Raw Data

Tokenization

Cornell CS 5740: Natural Language Processing
Yoav Artzi, Spring 2023

Tokenization

- How do we represent an input text?

Tokenization: splitting a string into a sequence of tokens

- Given a piece of text $\bar{x}$, we said it's a sequence $\langle x_1, \ldots, x_n \rangle$
- But how do you get from a string to $\langle x_1, \ldots, x_n \rangle$?
  - So far: we split to "words" according to white spaces

"I love Lucy, but adore Ethel"

$$\bar{x} = \langle \text{I, Love, Lucy, ,, but, adore, Ethel} \rangle$$

  - Actually, even here we can see it's more complex. Why?

Tokenization

"I love Lucy, but adore Ethel"

$$\bar{x} = \langle \text{I, Love, Lucy, ,, but, adore, Ethel} \rangle$$

- So, tokenization is not simple, and tokenizers require may specialized rules
- Such as, what will we do with the following strings:
  - "amazing!", "state-of-the-art", "un-thinkable", "prize-winning", "aren't", "O'Neill"
  - Some languages don't even use spaces to mark word boundaries!
- Check out spaCy's tokenizers! (https://spacy.io/)

Tokenization

Handling Unknown Words

- What happens when we encounter a word that we have never seen in our training data?
  - With word-level tokenization, not much we can do
  - Except assigning to it a special <UNK> token, or maybe do something a bit smarter with some clustering
    - Don’t forget to use UNK during training — why?
  - Why this is bad?

Tokenization

Limitations of <UNK>

- Generally, we lose most of the information the word conveys 😢
- Especially hurts in texts/languages with many rare words/entities

The chapel is sometimes referred to as "Hen Gapel Lligwy" ("hen" being the Welsh word for "old" and "capel" meaning "chapel").

The chapel is sometimes referred to as " Hen <unk> <unk> " (" hen " being the Welsh word for " old " and " <unk> " meaning " chapel ").

Tokenization

Other Limitations

- Word-level tokenization treats different forms of the same root as completely separate (e.g., “open”, “opened”, “opens”, “opening”, etc)
- This means separate features or embeddings!
- Why is this a problem? Especially with limited data?

Tokenization

Other Limitations

- Word-level tokenization treats different forms of the same root as completely separate (e.g., “open”, “opened”, “opens”, “opening”, etc)
- This means separate features or embeddings!
- Why is this a problem? Especially with limited data?
- We can use pre-trained embeddings (e.g., word2vec)
  - So we can learn similar embeddings given enough data
  - But still separate parameters, and will still hurt with rare words

Character-level Tokenization

- Let's reconsider how we split:
  - Instead of white spaces, just split to characters
- Impact on vocabulary size? Unknown word problem? Other input properties?

Character-level Tokenization

- Let's reconsider how we split:
  - Instead of white spaces, just split to characters
- Impact on vocabulary size? Unknown word problem? Other input properties?
  - Small vocabulary: just the number of unique characters in the training data!
  - Much longer input sequences
  - Need to learn from scratch how to combine characters to recover word meaning
    - Will BOW/NBOW models work?

Subword Tokenization

- “Word”-level: issues with unknown words and information sharing, and gets complex fast
  - Also, fits poorly to some languages
- Character-level: long sequences, the model needs to do a lot of heavy lifting in representing that is encoded in plain-sight
- Let’s find a middle ground!
- Subword tokenization first developed for machine translations
  - Based on byte pair encoding (Gage, 1994)
- Now, used everywhere

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

The main motivation behind this paper is that the translation of some words is transparent in that they are translatable by a competent translator even if they are novel to him or her, based on a translation of known subword units such as morphemes or phonemes.

Byte Pair Encoding (BPE)

- $\mathcal{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathcal{V} \leftarrow \mathcal{V} \cup \{lr\}$
- Return $\mathcal{V}$

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- $\mathscr{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathscr{V} \leftarrow \mathscr{V} \cup \{lr\}$
- Return $\mathscr{V}$

Word	Frequency
hug	10
pug	5
pun	12
bun	4
hugs	5

$$\mathscr{V} = \{\mathrm{b}, \mathrm{g}, \mathrm{h}, \mathrm{n}, \mathrm{p}, \mathrm{s}, \mathrm{u}\}$$

