
    print()
```

```
I am a contract-drafting em,
The loyalest of lawyers!
I draw up terms for deals 'twixt firms
To service my employers!

Stanza has 2 sentences.

I am a contract - drafting em , <NL> The loyalest of lawyers ! <NL>
I draw up terms for deals ' twixt firms <NL> To service my employers !
```

1.1.6 Replace newlines with white spaces

[9]:
```
stanza = "I am a contract-drafting em, " \
  "The loyalest of lawyers! " \
  "I draw up terms for deals 'twixt firms " \
  "To service my employers!"
print(stanza)
print()

doc = nlp(stanza)

print(f"Stanza has {len(list(doc.sents))} sentences.\n")

# Print tokens, one sentence per line.
for sent in doc.sents:
  for token in sent:
    print (token, end = ' ')
  print()
```

```
#for token in doc:
#  print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
#        token.shape_, token.is_alpha, token.is_stop)
```

```
I am a contract-drafting em, The loyalest of lawyers! I draw up terms for deals
'twixt firms To service my employers!

Stanza has 2 sentences.

I am a contract - drafting em , The loyalest of lawyers !
I draw up terms for deals ' twixt firms To service my employers !
```

1.2 BPE tokenization using `tiktoken`

[10]:
```
# Uncomment this line if tiktoken is not yet installed on your machine.
!pip install tiktoken

import tiktoken

# To get the tokeniser corresponding to a specific model in the OpenAI API:
enc = tiktoken.encoding_for_model("gpt-4")

# The .encode() method converts a text string into a list of token integers.
enc.encode("Predeal is soooo data sciency this Summer!")
```

```
Requirement already satisfied: tiktoken in
/Users/rbunescu/anaconda3/lib/python3.10/site-packages (0.4.0)
Requirement already satisfied: requests>=2.26.0 in
/Users/rbunescu/anaconda3/lib/python3.10/site-packages (from tiktoken) (2.32.3)
Collecting regex>=2022.1.18
  Downloading regex-2024.7.24-cp310-cp310-macosx_10_9_x86_64.whl (282 kB)
                           282.1/282.1
kB 3.7 MB/s eta 0:00:00a 0:00:01
Requirement already satisfied: certifi>=2017.4.17 in
/Users/rbunescu/anaconda3/lib/python3.10/site-packages (from
requests>=2.26.0->tiktoken) (2024.7.4)
Requirement already satisfied: idna<4,>=2.5 in
/Users/rbunescu/anaconda3/lib/python3.10/site-packages (from
requests>=2.26.0->tiktoken) (3.7)
Requirement already satisfied: charset-normalizer<4,>=2 in
/Users/rbunescu/anaconda3/lib/python3.10/site-packages (from
requests>=2.26.0->tiktoken) (3.3.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/Users/rbunescu/anaconda3/lib/python3.10/site-packages (from
requests>=2.26.0->tiktoken) (1.26.19)
Installing collected packages: regex
Successfully installed regex-2024.7.24
```

[10]:
```
[4808, 30739, 374, 779, 39721, 828, 39074, 2301, 420, 19367, 0]
```

[11]:
```
# The .decode() method converts a list of token integers to a string.
enc.decode([4808, 30739, 374, 779, 39721, 828, 39074, 2301, 420, 19367, 0])
```

[11]:
```
'Predeal is soooo data sciency this Summer!'
```

[12]:
```
# The .decode_single_token_bytes() method safely converts a single integer␣
 ↪token to the bytes it represents.
tokens = [enc.decode_single_token_bytes(token) for token in [4808, 30739, 374,␣
 ↪779, 39721, 828, 39074, 2301, 420, 19367, 0]]
print(tokens)
```

```
[b'Pre', b'deal', b' is', b' so', b'ooo', b' data', b' sci', b'ency', b' this',
b' Summer', b'!']
```

[13]:
```
# We usually combine .encode() with .decode_single_token_bytes() into one list␣
 ↪comprehension
#    to get the list of tokens as byte strings.
tokens = [enc.decode_single_token_bytes(token) for token in enc.encode("Predeal␣
 ↪is soooo data sciency this Summer!")]

# Note the 'b' in front of each string, which means that the string you see is␣
 ↪a sequence of bytes.
print(tokens)

# To translate to the standard representation (utf-8), you can use token.
 ↪decode('utf-8').
utf8_tokens = [token.decode('utf-8') for token in tokens]
print(utf8_tokens)
```

```
[b'Pre', b'deal', b' is', b' so', b'ooo', b' data', b' sci', b'ency', b' this',
b' Summer', b'!']
['Pre', 'deal', ' is', ' so', 'ooo', ' data', ' sci', 'ency', ' this', '
Summer', '!']
```

[14]:
```
# Let's see how tiktoken deals with different types of white space.
tokens = [enc.decode_single_token_bytes(token) for token in enc.encode("His \t␣
 ↪\n \n\t  ambivalence was perplexing.")]
#tokens = enc.decode_single_token_bytes(enc.encode("His ambivalence was␣
 ↪perplexing."))

print(tokens)
```

```
[b'His', b' \t', b' \n \n', b'\t ', b' amb', b'ivalence', b' was', b' perplex',
b'ing', b'.']
```

[15]:
```
tokens = [enc.decode_single_token_bytes(token) for token in enc.encode("I think␣
 ↪what she said is soooo craaaazy!")]
print(tokens)

tokens[1].strip().decode('utf-8')
```

```
[b'I', b' think', b' what', b' she', b' said', b' is', b' so', b'ooo', b' cra',
b'aa', b'azy', b'!']
```

[15]:
```
'think'
```

[16]:
```
# Another example showing subword tokens.
tokens = [enc.decode_single_token_bytes(token) for token in enc.encode("The␣
 ↪perplexing cat sat on the mat.")]
print(tokens)
```

```
[b'The', b' perplex', b'ing', b' cat', b' sat', b' on', b' the', b' mat', b'.']
```

[17]:
```
# Let's decode from the byte string representation to utf-8.
utf8_tokens = [token.decode('utf-8') for token in tokens]
print(utf8_tokens)
```

```
['The', ' perplex', 'ing', ' cat', ' sat', ' on', ' the', ' mat', '.']
```

[ ]:
```

```