Introduction to the tokenization

Thomas Gerald

January 16, 2026

Ressources (for exercices and lecture slides)

https://thomas-gerald.fr/TMC

Using subword units

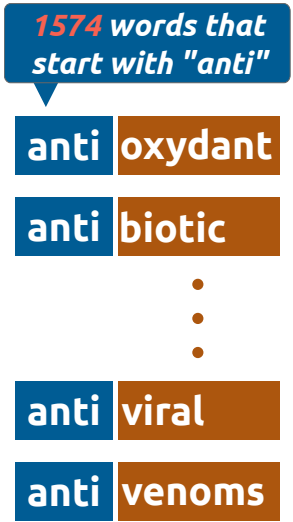
Tokenization: Splitting into subwords units

- Splitting words into subwords
- Subwords with ”semantic” meaning
- Suffix/prefix split

How to ?

- Use a database (dictionary)
- Do it automatically ? $\rightarrow$ Statistical tokenization

Tokenization: Splitting into subwords units



Automatic tokenizer ?

Consider a text corpus

- common words are frequent
- common subwords are frequent

→ Algorithms based on the frequency of subwords ?

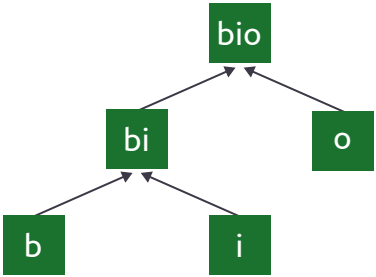
A naive algorithm:

1. Splitting words in all possible subwords
2. Ordering subwords by frequency
3. Add to the vocabulary the top-k most frequent subwords

Tokenization: BPE Algorithm

The Byte Pair Encoding Algorithm

Learn merging rules from characters to subwords



→ “A new algorithm for data compression”, P. Gage, C user Journal, 1994

→ “Neural Machine Translation of Rare Words with Subword Units”, R. Seenrich et al., ACL 2016

Neural Machine Translation of Rare Words with Subword Units

Rico Sennrich and **Barry Haddow** and **Alexandra Birch**
School of Informatics, University of Edinburgh
{rico.sennrich,a.birch}@ed.ac.uk,bhaddow@inf.ed.ac.uk

Abstract

Neural machine translation (NMT) models typically operate with a fixed vocabulary, but translation is an open-vocabulary problem. Previous work addresses the translation of out-of-vocabulary words by backing off to a dictionary. In this paper, we introduce a simpler and more effective approach, making the NMT model capable of open-vocabulary translation by encoding rare and unknown words as sequences of subword units. This is based on the intuition that various word classes are translatable via smaller units than words, for instance names (via character copying or transliteration), compounds (via com-

lem, and especially for languages with productive word formation processes such as agglutination and compounding, translation models require mechanisms that go below the word level. As an example, consider compounds such as the German *Abwasser|behandlungs|anlange* ‘sewage water treatment plant’, for which a segmented, variable-length representation is intuitively more appealing than encoding the word as a fixed-length vector.

For word-level NMT models, the translation of out-of-vocabulary words has been addressed through a back-off to a dictionary look-up (Jean et al., 2015; Luong et al., 2015b). We note that such techniques make assumptions that often do not hold true in practice. For instance, there is not al-

Tokenization: BPE Algorithm

Algorithm 1 BPE Tokenization

1: **Input:** Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
2: **Output:** Tokenized word $\mathcal{W}$

Tokenization: BPE Algorithm

Algorithm 1 BPE Tokenization

1: **Input:** Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
2: **Output:** Tokenized word $\mathcal{W}$
3: $\mathcal{W} \leftarrow$ split w into symbols $\in \mathcal{V}$

Tokenization: BPE Algorithm

Algorithm 1 BPE Tokenization

1: **Input:** Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
2: **Output:** Tokenized word $\mathcal{W}$
3: $\mathcal{W} \leftarrow$ split w into symbols $\in \mathcal{V}$
4: **for** $(p_1, p_2, v) \in \mathcal{M}$ **do**

