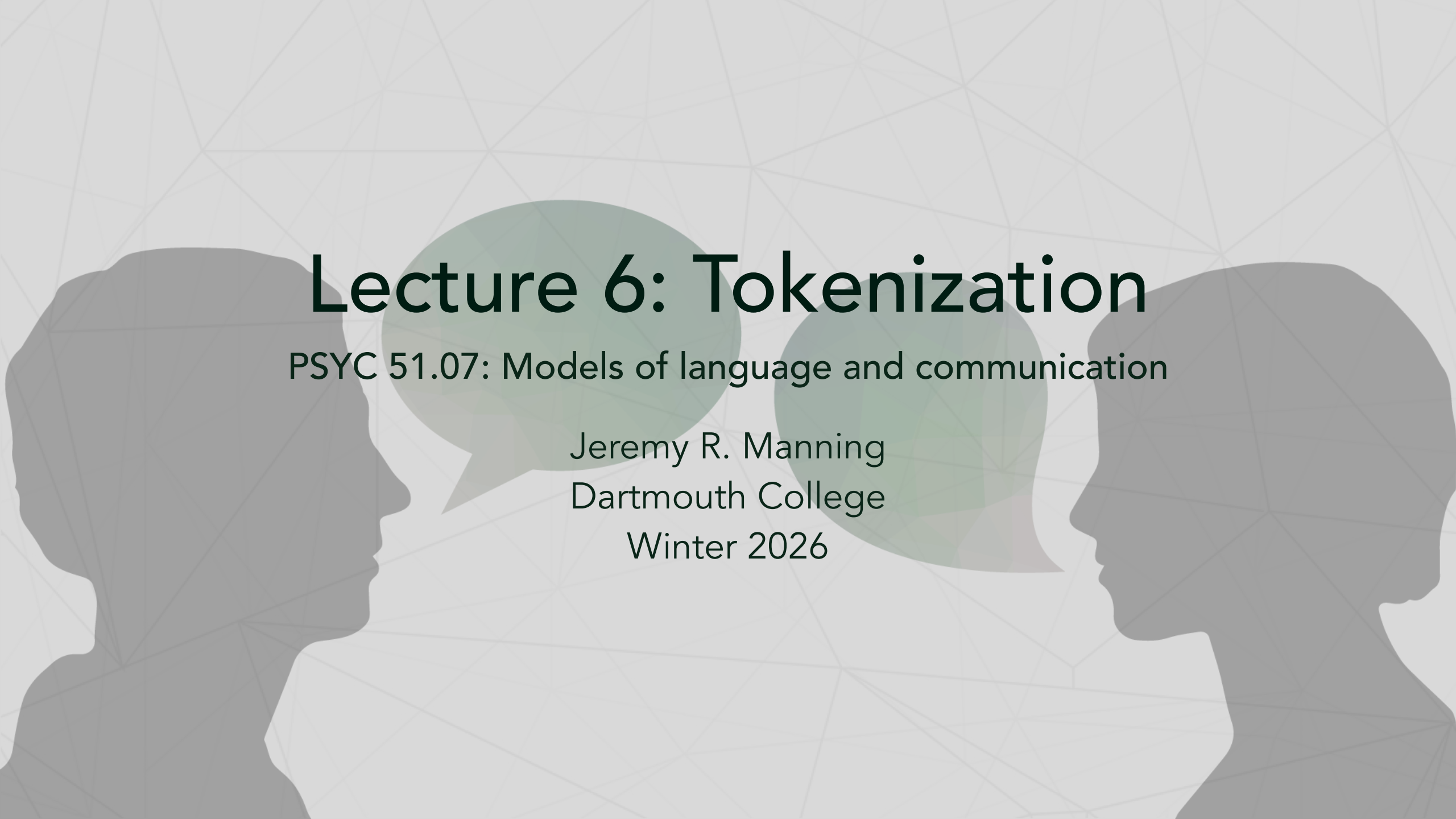


Lecture 6: Tokenization

PSYC 51.07: Models of language and communication

Jeremy R. Manning
Dartmouth College
Winter 2026

Learning objectives

By the end of this lecture, you will be able to...

1. Understand what tokenization is and why it matters
2. Explain the limitations of word-level tokenization
3. Describe how subword tokenization works (BPE, WordPiece, SentencePiece)
4. Compare different tokenization methods and their trade-offs
5. Implement and experiment with different tokenizers using HuggingFace

Central question

How do we break text into meaningful units for AI models?

