CS 784: Computational Linguistics
Lecture 4: Morphology and Tokenization

Freda Shi

School of Computer Science, University of Waterloo
fhs@uwaterloo.ca

January 20, 2025

What is a word?

Oxford Languages:

> *A single distinct **meaningful** element of speech or writing, used with others (or sometimes alone) to form a sentence and typically shown with a space on either side when written or printed.*

What is a word?

Oxford Languages:

> *A single distinct **meaningful** element of speech or writing, used with others (or sometimes alone) to form a sentence and typically shown with a space on either side when written or printed.*

The definition of word naturally connects to the study of **lexical semantics**.

What is a word?

Does it mean things in dictionaries?

What is a word?

Does it mean things in dictionaries?

Yes and No.

> *One of the most prolific areas of change and variation in English is vocabulary;* ***new words*** *are constantly being coined to the name or describe new inventions or innovations, or to better identify aspects of our rapidly changing world... Most general English dictionaries are designed to include only those words that* ***meet certain criteria*** *of usage across wide areas and over extended periods of time...*

[Source: Merriam-Webster, `https://www.merriam-webster.com`]

What is a word?

Does it mean things between spaces and punctuation?

What is a word?

Does it mean things between spaces and punctuation?

Yes and No.

This is English: The cat is cute.	This is Chinese: 猫很可爱。
This is French: Le chat est mignon.	This is Japanese: 猫はかわいいです。
This is Spanish: El gato es lindo.	This is Thai: แมวน่ารัก.

What is a word?

Does it mean the smallest unit that can be uttered in isolation?

What is a word?

Does it mean the smallest unit that can be uttered in isolation?

Yes and No.

- You could utter this word in isolation: *unimpressively*
- Also this one: *impress*

What is a word?

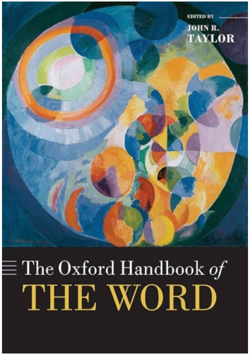
Does it mean the smallest unit that can be uttered in isolation?

Yes and No.

- You could utter this word in isolation: *unimpressively*
- Also this one: *impress*
- Probably also these when you talk about morphology: *un*, *ive*, *ly*

What is a word?

Each of the above points captures some, but likely not all aspects of what a word is.



- 42 chapters
- Nearly 900 pages
- Covers a lot of aspects of what makes a word word, “to anyone who shares a fascination with words”

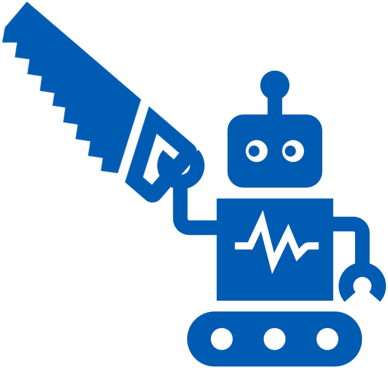
Outline of Today's Lecture

- Introduction to linguistic morphology
  –*the study of internal structures of words*
- Introduction to (conventional) lexical semantics
  –*the study of word meanings*
- Word tokenization
  –*the process of splitting texts into "words"*

colder

replayed

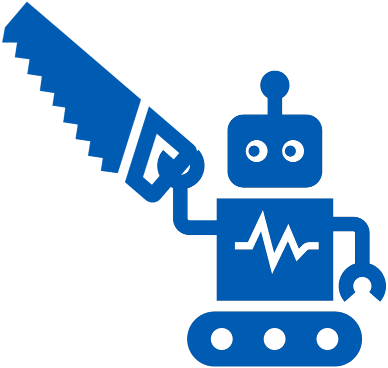
gameplay



cold|er

re|play|ed

game|play



Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morpheme: the smallest meaning-bearing unit in a language.

Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morpheme: the smallest meaning-bearing unit in a language.

Types of morphemes:

- **Stem**: the core meaning-bearing unit.
- **Affix**: a piece that attaches to a stem.

, ...

Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morpheme: the smallest meaning-bearing unit in a language.

Types of morphemes:

- **Stem**: the core meaning-bearing unit.
- **Affix**: a piece that attaches to a stem.
  - Prefix: **un**happy, **pre**define

, ...

Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morpheme: the smallest meaning-bearing unit in a language.

Types of morphemes:

- **Stem**: the core meaning-bearing unit.
- **Affix**: a piece that attaches to a stem.
  - Prefix: **un**happy, **pre**define
  - Suffix: cat**s**, walk**ed**

, ...

Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morpheme: the smallest meaning-bearing unit in a language.

Types of morphemes:

- **Stem**: the core meaning-bearing unit.
- **Affix**: a piece that attaches to a stem.
  - Prefix: **un**happy, **pre**define
  - Suffix: cat**s**, walk**ed**
  - Infix: (Malay) Gigi (*teeth*)
    $\rightarrow$ G**er**igi (*toothed blade*)

, ...

Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morpheme: the smallest meaning-bearing unit in a language.

