

Natural Language Processing (NLP) and Large Language Models (LLMs)

Lecture 4: Tokenization, subword embeddings, and evaluation of word embeddings

Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

WISE @ XMU

2025 年 3 月 18 日

Recap

- Deep learning; backpropagation; CNN
- Tutorials on pytorch:<https://pytorch.org/tutorials/>.

- ① Section 1: Subword and tokens
- ② Section 2: Tokenization
- ③ Section 3: Evaluate the token embeddings

① Section 1: Subword and tokens

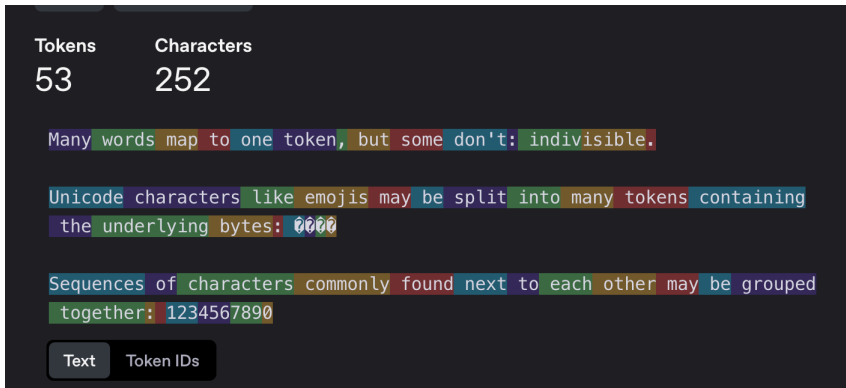
② Section 2: Tokenization

③ Section 3: Evaluate the token embeddings

Subword and token


- A **subword** (e.g., “whe”) is a unit of text smaller than a word (“where”) but larger than a character (“w”).
- A **token** is either a word or a subword.
- In NLP, a **tokenizer** (分词器) converts text into tokens; this process is called **tokenization**.
- Simplest English tokenizer: split by spaces using `yourtext.split(' ')`. Does not work for Chinese.

Tokenizer of GPT-4o



You can try it from <https://platform.openai.com/tokenizer>

Token details of DeepSeek

 DeepSeek API Docs

English ▾ DeepSeek Platform [

Quick Start ▾
Your First API Call
Models & Pricing
The Temperature Parameter
Token & Token Usage
Rate Limit
Error Codes
News ▾
DeepSeek-R1 Release
2025/01/20
DeepSeek APP 2025/01/15
Introducing DeepSeek-V3
2024/12/26
DeepSeek-V2.5-1210 Release
2024/12/10

🏠 > Quick Start > Token & Token Usage

Calculate token usage offline

Token & Token Usage

Tokens are the basic units used by models to represent natural language text, and also the units we use for billing. They can be intuitively understood as 'characters' or 'words'. Typically, a Chinese word, an English word, a number, or a symbol is counted as a token.

Generally, the conversion ratio between tokens in the model and the number of characters is approximately as following:

- 1 English character \approx 0.3 token.
- 1 Chinese character \approx 0.6 token.

However, due to the different tokenization methods used by different models, the conversion ratios can vary. The actual number of tokens processed each time is based on the model's return, which you can view from the usage results.

Why do we need subwords?

- Capture shared roots or etyma (e.g., “dog” and “dogs”).
- Generate words unseen in vocabulary (e.g., “pre” + “sent” = “present”).
- Potentially reduce vocabulary size (an extreme case: all English text from 26 letters, punctuation, and spaces).
- Facilitate cross-lingual transfer (e.g., shared Latin roots such as “e.g.”).
- Robustness to typos and minor variations.

What is a good tokenizer

- Break a word to too **many** tokens: it is hard to model a word or a sequence (e.g., “where” to “w”, “h”, “e”, “r”, “e”).
- Too Few Tokens: the model misses opportunities for parameter sharing and generalization. (e.g., “dog” and “dogs”)
- Linguistically meaningful: “unhappiness” to “un”, “happy”, “ness”
- Statistically meaningful: tokens should appear frequently enough in the training data.

