Tokenization

CSE 5525: Foundations of Speech and Natural Language Processing

https://shocheen.github.io/courses/cse-5525-spring-2026

THE OHIO STATE UNIVERSITY

Logistics

- Hw2 due today.

- Project proposal samples shared on teams– proposals due on Feb 20.

- HW3 will be released on Feb 21st (due two weeks after).

Outline: Tokenization

I. Word-level tokenization
II. Character / byte level tokenization
III. Subword Tokenization
    I. BPE (primary topic)
    II. WordPiece
    III. Unigram (depending on time)

What's tokenization

Recall: A language model is a probability distribution over a sequence of "tokens"; each token is from a "vocabulary".

What is a token: "basic unit that need not be decomposed for further processing" [Webster and Kit, 1992]

Tokenization: Splitting or segmenting a "string" of text into a sequence of *tokens*

"I love watching the television"

$$\bar{x} = \langle \text{I, love, watching, the, television} \rangle$$

Why do we need to tokenize?

- Neural networks work with real-valued numbers. Most machine learning deals with continuous data but text is discrete

- Text needs to be converted to a form that a model can consume/generate.

Set of all unique tokens form a *vocabulary*

- Given a tokenization \`algorithm\`
  - Tokenize a corpus of text
  - Collect all unique tokens (also known as *types*) → vocabulary

- The role of vocabulary in a language model
  - The vocabulary and its size is part of the model architecture
  - It defines the size of the input embedding/unembedding table and final output layer
    - large vocabulary = more parameters

A simple tokenizer: Split by whitespace?

"I love watching the television"

$$\bar{x} = \langle \text{I, love, watching, the, television} \rangle$$

- Tokenization is not always this simple
- Such as, what will we do with the following strings:
  - "amazing!", "state-of-the-art", "unthinkable", "prize-winning", "aren't", "O'Neill"
  - Some languages don't even use spaces to mark word boundaries!

私は日本語を勉強しています

I am studying Japanese.

Çekoslovakyalılaştıramadıklarımızdanmışsınız

You are one of those whom we could not turn into a Czechoslovakian.

What about a word level tokenizer?

- Define each token as a word or a punctuation:
  - What is a word: smallest unit of language that carries meaning and can stand alone or combine with other units to create more complex meanings
- How to split into words?
  - For languages like English:
    - Could be simple regexes: split on all spaces and punctuation
      - What about “The value of pi = 3.14”, “the IP address is 0.0.0.0”
      - What about “He got cold feet” → should cold feet be one or two words?
  - For languages like Chinese:
    - 我爱自然语言处理 (I love Natural Language Processing)
    - Need specialized tools (e.g. jieba)

Some related terminology

- A **morpheme** is the smallest meaning-bearing unit of a language
    - “unlikeliest” has the morphemes {un-, likely, -est}
- **Morphology** is the study of the way words are built up from morphemes
- **Word forms** are the variations of a word that express different grammatical categories typically through the use of morphemes.
    - **Tense** (when something happened; past, present, future)
    - **Case** (inflecting nouns/pronoun and their modifiers to express their relationship with other words; English has largely lost its inflected case system)
    - **Number** (singular/plural)
    - **Gender** (masculine, feminine, neuter; not extensively used in English)
  - and thus help convey the specific meaning and function of the word in a sentence

Word level tokenizer – How to define a vocabulary?

- Given a tokenization algorithm
    - Tokenize a corpus of text
    - Collect all unique tokens → vocabulary
- For a English corpus, a large corpus can have 1M+ unique words. Typically in any languages, the set of words can be arbitrarily large, new words are always getting invented.
    - Language vocabulary becomes too large or unbounded or “open”.
- A popular solution: Cut this list to include only K tokens.
    - How to chose K – based on frequency
    - What to do with the rest? – replace with an UNK “unknown” token
    - Word level tokenizers lead to “closed” vocabulary models.

Handling Unknown Words

- What happens when we encounter a word that we have never seen in our training data?
  - Not much we can do
  - Except assigning to it a special <UNK> token
  - Why this is bad?

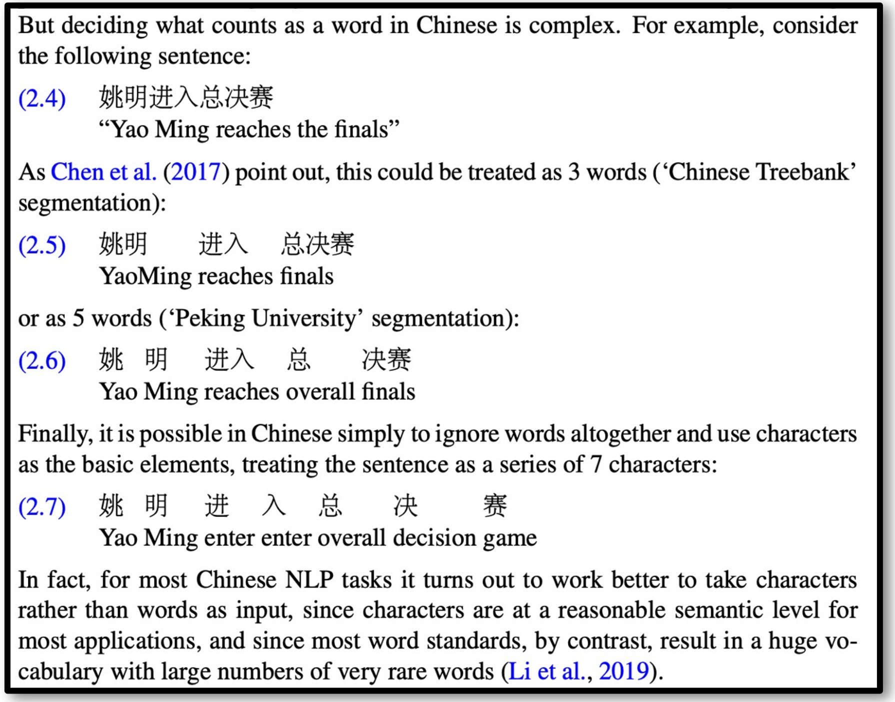
Limitations of <UNK>

- Generally, we lose most of the information the word conveys. UNKs don't give features for novel words that are useful anchors of meaning
- What if you want to generate a word which is not in the vocabulary? Imagine ChatGPT generating UNK.
- Especially hurts in [productive] 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 ").*

Word Level 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.



But deciding what counts as a word in Chinese is complex. For example, consider the following sentence:

(2.4) 姚明进入总决赛
“Yao Ming reaches the finals”

As Chen et al. (2017) point out, this could be treated as 3 words (‘Chinese Treebank’ segmentation):

(2.5) 姚明 进入 总决赛
YaoMing reaches finals

or as 5 words (‘Peking University’ segmentation):

(2.6) 姚 明 进入 总 决赛
Yao Ming reaches overall finals

Finally, it is possible in Chinese simply to ignore words altogether and use characters as the basic elements, treating the sentence as a series of 7 characters:

(2.7) 姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

In fact, for most Chinese NLP tasks it turns out to work better to take characters rather than words as input, since characters are at a reasonable semantic level for most applications, and since most word standards, by contrast, result in a huge vocabulary with large numbers of very rare words (Li et al., 2019).

