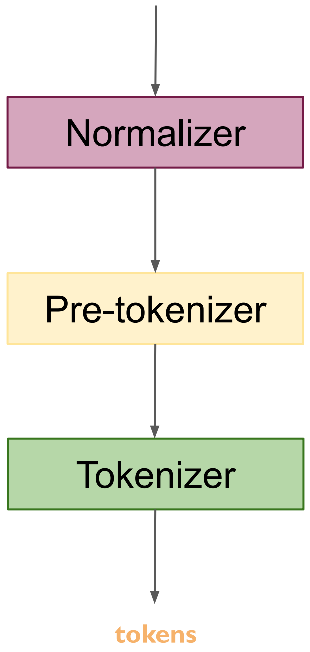
Topics in Natural Language Processing

Look Before You Leap: Pre-tokenization and Tokenization

Dr. Maunendra Sankar Desarkar

Topics

+ Text Normalization
+ Text Tokenization
+ General Tokenization algorithms
  + Byte-Pair Encoding
  + Word Piece
  + Unigram language model
  + Sentence Piece



Text or Lexical Normalization

Translating non-standard text to a standard form

Why do we need it?

1. Reducing vocabulary size
2. Bring commonality in meaning across multiple inputs where the same word is intended to be used but they appear in different forms

```
new pix coming tomoroe
```

⇩

```
new pictures coming tomorrow
```

Source: Lexical Normalization | NLP-progress (Ruder)

Datasets for such tasks

1. LexNorm
2. MultiLexNorm

Rob van der Goot. 2019. MoNoise: A Multi-lingual and Easy-to-use Lexical Normalization Tool. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 201–206, Florence, Italy. Association for Computational Linguistics.

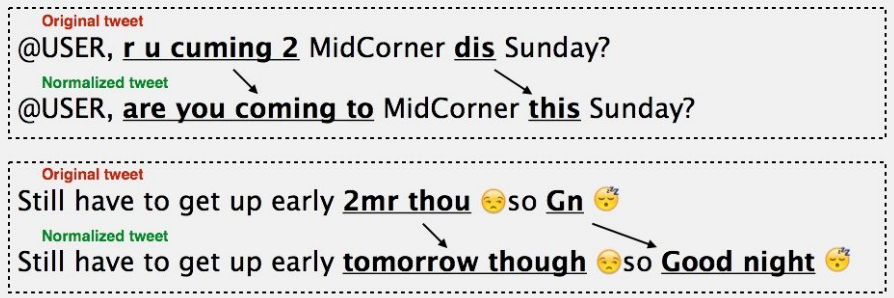
Examples from MultiLexNorm

Lang.	Language name	Normalization example
DA	Danish	De skarpe lamper gjorde destromindre ek bedre .
		De skarpe lamper gjorde destro mindre ikke bedre .
DE	German	ogäj isch hätts auch dwiddern könn
		Okay ich hätte es auch twittern können
EN	English	u hve to let ppl decide what dey want to do
		you have to let people decide what they want to do
ES	Spanish	@username cuuxamee sii peroo veen yaa eem
		@username escúchame sí pero ven ya eh
HR	Croatian	svi frendovi mi nešto rade , veceras san osta sam .
		svi frendovi mi nešto rade , večeras sam ostao sam .
ID-EN	Indonesian-English	pdhal not fully bcs those ppl jg sih .
		padahal not fully because those people juga sih .
IT	Italian	a Roma è cosí primavera che sembra gia giov
		a Roma è così primavera che sembra già giovedì
NL	Dutch	Kga me wss trg rolle vant lachn
		Ik ga me waarschijnlijk terug rollen van het lachen

Lexical Normalization

Non-standard words (NSWs) are normalised to one or more canonical English words based on a pre-defined lexicon.

- *l o v e* should be normalised to *love* (many-to-one normalisation),
- *tmrw to tomorrow* (one-to-one normalisation),
- *cu* to see you (one-to-many normalisation).
- NOTE: *IBM* should be left untouched as it is in the lexicon and in its canonical form, and the informal *lol* should be expanded to *laughing out loud*.