Types of morphemes:

- **Stem**: the core meaning-bearing unit.
- **Affix**: a piece that attaches to a stem.
  - Prefix: **un**happy, **pre**define
  - Suffix: cat**s**, walk**ed**
  - Infix: (Malay) Gigi (*teeth*) → G**er**igi (*toothed blade*)
  - Circumfix: (German) mach (root of *machen*; *to make*) → **ge**mach**t** (*made*; past participle)

, ...

Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morpheme: the smallest meaning-bearing unit in a language.

Types of morphemes:

- **Stem**: the core meaning-bearing unit.
- **Affix**: a piece that attaches to a stem.
  - Prefix: **un**happy, **pre**define
  - Suffix: cat**s**, walk**ed**
  - Infix: (Malay) Gigi (*teeth*)
    $\rightarrow$ G**er**igi (*toothed blade*)
  - Circumfix: (German) mach (root of *machen*; *to make*)
    $\rightarrow$ **ge**mach**t** (*made*; past participle)
  - Interfix*: speed**o**meter, ...

Morphology

Morphology: the study of how words are built from smaller **meaning-bearing** units.

Morpheme: the smallest meaning-bearing unit in a language.

Types of morphemes:

- **Stem**: the core meaning-bearing unit.
- **Affix**: a piece that attaches to a stem.
  - Prefix: **un**happy, **pre**define
  - Suffix: cat**s**, walk**ed**
  - Infix: (Malay) Gigi (*teeth*)
    → G**er**igi (*toothed blade*)
  - Circumfix: (German) mach (root of *machen*; *to make*)
    → **ge**mach**t** (*made*; past participle)
  - Interfix*: speed**o**meter, ...

See more in Chap. 6.2 of Julianne Doner. The Linguistic analysis of word and sentences structures

Types of Word Formation

Inflection: adding morphemes to a word to indicate grammatical information.

- *walk* $\rightarrow$ *walk**ed***
- *cat* $\rightarrow$ *cat**s***

Types of Word Formation

Inflection: adding morphemes to a word to indicate grammatical information.

- *walk* → *walk**ed***
- *cat* → *cat**s***

Derivation: adding morphemes to a word to create a new word with a **different meaning**.

- *happy* → *happi**ness***
- *define* → ***pre**define*

Types of Word Formation

Inflection: adding morphemes to a word to indicate grammatical information.

- *walk* → *walk**ed***
- *cat* → *cat**s***

Derivation: adding morphemes to a word to create a new word with a **different meaning**.

- *happy* → *happi**ness***
- *define* → ***pre**define*

Compounding: combining two or more words to create a new word.

- key + board → keyboard
- law + suit → lawsuit
- book + case → bookcase

Isolating Language

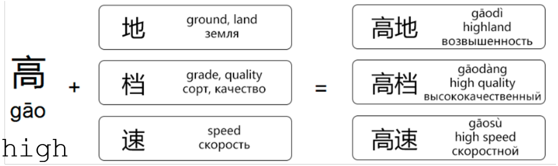
In languages like Classical Chinese, Vietnamese, and Thai:

- Each word form typically consists of one single morpheme;
- There is little morphology other than compounding.

Isolating Language

In languages like Classical Chinese, Vietnamese, and Thai:

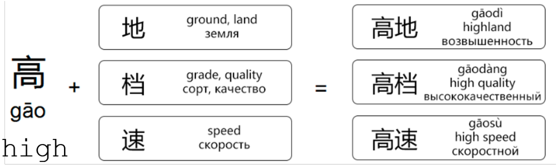
- Each word form typically consists of one single morpheme;
- There is little morphology other than compounding.



Isolating Language

In languages like Classical Chinese, Vietnamese, and Thai:

- Each word form typically consists of one single morpheme;
- There is little morphology other than compounding.



Chinese is a champion in the realm of compounding—up to 80% of Chinese words are compounds.

Morphological Decomposition

Usually, morphological decomposition is simply splitting a word into its morphemes.

- `walked` = `walk` + `ed`
- `unhappiness` = `un` + `happy` + `ness`

Morphological Decomposition

Usually, morphological decomposition is simply splitting a word into its morphemes.

- `walked` = `walk` + `ed`
- `unhappiness` = `un` + `happy` + `ness`

However, it can actually be a hierarchical structure.

- `unbreakable` = `un` + (`break` + `able`)
- `internationalization` = (((`inter` + `nation`) + `al`) + `iz(e)`) + `tion`

Morphological Decomposition

Usually, morphological decomposition is simply splitting a word into its morphemes.

- `walked` = `walk` + `ed`
- `unhappiness` = `un` + `happy` + `ness`

However, it can actually be a hierarchical structure.

- `unbreakable` = `un` + (`break` + `able`)
- `internationalization`
  = (((`inter` + `nation`) + `al`) + `iz(e)`) + `tion`

There is ambiguity in hierarchical decomposition.

`The door is unlockable.`

Morphological Decomposition

Usually, morphological decomposition is simply splitting a word into its morphemes.

- walked = walk + ed
- unhappiness = un + happy + ness

However, it can actually be a hierarchical structure.

- unbreakable = un + (break + able)
- internationalization = (((inter + nation) + al) + iz(e)) + tion

There is ambiguity in hierarchical decomposition.