Tokenization: BPE Algorithm

Algorithm 1 BPE Tokenization

Input: Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
Output: Tokenized word $\mathcal{W}$
$\mathcal{W} \leftarrow$ split w into symbols $\in \mathcal{V}$
for $(p_1, p_2, v) \in \mathcal{M}$ **do**
    $i \leftarrow 0$

Tokenization: BPE Algorithm

Algorithm 1 BPE Tokenization

Input: Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
Output: Tokenized word $\mathcal{W}$
$\mathcal{W} \leftarrow$ split w into symbols $\in \mathcal{V}$
for $(p_1, p_2, v) \in \mathcal{M}$ **do**
    $i \leftarrow 0$
    while $i < len(W)$ **do**

Tokenization: BPE Algorithm

Algorithm 1 BPE Tokenization

Input: Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
Output: Tokenized word $\mathcal{W}$
$\mathcal{W} \leftarrow$ split w into symbols $\in \mathcal{V}$
for $(p_1, p_2, v) \in \mathcal{M}$ **do**
    $i \leftarrow 0$
    while $i < len(W)$ **do**
        if $W_i = p_1 \wedge W_{i+1} = p_2$ **then**

Tokenization: BPE Algorithm

Algorithm 1 BPE Tokenization

Input: Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
Output: Tokenized word $\mathcal{W}$
$\mathcal{W} \leftarrow$ split w into symbols $\in \mathcal{V}$
for $(p_1, p_2, v) \in \mathcal{M}$ **do**
    $i \leftarrow 0$
    while $i < len(W)$ **do**
        if $W_i = p_1 \wedge W_{i+1} = p_2$ **then**
            $W_i \leftarrow v$
            $W_{i+1:len(W)} \leftarrow W_{i+2:len(W)}$

Tokenization: BPE Algorithm

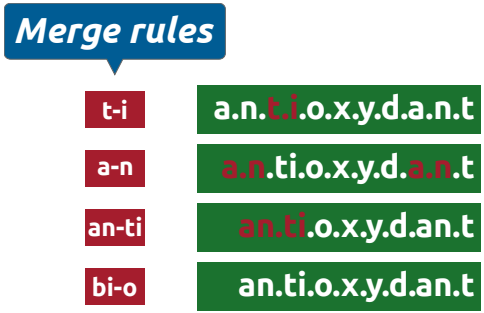
Algorithm 1 BPE Tokenization

Input: Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
Output: Tokenized word $\mathcal{W}$
$\mathcal{W} \leftarrow$ split w into symbols $\in \mathcal{V}$
for $(p_1, p_2, v) \in \mathcal{M}$ **do**
    $i \leftarrow 0$
    while $i < len(W)$ **do**
        if $W_i = p_1 \wedge W_{i+1} = p_2$ **then**
            $W_i \leftarrow v$
            $W_{i+1:len(W)} \leftarrow W_{i+2:len(W)}$
        else
            $i \leftarrow i + 1$
        end if
    end while
end for

Tokenization: BPE Algorithm

Algorithm 1 BPE Tokenization

Input: Word w; Vocabulary $\mathcal{V}$; Merge rules $\mathcal{M}$
Output: Tokenized word $\mathcal{W}$
$\mathcal{W} \leftarrow$ split w into symbols $\in \mathcal{V}$
for $(p_1, p_2, v) \in \mathcal{M}$ **do**
    $i \leftarrow 0$
    while $i < len(W)$ **do**
        if $W_i = p_1 \wedge W_{i+1} = p_2$ **then**
            $W_i \leftarrow v$
            $W_{i+1:len(W)} \leftarrow W_{i+2:len(W)}$
        else
            $i \leftarrow i + 1$
        end if
    end while
end for
return $\mathcal{W}$



Tokenization: BPE Algorithm

Vocabulary $\rightarrow A, B \dots Z, a, b \dots z$

Find the most frequent couple of tokens

Words in the text

W		Occ
a.n.t.i.v.i.r.a.l	×	2
a.n.t.i.o.x.i.d.a.n.t	×	1
a.n.t.i.b.i.o.t.i.c	×	4
b.i.o.l.o.g.i.c	×	6
a.n	×	3