MoNoise for Lexical Normalization

Two steps - Candidate generation and Ranking

Candidate generation

1. Embedding-based
2. Aspell (edit-distance based)
3. Lookup list from training data
4. Split and check in valid word list (for longer words)
5. Original word

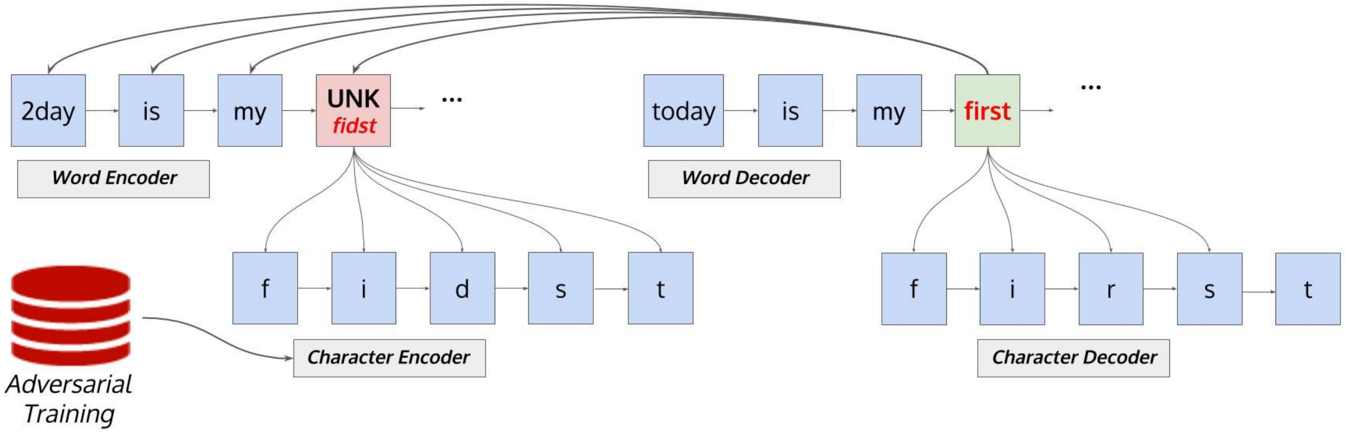
Candidate Ranking

1. Is Original
2. Embedding-distance
3. Aspell Ranking
4. Lookup List
5. N-Gram Probability
6. Character Order Match

What evaluation metrics can be used?

Seq2Seq for Lexical Normalization

- Recent trend - end-to-end
- Can we adapt Seq2Seq Models for this task?
  - Word-based seq2seq: does not work well with OOV/low-probability works
  - Char-based seq2seq: Longer training time. Also, ot robust for OOV



Lourentzou, Ismini, Kabir Manghnani, and ChengXiang Zhai. "Adapting sequence to sequence models for text normalization in social media" *Proceedings of the international AAAI conference on web and social media*. Vol. 13. 2019.

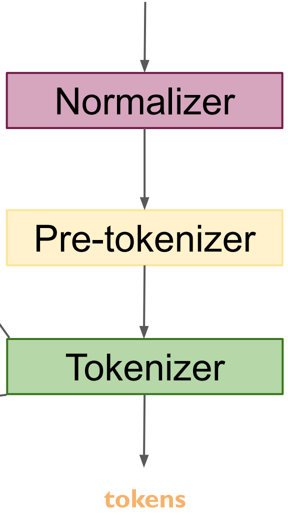
More on Lexical Normalization

- Muller, Benjamin, Benoît Sagot, and Djamé Seddah. "[Enhancing BERT for Lexical Normalization]." *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*. 2019.
- Ashmawy, Mohamed, Mohamed Waleed Fakhr, and Fahima A. Maghraby. "[Lexical normalization using generative transformer model (LN-GTM)]." *International Journal of Computational Intelligence Systems* 16.1 (2023): 183.
- Bikaun, Tyler, Melinda Hodkiewicz, and Wei Liu. "MaintNorm: A corpus and benchmark model for lexical normalisation and masking of industrial maintenance short text." *Proceedings of the Ninth Workshop on Noisy and User-generated Text (W-NUT 2024)*. 2024.
- Bucur, Ana-Maria, Adrian Cosma, and Liviu P. Dinu. "Sequence-to-Sequence Lexical Normalization with Multilingual Transformers." *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*. 2021.

