Natural Language Processing

Lecture 7 - Tokenization

May 2025

Eyal Ben-David

*Partially Based on Umass's CS685, and Cornell's CS 5740

(DON’T FORGET RECORDING!!)

Course Logistics & updates

- First assignment is over
  - An important milestone

- Second assignment will be released on Tuesday
  - This class is very relevant

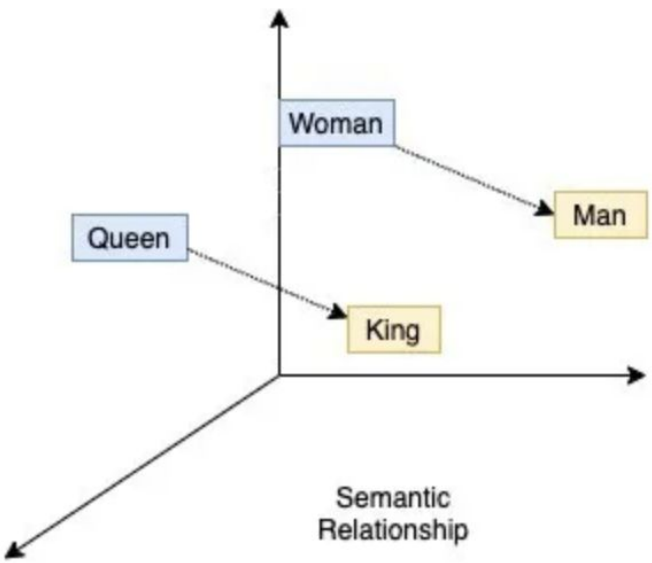
Recp

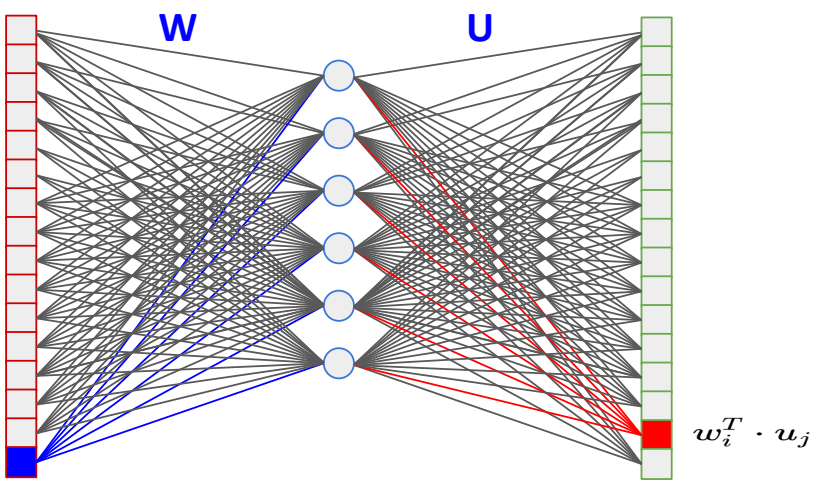
In the past few weeks, we discussed several topics

- (Dense) static word representation
  - W2V, Glove
- RNNs

Recp - Dense-Static Word Embeddings

- Mainly focused on W2V & Glove.