Number of couple occurences

Tuple		Occ
t.i	$2+1+8+0+0$	11
i.o	$0+1+4+6+0$	11
a.n	$2+1+4+0+3$	10
b.i	$0+0+4+6+0$	10
i.c	$0+0+4+6+0$	10
n.t	$2+2+4+0+0$	8
⋮	⋮	⋮

Tokenization: BPE Algorithm

Vocabulary $\rightarrow A, B \dots Z, a, b \dots z, ti$

Find the most frequent couple of tokens

Words in the text

W		Occ
a.n.ti.v.i.r.a.l	×	2
a.n.ti.o.x.i.d.a.n.t	×	1
a.n.ti.b.i.o.ti.c	×	4
b.i.o.l.o.g.i.c	×	6
a.n	×	3

Number of couple occurences

Tuple		Occ
a.n	$2+1+4+0+3$	10
b.i	$0+0+4+6+0$	10
i.o	$0+0+4+6+0$	10
n.ti	$2+1+4+0+0$	7
i.c	$0+0+0+6+0$	6
⋮	⋮	⋮

Tokenization: BPE Algorithm

Vocabulary $\rightarrow A, B \dots Z, a, b \dots z, ti, an$

Find the most frequent couple of tokens

Words in the text

W		Occ
an.ti.v.i.r.a.l	×	2
an.ti.o.x.i.d.an.t	×	1
an.ti.b.i.o.ti.c	×	4
b.i.o.l.o.g.i.c	×	6
an	×	3

Number of couple occurences

Tuple		Occ
b.i	$0+0+4+6+0$	10
i.o	$0+0+4+6+0$	10
an.ti	$2+2+4+0+0$	7
texttti.c	$0+0+0+6+0$	6
⋮	⋮	⋮

Vocabulary $\rightarrow A, B \dots Z, a, b \dots z, ti, an, bi$

Find the most frequent couple of tokens

Words in the text

W		Occ
an.ti.v.i.r.a.l	×	2
an.ti.o.x.i.d.an.t	×	1
an.ti.bi.o.ti.c	×	4
bi.o.l.o.g.i.c	×	6
an	×	3

Number of couple occurences

Tuple		Occ
bi.o	$0+0+4+6+0$	10
an.ti	$2+2+4+0+0$	7
i.c	$0+0+0+6+0$	6
⋮	⋮	⋮

Tokenization: BPE Algorithm

Vocabulary $\rightarrow A, B \dots Z, a, b \dots z, ti, an, bi, bio$

Find the most frequent couple of tokens

Words in the text

W		Occ
an.ti.v.i.r.a.l	×	2
an.ti.o.x.i.d.an.t	×	1
an.ti.bio.ti.c	×	4
bio.l.o.g.i.c	×	6
an	×	3

Number of couple occurences

Tuple		Occ
an.ti	$2+2+4+0+0$	7
i.c	$0+0+0+6+0$	6
⋮	⋮	⋮

Tokenization: BPE Algorithm

Vocabulary $\rightarrow A, B \dots Z, a, b \dots z, ti, an, bi, bio, anti$

Find the most frequent couple of tokens

Words in the text

W		Occ
an.ti.v.i.r.a.l	×	2
an.ti.o.x.i.d.an.t	×	1
an.ti.bio.ti.c	×	4
bio.l.o.g.i.c	×	6
an	×	3

Number of couple occurences

Tuple		Occ
an.ti	$2+2+4+0+0$	7
i.c	$0+0+0+6+0$	6
⋮	⋮	⋮

Tokenization: BPE Algorithm

Vocabulary $\rightarrow A, B \dots Z, a, b \dots z$

Find the most frequent couple of tokens

Words in the text

W		Occ
anti.v.i.r.a.l	×	2
anti.o.x.i.d.an.t	×	1
anti.bio.ti.c	×	4
bio.l.o.g.i.c	×	6
an	×	3

Etc....