What is tokenization?

> Definition
>
> **Tokenization** is the process of converting text into smaller units (tokens).

Why tokenize?

- Neural networks process numbers, not text
- Need discrete units to create vocabulary
- First step in any NLP pipeline

What can tokens be?

- **Characters:** a, b, c, ... (very fine-grained)
- **Words:** "hello", "world" (intuitive but limited)
- **Subwords:** "un", "happiness" (sweet spot!)

The tokenization spectrum

Fine-grained to coarse-grained tokenization

Granularity	Vocabulary size	Sequence length
Character	~100s	Very long
Subword	30k-50k	Medium

Splitting on spaces fails for three reasons

1. Vocabulary explosion

English has ~170,000 words, each inflection counts separately ("run", "runs", "running", "ran"), and compounds vary ("ice cream", "icecream", "ice-cream")

2. Unknown words

OOV for new terms ("COVID-19", "selfie"), rare words ("supercalifragilisticexpialidocious"), and typos ("teh")

3. Languages without spaces

Chinese: 没有空格 | Japanese: 日本語も同様

Word-level limitations illustrated

Vocabulary growth as training data increases

Observation

Word-level vocabulary keeps growing! Subword vocabulary stabilizes at a reasonable size.

The OOV problem in action

```python
# Simple word-level vocabulary
vocab = {"hello", "world", "the", "cat", "sat"}

def tokenize_word_level(text, vocab):
 tokens = text.lower().split()
 result = []
 for token in tokens:
 if token in vocab:
 result.append(token)
 else:
 result.append("<UNK>") # Unknown token
 return result

# Example
text = "Hello! The cat jumped"
```

Problem

We lose information! "jumped" becomes meaningless `<UNK>`

Most words are made of smaller meaningful pieces

Examples:

- "unhappiness" = "un" + "happiness"
- "preprocessing" = "pre" + "process" + "ing"
- "antiestablishment" = "anti" + "establish" + "ment"

Benefits:

- Fixed vocabulary (30k-50k tokens)
- No OOV: break into known parts
- Captures morphology (prefixes, suffixes)

Subword tokenization solves the OOV problem

```
# Word-level vocabulary (only words seen in training)
vocab = {"the", "cat", "sat", "on", "mat", "dog", "ran"}

# Trying to tokenize a new sentence:
```

The subword solution:

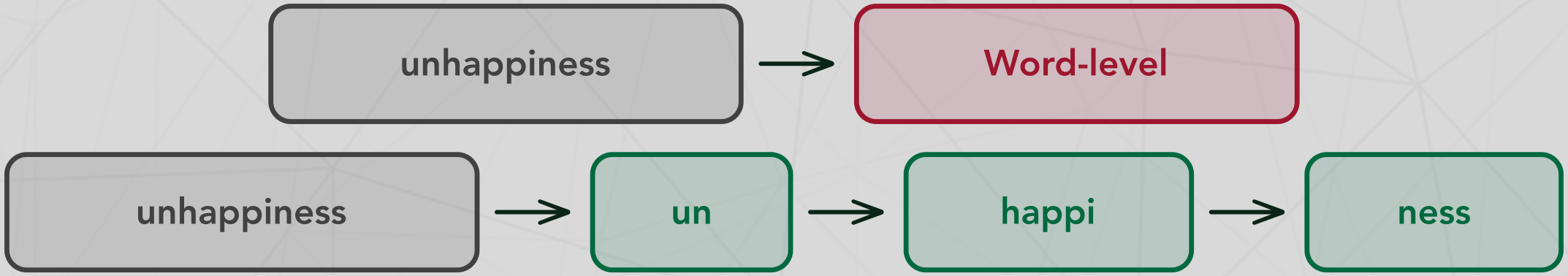
```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")

# Same difficult sentence:
tokens = tokenizer.tokenize("supercalifragilisticexpialidocious")
print(tokens)
```

Key insight

Subwords preserve meaning even for novel words!

Visual: How "unhappiness" gets tokenized



Word-level (1 token, may be OOV) vs. Subword BPE (3 tokens, always known)

Trade-off

More tokens = longer sequences, but smaller vocabulary and no OOV!

Byte-Pair Encoding (BPE)

Further reading

Sennrich, Haddow, & Birch (2016, *ACL*): Neural Machine Translation of Rare Words with Subword Units

- **Original use:** Data compression (1994)
- **Adapted for NLP:** Sennrich et al. (2016)
- **Used by:** GPT, GPT-2, GPT-3, RoBERTa, BART, many others!