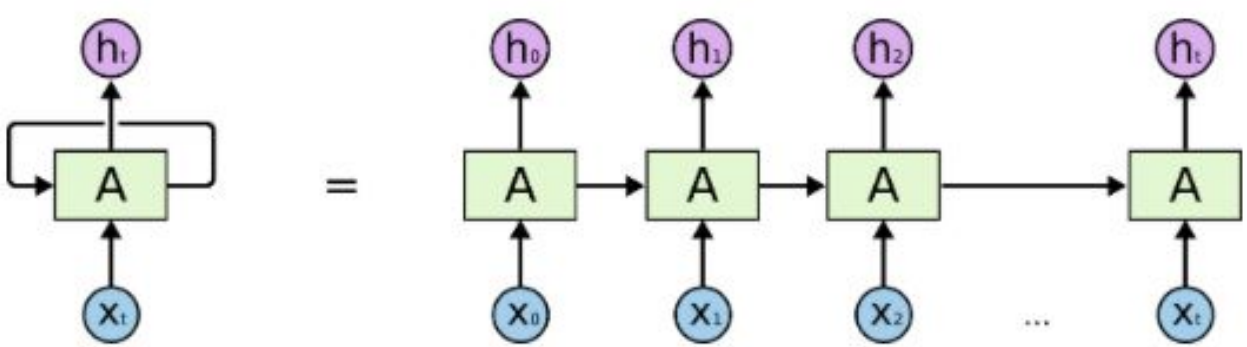




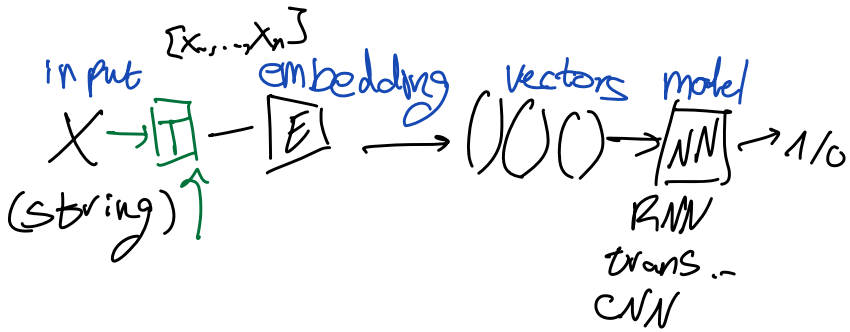
Recap - Why word embeddings are not enough

Recp - RNNs

- A neural network with a directed cycle.
- The output at time step t is part of the input of at time step t+1.
- The cyclic connection serves as a memory state, updated at every time step given the last and current inputs.
- Allows information to propagate through time.



Tody



- Taking one step back
- We always assume to have a sequence X1, ... , Xn
- But how do we get this sequence?
- What is exactly this sequence?

Representing input text

- How do we represent an input text?

  Tokenization: splitting a string into a sequence of tokens

- Given a piece of text ***X***, we said that it's a sequence <***x1, x2, ..., xn***>
- But how do we get from a string to <***x1, x2, ..., xn***>

Vanilla

Word-level tokenization

- How do we represent an input text?
- So far, we chop string into *words*

“I love Ketchup, but hate tomatoes.”

X = < I , love , Ketchup, , but , hate , tomatoes >

Word-level tokenization

So far, we chop string into *words* `str.split(' ')`

"I love Ketchup, but hate tomatoes."

X = < I , love , Ketchup, , but , hate , tomatoes >

- But should *Ketchup,* be a single token?
- So, tokenization is not simple
- Even vanilla tokenizers require specialized rules

Word-level tokenization

- What about the following strings:

  “amazing!”, “state-of-the-art”, “un-thinkable”,

  “prize-winning”, “aren’t”, “O’Neill”

- Also, what about this:

  “ကြောင်က ကြွက်ကို လိုက်သွားတယ်။”

  “猫追老鼠”

  (“the cat chased the mouse” in Burmese and Chinese)

How will it look

```python
def split_to_words(text):
    words = []
    current_word = ''
    special_chars = set([
        ' ', '\n', '\t', '.', ',', '!', '?', ':', ';',
        '(', ')', '[', ']', '{', '}', '"', '/', '\\',
    '|',
        '_', '+', '=', '*', '&', '^', '%', '$', '#', '@',
        '~', '`', '<', '>', '—', '…', …
    ])

    for char in text:
        …
        …
```

Handling <UNK> words

What happens when we encounter a word that we have never seen in our training data?

- Not much we can do with word-level tokenization
- We assign a special token, <UNK>
- Don’t forget using <UNK> in training
  - Why?
  - How will we use it?

Limitations - QA

Q: What is the recommended dosage of belimumab for pediatric patients with systemic lupus erythematosus?

(Easy, right?:))

Limitations - QA

Q: What is the recommended dosage of belimumab for pediatric patients with systemic lupus erythematosus?

T: ["What", "is", "the", "recommended", "dosage", "of", "<UNK>", "for", "pediatric", "patients", "with", "systemic", "lupus", "<UNK>", "?"]

Limitations - Named Entity Recognition (NER)

Task: An NER model is designed to scan clinical trial reports or patient notes and identify mentions of medication names (tagging them as MEDICATION).

Text: Initial trials of ziltivekimab showed promising results in reducing cardiovascular events for patients with CKD. Further analysis of pharmacodynamics is ongoing.