BERT tokenizer

- Install transformers
- BERT-tokenizer.ipynb
- BERT is not good at Chinese

GPT-2 tokenizer

- GPT2-tokenizer.ipynb
- Ġ represents the “space”.

Deepseek tokenizer

- deepseek-tokenizer.ipynb
- Download it from https://api-docs.deepseek.com/quick_start/token_usage.
- The downloaded tokenizer is garbled for other languages besides English.

Challenges of tokenization in Chinese

- Unlike English, we should not just split a Chinese sentence based on the space (Chinese characters will be split one by one) why?
- Different meanings of one character: “打水”, “打人”, “打篮球”
- Different meanings of different tokenization: “海贼王路飞”

NLP tools for Chinese

- HanLP;THULAC;
- HanLP tokenizer: `hanlp-tokenizer.ipynb`

The fastText model

- The fastText model is a subword analogue of the skip-gram model, used to **compute subword embeddings**.
- Reading material: Enriching Word Vectors with Subword Information, Bojanowski et al., 2017.
- For a center word (e.g., “where”), consider all its subwords of length 3–6.
- Use “<” and “>” as boundary symbols to distinguish prefixes and suffixes clearly.
- Example: all subwords of length 3 for “<where>” are {“<wh”, “whe”, “her”, “ere”, “re>”}.

Computing Subword Embeddings Using Skip-Gram

- For a **center word** c and a context word o , denote their embeddings by \mathbf{v}_c and \mathbf{u}_o , respectively.
- Recall that the skip-gram model learns embeddings by maximizing the likelihood:

$$p(o|c) = \frac{\exp(\mathbf{u}_o^T \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^T \mathbf{v}_c)}$$

Computing Subword Embeddings Using Skip-Gram

- Define \mathcal{G}_c and \mathcal{G}_o as the sets of all subwords (length 3–6) of words c and o , respectively.
- Represent embeddings as sums of subword embeddings:

$$\mathbf{v}_c = \sum_{g \in \mathcal{G}_c} \mathbf{v}_g, \quad \mathbf{v}_g \text{ is the embedding of subword } g,$$

$$\mathbf{u}_o = \sum_{h \in \mathcal{G}_o} \mathbf{u}_h, \quad \mathbf{u}_h \text{ is the embedding of subword } h.$$

- Optimize subword embeddings \mathbf{v}_g and \mathbf{u}_h by maximizing likelihood via backpropagation.

Weaknesses of the fastText model

- Compared to the skip-gram model, fastText has a larger vocabulary due to subwords, resulting in more parameters.
- Computing a word's embedding requires summing all its subword embeddings, increasing computational complexity.
- The vocabulary size cannot be explicitly predefined, as it depends on the extracted subwords of specified lengths (e.g., 3–6 characters).

① Section 1: Subword and tokens

② Section 2: Tokenization

③ Section 3: Evaluate the token embeddings

Byte pair encoding (BPE, Sennrich et al, 2015)

- Input: a training corpus; Output: a vocabulary and a tokenizer
- Initialize the vocabulary \mathcal{V} as the set of all characters.
- Find a pair $w, w' \in \mathcal{V}$ that co-occur the most times and generate a new symbol ww'
- Add ww' to \mathcal{V} .

Example

- Corpus: the dog, the fog, the fox
- Initialize $\mathcal{V} = \{t, h, e, \sqcup, d, o, g, f, x\}$
- Tokenizer: $[t, h, e, \sqcup, d, o, g], [t, h, e, \sqcup, f, o, g], [t, h, e, \sqcup, f, o, x]$

Update the tokenizer and the vocabulary

- Tokenizer:
 - [t,h,e,▯,d,o,g], [t,h,e,▯,f,o,g], [t,h,e,▯,f,o,x]
 - [th,e,▯,d,o,g], [th,e,▯,f,o,g], [th,e,▯,f,o,x] (“th” occurs 3 times)
 - [the,▯,d,o,g], [the,▯,f,o,g], [the,▯,f,o,x] (“the” occurs 3 times)
 - [the,▯,d,og], [the,▯,f,og], [the,▯,f,o,x] (“og” occurs twice)
- Vocabulary: $\mathcal{V} = \{t, h, e, _, d, o, g, f, x, th, the, og\}$