Tokenization: Early

> In NLP studies, it is conventional to concentrate on pure analysis or generation while taking the basic units, namely words, for granted. It is an obvious truth, however, that without these basic units clearly segregated, it is impossible to carry out any analysis or generation. But too little attention has so far been paid to the process, a kind of preprocessing in a sense, of identifying basic units to be processed. The simplicity of

gist → *basic units clearly segregated*



Question: What are the basic units? Or How do we define basic units?
Possible answers:

1. Words (Human way)
2. Characters
3. Subwords
4. Morphemes (Linguistic way)

Jonathan J. Webster and Chunyu Kit. 1992. [Tokenization as the Initial Phase in NLP](). In *COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics*.

Tokenization: Big Picture

Tokens → unique → Vocabulary → vectorization → Embeddings | ….

Model architecture

- Current NLP models built upon the paradigm of:
  - **Pretraining + Fine Tuning**
- Fine Tuning ⇒ Typically multiple tasks
- Fixed vocabulary after tokenization (*adaptation techniques exists.*)
- Embeddings of fixed vocabulary are learned

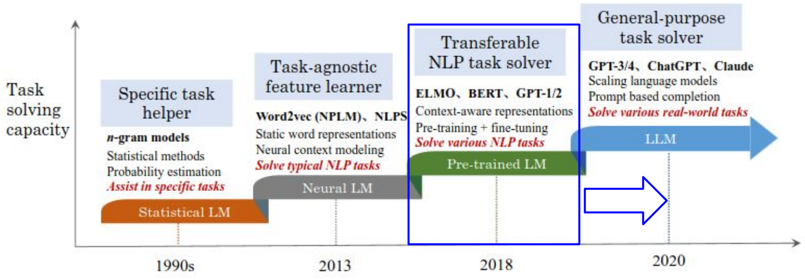


Image credits: https://arxiv.org/pdf/2303.18223

Typically in downstream task, same vocabulary as pre training is used ⇒ vocabulary should be good as per the downstream task too

Tokenization: What?

Breaking into smaller units or pieces called tokens

or

Identifying *word* boundaries

Ideally: Character tokens would be good 😊
Why?

1. Fixed in numbers
2. Solves Unknown token problem

Cons:

1. Increase in text length
2. Difficulty to capture the word level understanding
3. Existing architectures are not that good

Tokenization: What?

Breaking into smaller units or pieces called tokens

or

Identifying *word* boundaries

Ideally: <u>Character tokens</u> would be good 😊
Why?

1. Fixed in numbers
2. Solves Unknown token problem

Cons:

1. Increase in text length
2. Difficulty to capture the word level understanding
3. Existing architectures are not that good

Can we tokenize at word level ?

Tokenization: What?

Breaking into smaller units or pieces called tokens

or

Identifying *word* boundaries

Ideally: Character tokens would be good 😊
Why?

1. Fixed in numbers
2. Solves Unknown token problem

Cons:

1. Increase in text length
2. Difficulty to capture the word level understanding
3. Existing architectures are not that good

Can we tokenize at word level ?

Why? Easy for models to capture semantics
Cons:

1. Large vocabulary
2. Difficult to scale in Multilingual setting (7000+ languages across globe)
3. Unknown token problem exists

Tokenization: What?

Breaking into smaller units or pieces called tokens

or

Identifying *word* boundaries

Ideally: Character tokens would be good 😊
Why?

1. Fixed in numbers
2. Solves Unknown token problem

Cons:

1. Increase in text length
2. Difficulty to capture the word level understanding
3. Existing architectures are not that good

Can we tokenize at word level ?

Is there something between Character and Word level?

Why? Easy for models to capture semantics
Cons:

1. Large vocabulary
2. Difficult to scale in Multilingual setting (7000+ languages across globe)
3. Unknown token problem exists

Tokenization: Subword (the current standard)

- Handles unknown token problem
- Rare words are handled efficiently
- Open vocabulary
- Falls in between Word and Character level
- Controllable vocabulary size
- Variants:
  - Characters
  - Bytes (Currently adapted by LLMs!)

Subword algorithms:

1. Byte-Pair Encoding (BPE)
2. Word Piece
3. Unigram language model
4. Sentence Piece

Tokenization: Subword : BPE

Algorithm 1 Byte-pair encoding (Sennrich et al., 2016; Gage, 1994)

Input: set of strings D, target vocab size k
procedure BPE(D, k)
    $V \leftarrow$ all unique characters in D
        (about 4,000 in English Wikipedia)
    while $|V| < k$ **do** $\triangleright$ Merge tokens
        $t_L, t_R \leftarrow$ Most frequent bigram in D
        $t_{\text{NEW}} \leftarrow t_L + t_R$ $\triangleright$ Make new token
        $V \leftarrow V + [t_{\text{NEW}}]$
        Replace each occurrence of t_L, t_R in
            D with t_{NEW}
    end while
    return V
end procedure

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Tokenization: Subword : BPE Example

Steps:

1. Compute unique set of words
2. Build character vocabulary
3. Add new tokens by merging most frequency pairs
4. Continue till the desired vocabulary in achieved

Unique word	Freq.
hug	10
pug	5
pun	12
bun	4
hugs	5

Base characters: unique character in a language.