`The door is unlockable.`

- (un + lock) + able: able to be unlocked.
- un + (lock + able): not able to be locked.

Morphology in Computational Linguistics/NLP

Individual tasks that address morphology:

- **Lemmatization**: putting words/tokens in a standard format.
  - **Lemma**: canonical/dictionary form of a word.
  - **Wordform**: fully inflected or derived form of a word as it appears in text.

Wordform	Lemma
run	run
ran	run
running	run

Morphology in Computational Linguistics/NLP

Individual tasks that address morphology:

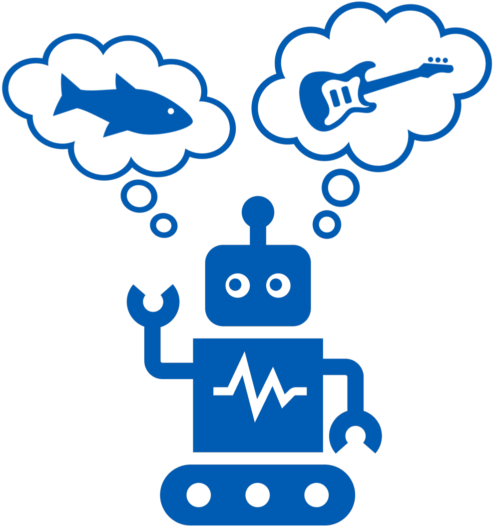
- **Lemmatization**: putting words/tokens in a standard format.
  - **Lemma**: canonical/dictionary form of a word.
  - **Wordform**: fully inflected or derived form of a word as it appears in text.

Wordform	Lemma
run	run
ran	run
running	run

- **Stemming**: reducing words to their stems (approximately) by removing affixes.
  More conventional engineering-oriented approach used in applications such as retrieval.

$$\texttt{words} \rightarrow \texttt{word}$$
$$\texttt{imaginative} \rightarrow \texttt{imagin}$$
$$\texttt{airplanes} \rightarrow \texttt{airplan}$$

bass



Variability and Ambiguity in Words

Lemmatization and stemming tackles the problem of variability—multiple forms could share the same or similar meanings.

Variability and Ambiguity in Words

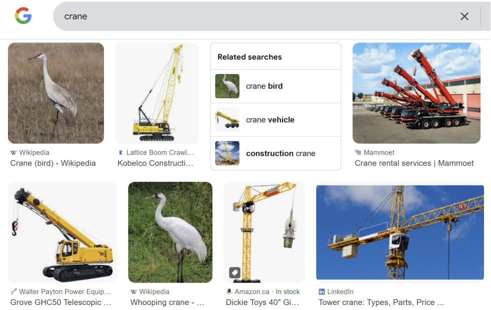
Lemmatization and stemming tackles the problem of variability—multiple forms could share the same or similar meanings.

On the other hand, one wordform could refer to multiple meanings.

Variability and Ambiguity in Words

Lemmatization and stemming tackles the problem of variability—multiple forms could share the same or similar meanings.

On the other hand, one wordform could refer to multiple meanings.



[Source: Google Images]

Polysemy vs. Homonymy

Polysemy: a word has multiple **related** meanings.

- She is a **star** in the movie.
- (Point to the sky) The **star** is shining.

Polysemy vs. Homonymy

Polysemy: a word has multiple **related** meanings.

- She is a **star** in the movie.
- (Point to the sky) The **star** is shining.

Homonymy: a word has multiple meanings originated from different sources.

- I need to go to the **bank** as I don't have enough cash.
- I am sitting on the **bank** of the river.

Polysemy vs. Homonymy

Polysemy: a word has multiple **related** meanings.

- She is a **star** in the movie.
- (Point to the sky) The **star** is shining.

Homonymy: a word has multiple meanings originated from different sources.

- I need to go to the **bank** as I don't have enough cash.
- I am sitting on the **bank** of the river.

Question: Which one is the case for **crane**?

Synomyms

Synonyms: (informal definition) words that have the same meanings, according to some criteria.

- `couch` and `sofa`
- `big` and `large`
- `water` and H_2O

Synomyms

Synonyms: (informal definition) words that have the same meanings, according to some criteria.

- `couch` and `sofa`
- `big` and `large`
- `water` and H_2O

There are very few (or no) examples of perfect synonymy.

Synomyms

Synonyms: (informal definition) words that have the same meanings, according to some criteria.

- `couch` and `sofa`
- `big` and `large`
- `water` and H_2O

There are very few (or no) examples of perfect synonymy. Synonymy is a relation between **senses** rather than words.

- How **big** is the plane?
- How **large** is the plane?

Synomyms

Synonyms: (informal definition) words that have the same meanings, according to some criteria.

- `couch` and `sofa`
- `big` and `large`
- `water` and H_2O

There are very few (or no) examples of perfect synonymy. Synonymy is a relation between **senses** rather than words.

- How **big** is the plane?
- How **large** is the plane?

- Miss Nelson became a kind of **big** sister to Benjamin.
- Miss Nelson became a kind of **large** sister to Benjamin. (*)

Antonyms

Antonyms: senses that are opposite with respect to (at least) one feature of meaning.