Apply the tokenizer

- New corpus: [the log]
- Apply the tokenizer:
 - [t, h, e, _▯, l, o, g]
 - [th, e, _▯, l, o, g]
 - [the, _▯, l, o, g]
 - [the, _▯, l, og]

Run BPE on bytes



- There is a huge amount (144,697) of unicode characters (such as a letter or a Chinese character) especially in the multilingual setting.
- Applying BPE on all characters is time-consuming and we can hardly see all characters in the training data.
- Usually we apply BPE on **bytes** from unicode
- Chinese characteristic to unicode bytes: 厦门大学 = [u53,a6,u95,e8,u5,927,u5,b66]
- There is a much smaller size of bytes (such as 256).

The Unigram Model

- Instead of splitting tokens solely based on frequency, the probability of each token can be updated using machine learning-based methods.
- **Reading Material:** *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates* by Taku Kudo, 2018.

Probabilistic models of tokenization (frequency)

- Training data: "ababc"
- Vocabulary and frequency:

token	# of occurrence	frequency
a	2	2/12
b	2	2/12
c	1	1/12
ab	2	2/12
ba	1	1/12
bc	1	1/12
aba	1	1/12
bab	1	1/12
abc	1	1/12

- For a new sentence "abc", calculate the probability of tokenization (e.g. $\text{Prob}(\text{"a"}, \text{"b"}, \text{"c"}) = 2/12 \times 2/12 \times 1/12$)

Probabilistic models of tokenization

- For a sentence S , let $\mathbf{x} = (x_1, \dots, x_L)$ be its tokenization.
- In the previous example, $x_1 = \text{"a"} , x_2 = \text{"b"} , x_3 = \text{"c"} .$
- Beyond simple frequency counting, we can model the probability of tokenization using:

$$p_{\theta}(\mathbf{x} \mid S) = \prod_{i=1}^L p_{\theta}(x_i),$$

where θ parameterizes the model.

- With a **pre-defined vocabulary** \mathcal{V} , $\sum_{v \in \mathcal{V}} p_{\theta}(v) = 1$.
- p_{θ} could be a (recurrent) neural network (coming soon...) or a CNN.



Training

- Let $\mathcal{T}(S)$ denote the set of all possible tokenizations of a sentence S .
- Given a training corpus containing n sentences S_1, \dots, S_n , the log-likelihood is defined as:

$$\mathcal{L}(\theta) = \sum_{i=1}^n \log \left(\sum_{\mathbf{x} \in \mathcal{T}(S_i)} p_{\theta}(\mathbf{x} \mid S_i) \right).$$

- The conditional probability of x_i is not observable; use the **EM algorithm**.

EM algorithm

- The Expectation–Maximization (EM) algorithm is widely used for computing the maximum likelihood estimates of models with unobserved latent variables.
- Observed variable: s ; Latent (unobserved) variable: z
- Main idea: $p(s) = \mathbb{E}_z[p(s, z)]$
- The log-likelihood is expressed as:

$$\mathbb{E}_z[\log p(s, z)]$$

EM algorithm

- Expectation step: given the current estimate $\theta^{(t)}$, calculate

$$q(\theta|\theta^{(t)}) = \mathbb{E}_{\mathbf{z} \sim p_{\theta^{(t)}}(\cdot|\mathbf{s})} [\log p_{\theta}(\mathbf{s}, \mathbf{z})]$$

- Maximization step: update

$$\theta^{(t+1)} = \operatorname{argmax}_{\theta} q(\theta|\theta^{(t)})$$

Apply the tokenizer

- Given a sentence, a probability $p_{\theta}(\mathbf{x}|S)$ is trained.
- Find \mathbf{x}^* that maximizes $p_{\theta}(\mathbf{x}|S)$
- Use the Viterbi algorithm to maximize it.
- The Viterbi algorithm is a dynamic programming algorithm for obtaining the **maximum a posteriori probability estimate**.

The vocabulary may not be predefined

- Start with a “reasonably big” seed vocabulary \mathcal{V} .
- Given \mathcal{V} , optimize p_θ and the tokenization (find \mathbf{x}^*)
- For each token $x \in \mathcal{V}$, calculate $-\text{LogLikelihood}(x)$ (the loss of x).
- Sort by loss and keep the top 80% tokens in the vocabulary.