A simpl*er* tokenizer: Character-level

I love watching the television

<I _ l o v e _ w a t c h i n g _ t h e _ t e l e v i s i o n>

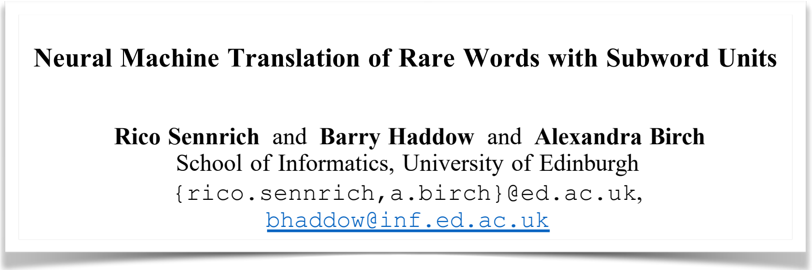
Issues

- Make sequences very long
- Word-level models provide an inductive bias to models. BUT character-level models must learn to compose characters into words.
    - Need deeper models

[Jurafsky and Martin Sec 2.4 p17]

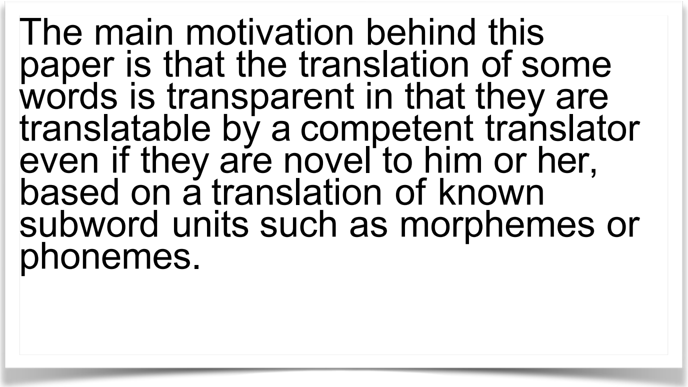
Current standard: *Subword* Tokenization

- "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 translation
  - 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.

A redefinition of the notion of tokenization

Current text segmentation follows two steps: rule based “pre-tokenization” + learned sub-word tokenization.

Pretokens are computed using regular expressions (such as split by space + punctuation) = basically word-level tokenizers.

- https://github.com/huggingface/tokenizers/tree/main/tokenizers/src/pre_tokenizers

Unseen word can be represented by some sequence of known subwords; “lower”=”low”+ “er”

[Mielke et al., 2021]

Byte-Pair-Encoding (BPE)

[coined by Gage et al., 1994; adapted to the task of word segmentation by Sennrich et al., 2016; see Gallé (2019) for more]

Main idea: Use our data to automatically tell us what the tokens should be

Token learner:

Raw train corpus ⇒ Vocabulary (a set of tokens)

Token segmenter:

Raw sentences ⇒ Tokens in the vocabulary

[Jurafsky & Martin (2023)]

Byte-Pair-Encoding (BPE) – **Token learner**

[coined by Gage et al., 1994; adapted to the task of word segmentation by **Sennrich et al., 2016**; see Gallé (2019) for more]

Raw train corpus ⇒ Vocabulary (a set of tokens)

- Pre-tokenize the corpus in words & append a special end-of-word symbol _ to each word
- Initialize vocabulary with the set of all individual characters
- Choose 2 tokens that are most frequently adjacent (“A”, “B”)
  - Respect word boundaries: Run the algorithm inside words
- Add a new merged symbol (“AB”) to the vocabulary
- Change the occurrence of the 2 selected tokens with the new merged token in the corpus
- Continues doing this until k merges are done

[Jurafsky & Martin (2023)]

Byte-Pair-Encoding (BPE) – Token learner *Example*

end-of-word symbol

corpus	
5	l o w _
2	l o w e s t _
6	n e w e r _
3	w i d e r _
2	n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w

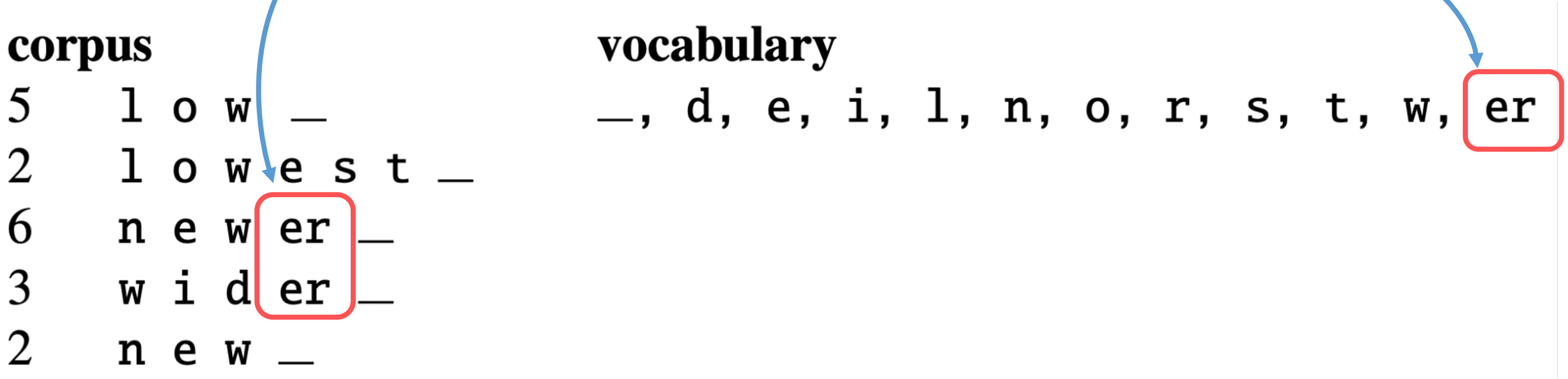
each word is split into characters

word occurrence count in the corpus

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair `e r` [a total of 9 occurrences]
- Merge these symbols, treating `er` as one symbol, & add the new symbol to the vocabulary



corpus

5	l o w _
2	l o w e s t _
6	n e w er _
3	w i d er _
2	n e w _

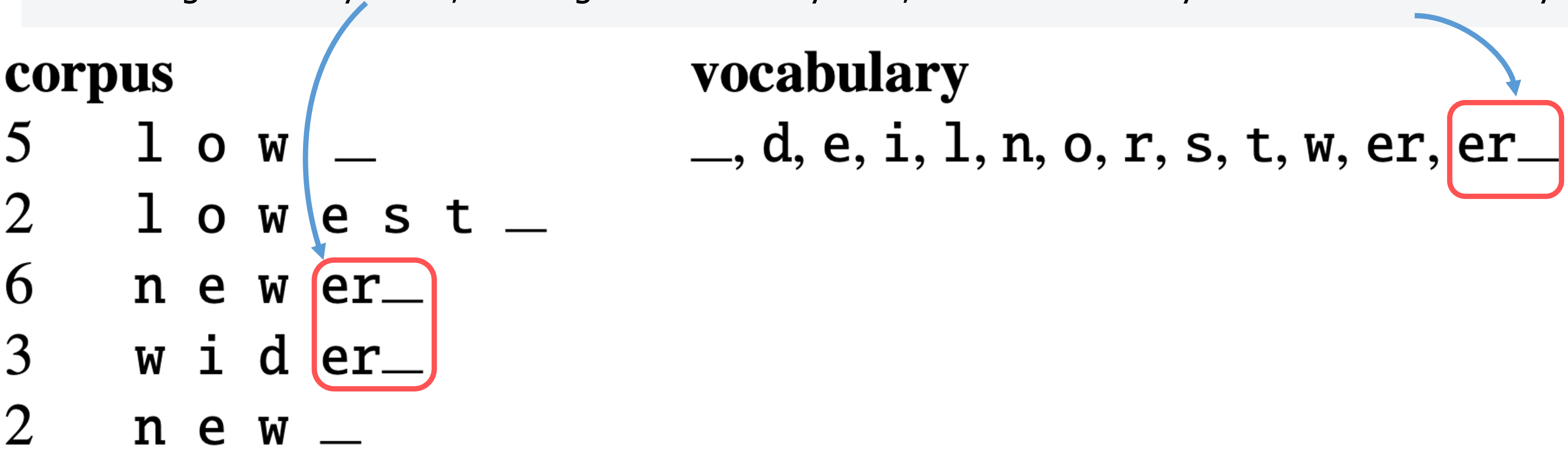
vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair `e r _`
- Merge these symbols, treating `er_` as one symbol, & add the new symbol to the vocabulary