- *dark* and *light*
- *dark* and *bright*
- *hot* and *cold*
- *in* and *out*

Hyponymy/Hypernymy, and Meronym/Holonym

Sense A is a **hyponym** of sense B if A is more specific, denoting a **subclass** of B.

- *dog* is a hyponym of *animal*
- *corgi* is a hyponym of *dog*

Hyponymy/Hypernymy, and Meronym/Holonym

Sense A is a **hyponym** of sense B if A is more specific, denoting a **subclass** of B.

- *dog* is a hyponym of *animal*
- *corgi* is a hyponym of *dog*

Conversely, sense B is a **hypernym** of sense A.

Hyponymy/Hypernymy, and Meronym/Holonym

Sense A is a **hyponym** of sense B if A is more specific, denoting a **subclass** of B.

- *dog* is a hyponym of *animal*
- *corgi* is a hyponym of *dog*

Conversely, sense B is a **hypernym** of sense A.

Sense A is a **meronym** of sense B if A is a **part** of B.

- hand is a meronym of body
- finger is a meronym of hand

Hyponymy/Hypernymy, and Meronym/Holonym

Sense A is a **hyponym** of sense B if A is more specific, denoting a **subclass** of B.

- *dog* is a hyponym of *animal*
- *corgi* is a hyponym of *dog*

Conversely, sense B is a **hypernym** of sense A.

Sense A is a **meronym** of sense B if A is a **part** of B.

- hand is a meronym of body
- finger is a meronym of hand

Conversely, sense B is a **holonym** of sense A.

Hyponymy/Hypernymy, and Meronym/Holonym

Sense A is a **hyponym** of sense B if A is more specific, denoting a **subclass** of B.

- *dog* is a hyponym of *animal*
- *corgi* is a hyponym of *dog*

Conversely, sense B is a **hypernym** of sense A.

Sense A is a **meronym** of sense B if A is a **part** of B.

- hand is a meronym of body
- finger is a meronym of hand

Conversely, sense B is a **holonym** of sense A.

The WordNet Database: `https://wordnet.princeton.edu/`

Word Sense Disambiguation

Word Sense Disambiguation (WSD): the task of determining which sense of a word is used in a particular context, given a set of possible senses.

Relatedly, word sense induction (WSI) requires clustering word usages into senses without predefined ground truths.

Word Sense Disambiguation

Word Sense Disambiguation (WSD): the task of determining which sense of a word is used in a particular context, given a set of possible senses.

Relatedly, word sense induction (WSI) requires clustering word usages into senses without predefined ground truths.

Default solution (as of 2025): encode the context of the word with a pretrained model and train a neural network to predict the sense.

The Role of Word Senses in 2025?

The Role of Word Senses in 2025?

A practical question: We have powerful neural language models, which do not distinguish word senses. Do we still need WSD in applications?

The Role of Word Senses in 2025?

A practical question: We have powerful neural language models, which do not distinguish word senses. Do we still need WSD in applications?

A philosophical question in lexical semantics: Do discrete word senses even exist?

[Jiangtian Li. (2024). *Semantic minimalism and the continuous nature of polysemy*. Mind and Language.]

Tokenization

Tokenization: the process that converts running text (i.e., a sequence of characters) into a sequence of **tokens**.

Tokenization

Tokenization: the process that converts running text (i.e., a sequence of characters) into a sequence of **tokens**.

```
"Oh!" said Lydia stoutly, "I am not afraid; for
though I _am_ the youngest, I'm the tallest."
```

```
" Oh ! " said Lydia stoutly , " I am not afraid ; for
though I _ am _ the youngest , I 'm the tallest . "
```

Conventions in Rule-Based Tokenizers

	Penn Treebank	Moses
don't	do n't	don 't
can't	ca n't	can 't
aren't	are n't	aren 't
won't	wo n't	won 't

Conventions in Rule-Based Tokenizers

	Penn Treebank	Moses
don't	do n't	don 't
can't	ca n't	can 't
aren't	are n't	aren 't
won't	wo n't	won 't

It is important to check & ensure the consistency when comparing results across different tokenizers.

Conventions in Rule-Based Tokenizers

	Penn Treebank	Moses
don't	do n't	don 't
can't	ca n't	can 't
aren't	are n't	aren 't
won't	wo n't	won 't

It is important to check & ensure the consistency when comparing results across different tokenizers.

See `nltk.tokenizer`, which also works for sentence tokenization.

Tokenization across Languages

There is no explicit whitespace between words in some languages, and tokenization becomes highly nontrivial in these cases.

姚明	进入	总决赛	**Chinese Treebank**
YaoMing	*reaches*	*finals*	

姚	明	进入	总	决赛	**Peking University**
Yao	*Ming*	*reaches*	*overall*	*finals*	

[Source: Chen et al. (2017)]

Word Types vs. Word Tokens

```
“ oh !  ”  said lydia stoutly , “ i am not afraid ; for
though i _ am _ the youngest , i 'm the tallest .  ”
```

Count	Word Type
3	i
2	,
2	_
2	am
2	the
2	“
2	”

Count	Word Type
1	!
1	.
1	;
1	afraid
1	for
1	lydia
1	not

Count	Word Type
1	oh
1	said
1	stoutly
1	tallest
1	though
1	youngest
1	’m

Word Types vs. Word Tokens

“ oh ! ” said lydia stoutly , “ i am not afraid ; for
though i _ am _ the youngest , i 'm the tallest . ”

Count	Word Type
3	i
2	,
2	_
2	am
2	the
2	“
2	”

Count	Word Type
1	!
1	.
1	;
1	afraid
1	for
1	lydia
1	not

Count	Word Type
1	oh
1	said
1	stoutly
1	tallest
1	though
1	youngest
1	’m

Type: a unique word form in the text, or an entry in a vocabulary.