Number of couple occurences

Tuple		Occ
i.c	$0+0+0+6+0$	6
⋮	⋮	⋮

Tokenization: BPE Algorithm

Principle:

Initialize vocabulary with characters

1. Tokenize the text (with current rules)
2. Finding tokens that appears the most consecutively
3. Add a merge rule between the two tokens
4. Repeat the procedure

Algorithm 2 Training BPE (big picture)

1: **Input:** Corpus C; Vocabulary $\mathcal{V}$; size S
2: **Output:** $\mathcal{V}, \mathcal{M}$

Tokenization: BPE Algorithm

Principle:

Initialize vocabulary with characters

1. Tokenize the text (with current rules)
2. Finding tokens that appears the most consecutively
3. Add a merge rule between the two tokens
4. Repeat the procedure

Algorithm 2 Training BPE (big picture)

1: **Input:** Corpus C; Vocabulary $\mathcal{V}$; size S
2: **Output:** $\mathcal{V}, \mathcal{M}$
3: $W \leftarrow tokenize(C)$
4: $\mathcal{M} \leftarrow \emptyset$

Tokenization: BPE Algorithm

Principle:

Initialize vocabulary with characters

1. Tokenize the text (with current rules)
2. Finding tokens that appears the most consecutively
3. Add a merge rule between the two tokens
4. Repeat the procedure

Algorithm 2 Training BPE (big picture)

1: **Input:** Corpus C; Vocabulary $\mathcal{V}$; size S
2: **Output:** $\mathcal{V}, \mathcal{M}$
3: $W \leftarrow \textit{tokenize}(C)$
4: $\mathcal{M} \leftarrow \emptyset$
5: **while** $\mathcal{V} < S$ **do**

Tokenization: BPE Algorithm

Principle:

Initialize vocabulary with characters

1. Tokenize the text (with current rules)
2. Finding tokens that appears the most consecutively
3. Add a merge rule between the two tokens
4. Repeat the procedure

Algorithm 2 Training BPE (big picture)

Input: Corpus C; Vocabulary $\mathcal{V}$; size S
Output: $\mathcal{V}, \mathcal{M}$
$W \leftarrow tokenize(C)$
$\mathcal{M} \leftarrow \emptyset$
while $\mathcal{V} < S$ **do**
    $F \leftarrow compute\_pairs\_freq(W)$
    $P \leftarrow best\_pairs(F)$

Tokenization: BPE Algorithm

Principle:

Initialize vocabulary with characters

1. Tokenize the text (with current rules)
2. Finding tokens that appears the most consecutively
3. Add a merge rule between the two tokens
4. Repeat the procedure

Algorithm 2 Training BPE (big picture)

Input: Corpus C; Vocabulary $\mathcal{V}$; size S
Output: $\mathcal{V}, \mathcal{M}$
$W \leftarrow tokenize(C)$
$\mathcal{M} \leftarrow \emptyset$
while $\mathcal{V} < S$ **do**
    $F \leftarrow compute\_pairs\_freq(W)$
    $P \leftarrow best\_pairs(F)$
    $\mathcal{V} \leftarrow V \cup (P_0.P_1)$
    $m \leftarrow (P_0, P_1, \{j | V_j = P_0.P_1\})$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{m\}$

Tokenization: BPE Algorithm

Principle:

Initialize vocabulary with characters

1. Tokenize the text (with current rules)
2. Finding tokens that appears the most consecutively
3. Add a merge rule between the two tokens
4. Repeat the procedure

Algorithm 2 Training BPE (big picture)

Input: Corpus C; Vocabulary $\mathcal{V}$; size S
Output: $\mathcal{V}, \mathcal{M}$
$W \leftarrow tokenize(C)$
$\mathcal{M} \leftarrow \emptyset$
while $\mathcal{V} < S$ **do**
    $F \leftarrow compute\_pairs\_freq(W)$
    $P \leftarrow best\_pairs(F)$
    $\mathcal{V} \leftarrow V \cup (P_0.P_1)$
    $m \leftarrow (P_0, P_1, \{j | V_j = P_0.P_1\})$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{m\}$
    $W \leftarrow merge\_pair(m, W)$
end while