corpus	
5	l o w _
2	l o w e s t _
6	n e w er_
3	w i d er_
2	n e w _

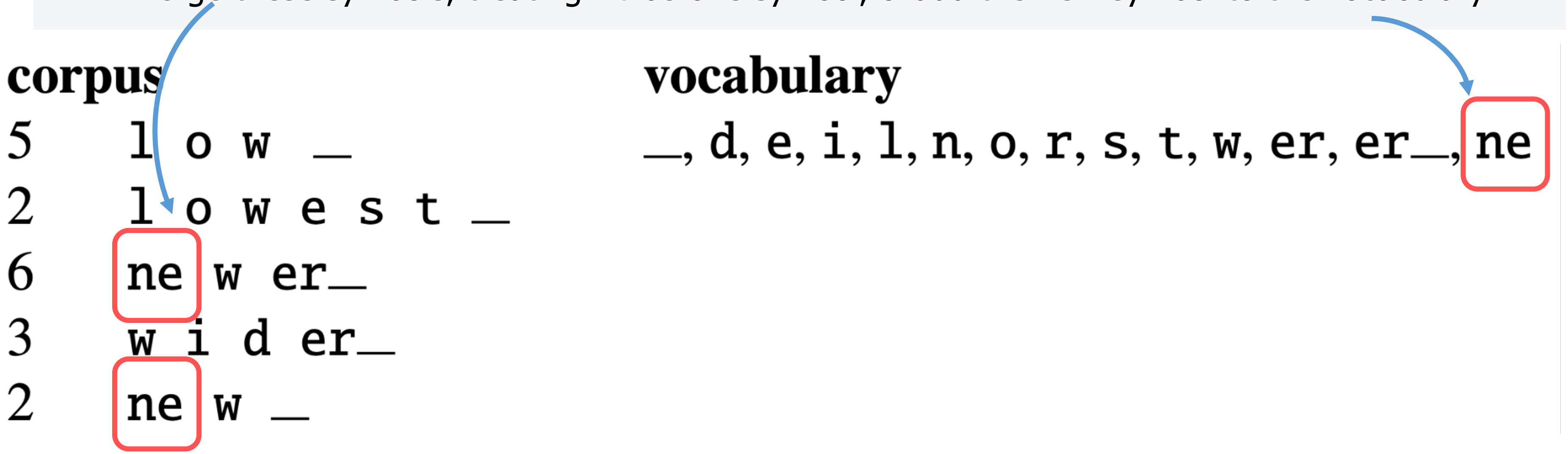
vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair n e
- Merge these symbols, treating ne as one symbol, & add the new symbol to the vocabulary



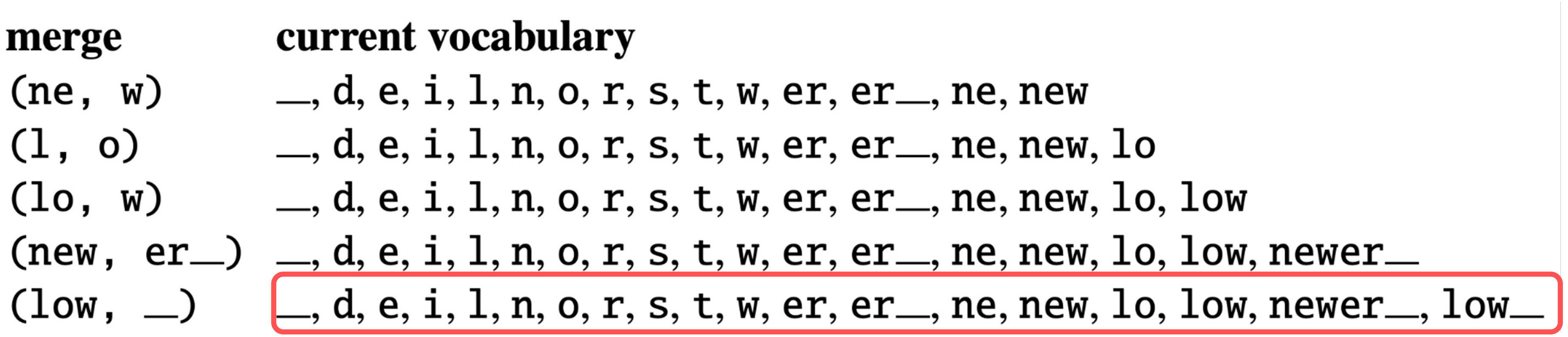
corpus	
5	l o w _
2	l o w e s t _
6	ne w er_
3	w i d er_
2	ne w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_, ne

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*



merge	current vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Final vocabulary

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – **Token segmenter**

[coined by Gage et al., 1994; adapted to the task of word segmentation by **Sennrich et al., 2016**; see Gallé (2019) for more]

- The token segmenter is used to tokenize a test sentence
    - Goal: Apply the merges we've learned from the training data, greedily, in the order we learned them

- First, we segment each test sentence word into characters

- Then, we apply the first merge rule
  - E.g., replace every instance of e r in the test corpus with er

- Then the second merge rule
  - E.g., replace every instance of er _ in the test corpus with er_

- And so on

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – **Token segmenter**

[coined by Gage et al., 1994; adapted to the task of word segmentation by **Sennrich et al., 2016**; see Gallé (2019) for more]

- Test example: slow_ → s l o w_ → s lo w_ -> s low_

- Test example: now → n o w

BPE can tokenize a word never seen at training time.

Leads to an open vocabulary model*

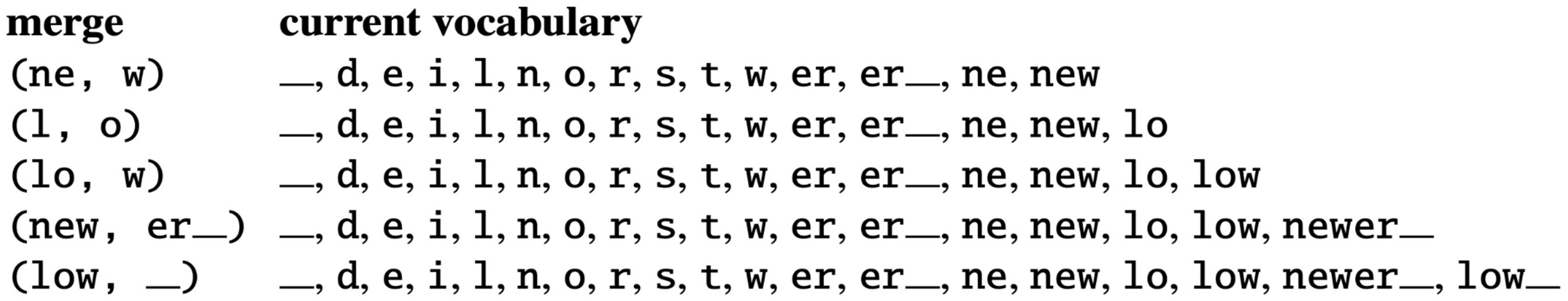
Can often learn morphological segmentations

Deescalation → De escalat ion

A variant of BPE: WordPiece

used in BERT and some follow ups

- Training algorithm: Same as BPE
- Segmentation algorithm: greedily pick the longest prefix that exists in the vocabulary (and repeat)
    - Slow_ → s low_ [stop]



merge	current vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE/Wordpiece summary

- Start with a character vocabulary
- Merge frequent bigrams
- Repeat until a desired vocab size/merge size is reached.

Unigram LM Tokenizer

1. Start with a **large base vocabulary**, **remove tokens** until a desired size is reached.

1. **How to construct a base vocabulary**: all substrings of pre-tokenized words OR start with a large BPE vocabulary.

1. **How to remove tokens**:
    a. Compute **the unigram LM loss** over the corpus (more details later)
    b. Removing tokens increases this loss.
    c. Select and remove tokens that increase it the least.
    d. Repeat

Base Vocabulary

- The corpus:

  ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

- Initial Vocabulary (all strict substrings)

  ("h", 15) ("u", 36) ("g", 20) ("hu", 15) ("ug", 20) ("p", 17) ("pu", 17) ("n", 16)

  ("un", 16) ("b", 4) ("bu", 4) ("s", 5) ("hug", 15) ("gs", 5) ("ugs", 5)

Unigram LM loss

- Unigram LM loss = negative log **probability of the corpus**.