Word Types vs. Word Tokens

`“ oh ! ” said lydia stoutly , “ i am not afraid ; for though i _ am _ the youngest , i 'm the tallest . ”`

Count	Word Type
3	i
2	,
2	_
2	am
2	the
2	“
2	”

Count	Word Type
1	!
1	.
1	;
1	afraid
1	for
1	lydia
1	not

Count	Word Type
1	oh
1	said
1	stoutly
1	tallest
1	though
1	youngest
1	’m

Type: a unique word form in the text, or an entry in a vocabulary.

Token: an instance of a type in the text.

Word Types vs. Word Tokens

```
“ oh !  ”  said lydia stoutly , “ i am not afraid ; for
though i _ am _ the youngest , i 'm the tallest .  ”
```

Count	Word Type
3	i
2	,
2	_
2	am
2	the
2	“
2	”

Count	Word Type
1	!
1	.
1	;
1	afraid
1	for
1	lydia
1	not

Count	Word Type
1	oh
1	said
1	stoutly
1	tallest
1	though
1	youngest
1	'm

Type: a unique word form in the text, or an entry in a vocabulary.

Token: an instance of a type in the text.

type count = 21, token count = 29

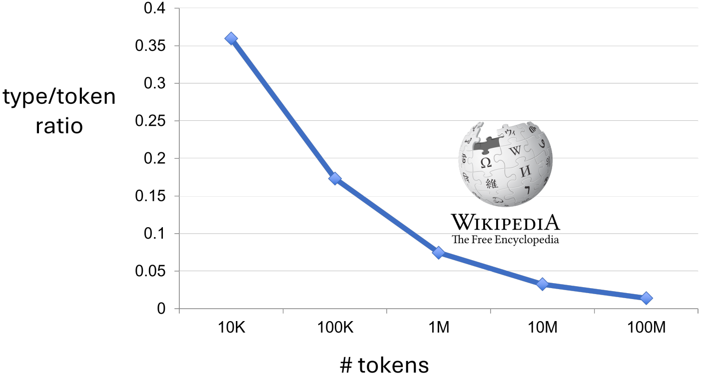
Type-Token Ratio

How does the type/token ratio change when adding more data?

Type-Token Ratio

How does the type/token ratio change when adding more data?

Words don't appear out of nowhere!



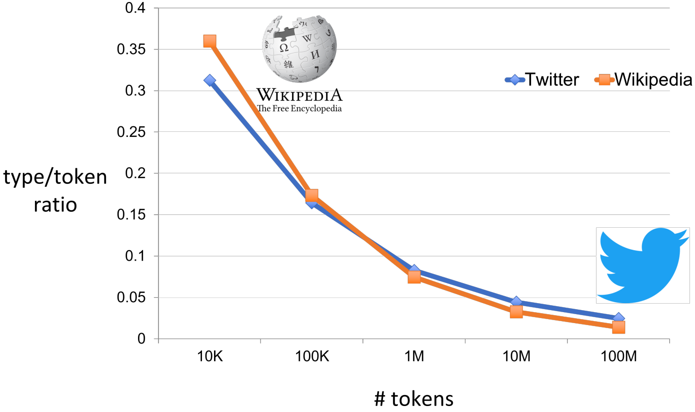
[Figure credit: Kevin Gimpel]

Type-Token Ratio: Wikipedia vs. Twitter

How do the type-token ratio curves compare between Wikipedia and Twitter?

Type-Token Ratio: Wikipedia vs. Twitter

How do the type-token ratio curves compare between Wikipedia and Twitter?



[Figure credit: Kevin Gimpel]

Word	#
really	224571
rly	1189
realy	1119
rlly	731
reallly	590
realllly	234
reallyy	216
relly	156
reallllly	146
rily	132
reallyyy	104
realllllly	89
reeeally	89
reaaally	84
reaally	82

Word	#
reeeeally	72
reaaaally	65
reallyyyy	57
rilly	53
reallllllly	50
reeeeeally	48
reeally	41
really2	38
reaaaaally	37
reallyyyyy	35
reely	31
realllyyy	30
reaaly	27
realllyy	27
realllyyyy	26

Word	#
realllllllly	25
reaaallly	22
really-	21
reeaally	19
reallllyyy	18
reaaaallly	16
reaallly	15
reallllllllly	15
reallllyy	15
reallyreally	15
realyy	15
reallllyyyy	14
reeeeeeally	14
reeeaaally	13
reaaaaaally	12