Core idea

Iteratively merge the most frequent pair of characters/subwords.

Algorithm

1. Start with character-level vocabulary
2. Count all adjacent pairs in corpus
3. Merge most frequent pair → create new token
4. Repeat until desired vocabulary size

BPE example: Step by step

Training data: "low low low lower lower newest newest newest newest newest widest"

Initial tokens (characters): `l, o, w, e, r, n, s, t, i, d`

Iteration 1

Most frequent pair = `e, s` (appears 4 times)

- Merge → new token: `es`
- Vocabulary: `l, o, w, e, r, n, s, t, i, d, es`

Iteration 2

Most frequent = `es, t`

- Merge → new token: `est`
- Vocabulary: `l, o, w, e, r, n, s, t, i, d, es, est`

Iteration 3

Most frequent = `l, o`

- Merge → new token: `lo`

Continue until reaching target vocabulary size (e.g., 30,000)...

BPE learns to merge frequent character pairs

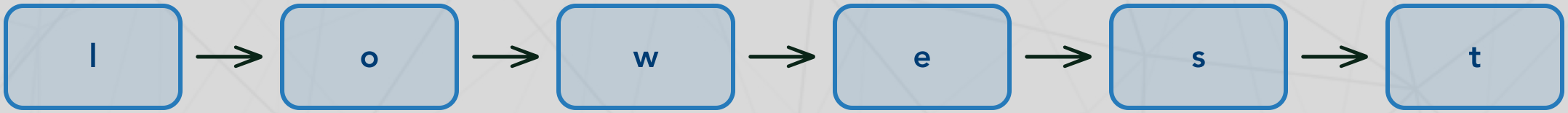
```
Training corpus: "low" (x3), "lower" (x2), "newest" (x4), "widest" (x1)

After training, vocabulary includes:
- Characters: l, o, w, e, r, n, s, t, i, d
- Merges learned: es → est → lo → low → er → ...

Tokenizing "lowest":
 Step 1: Split into characters → [l, o, w, e, s, t]
 Step 2: Apply merge rules in order learned:
 [l, o, w, e, s, t]
 → [lo, w, e, s, t] (merge l+o)
 → [low, e, s, t] (merge lo+w)
 → [low, es, t] (merge e+s)
```

Result: "lowest" → `["low", "est"]` (2 tokens instead of 6 characters!)

BPE visualization



Step 0: Character-level

After merges: "lowest" becomes 3 tokens

Result

"lowest" → `[lo, w, est]` — captures common patterns without explicit

BPE in practice with HuggingFace

```python
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")  # GPT-2 uses BPE

text = "I'm learning about tokenization!"
tokens = tokenizer.tokenize(text)
print("Tokens:", tokens)
# ['I', "'m", 'Ġlearning', 'Ġabout', 'Ġtoken', 'ization', '!']
# Note: 'Ġ' represents space

token_ids = tokenizer.encode(text)  # Get token IDs
print("Token IDs:", token_ids)  # [40, 1101, 4673, 546, 11241, 1634, 0]
```

Try it out!

Use the Tokenization Explorer Demo to experiment with different tokenizers interactively.

WordPiece merges "surprisingly common" pairs

Further reading

Wu et al. (2016, *arXiv*): Google's Neural Machine Translation System

Key difference from BPE

Instead of merging most *frequent* pair, merge pair that maximizes *likelihood*:

$$\text{score}(x, y) = \frac{P(xy)}{P(x) \times P(y)}$$

- **Used by:** BERT, DistilBERT, Electra
- **Special tokens:** `##` prefix for continuation ("playing" → `["play", "##ing"]`)

WordPiece example with BERT

```python
from transformers import AutoTokenizer

# BERT uses WordPiece
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

text = "I'm learning about tokenization!"

tokens = tokenizer.tokenize(text)
print("Tokens:", tokens)
# Output: ['i', "'", 'm', 'learning', 'about', 'token', '##ization', '!']
# ^^^ Note the ##
```

Note

BERT lowercases by default (unless using cased model)

SentencePiece works for languages without spaces

Further reading

Kudo & Richardson (2018, *EMNLP*): SentencePiece: A simple and language independent subword tokenizer

Problem with BPE/WordPiece:

- Assume pre-tokenized text
- Fail on Chinese, Japanese, Thai...