- **Probability of a corpus** = product of marginal **probability of individual words**

  p(pug hugs bugs) = p(pug) x p(hugs) x p (bugs)

Probability of a word

- product of marginal probability of its subwords (based on frequency)

  p(pug) = p(“p”) x p(“u”) x p(“g”)

  Or, p(pug) = p(“pu”) x p(“g”)

  Or, p(pug) = p(“p”) x p(“ug”)

  Choose highest of all possible splits (why?)

- How to this efficiently: Dynamic programming (Viterbi algorithm)

Unigram Tokenization Algorithm

1. Start with a base vocabulary

2. Compute the unigram loss, L, over the corpus

VERY SLOW!

3. For every token, w, in the vocabulary
    a. Remove w from the vocabulary and recompute the loss, L'(w)
    b. Define score(w) = L'(w) - L

4. Compute w* = min_w score(w).
    a. Remove w* from the vocabulary.
    b. Got to step 2. Repeat until a desired vocabulary size is reached.

Unigram Tokenization Algorithm (Slightly Faster)

1. Start with a base vocabulary

2. Compute the unigram loss, L, over the corpus

3. For every token, w, in the vocabulary
    a. Remove w from the vocabulary and recompute the loss, L'(w)
    b. Define score(w) = L'(w) - L

4. Compute **W** = x% of tokens with the lowest score.
    a. Remove w* from the vocabulary.
    b. Got to step 2. Repeat until a desired vocabulary size is reached.

Unigram Tokenization Algorithm (Slightly Faster)

1. Start with a base vocabulary

1. Compute the unigram loss, L

1. For every token, w, in the vocabulary
    a. Remove w from the vocabulary and recompute the loss, L'(w)
    b. Define score(w) = L'(w) - L

1. Compute **W** = x% of tokens with the lowest score.
    a. Remove **W** from the vocabulary.
    b. Go to step 2.

How to tokenize once the vocabulary is decided

- Tokenization which maximizes the unigram probability of the word
- (or find top *k* tokenizations)
- “Unhug” (For each position, the subwords with the best scores ending in that position:)

  Character 0 (u): "u" (score 0.171429)

  Character 1 (n): "un" (score 0.076191)

  Character 2 (h): "un" "h" (score 0.005442)

  Character 3 (u): "un" "hu" (score 0.005442)

  Character 4 (g): "un" "hug" (score 0.005442) **[final tokenization]**

Unigram vs BPE

1. Why unigram over BPE and WordPiece?
   a. Unigram finds optimal coding length of a sequence (according to Shannon’s entropy).
   b. Unigram allows sampling multiple tokenizations for every word – subword regularization – robustness

1. Why is Unigram tokenization not more popular?
   a. In many papers simply referred to as SentencePiece, which is actually not a tokenizer but a library wrapping several tokenizers
   b. Does the actual subword tokenizer matter as we scale up models?

Subword methods are not TRULY open-vocabulary

- What if you encounter a character not seen at training time?

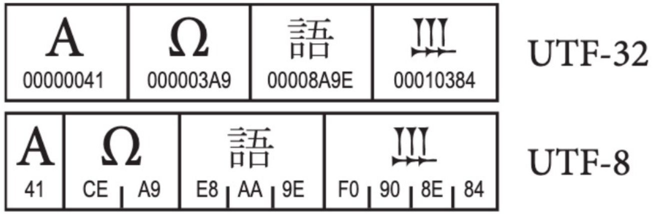
- For example, a BPE model trained on only English encounters a Chinese character at test time --- it gets assigned UNK

- How to solve this issue?
  - Train on a mix of all characters? Accounting for all languages, they are millions of characters – vocabulary size would be too large

Solution: Byte-level subword Models (BBPE)

- Every written language is represented in Unicode

Unicode is a text encoding standard designed to support the use of text in all of the world's writing systems that can be digitized. It has multiple versions like UTF-8, UTF-16, UTF-32. UTF-8 is the most common one.



- Each character is a sequence of “bytes”.
  - Each byte is 8-bit. Total 256 unique byte values
  - Each character requires between 1 to 4 bytes.

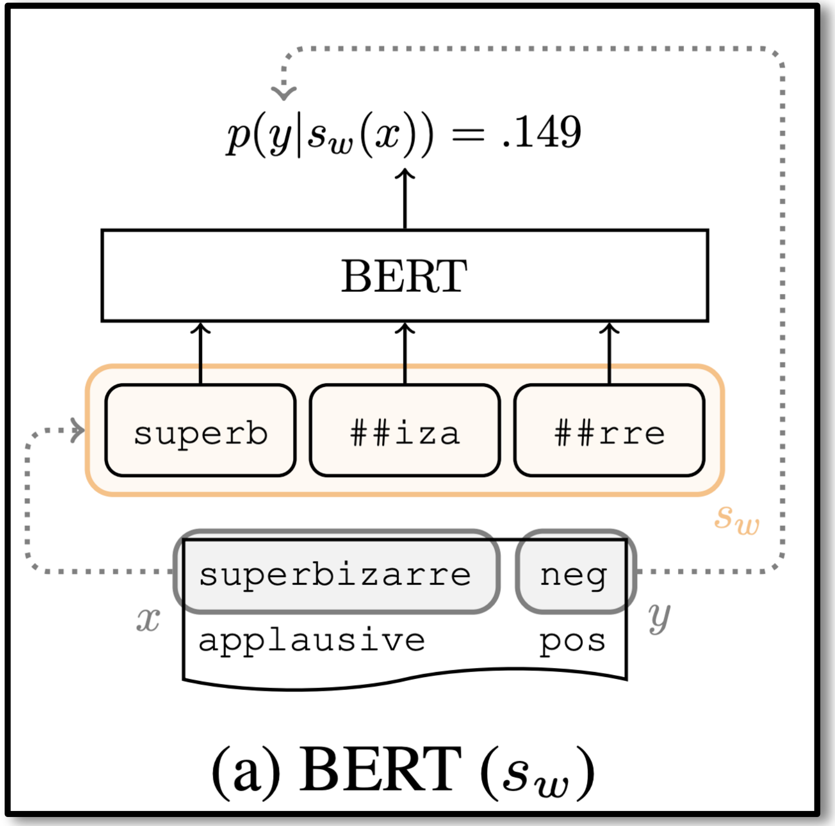
Solution: Treat a corpus as a sequence of byte. Start the vocabulary with all bytes (256) and train a BPE model. TRULY Open Vocabulary. Works for all characters!*

Subword Models -- Summary

- Split text in tokens learned statistically from the training corpus
- Makes the model open vocabulary
- Subword methods prove to be an effective method of “compressing text”

Issues with subword models

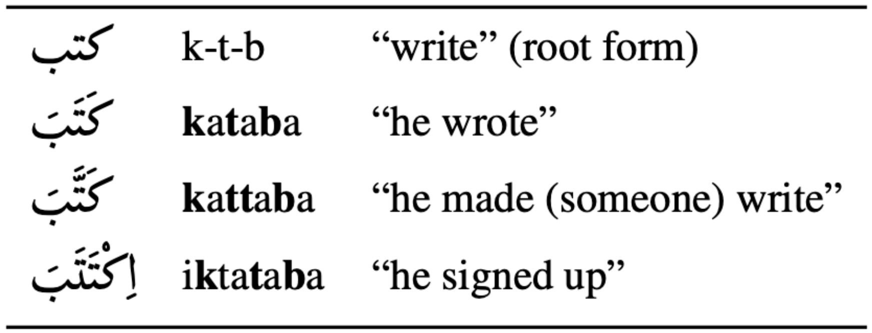
BERT thinks the sentiment of "superbizarre" is positive because its tokenization contains the token "superb"



(a) BERT (s_w)

[Hofmann et al., 2021]

Non-concatenative Languages



كتب	k-t-b	“write” (root form)
كَتَبَ	**k**a**t**a**b**a	“he wrote”
كَتَّبَ	**k**a**tt**a**b**a	“he made (someone) write”
اِكْتَتَبَ	i**k**ta**t**a**b**a	“he signed up”

Table 1: Non-concatenative morphology in Arabic.[4] The root contains only consonants; when conjugating, vowels, and sometimes consonants, are interleaved with the root. The root is not separable from its inflection via any contiguous split.