Word	#
reeeealy	7
reeeeeeeeeally	7
relaly	7
r-e-a-l-l-y	6
r-really	6
reaaaaaallly	6
realllllllllly	6
realllyyyyy	6
realyl	6
reeeaaaally	6
reeeaaallly	6
reeeaaalllyyy	6
reaaaaallly	5
reaaaalllly	5
reaalllyy	5

Word	#
reallllllllllllly	5
realllllllllllly	5
reeallyyy	5
reeeeaaallly	5
reeeeaally	5
reeeeeeeeally	5
rellly	5
rrly	5
rrrreally	5
reaaaaly	4
reaaalllly	4
reaaalllyy	4
reaalllly	4
reaalllyyy	4
realllllllyyyy	4

Word	#
realllllyyyy	4
reeaaaally	4
reeealy	4
reeeeeeeeeeally	4
rllly	4
r34lly	3
r]eally	3
reaaaaaaaaally	3
reaaaaaly	3
reaaaalllllly	3
reaaaallyy	3
reaaallyy	3
reaalllllly	3
reaallyyyy	3
realiy	3

And many pages more...

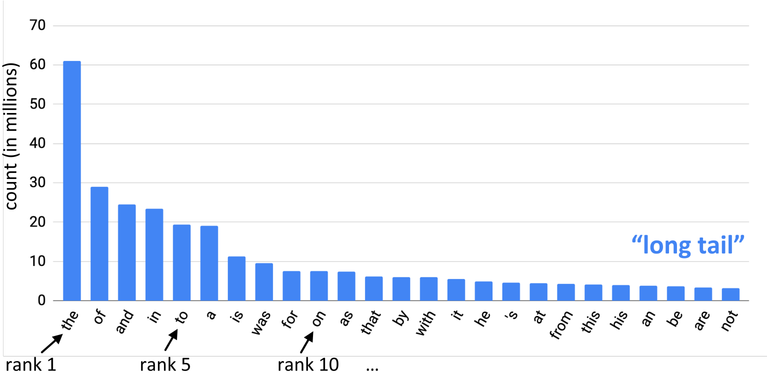
Zipf's Law

The size of vocabulary continues to grow as more data is added, but the rate of growth slows down.

Zipf's Law

The size of vocabulary continues to grow as more data is added, but the rate of growth slows down.

Zipf's Law: the frequency of a word is **inversely proportional** to its **rank** in the frequency table.



[Figure credit: Kevin Gimpel]

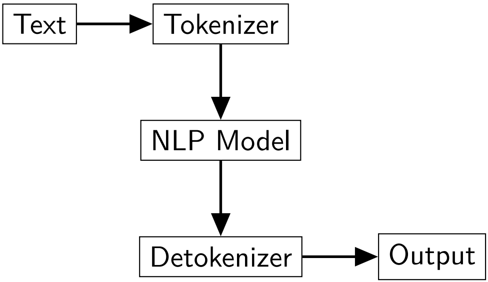
Tokenization in Modern NLP Systems

There are so many word types, but the words have shared **internal structures** and **meanings**.

Tokenization in Modern NLP Systems

There are so many word types, but the words have shared **internal structures** and **meanings**.

Modern NLP systems always covert tokens into numerical indices for further processing. Can we do better than assign each word a unique index?



Data-Driven Subword-Based Tokenizers

Data-driven tokenizers offer an option that **learns** the tokenization rules from data, tokenizing texts into **subword units** (a.k.a. **wordpieces**) using statistics of character sequences in the dataset.

Data-Driven Subword-Based Tokenizers

Data-driven tokenizers offer an option that **learns** the tokenization rules from data, tokenizing texts into **subword units** (a.k.a. **wordpieces**) using statistics of character sequences in the dataset.

Two most popular methods:

- **Byte Pair Encoding (BPE)**: Gage (1994), Sennrich et al. (2016)
- **SentencePiece**: Kudo (2018)

Byte Pair Encoding

Originally introduced by Gage (1994) for data compression, and later adapted (and revived) by Sennrich et al. (2016) for NLP.

[Phillip Gage. (1994). A new algorithm for data compression. The C Users Journal.]

Byte Pair Encoding

Originally introduced by Gage (1994) for data compression, and later adapted (and revived) by Sennrich et al. (2016) for NLP.

Key idea: "**merge**" symbols with a greedy algorithm.

Initialize the vocabulary with the set of characters, and iteratively merge the most frequent pair of symbols to extend the vocabulary.

[Phillip Gage. (1994). A new algorithm for data compression. The C Users Journal.]

Byte Pair Encoding

Originally introduced by Gage (1994) for data compression, and later adapted (and revived) by Sennrich et al. (2016) for NLP.

Key idea: "**merge**" symbols with a greedy algorithm.

Initialize the vocabulary with the set of characters, and iteratively merge the most frequent pair of symbols to extend the vocabulary.

Training Corpus
c a t
c a t s
c o n c a t e n a t i o n
c a t e g o r i z a t i o n

[Phillip Gage. (1994). A new algorithm for data compression. The C Users Journal.]

Byte Pair Encoding

Originally introduced by Gage (1994) for data compression, and later adapted (and revived) by Sennrich et al. (2016) for NLP.

Key idea: "**merge**" symbols with a greedy algorithm.

Initialize the vocabulary with the set of characters, and iteratively merge the most frequent pair of symbols to extend the vocabulary.

