Byte-Pair Encoding (BPE) & Tokenization:

Comprehensive Notes — Data Preparation & Tokenization

Part I — Foundations (Course Walkthrough)

Koorosh Asil Ghrehbaghi
K.N. Toosi University of Technology (KNTU) — B.Sc. Computer Science
Instructor: Dr. Maryam Abdolali

Compiled on: September 7, 2025

Abstract

These notes summarize and explain principles of corpus-based data preparation and tokenization used in modern natural language processing, with a strong focus on the Byte-Pair Encoding (BPE) subword algorithm. The presentation includes precise definitions, worked examples, pseudocode, practical Unix commands, mathematical background (Heaps' / Herdan's law), design choices (why characters, words or subwords), and common uses and caveats. This document is intended as a study-friendly **notebook** and reference for the unit:

Part I – Foundations: Data Preparation & Tokenization.

Syllabus position: This note covers *Data Preparation & Tokenization* (the first section of Part I). Future units (not covered here) include N-Gram models, embeddings, sequence models, and transformers.

Contents

1 Overview

Natural language data must be prepared before algorithmic processing. Preparation includes normalization, tokenization, sentence segmentation, and optional lemma/stem processing. Tokenization is the process of mapping raw text into a sequence of atomic units (tokens) that the NLP model will see. The choice of token unit (characters, words, subwords) affects model capacity, vocabulary size, and handling of unseen words.

2 Key Definitions

Definition

- **Corpus / Corpora:** A machine-readable collection of text or speech samples used for training, evaluation, or analysis.
- **Token:** A single instance of a unit in running text (e.g., an occurrence of a word or punctuation).
- **Type:** A distinct token form (the vocabulary item). The vocabulary size is the number of types.
- **Vocabulary (V):** The set of all types (tokens) a model knows and can output.
- **Lemma:** The base or dictionary form of a set of wordforms that share the same meaning (e.g., `run, runs, ran` $\rightarrow$ lemma `run`).
- **Stemming:** A heuristic process for conflating wordforms by removing affixes (may produce non-words).
- **Tokenization:** The segmentation of raw text into tokens (words, subwords, or characters).
- **Subword:** A token smaller than a whole word, often a morpheme (meaning-bearing unit) or a frequent substring. Subwords allow representing unseen words as a sequence of known pieces.
- **Disfluency:** Filler words or fragments in spoken language (e.g., *uh*, *um*, broken words). Treatment depends on task.

3 Important Observations & Use Cases

Importance & Use Cases

- Models trained only on one language variety (e.g., standard written English) may perform poorly on dialectal text (e.g., AAL) or code-switched text. Data diversity matters.
- Tokenization choice is task-dependent: information retrieval often benefits from case-folding; named-entity recognition and machine translation may require preserving case.

- Subword tokenization (like BPE) is a practical compromise between full-word vocabularies (which explode in size) and character models (which may require very deep models to learn meaning).
- Heaps' Law quantifies how vocabulary grows with corpus size; hence token budget and number of merges in BPE should be chosen with corpus scale in mind.

4 Corpus Statistics and Heaps' (Herdan's) Law

Let N be the number of tokens (running words) in a corpus, and $|V|$ be the number of types (vocabulary size). Empirically,

$$|V| \;=\; k\,N^{\beta}, \qquad 0 < \beta < 1,$$

where k, β are constants depending on genre and language. Typical β for large English corpora is ≈ 0.67–0.75.

Example

Rough numbers (illustrative):

Corpus	Tokens (N)	Types ($\|V\|$)
Brown corpus	1×10^6	38,000
COCA	4.4×10^8	2,000,000
Google N-grams	1×10^{12}	13,000,000 (thresholded)

5 Practical Text Normalization

Text normalization frequently includes:

- Case-folding (optional)
- Unicode normalization (NFC/NFKC)
- Normalizing punctuation and whitespace
- Expanding or handling contractions/clitics (e.g., `we're` $\rightarrow$ `we are` if desired)
- Special-token detection: URLs, emails, hashtags, currencies, dates

Example

UNIX quick token frequency (naive):

```
# one-line pipeline (Linux / Unix)
tr -sc 'A-Za-z' '\n' < corpus.txt | tr A-Z a-z | sort | uniq -c | sort -n -r
```

This yields approximate word counts (lowercased, punctuation removed).

6 Lemmatization, Normalization, and Standardization

Definition

- **Word Normalization:** The process of converting multiple surface variants of expressions into a canonical form to improve consistency across texts (e.g., `USA` → `US`; `e-mail` → `email`).
- **Case Folding:** Converting all text to a single case (usually lower case) so that case differences do not create distinct tokens (e.g., `Washington` and `washington` become identical).
- **Unicode Normalization:** Converting text to a canonical Unicode representation (NFC/NFKC) to ensure that visually identical characters use the same code points.
- **Lemmatization:** The process of mapping wordforms to their dictionary (lemma) form using morphological analysis and part-of-speech (POS) information where necessary (e.g., `am`, `are`, `is` → `be`; `children` → `child`).
- **Stemming:** A rule-based or heuristics-driven reduction of words to a root form by chopping affixes (e.g., Porter stemmer: `running` → `run`; `happiness` → `happi`). Stemming may produce non-words.
- **Standardization:** Broader normalization that includes formatting of dates, numbers, phone numbers, currency, and domain-specific canonicalization (e.g., `01/02/06` → `2006-02-01` in ISO format).