SentencePiece solution:

- Raw character stream input
- Space = just another char (`_`)

SentencePiece in action

```python
from transformers import AutoTokenizer

# T5 uses SentencePiece
tokenizer = AutoTokenizer.from_pretrained("t5-small")

text = "I'm learning about tokenization!"

tokens = tokenizer.tokenize(text)
print("Tokens:", tokens)
# Output: ['I', "'", 'm', 'learning', 'about', 'token', 'ization', '!']
# ^^^ Note the for spaces

# Works seamlessly with other languages!
```

Key advantage

No language-specific preprocessing required!

Tokenization methods comparison

Method	Approach	Used By	Key Feature
BPE	Frequency merging	GPT-2, RoBERTa	Bottom-up
WordPiece	Likelihood merging	BERT, DistilBERT	Principled scoring
SentencePiece (Unigram)	Top-down pruning	T5, mT5	Multilingual

Comparing tokenizers side-by-side

```
from transformers import AutoTokenizer
text = "The unhappiest researchers couldn't preprocess data!"

models = {"GPT-2 (BPE)": "gpt2", "BERT (WordPiece)": "bert-base-uncased",
          "T5 (SentencePiece)": "t5-small"}

for name, model_name in models.items():
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    tokens = tokenizer.tokenize(text)
    print(f"{name}: Tokens ({len(tokens)}): {tokens}")

# Observe: Different handling of "unhappiest", spaces, and token counts!
```

Tokenizer comparison: Actual output

Input: `"The unhappiest researchers couldn't preprocess data!"`

Tokenizer	Tokens	Count
GPT-2 (BPE)	`['The', 'Ġun', 'happ', 'iest', 'Ġresearchers', 'Ġcouldn', "'t", 'Ġpre', 'process', 'Ġdata', '!']`	11
BERT (WordPiece)	`['the', 'un', '##hap', '##pie', '##st', 'researchers', 'couldn', "'", 't', 'pre', '##process', 'data', '!']`	13
T5 (SentencePiece)	`['The', 'un', 'happiest', 'researchers', 'couldn', "'", 't', 'pre', 'process', 'data', '!']`	11

Key observations

- `Ġ` (GPT-2) and `_` (T5) mark word starts (spaces)
- `##` (BERT) marks continuation subwords
- "unhappiest" is split differently by each
  BERT lowercases; GPT-2/T5 preserve case

Tokenizers add special tokens for model-specific purposes

Token	Purpose	Used By
[CLS]	Classification token (start)	BERT
[SEP]	Separator between segments	BERT
[PAD]	Padding to same	Most models

Working with special tokens

```python
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

text = "Hello world"
encoded = tokenizer(text, return_tensors="pt")
print("Input IDs:", encoded['input_ids'])  # [101, 7592, 2088, 102]
# ^^^ [CLS] and [SEP] added automatically!

full_decode = tokenizer.decode(encoded['input_ids'][0])
print("With special:", full_decode)  # "[CLS] hello world [SEP]"

clean_decode = tokenizer.decode(encoded['input_ids'][0], skip_special_tokens=True)
print("Without special:", clean_decode)  # "hello world"
```

Vocabulary size trade-offs

Smaller vocabulary (e.g., 10k tokens):

- Faster training (smaller embedding matrix)
- Less memory
- Longer sequences (more subwords per word)
- May lose semantic

Larger vocabulary (e.g., 100k tokens):

- Shorter sequences (closer to word-level)
- Better semantic preservation
- Slower training (larger embeddings)

Tokenization pitfalls and gotchas

Common issues to watch out for

1. **Tokenizer-model mismatch:** Always use the tokenizer that matches your model! GPT-2 tokenizer ≠ BERT tokenizer
2. **Maximum sequence length:** BERT: 512 tokens | GPT-2: 1024 | GPT-3: 2048 — Text gets truncated if too long!
3. **Case sensitivity:** `bert-base-uncased` lowercases everything; `bert-base-cased` preserves case
4. **Rare words → many tokens:** "antidisestablishmentarianism" → 10+ tokens — Can hit sequence limit faster than expected!
5. **Special characters:** Emoji, Unicode, accents may be split unexpectedly

Debugging tokenization

```python
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
text = "The antidisestablishmentarianism debate continues!"

tokens = tokenizer.tokenize(text)
token_ids = tokenizer.encode(text)
print(f"Tokens ({len(tokens)}): {tokens}\n")

for i, (token, token_id) in enumerate(zip(tokens, token_ids)):
    decoded = tokenizer.decode([token_id])
    print(f"{i:2d}. ID {token_id:5d} | Token: {token:20s} | Decoded:
{decoded}")

print(f"\nVocabulary size: {tokenizer.vocab_size}")
```