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- $\mathscr{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathscr{V} \leftarrow \mathscr{V} \cup \{lr\}$
- Return $\mathscr{V}$

$$\mathscr{V} = \{\mathrm{b, g, h, n, p, s, u}\}$$

Word	Frequency
h+u+g	10
p+u+g	5
p+u+n	12
b+u+n	4
h+u+g+s	5

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- $\mathscr{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathscr{V} \leftarrow \mathscr{V} \cup \{lr\}$
- Return $\mathscr{V}$

$$\mathscr{V} = \{\mathrm{b, g, h, n, p, s, u}\}$$

Word	Frequency
h+u+g	10
p+u+g	5
p+u+n	12
b+u+n	4
h+u+g+s	5

Pair	Frequency
u+g	20
p+u	17
u+n	16
h+u	15
g+s	5

⋮

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- $\mathcal{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathcal{V} \leftarrow \mathcal{V} \cup \{lr\}$
- Return $\mathcal{V}$

$$\mathcal{V} = \{\text{b, g, h, n, p, s, u}\}$$

Word	Frequency
h+u+g	10
p+u+g	5
p+u+n	12
b+u+n	4
h+u+g+s	5

Pair	Frequency
u+g	20
p+u	17
u+n	16
h+u	15
g+s	5

⋮

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- $\mathscr{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathscr{V} \leftarrow \mathscr{V} \cup \{lr\}$
- Return $\mathscr{V}$

$$\mathscr{V} = \{\text{b, g, h, n, p, s, u, ug}\}$$

Word	Frequency
h+u+g	10
p+u+g	5
p+u+n	12
b+u+n	4
h+u+g+s	5

Pair	Frequency
u+g	20
p+u	17
u+n	16
h+u	15
g+s	5
⋮	

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- $\mathscr{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathscr{V} \leftarrow \mathscr{V} \cup \{lr\}$
- Return $\mathscr{V}$

$$\mathscr{V} = \{\mathrm{b, g, h, n, p, s, u, ug}\}$$

Word	Frequency
h+ug	10
p+ug	5
p+u+n	12
b+u+n	4
h+ug+s	5

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- $\mathscr{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathscr{V} \leftarrow \mathscr{V} \cup \{lr\}$
- Return $\mathscr{V}$

$$\mathscr{V} = \{\mathrm{b, g, h, n, p, s, u, ug}\}$$

Word	Frequency
h+ug	10
p+ug	5
p+u+n	12
b+u+n	4
h+ug+s	5

Pair	Frequency
u+n	16
h+ug	15
p+u	12
p+ug	5
ug+s	5

⋮

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- $\mathscr{V} \leftarrow$ All characters in the training data (as base **tokens**)
- For k steps:
  - Tokenize the data, taking the longest prefix each time
  - Count the frequency of adjacent token pairs in the data
  - Choose the pair $\langle l, r \rangle$ that occurs most frequently
  - Add the pair to the vocabulary as a new token $\mathscr{V} \leftarrow \mathscr{V} \cup \{lr\}$
- Return $\mathscr{V}$

$$\mathscr{V} = \{\text{b}, \text{g}, \text{h}, \text{n}, \text{p}, \text{s}, \text{u}, \text{ug}, \text{un}, \text{hug}\}$$

Word	Frequency
hug	10
p+ug	5
p+un	12
b+un	4
hug+s	5

Example from: https://huggingface.co/docs/transformers/tokenizer_summary

Byte Pair Encoding (BPE)

- To avoid <UNK> altogether, must add all possible characters/ symbols
  - Oops: there are ~138K unicode symbols
- Instead, use bytes!
  - GPT-2 does this with some rules to prevent certain types of merges
  - Commonly vocabulary sizes are 32-64K
- Package to help with tokenization: tokenizers from Hugging Face (https://github.com/huggingface/tokenizers)

Other Subword Encoding Schemes

- WordPiece (Schuster et al., 2012): merge to increase likelihood as measured by a language model (vs. frequency as in BPE)
- SentencePiece (Kudo et al., 2018): can do subword tokenization without pre-tokenization (i.e., using white spaces)
  - Good for words without such word boundaries
  - Although pre-tokenization still usually helps

Subword Tokenization

What do Subwords Capture?

- Subwords can be arbitrary strings
- But can also be meaning-bearing units
  - Can capture morphemes (the smallest meaning-bearing unit)
    - "unlikeliest" → [un-, likely, -est]
  - Can separate single form from plural
  - etc
- Importantly: this arises from the data

Subword Tokenization

Limitations

- Does not work well with languages that have more complex morphology (word forms), such as Turkish and Arabic
- Pre-tokenization using spaces doesn’t work on some languages (e.g., Chinese and Thai don’t use spaces between words)
- There are other recipes:
  - Tokenizer free, just work with bytes (e.g., ByT5)
  - Other learning techniques with soft tokenization (e.g., Charformer)

Acknowledgements

- The slides in this deck were adapted from Mohit Iyyer, with some modifications following slides by Ana Marasović.