(Easy, right?:))

Limitations - Named Entity Recognition (NER)

Task: An NER model is designed to scan clinical trial reports or patient notes and identify mentions of medication names (tagging them as MEDICATION).

Text: Initial trials of <UNK> showed promising results in reducing cardiovascular events for patients with CKD. Further analysis of <UNK> is ongoing.

(How about now?:)

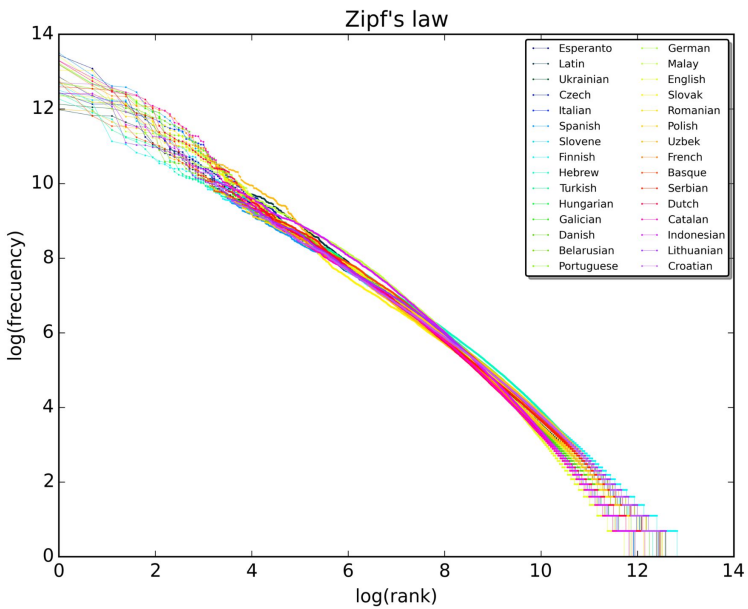
Limitations - more

- Especially hurts in texts/languages with many rare words/entities
- Treating words with the same roots differently
  - e.g., “play”, “played”, “plays”, “playing”, etc.
- Why is this a problem?
  - Especially with limited data?

Word frequency - Zipf’s law

f(r) ≅ C / r (r - rank, f - frequency)

- The most frequent word appears approximately twice as often as the second most frequent word.
- The third most frequent word appears approximately one-third as often as the most frequent word, and so on.




WHAT ELSE
CAN WE DO?

An alternative: Character-level tokenization

- How do we split input text?
- Very simple, instead of white spaces, we split to characters.

“I love Ketchup, but hate tomatoes.”

X = < *I* , , *l*, *o*, *v* , *e*, , *K*, *e*, *t*, *c*, *h*, *u*, *p*, , *b*, *u*, *t*, , *h*, *a*, *t*, *e*, , *t*, *o*, *m*, *a*, *t*, *o*, *e*, *s*>

How will it look (very elegant)

```python
def split_to_characters(text):
    characters = []
    for char in text:
        characters.append(char)
    …
```

Character-level tokenization

- Impact on vocabulary size? Unknown word problem? Other concerns?

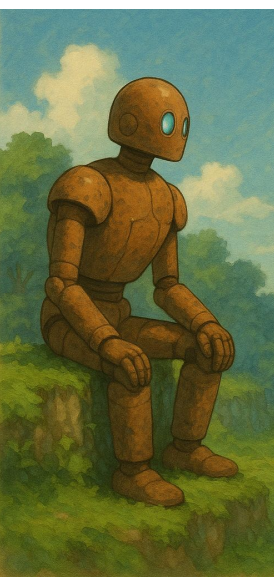
Character-level tokenization

- Impact on vocabulary size? Unknown word problem? Other concerns?
  - Vocabulary size: Very small, just the number of unique characters
  - We don't have <UNK> anymore :)
  - Input sequence length: Turns to be much longer
  - (for generative model) Output sequence length: Also much longer
  - We will need to learn how to stitch character together into words
  - What about informative word embeddings?

Character-level tokenization

- Impact on vocabulary size? Unknown word problem? Other concerns?
  - Vocabulary size: Very small, just the number of unique characters
  - We don’t have <UNK> anymore :)
  - Input sequence length: Turns to be much longer
  - (for generative model) Output sequence length: Also much longer
  - We will need to learn how to stitch character together into words
  - What about informative word embeddings?
- **Major limitation: processing/generation time.**

Character-level vs. Word-level



Word-level

Pros:

- Carry clear semantic meaning.
- Efficiently captures contextual information.
- Shorter sequences, speeding up training and inference.