Training Corpus
c a t
c a t s
c o n c a t e n a t i o n
c a t e g o r i z a t i o n

Initial Vocabulary
a c e g i n o r s t z

[Phillip Gage. (1994). A new algorithm for data compression. The C Users Journal.]

Byte Pair Encoding

Originally introduced by Gage (1994) for data compression, and later adapted (and revived) by Sennrich et al. (2016) for NLP.

Key idea: "**merge**" symbols with a greedy algorithm.

Initialize the vocabulary with the set of characters, and iteratively merge the most frequent pair of symbols to extend the vocabulary.

Training Corpus

`c a t`
`c a t s`
`c o n c a t e n a t i o n`
`c a t e g o r i z a t i o n`

Initial Vocabulary

`a c e g i n o r s t z`

Count Symbol-Pair Frequencies

<a t>: 6, <c a>: 4, <o n>: 3, ...

[Phillip Gage. (1994). A new algorithm for data compression. The C Users Journal.]

Byte Pair Encoding

Originally introduced by Gage (1994) for data compression, and later adapted (and revived) by Sennrich et al. (2016) for NLP.

Key idea: "**merge**" symbols with a greedy algorithm.

Initialize the vocabulary with the set of characters, and iteratively merge the most frequent pair of symbols to extend the vocabulary.

Training Corpus

c a t

c a t s

c o n c a t e n a t i o n

c a t e g o r i z a t i o n

Initial Vocabulary

a c e g i n o r s t z

Count Symbol-Pair Frequencies

<a t>: 6, <c a>: 4, <o n>: 3, ...

Update Vocabulary

a c e g i n o r s t z at

[Phillip Gage. (1994). A new algorithm for data compression. The C Users Journal.]

Byte Pair Encoding (Cont.)

Update the corpus by replacing the instances of merged pairs with the new symbol.

Training Corpus
c at
c at s
c o n c at e n at i o n
c at e g o r i z at i o n

Byte Pair Encoding (Cont.)

Update the corpus by replacing the instances of merged pairs with the new symbol.

Current Vocabulary

`a c e g i n o r s t z at`

Training Corpus

`c at`
`c at s`
`c o n c at e n at i o n`
`c at e g o r i z at i o n`

Byte Pair Encoding (Cont.)

Update the corpus by replacing the instances of merged pairs with the new symbol.

Current Vocabulary

a c e g i n o r s t z at

Training Corpus

c at

c at s

c o n c at e n at i o n

c at e g o r i z at i o n

Count Symbol-Pair Frequencies

<c at>: 4, <o n>: 3, <t e>: 2, ...

Byte Pair Encoding (Cont.)

Update the corpus by replacing the instances of merged pairs with the new symbol.

Training Corpus

c at
c at s
c o n c at e n at i o n
c at e g o r i z at i o n

Current Vocabulary

a c e g i n o r s t z at

Count Symbol-Pair Frequencies

<c at>: 4, <o n>: 3, <t e>: 2, ...

Update Vocabulary

a c e g i n o r s t z at cat

Byte Pair Encoding (Cont.)

Update the corpus by replacing the instances of merged pairs with the new symbol.

Training Corpus

c at
c at s
c o n c at e n at i o n
c at e g o r i z at i o n

Current Vocabulary

a c e g i n o r s t z at

Count Symbol-Pair Frequencies

<c at>: 4, <o n>: 3, <t e>: 2, ...

Update Vocabulary

a c e g i n o r s t z at cat

Repeat the above steps until the desired vocabulary size is reached.

Upon completion, the training corpus is tokenized into vocabulary symbols.

Issues with Byte Pair Encoding

The BPE proposal is not optimal in terms of compression rate.

Training Corpus
cat
cat s
c o n cat e n at i o n
cat e g o r i z at i o n

Termination Vocabulary
a c e g i n o r s t z at cat

Issues with Byte Pair Encoding

The BPE proposal is not optimal in terms of compression rate.

Training Corpus
cat
cat s
c o n cat e n at i o n
cat e g o r i z at i o n

Termination Vocabulary
a c e g i n o r s t z **at** cat

A Better Vocabulary
a c e g i n o r s t z cat **on**

Issues with Byte Pair Encoding

The BPE proposal is not optimal in terms of compression rate.

Training Corpus
cat
cat s
c o n cat e n at i o n
cat e g o r i z at i o n

Termination Vocabulary
a c e g i n o r s t z **at** cat

A Better Vocabulary
a c e g i n o r s t z cat **on**

However, with its simplicity and practical effectiveness, BPE has been widely adopted in NLP systems such as GPT-2.

Issues with Byte Pair Encoding

The BPE proposal is not optimal in terms of compression rate.

Training Corpus

`cat`
`cat s`
`c o n cat e n at i o n`
`cat e g o r i z at i o n`

Termination Vocabulary

`a c e g i n o r s t z` **`at`** `cat`

A Better Vocabulary

`a c e g i n o r s t z cat` **`on`**

However, with its simplicity and practical effectiveness, BPE has been widely adopted in NLP systems such as GPT-2.