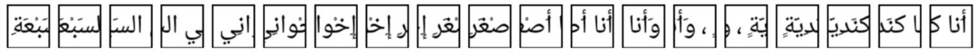
Subword Tokenization and “noise”

- He fell and broke his **coccix** (vs coccyx)
- Neighbor = 1 token, neighbour = two tokens
- John Smith = 2 tokens, Srinivas Ramanujam = ??

Subword Tokenization and “numbers”

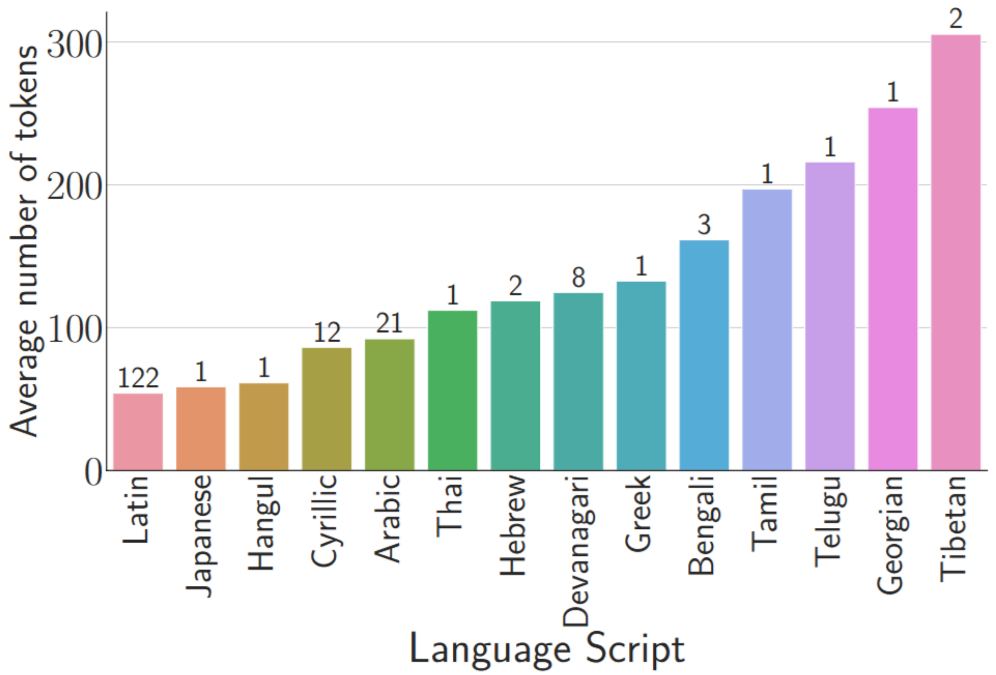
- Why are LMs bad at basic arithmetic?
  - 3.14 is tokenized as 3.14
  - 3.15 is tokenized as 3 . 15

Natural phenomena like diacritics or a little easily human-readable noise lead to unexpected BPE sequences and catastrophic translation failure

Arabic–English	
src	أنا كندية، وأنا أصغر أخواني السبعة
`diacritics` 1.0	أَنا كَنَدِيَّةٍ ، وَأَنا أَصْغَرِ إِخْوانِي السَبْعَةِ
ref	I'm Canadian, and I'm the youngest of seven kids.
in_{vis}	
out_{vis}	I'm a Canadian, and I'm the youngest of my seven sisters.
COMET	0.764
in_{text}	_ أ َ ان _ ك َ ن َ د ِ ي ّ َ ة ٍ _ , _ و َ _ أ َ ان _ أ َ ص ْ غ َ ر ِ _ إ ِ خ ْ او ن ِ ي _ الس َ ب ْ ع َ ة _ .
out_{text}	We grew up as a teacher, and we gave me a hug.
COMET	-1.387

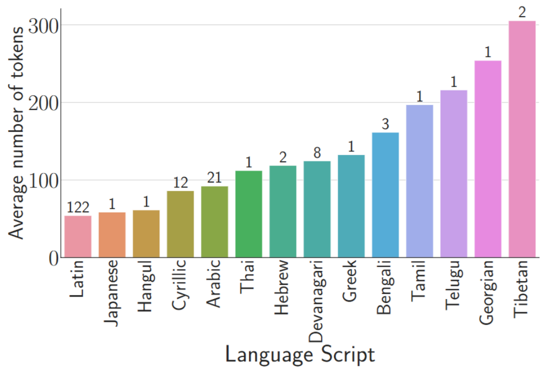
[Salesky et al., 2021]

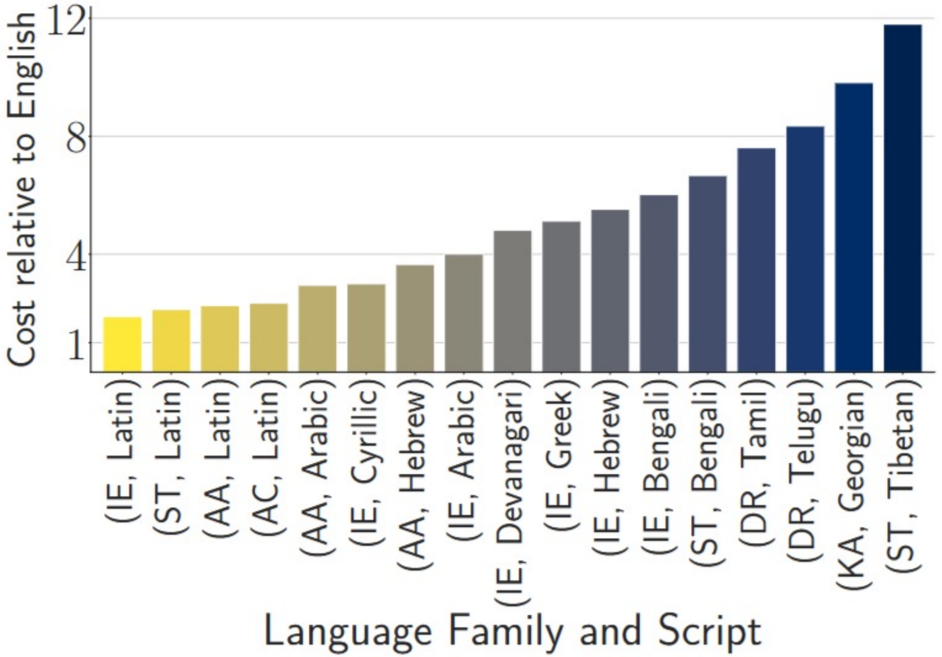
Sequence Lengths, Costs, and Performance



[[2305.13707] Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models]()