Cons:

- Large vocabulary
- OOV
- Complex preprocessing

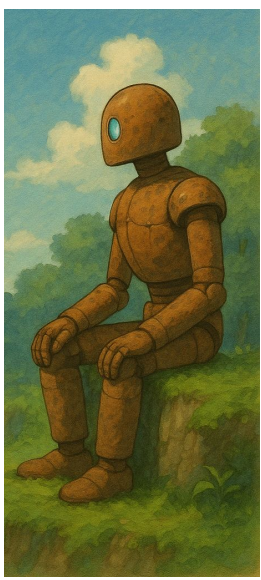
Character-level

Pros:

- Minimal vocabulary size
- Eliminating OOV issues.
- Easily handles morphological variations, misspellings, and noisy text.
- Simpler preprocessing pipeline

Cons:

- Longer sequences; expensive.
- Less semantic information per token
- Requires larger/deeper models for comparable performance.



Let’s find a middle ground

The middle ground: Sub-word tokenization

- Developed for machine translation by Sennrich et al., ACL 2016
- Based on byte pair encoding (Gage, 1994)
- Today, this is used everywhere.

"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**."

Why Subword Units Help in Translation

Motivation: Some words are translatable even if unseen before — thanks to familiar subword units (e.g., characters, morphemes).

Type	**English (Tokens)**	**German (Tokens)**	**Russian (Tokens)**	**Japanese (Tokens)**
Named Entity	[Ba][ra][ck] [O][ba][ma]	[Ba][ra][ck] [O][ba][ma]	[Б][а][р][а][к] [О][б][а][м][а]	[バ][ラ][ク] [オ][バ][マ]
Cognate / Loanword	[cl][au][st][r][o][ph][o][b][ia]	[Kl][au][st][ro][ph][ob][ie]	[К][л][а][у][с][т][р][о][ф][о][б][и][я]	
Morphologically Complex	[solar] [system]	[Sonne][System]		

Byte Pair Encoding (BPE) - A Macro Overview

- Relies on pre-tokenization that splits the training data into words.
- Use a large training corpus.
- After pre-tokenization, get the set of unique words and their frequency.
- Create a basic vocabulary: All unique characters/symbols in the data.
- **Merge tokens**: At each step, merge the most frequent pair of tokens in the vocab, to create a new (longer) token.
- Repeat K times.

BPE - Example (Training)

Training data

Pre-tokens	Frequency
_hug	10
_pug	5
_pun	12
_bun	4
_bugs	5

Example from https://huggingface.co/transformers/tokenizer_summary.html

BPE - Example (Training) - Init

Training data

Pre-tokens	Frequency
hug	10
pug	5
pun	12
bun	4
bugs	5

Initial vocabulary

_b, g, _h, n, _p, s, u,

Example from https://huggingface.co/transformers/tokenizer_summary.html

BPE - Example (Training) - Iteration #1

Pre-tokens	Frequency
_h+u+g	10
_p+u+g	5
_p+u+n	12
_b+u+n	4
_b+u+g+s	5

Merge cand	Frequency
_hu	10
ug	20
_pu	17
un	16
_bu	9
gs	5

Example from https://huggingface.co/transformers/tokenizer_summary.html

BPE - Example (Training) - - Iteration #2

Pre-tokens	Frequency
_h+ug	10
_p+ug	5
_p+u+n	12
_b+u+n	4
_b+ug+s	5

Merge cand	Frequency
_hug	10
_pug	5
_pu	12
un	16
_bu	4
_bug	5
ugs	5

Rules	score
ug	20

vocab	
_h	u
s	g
_p	n
_b	ug

Example from https://huggingface.co/transformers/tokenizer_summary.html

BPE - Example (Training) - - Iteration #3

Pre-tokens	Frequency
_h+ug	10
_p+ug	5
_p+un	12
_b+un	4
_b+ug+s	5

Merge cand	Frequency
_hug	10
_pug	5
_pun	12
_bun	4
_bug	5
ugs	5

Rules	score
ug	20
un	16

vocab		
_h	u	un
s	g	
_p	n	
_b	ug	

Example from https://huggingface.co/transformers/tokenizer_summary.html

BPE - Inference - Merge using the rules

- Split string to words.
- Per word:
  - Init - Map word to chars or bytes.
    - Don’t forget to include ‘ ‘
  - Repeat:
    - Extract all possible subsequent pairs
    - Find all merged pairs that appear in the vocabulary
      - If empty – return the tokens you found. Break
      - If not – choose the merge with the highest score. Apply the new merge to the word. Repeat