Here: [“b”, “g”, “h”, “n”, “p”, “s”, “u”] ⇒ 7 ***character encoded*** symbols

h u g	10
p u g	5
p u n	12
b u n	4
h u g s	5

Tokenization: Subword : BPE Example

Steps:

1. Compute unique set of words
2. Build character vocabulary
3. Add new tokens by merging most frequency pairs
4. Continue till the desired vocabulary in achieved

Vocab: [“b”, “g”, “h”, “n”, “p”, “s”, “u”]

h u g	10
p u g	5
p u n	12
b u n	4
h u g s	5

Most frequent pairs

u g ⇒ 20

Merge rule:
“u” “g” ⇒ “ug”

New Vocab: [“b”, “g”, “h”, “n”, “p”, “s”, “u”, “ug”]

h ug	10
p ug	5
p u n	12
b u n	4
h ug s	5

Most frequent pairs

u n ⇒ 16

Tokenization: Subword : BPE Example

Steps:

1. Compute unique set of words
2. Build character vocabulary
3. Add new tokens by merging most frequency pairs
4. Continue till the desired vocabulary in achieved

Vocab: [“b”, “g”, “h”, “n”, “p”, “s”, “u”, “ug”]

h ug	10
p ug	5
p u n	12
b u n	4
h ug s	5

Most frequent pairs

u n ⇒ 16

Updated Merge rule:
“u” “g” ⇒ “ug”
“u” “n” ⇒ “un”

h ug	10
p ug	5
p un	12
b un	4
h ug s	5

New Vocab: [“b”, “g”, “h”, “n”, “p”, “s”, “u”, “ug”, “un”]

Tokenization: Subword : BPE Example

Steps:
1. Compute unique set of words
2. Build character vocabulary
3. Add new tokens by merging most frequency pairs
4. Continue till the desired vocabulary in achieved

Vocab: [“b”, “g”, “h”, “n”, “p”, “s”, “u”, “ug”, “un”]

h ug	10
p ug	5
p un	12
b un	4
h ug s	5

Most frequent pairs

h ug ⇒ 15

hug	10
p ug	5
p un	12
b un	4
hug s	5

Updated Merge rule:

“u” “g” ⇒ “ug”
“u” “n” ⇒ “un”
“h” “ug” ⇒ “hug”

Vocab: [“b”, “g”, “h”, “n”, “p”, “s”, “u”, “ug”, “un”, “hug”]

Tokenization: Subword : BPE Example

Learned Merge rule:
“u” “g” ⇒ “ug”
“u” “n” ⇒ “un”
“h” “ug” ⇒ “hug”

Vocab: [“b”, “g”, “h”, “n”, “p”, “s”, “u”, “ug”, “un”, “hug”]

How do we tokenize new words?

⇒ Using Merge rules

Examples:

1. bug ⇒ [“b”, “ug”]
2. mug ⇒ [“UNK”, “ug”]
3. unhug ⇒ [“un”, “hug”]

Tokenization: Subword : WordPiece

- Developed by Google

- Used for pretraining popular BERT

- Similar to BPE differs in:
    - Initialization of Base vocabulary
    - Selection of pairs to be merged
    - Encoding process
    - Only saves final vocabulary
    - Uses prefix ## to identify subword

Tokenization: Subword : WordPiece Example

Unique word	Freq.
hug	10
pug	5
pun	12
bun	4
hugs	5

h ##u ##g	10
p ##u ##g	5
p ##u ##n	12
b ##u ##n	4
h ##u ##g ##s	5

Base characters:
(a) beginning character of a word
(b) character present inside a word preceded by the ##

Here: [“b”, “h”, “p”, “##g”, “##n”, “##s”, “##u”]