Sequence Lengths, Costs, and Performance

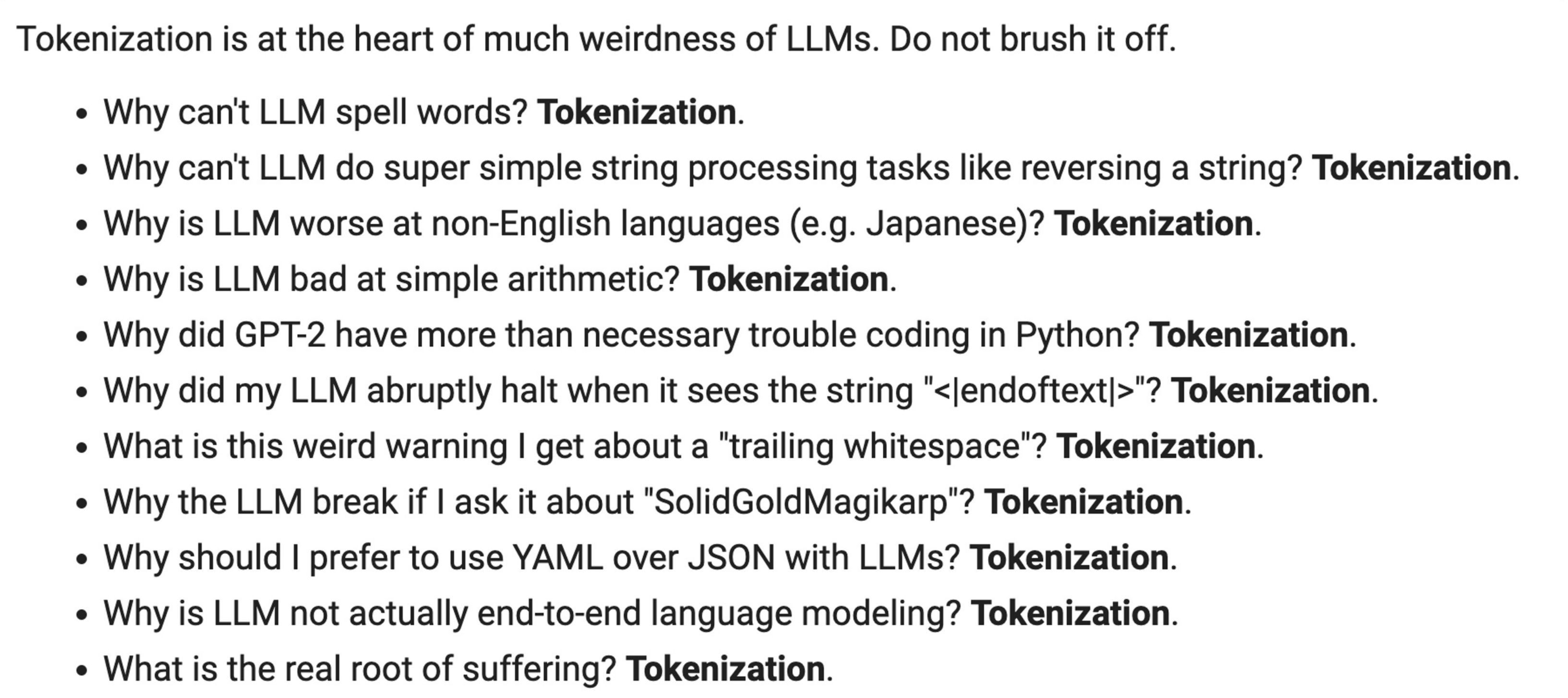




LLM APIs charge per token

[[2305.13707] Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models]

Tokenization



Tokenization is at the heart of much weirdness of LLMs. Do not brush it off.

- Why can't LLM spell words? **Tokenization**.
- Why can't LLM do super simple string processing tasks like reversing a string? **Tokenization**.
- Why is LLM worse at non-English languages (e.g. Japanese)? **Tokenization**.
- Why is LLM bad at simple arithmetic? **Tokenization**.
- Why did GPT-2 have more than necessary trouble coding in Python? **Tokenization**.
- Why did my LLM abruptly halt when it sees the string "<|endoftext|>"? **Tokenization**.
- What is this weird warning I get about a "trailing whitespace"? **Tokenization**.
- Why the LLM break if I ask it about "SolidGoldMagikarp"? **Tokenization**.
- Why should I prefer to use YAML over JSON with LLMs? **Tokenization**.
- Why is LLM not actually end-to-end language modeling? **Tokenization**.
- What is the real root of suffering? **Tokenization**.

[`Let's build the GPT Tokenizer’ by Andrej Karpathy]

Cody Blakeney
@code_star

Do you guys ever think about tokenizers?

Prithviraj (Raj) Ammanabrolu @rajammanabrolu · Sep 26, 2023

If there's one thing standing in the way of AGI, it's tokenizers

Luca Soldaini 🎀 @soldni · Aug 21

Recent tweaks to subword tokenizers

- Tokenize each digit separately (i.e. no merge on digits)

- Add special tokens to deal with edge cases:
  - E.g. special tokens for keywords from programming languages

- Train the tokenizer on a more balanced dataset.
  - Apply tricks like “alpha-sampling” – up sample lower resource languages and scripts to up their frequency.
  - Does not solve all issues but helps.

Masked Language Models

Reminder: Different senses, but same embedding 👎

- *The Amazon **Basin** is home to the largest rainforest on Earth.*
- *She filled the **basin** with water to wash the dishes.*
- *The neurosurgeon examined the cranial **basin** for signs of trauma.*

word2vec, GloVe, or other **static embeddings** assign exactly the same embedding to each of these occurrences of the word "basin" despite their different senses

Our next goal:
Assign different, **contextualized embeddings** based on the surrounding context

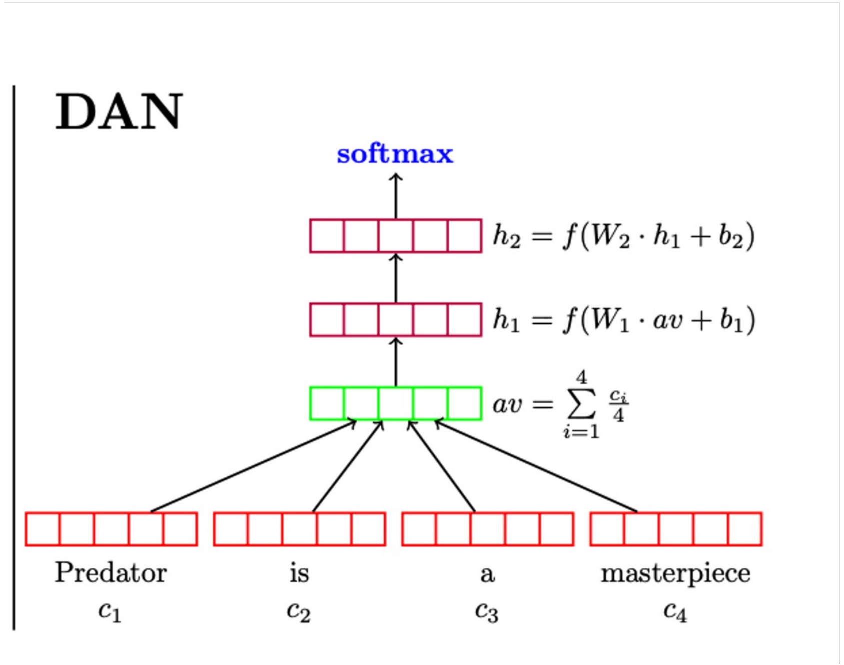
The final transformer encoder representation of each token is highly contextualized

Reminder: Token order & long-range dependencies with DAN 👎

Transformer considers token order with positional embeddings

Due to multi-headed self-attention & cross-attention, we are able to model long-range dependencies

Deep Averaging Network (DAN)


DAN
softmax
$h_2 = f(W_2 \cdot h_1 + b_2)$
$h_1 = f(W_1 \cdot av + b_1)$
$av = \sum_{i=1}^{4} \frac{c_i}{4}$
Predator c_1
is c_2
a c_3
masterpiece c_4