① Section 1: Subword and tokens

② Section 2: Tokenization

③ Section 3: Evaluate the token embeddings

- So far, we have discussed tokenization, which splits sentences into tokens.
- Each token represents a word or subword, and its embedding (universally referred to as a token embedding) can be computed using methods such as GloVe.
- How can we evaluate the quality of these embeddings?



Extrinsic Evaluation

- Consider a realistic QA task: Question \Rightarrow Tokenization and Embedding Calculation \Rightarrow Deep Learning System \Rightarrow Answer.
- Obtaining a good answer requires high-quality embeddings.
- **Extrinsic evaluation** assesses embeddings based on their performance in real-world tasks.
 - Evaluation can be slow due to the deep learning system.
 - Difficult to pinpoint issues: embeddings or other sub-systems? The system can act as a complicated black-box, with only answers observable.

Intrinsic Evaluation

- Evaluation on a specific intermediate task (e.g., analogy completion).
- Fast computation of performance; helps to understand the subsystem.
- Does not directly indicate usefulness in real-world tasks.

Analogy completion

- Semantic analogies:
 - Man \Rightarrow King; Woman \Rightarrow ?
 - State of Georgia \Rightarrow Atlanta; State of New York \Rightarrow ?
- Grammatical analogies: run \Rightarrow running; eat \Rightarrow ?
- Synonyms and antonyms: hot \Rightarrow cold; light \Rightarrow ?

Mathematical modeling

- We may simply expect

$$\textit{Embedding}(\text{King}) - \textit{Embedding}(\text{Man}) \approx \textit{Embedding}(?) - \textit{Embedding}(\text{Woman})$$

- Or equivalently:

$$\textit{Embedding}(\text{King}) - \textit{Embedding}(\text{Man}) + \textit{Embedding}(\text{Woman}) \approx \textit{Embedding}(?)$$

Mathematical modeling

- For an analogy completion problem involving 3 words a, b, c

$$a : b :: c : ?,$$

and let $\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c$ be the embeddings of a, b, c , correspondingly.

- The predicted word d is such that its embedding \mathbf{x}_d **maximizes the similarity**

$$\cos(\mathbf{x}_b - \mathbf{x}_a + \mathbf{x}_c, \mathbf{x}) = \frac{(\mathbf{x}_b - \mathbf{x}_a + \mathbf{x}_c)^T \mathbf{x}}{\|\mathbf{x}_b - \mathbf{x}_a + \mathbf{x}_c\| \|\mathbf{x}\|}$$

among all \mathbf{x} in the vocabulary.

Glove models

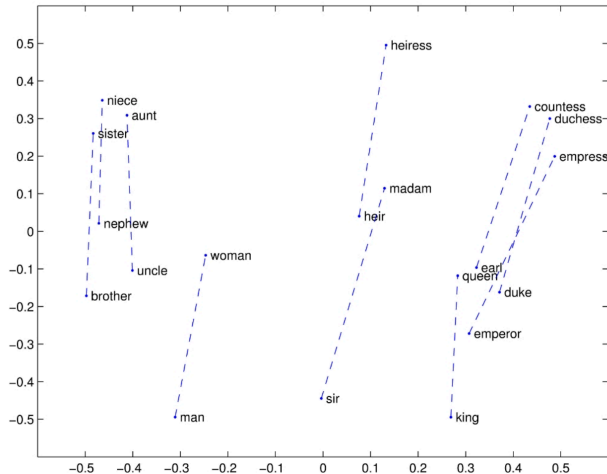


Figure is from CS224n

Other intrinsic evaluation: Meaning similarity

- Word vector distances and their correlation with human judgments

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Figure is from CS224n

Extrinsic word vector evaluation

- Such as the mis-classification error of NER.

Assignment 2 (Partial: 5%?, TBD)

- Implement and train a skip-gram model with negative sampling using PyTorch (or alternative deep learning frameworks such as TensorFlow).
- Do **not** use pre-existing word2vec libraries.
- LLM tools are permitted; however, **I am pretty sure they are not entirely accurate for this task at this stage**. Buggy code will be punished.
- Submit a .ipynb file with detailed comments