Open problem: there doesn't exist strong theoretical justification on the compression rate! [cf. Kozma and Voderholzer (2024)]

Note: whenever there is an open problem note, it means you'll receive full marks in the course if you solve it or improve the state of the art nontrivially, no matter how you perform in the rest of the course.

Apply Trained BPE to Another Corpus

Greedy encoding: at each time, apply the longest possible token in the vocabulary – this can be easily implemented with a Trie within $\mathcal{O}(\sum_{v \in V} |v| + n)$ time, where n is the length of the input text, and V is the vocabulary.

Vocabulary

`a c e g i n o r s t`

`z at cat cate er`

Apply Trained BPE to Another Corpus

Greedy encoding: at each time, apply the longest possible token in the vocabulary – this can be easily implemented with a Trie within $\mathcal{O}(\sum_{v \in V} |v| + n)$ time, where n is the length of the input text, and V is the vocabulary.

Vocabulary
`a c e g i n o r s t`
`z at cat cate er`

`cater` $\rightarrow$ `cate r`

Apply Trained BPE to Another Corpus

Greedy encoding: at each time, apply the longest possible token in the vocabulary – this can be easily implemented with a Trie within $\mathcal{O}(\sum_{v \in V} |v| + n)$ time, where n is the length of the input text, and V is the vocabulary.

Vocabulary
`a c e g i n o r s t`
`z at cat cate er`

`cater` $\rightarrow$ `cate r`

`categorial` $\rightarrow$ `cate g o r i a ?`

Apply Trained BPE to Another Corpus

Greedy encoding: at each time, apply the longest possible token in the vocabulary – this can be easily implemented with a Trie within $\mathcal{O}(\sum_{v\in V}|v|+n)$ time, where n is the length of the input text, and V is the vocabulary.

Vocabulary
`a c e g i n o r s t`
`z at cat cate er`

$$\texttt{cater} \rightarrow \texttt{cate r}$$

$$\texttt{categorial} \rightarrow \texttt{cate g o r i a ?}$$

Yet another issue with BPE: the tokenizer may fail on corpus with unseen characters—keeping a special token for unknown characters is necessary in practice.

Sentencepiece Tokenization

The name `sentencepiece` is ambiguous, as it may refer to:

Sentencepiece Tokenization

The name `sentencepiece` is ambiguous, as it may refer to:

- The algorithm proposed by Kudo (2018).

- The Google command-line toolkit that implements the Kudo (2018) algorithm and some others (including BPE): `https://github.com/google/sentencepiece`.

Sentencepiece Tokenization

The name `sentencepiece` is ambiguous, as it may refer to:

- The algorithm proposed by Kudo (2018).
  Key idea: find the vocabulary for a unigram language model that maximizes the likelihood of the training corpus.

$$P(s = \langle x_1, x_2, \ldots, x_n \rangle) = \prod_{i=1}^{n} P(x_i)$$

$$P(x_i) \propto \textit{optional-smoothing}\left(\frac{c(x_i)}{\sum_{x \in V} c(x)}\right)$$

- The Google command-line toolkit that implements the Kudo (2018) algorithm and some others (including BPE):
  `https://github.com/google/sentencepiece`.

Sentencepiece Tokenization

The name `sentencepiece` is ambiguous, as it may refer to:

- The algorithm proposed by Kudo (2018).
  Key idea: find the vocabulary for a unigram language model that maximizes the likelihood of the training corpus.

$$P(s = \langle x_1, x_2, \ldots, x_n \rangle) = \prod_{i=1}^{n} P(x_i)$$

$$P(x_i) \propto \textit{optional-smoothing} \left(\frac{c(x_i)}{\sum_{x \in V} c(x)} \right)$$

- The Google command-line toolkit that implements the Kudo (2018) algorithm and some others (including BPE):
  `https://github.com/google/sentencepiece`.
  In practice, we can simply apply the command-line tool to train a tokenizer on a corpus.

A Practical Question

How does ChatGPT (GPT-2 etc.) tokenize texts from different languages, with a unified tokenizer and fixed vocabulary size?

A Practical Question

How does ChatGPT (GPT-2 etc.) tokenize texts from different languages, with a unified tokenizer and fixed vocabulary size?

Byte-level BPE (BBPE)

That's great 👍

54 68 61 74 2019 73 **20** 67 72 65 61 74 **20** 1F44D

All in hexadecimal.

A Practical Question

How does ChatGPT (GPT-2 etc.) tokenize texts from different languages, with a unified tokenizer and fixed vocabulary size?

Byte-level BPE (BBPE)

That's great 👍

54 68 61 74 2019 73 **20** 67 72 65 61 74 **20** 1F44D

All in hexadecimal.

Prepend zeros to fix the length of tokens, and do BPE on the bit/multi-bit level.

A Practical Question

How does ChatGPT (GPT-2 etc.) tokenize texts from different languages, with a unified tokenizer and fixed vocabulary size?

Byte-level BPE (BBPE)

That's great 👍

54 68 61 74 2019 73 **20** 67 72 65 61 74 **20** 1F44D

All in hexadecimal.

Prepend zeros to fix the length of tokens, and do BPE on the bit/multi-bit level.

GPT-2 vocabulary size: 50257

Next

Edit Distance, Distributional Lexical Semantics