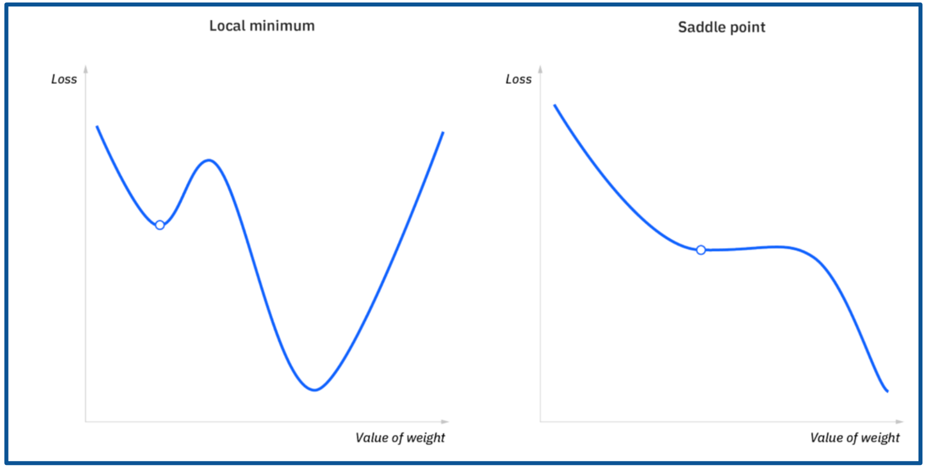
[Iyyer et al., 2015]

Reminder: Non-convex optimization, so initialization matters

Instead of starting from randomly initialized weights, we are going to make use of **large corpora** to **pretrain a model** and find weights that are a much better **starting point for all NLP tasks** than random weights

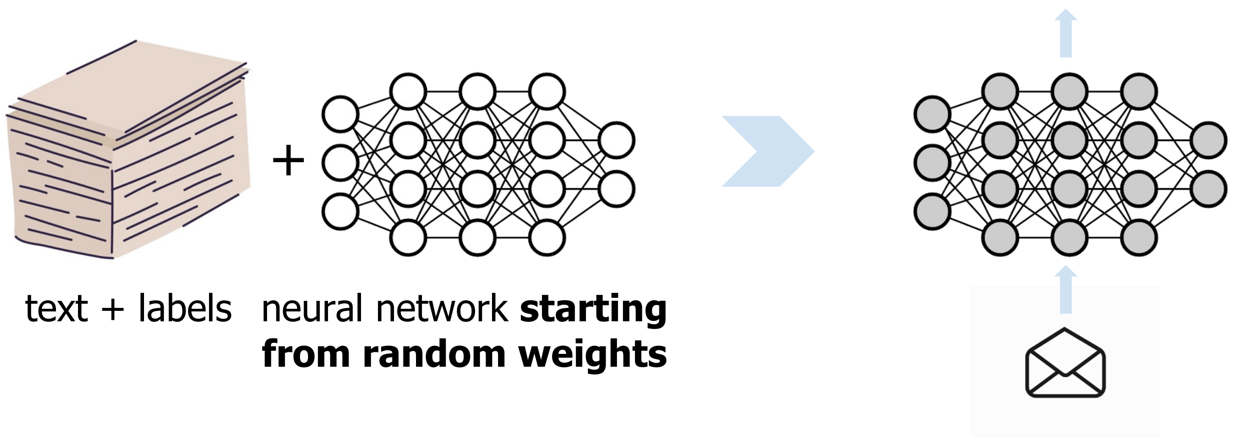
We are going to talk about this next!

Randomly initialized transformer still has this issue



Source: https://www.ibm.com/topics/gradient-descent

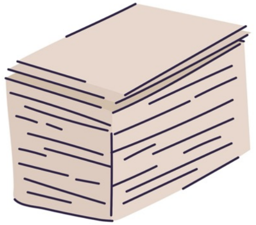
Standard Supervised Deep Learning



Pretrain-then-Finetune (also called post-train)

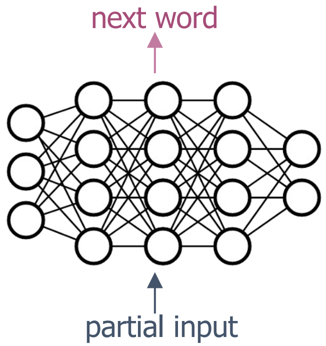
A form of transfer learning

Stage 1:
Pretrain a model



text

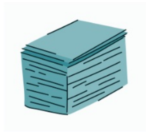
+



neural network **starting from <u>random</u> weights**

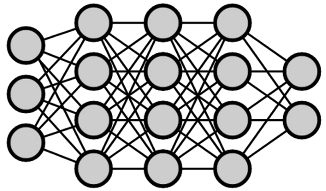
Objective: generate next or masked word
(does not require that people label the next word)

Stage 2:
Finetune the model



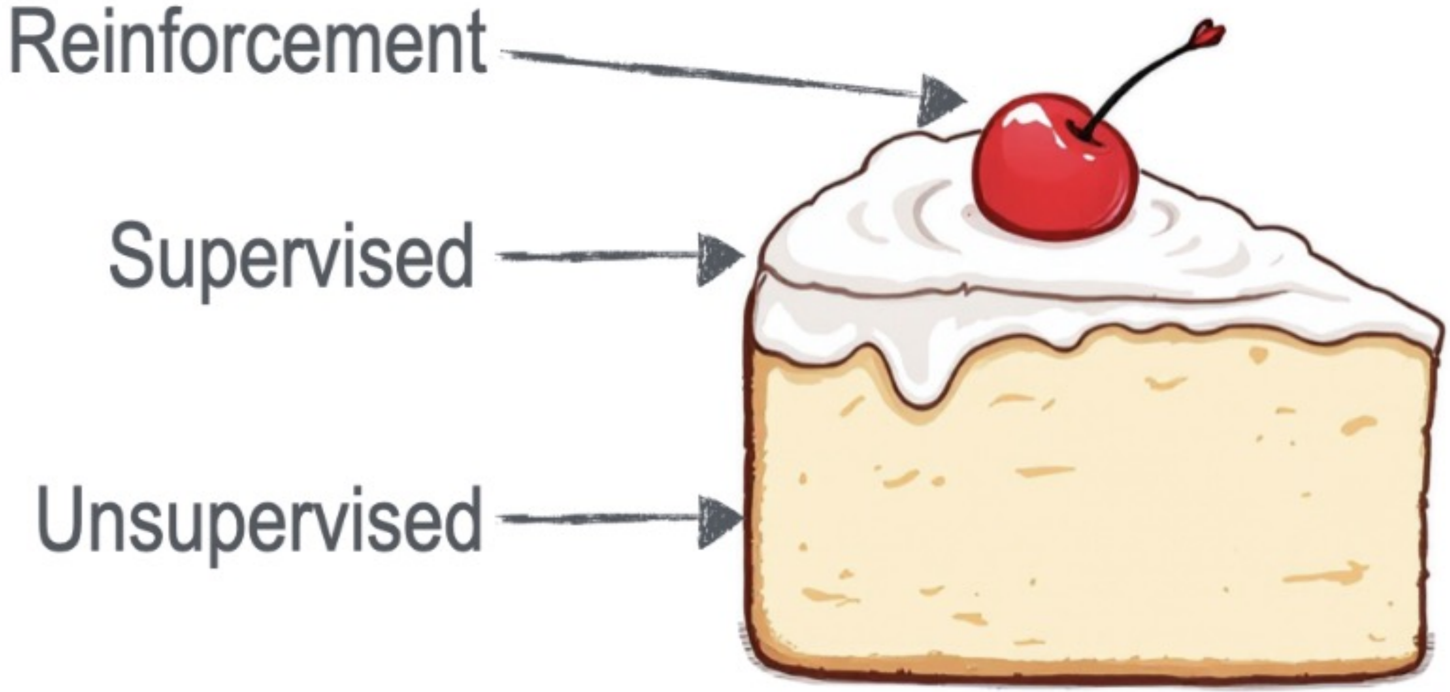
text + **labels**

+



neural network **starting from <u>pretrained</u> weights**

Objective: standard supervised training



arxiv.org/pdf/2502.03038

LM pretraining

- LMs so far: predict the next token given the previous tokens also sometimes called “Causal Language Modeling”.

- This enables a self-supervised task – train on only raw text (similar to word2vec).