Tokenization: Subword : WordPiece Example

Vocab: [“b”, “h”, “p”, “##g”, “##n”, “##s”, “##u”]

h ##u ##g	10
p ##u ##g	5
p ##u ##n	12
b ##u ##n	4
h ##u ##g ##s	5

Find pairs according to score & merge

h ##u ##g	10
p ##u ##g	5
p ##u ##n	12
b ##u ##n	4
h ##u ##gs	5

New Vocab: [“b”, “h”, “p”, “##g”, “##n”, “##s”, “##u”, “##gs”]

How to compute scores?

score (a,b) = f(a,b) / (f(a) * f(b))

score (“##g”, “##s”) = f(“##g ##s”) / (f(“##g”) * f(“##s”))
= 5/20*5
= 1/20

Tokenization: Subword : WordPiece Example

score (a,b) = f(a,b) / (f(a) * f(b))

Vocab: [“b”, “h”, “p”, “##g”, “##n”, “##s”, “##u” , “##gs”]

h ##u ##g	10
p ##u ##g	5
p ##u ##n	12
b ##u ##n	4
h ##u ##gs	5

Find pairs according to score & merge

hu ##g	10
p ##u ##g	5
p ##u ##n	12
b ##u ##n	4
hu ##gs	5

New Vocab: [“b”, “h”, “p”, “##g”, “##n”, “##s”, “##u” , “##gs”, “hu”]

score (“h”, “##u”) = f(“h ##u”) / (f(“h”) * f(“##u”))
= 15/15*36
= 1/36

Tokenization: Subword : WordPiece Example

score (a,b) = f(a,b) / (f(a) * f(b))

Vocab: [“b”, “h”, “p”, “##g”, “##n”, “##s”, “##u” , “##gs”, “hu”]

hu ##g	10
p ##u ##g	5
p ##u ##n	12
b ##u ##n	4
hu ##gs	5

Find pairs according to score & merge

hug	10
p ##u ##g	5
p ##u ##n	12
b ##u ##n	4
hu ##gs	5

New Vocab: [“b”, “h”, “p”, “##g”, “##n”, “##s”, “##u” , “##gs”, “hu”, “hug”]

score (“hu”, “##g”) = f(“hu ##g”) / (f(“hu”) * f(“##g”))
= 10/15*10
= 1/15

score (“hu”, “##gs”) = f(“hu ##gs”) / (f(“hu”) * f(“##gs”))
= 5/15*5
= 1/15

Tokenization: Subword : WordPiece Example

Vocab: [“b”, “h”, “p”, “##g”, “##n”, “##s”, “##u” , “##gs”, “hu”, “hug”]

How do we tokenize new words?

⇒ find longest subword in vocab and split

Examples:

1.	hugs	⇒	[“hug”, “##s”]	⇒	[“hug”, “##s”]
2.	bugs	⇒	[“b”, “##ugs”]	⇒	[“b”, “##u”, “##gs”]
3.	mugs	⇒	[UNK]		
4.	bum	⇒	[“b”, “##um”]	⇒	[UNK]

Subword Algorithms: Unigram LM

Unigram LM (Kudo, 2018)

Algorithm 2 Unigram LM (Kudo, 2018)

Input: set of strings D, target vocab size k
procedure UNIGRAMLM(D, k)
    $V \leftarrow$ all substrings occurring more than
        once in D (not crossing words)
    while $|V| > k$ **do** $\triangleright$ Prune tokens
        Fit unigram LM θ to D
        for $t \in V$ **do** $\triangleright$ Estimate token 'loss'
            $L_t \leftarrow p_\theta(D) - p_{\theta'}(D)$
            where θ' is the LM without token t
        end for
        Remove $\min(|V| - k, \lfloor \alpha |V| \rfloor)$ of the
        tokens t with highest L_t from V,
        where $\alpha \in [0, 1]$ is a hyperparameter
    end while
    Fit final unigram LM θ to D
    return V, θ
end procedure

- Works in opposite direction of BPE & WordPiece
- Starts with large vocabulary and remove symbols until desired vocab size
- At each step train unigram language model from current vocabulary

- Kudo, Taku. "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates." ACL 2018.
- Bostrom, Kaj, and Greg Durrett. "Byte Pair Encoding is Suboptimal for Language Model Pretraining." *EMNLP Findings* 2020.