Tokenization: BPE Algorithm

Principle:

Initialize vocabulary with characters

1. Tokenize the text (with current rules)
2. Finding tokens that appears the most consecutively
3. Add a merge rule between the two tokens
4. Repeat the procedure

Algorithm 2 Training BPE (big picture)

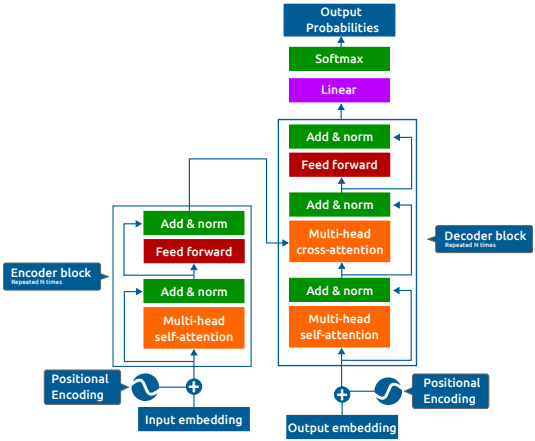
Input: Corpus C; Vocabulary $\mathcal{V}$; size S
Output: $\mathcal{V}, \mathcal{M}$
$W \leftarrow tokenize(C)$
$\mathcal{M} \leftarrow \emptyset$
while $\mathcal{V} < S$ **do**
    $F \leftarrow compute\_pairs\_freq(W)$
    $P \leftarrow best\_pairs(F)$
    $\mathcal{V} \leftarrow V \cup (P_0.P_1)$
    $m \leftarrow (P_0, P_1, \{j | V_j = P_0.P_1\})$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{m\}$
    $W \leftarrow merge\_pair(m, W)$
end while
return $\mathcal{V}, \mathcal{M}$

Tokenization: BPE Algorithm

→ Used in a lot of transformer models

Tokenization: BPE Algorithm

→ Used in a lot of transformer models



What are the models ?

- GPT, GPT2 (on byte level)
- BART
- RoBERTa
- . . .

Tokenization: BPE Algorithm

When to stop learning merging rules?

Tokenization: BPE Algorithm

When to stop learning merging rules?

- Threshold on the size of the vocabulary

Tokenization: BPE Algorithm

When to stop learning merging rules?

- Threshold on the size of the vocabulary
- Empirical size (depending on the languages/tasks/models)

Tokenization: BPE Algorithm

When to stop learning merging rules?

- Threshold on the size of the vocabulary
- Empirical size (depending on the languages/tasks/models)

$\rightarrow$ 32 000, 64 000 mostly (threshold between size and token expressivness)

Tokenization: Wordpiece Algorithm

Principle (similar to BPE)

- Intialize vocabulary with characters
- Compute for each token pairs t_1, t_2 $score(t_1, t_2) = \frac{p(t_1,t_2)}{p(t_1)\times p(t_2)}$
- Add a merge rule for couple of tokens with highest score

When to stop ?

- Threshold on the size of the vocabulary
- Threshold on the frequency

What are the models?

- BERT
- Most of transformer based on BERT

Tokenization: Unigrams

Unigram

- Different tokenization
- From large collection of subwords, reduce the vocabulary

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”

→ “Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates”, Taku Kudo, ACL 2018

Hypothesis : Training with different tokenization would help (regularization)

Unigram tokenization

Considering an input sentence X and $S(X)$ all the possible decomposition of X into subwords :

- The probability of the sequence $x = x_1, x_2, \ldots, x_m \in S(x)$ is given by $P(x) = \prod_{i=0}^{m} p(x_i)$
- $p(x_i)$ is the probability of a subword in the corpus

For the sequence X the segmentation is

$$x^* = \underset{x \in S(X)}{\arg\max}\, P(x)$$

$\rightarrow$ It is also possible to sample $x \sim S(x)$

Tokenization: Unigrams

Building vocabulary

1. Start from a large vocabulary
2. Compute how removing tokens affect the likelihood over the datasets
3. Remove tokens that affect (drop in likelihood) the most the model

Tokenization: Unigrams

Unigrams summary

- Different segmentation possible for an example
- Sampling most likely split

Do models use this appproach ?