BPE - Example (Inference) - Find basic symbols

Input word: ‘Bugunug’

Symbols: _b+u+g+u+n+u+g

Rules	score
ug	20
un	16

BPE - Example (Inference) - Extract all merges cands

Input word: ‘Bugunug’

Symbols: _b+u+g+u+n+u+g

Potential merges	In rules	Score
_bu	x	-
ug	x	-
gu	x	-
un	v	16
nu	x	-
ug	v	20

Rules	score
ug	20
un	16

BPE - Example (Inference) - Choose best merge

Input word: ‘Bugunug’

Symbols: _b+u+g+u+n+u+g

Potential merges	In rules	Score
_bu	x	-
ug	x	-
gu	x	-
un	v	16
nu	x	-
ug	v	20

Rules	score
ug	20
un	16

BPE - Example (Inference) - Apply new merge

Input word: ‘Bugunug’

Symbols: _b+ug+u+n+ug

Rules	score
ug	20
un	16

BPE - Example (Inference) - Extract all merges cands

Input word: ‘Bugunug’

Symbols: _b+ug+u+n+ug

Potential merges	In rules	Score
_bug	x	-
ugu	x	-
un	v	16
nug	x	-

Rules	score
ug	20
un	16

BPE - Example (Inference) - Apply new merge

Input word: ‘Bugunug’

Symbols: _b+ug+un+ug

Rules	score
ug	20
un	16

BPE - Example (Inference) - Extract all merges cands

Input word: ‘Bugunug’

Symbols: _b+ug+un+ug

Potential merges	In rules	Score
_bug	x	-
ugun	x	-
unug	x	-

Rules	score
ug	20
un	16

BPE - Example (Inference) - Return symbols

Input word: '**Bugunug**'

Final Symbols: **_b+ug+un+ug**

Other inference metrics

- Greedy
  - Longest prefix
  - Longest suffix
  - Longest token
- Merge rules-based inference methods
  - BPE
- Merge rules-based inference methods

Byte Pair Encoding

- We want to avoid OOV → Add all character and symbols
- OOPs: there are ~140K unicode symbols (constantly growing...)
- Instead: Start with bytes (2^8=256 overall)
  - They can represent all unicode symbols
- Common vocabulary sizes:
  - 32K-64K (~2018)
  - 200K (~Today)
  - Why is that??
- Tokenization Package from Hugging Face (https://github.com/huggingface/tokenizers)

Tokenization Goals

- Three main goals:
  - No OOV, handle any input
  - Speed (!!)
  - Token efficiency (also relates to speed, why?)
- Another (weaker) goal
  - Linguistic/task-driven tokenization
    - Empirically, we don’t see high gains on downstream tasks

What do subwords capture?

- No OOV – the model can provide an informative representation to anything
- Subword tokens may be meaningless sometimes
- But, they can also capture meaningful stuff
  - Morphemes (the smallest meaning-bearing unit)
    - “unlikeliest” ⟶ [un-, likely, -est]
  - Split single/plural form

- Whatever it catches – Was automatically learned from the data

Subword Limitations

- Pre-tokenization using spaces doesn’t work on some languages (e.g., Chinese and Thai)
- Morphologically rich languages

Actually, they work pretty well:)

Challenges with Morphologically Rich Languages

- Complex word formation
  - Build words by combining a root with multiple prefixes, suffixes, and infixes.
    - Note: infix – turning cupful to plural ⟶ cupsful
  - Each morpheme carries grammatical/semantic meaning
  - One word can often express what takes several words in languages
- Excessive Fragmentation
  - Over-segment complex words
  - Breaking down meaningful morphemes into smaller, less informative subword units

Challenges with Morphologically Rich Languages

Gözlükçülerimizinkilerden → from those of our opticians

- göz: eye
- -lük: makes it a noun related to eyes (glasses)
- -çü: agent suffix (maker/seller - optician)
- -ler: plural
- -imiz: our
- -in: genitive case (of)
- -ki: relative adjective marker (the one(s) of)
- -ler: plural (again, referring to the things)
- -den: from