- And get really interesting and useful representations.
    - With many usecases

Masked LMs

- LMs so far: predict the next token given the previous tokens
- Since we have a complete sentence?
  - So: can we incorporate future context to learn better representations
- How can we formulate a self-supervised prediction task?

Masked LMs

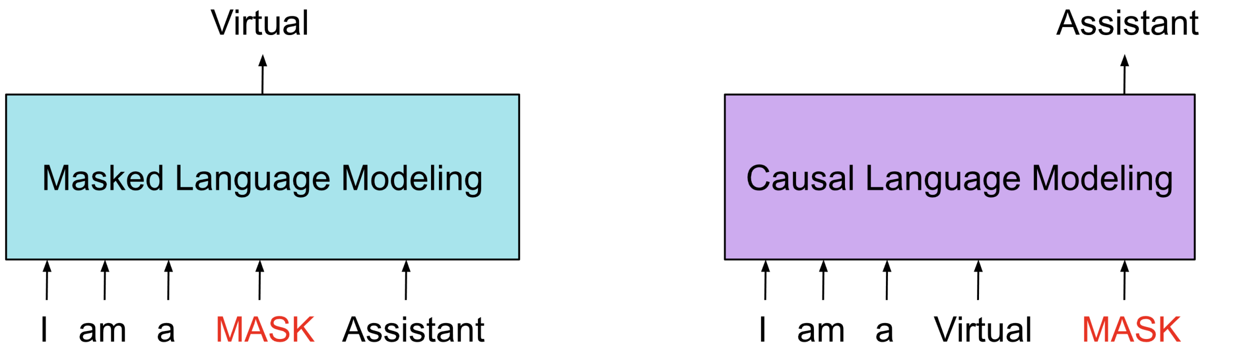
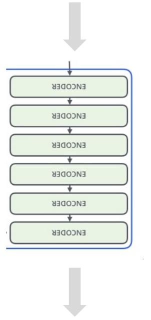


Image from https://www.holisticai.com/blog/from-transformer-architecture-to-prompt-engineering

Masked Language Modeling (MLM)

(text) Sylvester Stallone has made some **terrible** films in his lifetime, but this has got to be one of the **worst**. A totally **dull** story...

(masked text) Sylvester Stallone has made some **<mask>** films in his lifetime, but this has got to be one of the **<mask>**. A totally **<mask>** story...



(output final layer) vector(Sylvester) ... vector(**<mask>**) vector(films) vector(in) vector(his) ... vector(the) vector(**<mask>**) ... vector(**<mask>**) vector(story)...

Masked Language Modeling (MLM)

(text) Sylvester Stallone has made some **terrible** films in his lifetime, but this has got to be one of the **worst**. A totally **dull** story...

(masked text) Sylvester Stallone has made some **<mask>** films in his lifetime, but this has got to be one of the **<mask>**. A totally **<mask>** story...

ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER

(output final layer) vector(Sylvester) ... vector(**<mask>**) vector(films) vector(in) vector(his) ... vector(the) vector(**<mask>**) ... vector(**<mask>**) vector(story)...

loss(predicted_word,“terrible”)

Masked Language Modeling (MLM)

(text) Sylvester Stallone has made some **terrible** films in his lifetime, but this has got to be one of the **worst**. A totally **dull** story...

(masked text) Sylvester Stallone has made some **<mask>** films in his lifetime, but this has got to be one of the **<mask>**. A totally **<mask>** story...

ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER

(output final layer) vector(Sylvester) ... vector(**<mask>**) vector(films) vector(in) vector(his) ... vector(the) vector(**<mask>**) ... vector(**<mask>**) vector(story)...

loss(predicted_word,"dull")

loss(predicted_word,"worst")

loss(predicted_word,"terrible")

average these losses and do a gradient descent step

Masked LM: Only using Encoder transformer

- Encoders assume we have the complete sequence

- Self-attention computes weighted sum over entire context (i.e., entire sequence)

- There is no generation problem, we just want representations
  - We will learn how to use them later on

BERT

Bidirectional Encoder Representations from Transformers

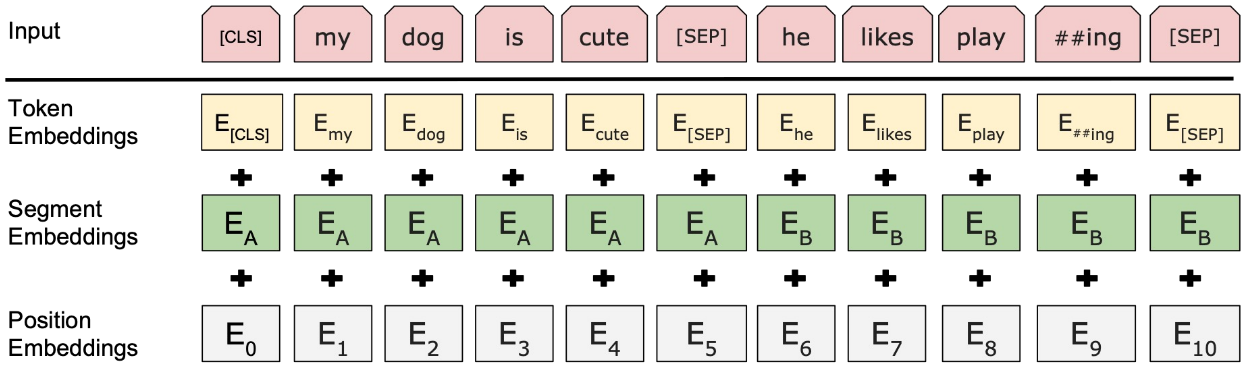
- Encoder transformer
- BERT Base: 12 transformer blocks, 768-dim word-piece tokens, 12 self-attention heads $\rightarrow$ 110M parameters
- BERT Large: 24 transformer blocks, 1024-dim word-piece tokens, 16 self-attention heads $\rightarrow$ 340M parameters
- 100's of variants since BERT came out.

[Devlin et al. 2018]

BERT

Inputs

- One or two sentences
  - Word-piece token embeddings
  - Position and segment embeddings



[figure from Devlin et al. 2018]

BERT
Training

- Data: raw text
- Two objectives:
  - Masked LM
  - Next-sentence prediction
- Later development in “RoBERTa”:
  - More data, no next-sentence prediction, dynamic masking

BERT

Masking Recipe for Training

- Randomly mask and predict 15% of the tokens
    - For 80% (of these 15%) replace the input token with a special token [MASK]
    - For 10% replace with a random token from the vocabulary
    - For 10% keep the same (input and output are the same, trivial task)

BERT

Next-sentence Prediction

- Input: [CLS] Text chunk 1 [SEP] Text chunk 2
- Training data: 50% of the time, take the true next chunk of text, 50% of the time take a random other chunk
- Predict whether the next chunk is the true next chunk
- Prediction is done on the [CLS] output representation

BERT

Related Techniques

- Central Word Prediction Objective (context2vec) [Melamud et al. 2016]
- Machine Translation Objective (CoVe) [McMann et al. 2017]
- Bi-directional Language Modeling Objective (ELMo) [Peters et al. 2018]
- Then BERT came ...
- ... and many more followed (the whole sesame street arrived)

BERT

What Do We Get?

- We can feed complete sentences to BERT
- For each token, we get a contextualized representation
    - Meaning: computed taking the other tokens in the sentence into acocunt
- In contrast to word2vec representations that fixed and do not depend on context
- While word2vec vectors are forced to mix multiple senses, BERT can provide more instance-specific vectors

BERT

How Do We Use It?

- Widely supported by existing frameworks
  - E.g., Transformers library by Hugging Face
- We will soon see how to use it when working with annotated data
- Large BERT models quickly outperformed human performance on several NLP tasks
  - But what it meant beyond benchmarking was less clear
- Started an arms race towards bigger and bigger models, which quickly led to the LLMs of today

BERT

What It Is Not Great For?

- primarily used for discriminative tasks, Cannot generate text
  - Can manipulate in weird ways to generate text but not the original purpose of this model