- T5
- ...

⚠ Same sentence can have a different tokenization

Tokenization: Lossless Tokenization

Example :

Let consider the following vocabulary :

Voc	id
the	1
cat	2
is	3
sit	4
t	5
ing	6
on	7
couch	8

What is the tokenization of (considering BPE):
"The cat is sitting on the couch"

↓

the-cat-is-sit-t-ing-on-the-couch ($[1, 2, 3, 4, 5, 6, 7, 1, 8]$)

How to detokenize ?

↓

"thecatissittingonthecouch"

⚠ We loss some information ont the space

Detokenization ?

1. Add a special character for inner-word tokens
   → cannot encode multiple space
2. Add a special token for space

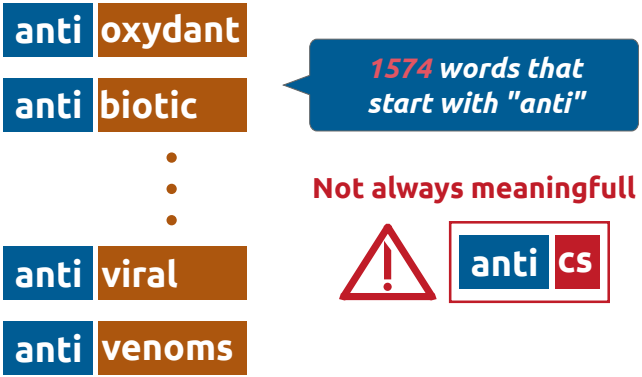
With **`wordpiece`** implementation

→ Using the token "*##token*" meaning it is an "inside word" token

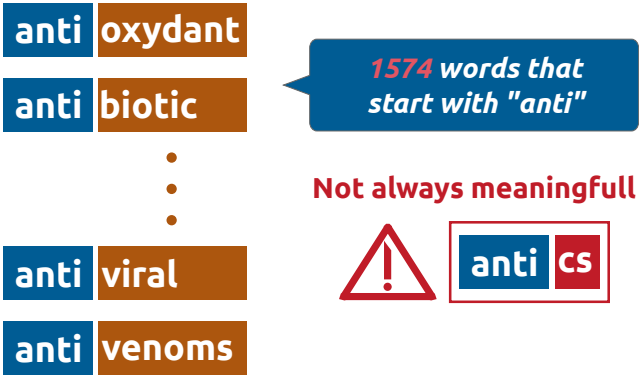
With **`sentencepiece`** implementation

→ Using the token "_" for space char

Tokenization: Some remaining issues

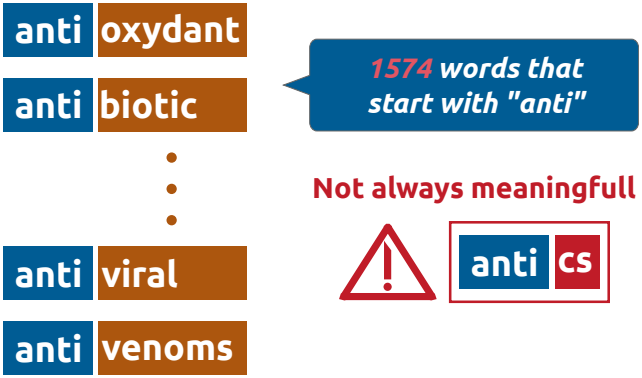


Tokenization: Some remaining issues



- Tokens are not always relevant

Tokenization: Some remaining issues



- Tokens are not always relevant
- Mostly depends on the size of the vocabulary or the training corpus

Tokenization: Conclusion

Conclusion:

- Principle and problems of tokenization
- Different approaches (BPE, Wordpiece, Unigram)

Tokenization: Conclusion

Conclusion:

- Principle and problems of tokenization
- Different approaches (BPE, Wordpiece, Unigram)

Exercise ?

- Implement a lemmatizer
- Implement the BPE algorithm