Subword Algorithms: Unigram LM : Example

Unique word	Freq.
hug	10
pug	5
pun	12
bun	4
hugs	5

Vocab: [“h”, “u”, “g”, “hu”, “ug”, “p”, “pu” , “n”, “un”, “b”, “bu”, “s”, “hug”, “gs”, “ugs”]

Subword Algorithms: Unigram LM : Example

Vocab: [“h”, “u”, “g”, “hu”, “ug”, “p”, “pu” , “n”, “un”, “b”, “bu”, “s”, “hug”, “gs”, “ugs”]

hug	10
pug	5
pun	12
bun	4
hugs	5

Frequency of subwords:

“h”	⇒	15 (hug, hugs)
“u”	⇒	36
“g”	⇒	20
“hu”	⇒	15
“ug”	⇒	20 (hug, pug, hugs)
“p”	⇒	17
“pu”	⇒	17
“n”	⇒	16
“un”	⇒	16
“b”	⇒	4
“bu”	⇒	4
“s”	⇒	5
“hug”	⇒	15
“gs”	⇒	5
“ugs	⇒	5 (hugs)

Probability of subword “ug”
= freq. of “ug” / total freq. of subwords

= 20 / 210

Subword Algorithms: Unigram LM : Example

How probabilities in Unigram are calculated?

hug	10
pug	5
pun	12
bun	4
hugs	5

Frequency of subwords:

“h”	⇒	15 (hug, hugs)
“u”	⇒	36
“g”	⇒	20
“hu”	⇒	15
“ug”	⇒	20 (hug, pug, hugs)
“p”	⇒	17
“pu”	⇒	17
“n”	⇒	16
“un”	⇒	16
“b”	⇒	4
“bu”	⇒	4
“s”	⇒	5
“hug”	⇒	15
“gs”	⇒	5
“ugs	⇒	5 (hugs)

Consider word “**pug**”

Possible tokens: [“p”, “u”, “g”], [“p”, “ug”], [“pu”, “g”]

Probabilities:

prob.([“p”, “u”, “g”]) = prob.(“p”) x prob.(“u”) x prob.(“g”)
= (17/120) x (36/210) x (20/210)
= 0.0013

prob.([“p”, “ug”]) = prob.(“p”) x prob.(“ug”)
= (17/210) x (20/210)
= 0.0077

prob.([“pu”, “g”]) = prob.(“pu”) x prob.(“g”)
= (17/210) x (20/210)
= 0.0077

Select one with highest probability

Subword Algorithms: Unigram LM : Example

hug	10
pug	5
pun	12
bun	4
hugs	5

At any given stage:

Word	**Possible subwords**		**Unigram Probabilities**
hug	[“hug”]	→	0.071428
pug	[“pu”, “g”]	→	0.007710
pun	[“pu”, “n”]	→	0.006168
bun	[“bu”, “n”]	→	0.001451
hugs	[“hug”, “s”]	→	0.001701

Loss? Negative log likelihood

Loss = 10 x (-log(0.071428)) + 5 x (-log(0.007710)) + 12 x (-log(0.006168)) + 4 x (-log(0.001451)) + 5 x (-log(0.001701))

= 169.8

Subword Algorithms: Unigram LM : Example

Which one to remove? (Exhaustive)

Let's only see two tokens: "pu" or "hug"

hug	10
pug	5
pun	12
bun	4
hugs	5

What happens if we remove "pu"?

Word	Possible subwords		Unigram Probabilities
hug	["hug"]	→	0.071428
pug	["p", "ug"]	→	0.007710
pun	["p", "un"]	→	0.006168
bun	["bu", "n"]	→	0.001451
hugs	["hug", "s"]	→	0.001701

Loss = 169.8

What happens if we remove "hug"?

Word	Possible subwords		Unigram Probabilities
hug	["hu", "g"]	→	0.006802
pug	["pu", "g"]	→	0.007710
pun	["pu", "n"]	→	0.006168
bun	["bu", "n"]	→	0.001451
hugs	["hu", "gs"]	→	0.001701

Loss = 193.317

Most likely "pu" will be removed

Subword Algorithms: SentencePiece

Raw text: Hello world.

Tokenization ↓ De-tokenization ↑ ?

Tokenized: [Hello] [world] [.]

Raw text: こんにちは世界。(Hello world.)

Tokenization ↓ De-tokenization ↑ ?

Tokenized: [こんにちは] [世界] [。]

Observation:

1. Raw text and tokenized sequences are not reversible
2. De-tokenization process are language dependent (Language-specific rules are expensive)

Motivation: How to achieve language-independent lossless de-tokenization?