Importance & Use Cases

Normalization and lemmatization are task-sensitive choices:

- For search and information retrieval, aggressive normalization (case folding, lemma mapping) often improves recall.
- For named-entity recognition or sentiment analysis, preserving case and some punctuation can be crucial for performance.
- Lemmatization requires accurate POS tagging for best results; incorrect POS tags can produce wrong lemmas.
- Standardization (dates, numbers) is essential for tasks that rely on structured numeric or temporal information (e.g., clinical text, financial reports).

6.1 Examples and Illustrations

Example

Case folding:

- Original: `Apple releases iOS; apple stock rises.`
- After case folding: `apple releases ios; apple stock rises.`

- Note: Downside — loses distinction between the company `Apple` and the fruit `apple` if case was informative.

Example

Unicode normalization:

- Pre-normalized: the same glyph can be encoded in different Unicode forms. For example, the character `é` may appear either as the single composed code point `U+00E9` ("composed"), or as the two-code-point decomposed sequence `U+0065 U+0301` ("decomposed": the letter `e` plus a combining acute accent).
- After NFC normalization: both forms become the same composed code point `U+00E9`, removing spurious token differences.

Example

Contraction expansion / clitics:

- Original: `We're going to New York.`
- After expansion: `We are going to New York.`
- Use-case: Some pipelines expand contractions to facilitate parsing or downstream matching.

Example

Lemmatization vs Stemming:

- Wordforms: `am, are, is` $\xrightarrow{\text{lemmatize}}$ `be`
- Wordforms: `running` $\xrightarrow{\text{lemmatize}}$ `run` (requires POS)
- Using Porter stemmer: `running` $\xrightarrow{\text{stem}}$ `run`
- Porter stemmer on `happiness` $\xrightarrow{\text{stem}}$ `happi` (non-lexical)

Example

Standardization examples (dates, numbers, currencies):

- Dates: `01/02/06` → `2006-02-01` (ISO 8601) when domain requires unambiguous ordering.
- Numbers: `555,500.50 (en)` vs `555 500,50 (fr)` → `555500.50` normalized numeric value.
- Currencies: `$45.55` → `45.55 USD` (explicit currency code).

6.2 Pipeline Recommendations

Importance & Use Cases

A recommended order for a conservative, task-aware preprocessing pipeline:

1. Unicode normalization (NFC/NFKC).
2. Tokenization (preserve tokens for URLs, emails, user handles).
3. Special-token handling and standardization (dates, currencies).
4. Optional case-folding (only if task permits).
5. POS tagging (if performing lemmatization).
6. Lemmatization (use POS information).
7. Subword tokenization (BPE/WordPiece) after normalization/lemmatization decisions are finalized.

Note: Order can vary depending on the goal; for example, if BPE is applied before lemmatization in some systems, lemmatization may be performed on de-tokenized text later for specific tasks.

7 Tokenization: Choices and Trade-offs

Token units may be:

1. **Characters:** Minimal units. Works for any input but produces long sequences.
2. **Words:** Intuitive but leads to huge vocabularies and out-of-vocabulary (OOV) problems.
3. **Subwords (BPE, WordPiece, Unigram):** Balanced approach; widely used for modern deep models.

Importance & Use Cases

Tokenization must handle punctuation, numbers, clitics, and multiword expressions. For Chinese, character-level tokenization is often preferred; for Japanese/Thai, learning segmentation is necessary.

8 Byte-Pair Encoding (BPE) — Full Explanation

8.1 Goal

BPE is an algorithm to learn a vocabulary of subword units from a training corpus so that:

- Frequent words become single tokens (compact representation).

- Rare/unseen words are represented as sequences of known subwords or characters.
- Vocabulary size is controlled by the number of merges k.

8.2 Algorithm (Token Learner)

We describe the token-learner portion of BPE. Start with corpus represented as sequences of characters (optionally word-boundaries marked).

Listing 1: BPE token-learner (pseudocode)

```
# Input: corpus C (list of token sequences), number of merges k
# Output: vocabulary V (initial chars + merged tokens)
V = set(all characters appearing in C)
for i in range(k):
    # Count adjacent pairs of tokens in C
    pair_freq = count_adjacent_pairs(C)
    (tL, tR) = most_frequent_pair(pair_freq)
    new_token = tL + tR # merge into one token
    V.add(new_token)
    # Replace all occurrences of the pair in C with new_token
    C = replace_pair_with_token(C, tL, tR, new_token)
return V
```

8.3 Token Segmentation (Test-time)

Given a learned vocabulary of tokens and a new word, segmentation proceeds greedily:

1. Start with characters of the new word.
2. Iteratively apply merges (in the same order learned during training) to join adjacent tokens when the merged token is in the vocabulary.
3. The final segmentation is the token sequence representing the word.