Connection to human language learning

> **Further reading**
>
> **Saffran, Aslin, & Newport (1996, *Science*):** Statistical learning by 8-month-old infants

Infant learning:

- Babies track statistical regularities in speech
- Identify word boundaries from transitional probabilities

Parallel with subword tokenization:

- BPE: Merge frequent character pairs → discover common morphemes
- Infants: Track frequent syllable pairs → discover

Statistical learning in action

Infant Learning:

Input stream:

`bidakupadotigolabu ...`

Learn high-probability sequences:

- `bi-da-ku` (word)
- `pa-do-ti` (word)

BPE Learning:

Input corpus:

`low low lower ...`

Merge high-frequency pairs:

- `l+o` → `lo`
- `lo+w` → `low`

Build vocabulary:

Hands-on exercise

Experiment with different tokenizers!

1. **Choose 3 models:** GPT-2, BERT, T5
2. **Test on diverse texts:**
   - Standard English: "The cat sat on the mat"
   - Complex words: "antidisestablishmentarianism"
   - Contractions: "I'm, you're, won't"
   - Typos: "teh qiuck brown fox"
   - Emoji: "I love this!"
   - Other languages: "这是中文" (Chinese)
3. **Compare results:** Number of tokens, how words are split, handling of unknown/rare words
4. **Reflect:** Which tokenizer works best for your use case? What are the trade-offs?

Try it out!

Use the Tokenization Explorer Demo to compare tokenizers interactively!

Discussion questions

Think about it...

1. **Linguistics vs. Statistics:** BPE discovers morphemes (un-, -ing, -ness) without linguistic rules. Is this "learning" morphology, or just pattern matching?
2. **Cross-lingual tokenization:** Should we use the same tokenizer for all languages? What are the trade-offs?
3. **Semantic preservation:** Does breaking "unhappy" into ["un", "happy"] preserve meaning? What about "butterfly"?
4. **Human vs. machine:** Humans don't consciously tokenize words. Why do machines need to?
5. **Future directions:** Will we move toward character-level or byte-level models that don't need tokenization?

Primary references

Foundational papers

- **Sennrich, Haddow, & Birch (2016).** Neural Machine Translation of Rare Words with Subword Units. *ACL*. — Introduced BPE for NLP
- **Kudo & Richardson (2018).** SentencePiece: A simple and language independent approach. *EMNLP*. — Language-agnostic tokenization
- **Wu et al. (2016).** Google's Neural Machine Translation System. *arXiv*. — WordPiece algorithm

HuggingFace resources

- Chapter 2.4: Tokenizers
- Chapter 6: Tokenizers (detailed)

Key takeaways

Summary

1. **Tokenization is fundamental:** Bridges raw text and neural networks
2. **Word-level has major limitations:** OOV problem, vocabulary explosion, language-dependent
3. **Subword tokenization is the sweet spot:** Balanced vocabulary size and sequence length
4. **Different methods, similar principles:** BPE: frequency-based | WordPiece: likelihood | SentencePiece: language-agnostic
5. **Statistical learning connects humans and machines:** Both discover structure from distributional patterns
6. **Always match tokenizer to model!** Critical for correct predictions

Looking ahead

Next lecture: X-Hour Text Classification Workshop

How tokenization connects:

- POS tagging: Token-level classification
- Sentiment: Sequence-level classification
- Both depend on good tokenization!

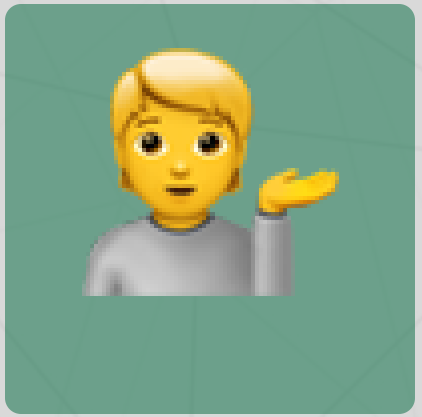
Prepare by...

- Experimenting with HuggingFace tokenizers
- Thinking about: How does token granularity affect downstream tasks?
- Exploring: Tokenizer artifacts and their impact

Questions? Want to chat more?

Email me

Join our Discord



Come to office hours

Next up

Lecture 7 — X-Hour: Text Classification Workshop