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Subword Algorithms: SentencePiece

DECODE (ENCODE (NORMALIZE (TEXT)) = NORMALIZE (TEXT)

Raw text: Hello world.

Normalize text: Hello_world.

Tokenized: [Hello] [_wor] [ld] [.]

- NORMALIZE
  - sequences are treated as a ***unicode characters***
  - NFKC-based normalization
  - White Spaces are escaped with _ (Lower One-Eighth block) unicode code point.
- ENCODE
  - Uses BPE or Unigram LM algorithm for segmentation
- DECODE
  - Subsitute _ with space

Unicode codepoint ref.: https://codepoints.net/U+2581?lang=en

SentencePiece (Highly recommended): https://github.com/google/sentencepiece

Subword Algorithms: Recap

1. BPE (Expansion)
    a. Starts with ***small*** base **character** vocabulary
    b. Merge most frequent pairs
2. WordPiece (Expansion)
    a. Starts with nearly similarly base byte as BPE
    b. Merge pairs based on scores
3. Unigram (Reduction)
    a. Starts with ***large*** vocabulary
    b. Remove tokens based on unigram loss

Total vocab. = Base vocab. + New vocab.

Subword Algorithms: Recap

1. BPE (Expansion)
   a. Starts with ***small*** base **character** vocabulary
   b. Merge most frequent pairs
2. WordPiece (Expansion)
   a. Starts with nearly similarly base byte as BPE
   b. Merge pairs based on scores
3. Unigram (Reduction)
   a. Starts with ***large*** vocabulary
   b. Remove tokens based on unigram loss

How many characters can a corpus in a specific language have?

Around 4000 in English Wikipedia

Subword Algorithms: Recap

1. BPE (Expansion)
    a. Starts with ***small*** base **character** vocabulary
    b. Merge most frequent pairs
2. WordPiece (Expansion)
    a. Starts with nearly similarly base byte as BPE
    b. Merge pairs based on scores
3. Unigram (Reduction)
    a. Starts with ***large*** vocabulary
    b. Remove tokens based on unigram loss

How Small?

How many characters can a corpus in a specific language have?

Around 4000 in English Wikipedia
(Very Large)

What about characters in other languages?

Subword Algorithms: Recap

1. BPE (Expansion)
   a. Starts with ***small*** base **character** vocabulary
   b. Merge most frequent pairs
2. WordPiece (Expansion)
   a. Starts with nearly similarly base byte as BPE
   b. Merge pairs based on scores
3. Unigram (Reduction)
   a. Starts with ***large*** vocabulary
   b. Remove tokens based on unigram loss

How Small?

What about characters in other languages?

149K unicode code point (v15.1)
(Very large)

UNICODE

As of Unicode version 15.1, there are 149,878 characters with code points, covering 161 modern and historical scripts, as well as multiple symbol sets. This article includes the 1,062 characters in the Multilingual European Character Set 2 (MES-2) subset, and some additional related characters.

Subword Algorithms: Recap

1. BPE (Expansion)
   a. Starts with ***small*** base **Byte** vocabulary
   b. Merge most frequent pairs
2. WordPiece (Expansion)
   a. Starts with nearly similarly base byte as BPE
   b. Merge pairs based on scores
3. Unigram (Reduction)
   a. Starts with ***large*** vocabulary
   b. Remove tokens based on unigram loss