Other Subword Tokenization Algorithms

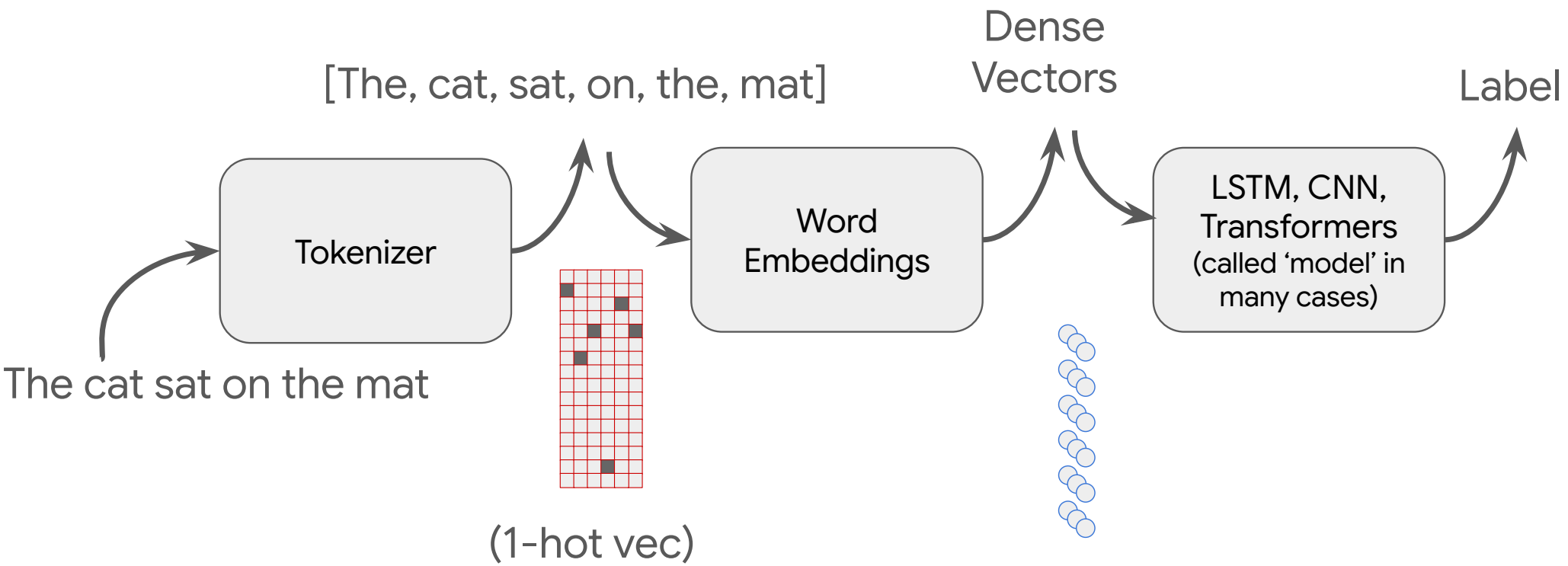
- **WordPiece** (Schuster et al., 2012): merge to increase likelihood
  - as measured by a language model (vs. frequency as in BPE)
- **SentencePiece** (Kudo et al., 2018):
  - Without pre-tokenization - not splitting to word
  - One of the initial symbols is ‘ ‘ (white spaces)
  - Then, just follow BPE.

Unigram (Kudo 2018)

- **Gist: work the opposite direction of BPE**
- **Init with a lot of base symbols**
- **Define a log-likelihood loss over the training data**
  - **Using the current vocabulary and a unigram language model**
- **In each iteration, drop the symbol the provides the lowest increase to the loss**
- **How to init?**
  - **Use BPE with a very high ‘iterations’ param**
  - **Or find the frequent sub-words/words appearing in the train set**

$$\mathcal{L} = -\sum_{i=1}^{N} \log \left(\sum_{x \in S(x_i)} p(x) \right)$$

Tokenizers in the NLP Pipeline



Next Week(s)

- Weeks 8 - LSTMs and syntactic structure!
  - LSTMs and RNNs
    - Bi-directional
    - Multi-layer
    - Attention
  - Dependency parsing
- Week 9 - Transformers!!

See you next week!