Example

If training learned merges: `(s + u) -> su`, `(su + n) -> sun`, `(flower) unchanged`, then test word `sunflower` segments as `sun flower` if both `sun` and `flower` are in V.

8.4 Important Implementation Notes

- Vocabulary grows additively: earlier merges remain available for reuse.
- Each merge adds exactly one token, so after k merges, vocabulary increases by k tokens (plus initial tokens).
- Common practical vocabulary sizes: 10k, 30k, 50k depending on model and corpus size.

9 Worked Example: Step-by-step BPE

Consider the small training corpus (space-separated words for clarity):

`low new newer lower`

We show a small sequence of merges.

Initial (chars)	After merges (illustrative)
l o w	merge `l + o` → `lo`
n e w	merge `n + e` → `ne`
n e w e r	merge `ne + w` → `new`
l o w e r	merge `lo + w` → `low` then `low + er` → `lower`

Example

Resulting vocabulary after illustrative merges might include:

$$\{\texttt{l, o, w, n, e, r, lo, ne, new, low, er, lower}\}$$

If the model later sees the test word `lowers`, it can tokenize as `lower` + `s`.

10 Representative Unix & Python Snippets

Example

Naive token frequency with Unix (already shown):

```
tr -sc 'A-Za-z' '\n' < corpus.txt | tr A-Z a-z | sort | uniq -c | sort -n -r
```

Listing 2: Minimal illustrative Python BPE merge step (non-optimized)

```
# NOTE: illustrative only, not optimized for large corpora.
from collections import Counter

def get_pair_freqs(corpus_tokens):
    pairs = Counter()
    for word in corpus_tokens:
        for i in range(len(word)-1):
            pairs[(word[i], word[i+1])] += 1
    return pairs

def apply_merge(corpus_tokens, pair_to_merge):
    merged = []
    a, b = pair_to_merge
    for word in corpus_tokens:
        i = 0
        new_word = []
        while i < len(word):
            if i < len(word)-1 and (word[i], word[i+1]) == (a, b):
                new_word.append(a+b) # merged token
                i += 2
            else:
                new_word.append(word[i])
                i += 1
        merged.append(new_word)
    return merged
```

11 Questions Addressed from the Notebook

11.1 Q: If we merge 'a'+'b' into new token 'Z', will 'abnormal' become 'Znormal'?

Yes. After that merge, every training occurrence of the character pair `ab` is replaced with `Z`. Thus `abnormal` will be represented as `Z n o r m a l` in the updated corpus. If later a merge of `Z`+`c` is learned (to form `X`), then `abc` can become `X`. All earlier merged tokens (`Z`) remain in the vocabulary.

11.2 Q: Will the vocabulary explode exponentially due to nested merges?

No. Each merge produces exactly one new token, and merges are only performed for frequent adjacent pairs observed in the data. The number of merges is a chosen hyperparameter k, thus vocabulary growth is linear in k (not exponential). Typical practice chooses k to yield a practical vocabulary size (e.g., 30k tokens).

12 Best Practices & Practical Recommendations

Importance & Use Cases

- Choose a vocabulary size appropriate for your model and compute budget (30k–50k is standard for many transformer models).
- Preserve tokens for special categories (URLs, emails, user handles) as special tokens or normalization rules.
- Consider whether to perform case-folding: do not lowercase when capitalization carries semantic information (e.g., named entities).
- Evaluate tokenization strategy with downstream metrics (e.g., perplexity, F1 for NER).

13 Summary

Definition

Byte-Pair Encoding (BPE) is a greedy, data-driven subword segmentation method that iteratively merges the most frequent adjacent symbol pairs to form a practical vocabulary of characters, subwords, and full words, allowing robust handling of rare and unseen words while controlling vocabulary size.

Importance & Use Cases

Use BPE (or similar subword methods) in modern sequence models to:

- Avoid OOV tokens,
- Reduce model parameter waste on extremely rare words,
- Preserve compositional information via subwords (morphemes).

14 References and Further Reading

- Sennrich, Haddow, and Birch (2016). Neural Machine Translation of Rare Words with Subword Units (BPE).
- Schuster and Nakajima (2012). WordPiece.
- Kudo (2018). Subword Regularization (Unigram).
- Bird, Klein, and Loper (2009). Natural Language Processing with Python (for basics).
- Jurafsky & Martin (2009, 2021). Speech and Language Processing — chapters on tokenization and corpora.

Prepared by: Koorosh Asil Ghrehbaghi
Course: KNTU Computer Science (Instructor: Dr. Maryam Abdolali)
Scope: Part I – Foundations: Data Preparation & Tokenization