149K unicode code points

Very large set to be considered as base vocabulary for BPE

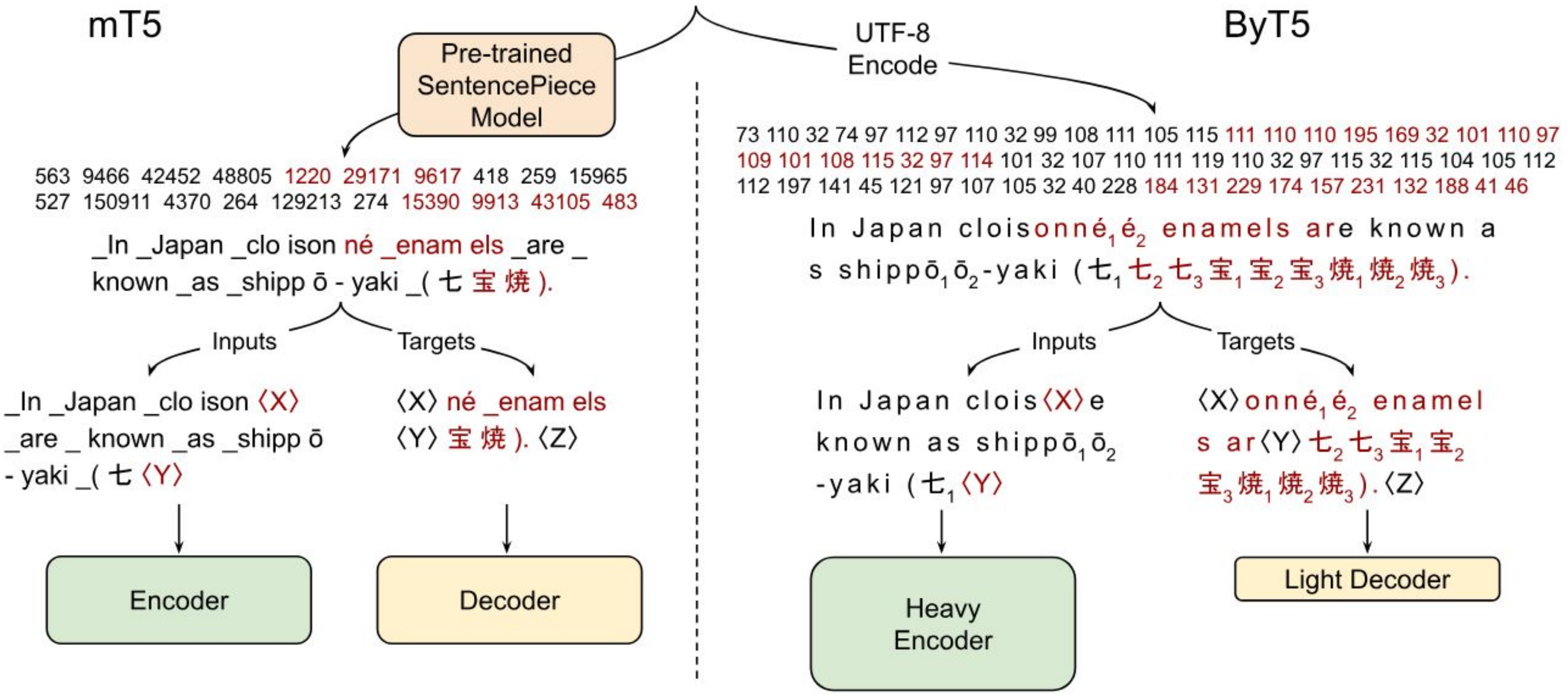
Solution: Unicode Encoding

Represent in **Bytes** instead of Characters.

UTF-8, UTF-16, UTF-32

UTF-8 can represent all unicode code points in 1-4 Bytes

ByT5


mT5
Pre-trained SentencePiece Model
563 9466 42452 48805 1220 29171 9617 418 259 15965
527 150911 4370 264 129213 274 15390 9913 43105 483
_In _Japan _clo ison né _enam els _are _
known _as _shipp ō - yaki _(七 宝 焼).
Inputs
Targets
_In _Japan _clo ison ⟨X⟩
_are _ known _as _shipp ō
- yaki _(七 ⟨Y⟩
⟨X⟩ né _enam els
⟨Y⟩ 宝 焼). ⟨Z⟩
Encoder
Decoder
UTF-8 Encode
ByT5
73 110 32 74 97 112 97 110 32 99 108 111 105 115 111 110 110 195 169 32 101 110 97
109 101 108 115 32 97 114 101 32 107 110 111 119 110 32 97 115 32 115 104 105 112
112 197 141 45 121 97 107 105 32 40 228 184 131 229 174 157 231 132 188 41 46
In Japan cloisonné$_1$é$_2$ enamels are known a
s shippō$_1$ō$_2$-yaki (七$_1$七$_2$七$_3$宝$_1$宝$_2$宝$_3$焼$_1$焼$_2$焼$_3$).
Inputs
Targets
In Japan clois⟨X⟩e
known as shippō$_1$ō$_2$
-yaki (七$_1$⟨Y⟩
⟨X⟩onné$_1$é$_2$ enamel
s ar⟨Y⟩七$_2$七$_3$宝$_1$宝$_2$
宝$_3$焼$_1$焼$_2$焼$_3$).⟨Z⟩
Heavy Encoder
Light Decoder