Foundations for Natural Language Processing

Evaluation for generation, tokenization

Ivan Titov
(with graphics/materials from Elena Voita)

School of informatics

Plan for today

Last time:

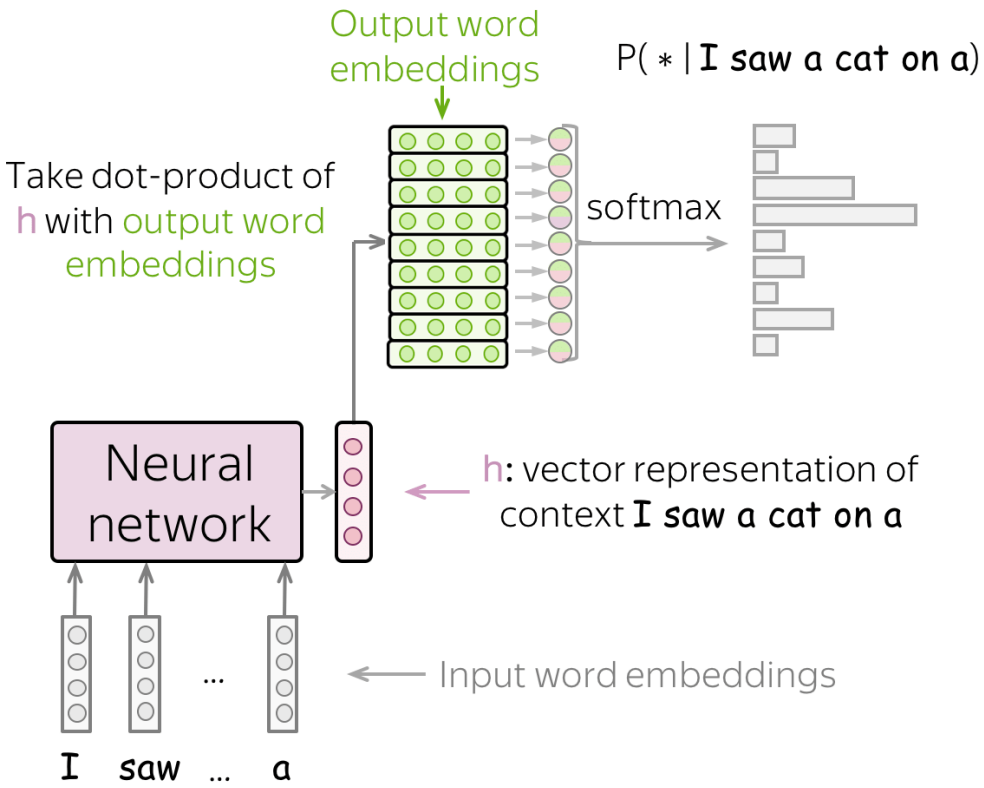
- Talked about generation and discussed inference algorithms
- Introduce vanilla encoder-decoder algorithms

Today, we will

- discuss how to evaluate text generation systems
- discuss tokenization and specifically subword tokenization

Recap: Generating text

To generate text using a language model, you could just *sample* tokens from the probability distribution predicted by a model



Recap: Sequence-to-Sequence modeling

x – input sentence,
y - its translation

Machine Translation

model parameters

$$y' = \arg\max_y p(y|x, \theta)$$

Questions we need to answer

- modeling

How does the model for $p(y|x, \theta)$ look like?

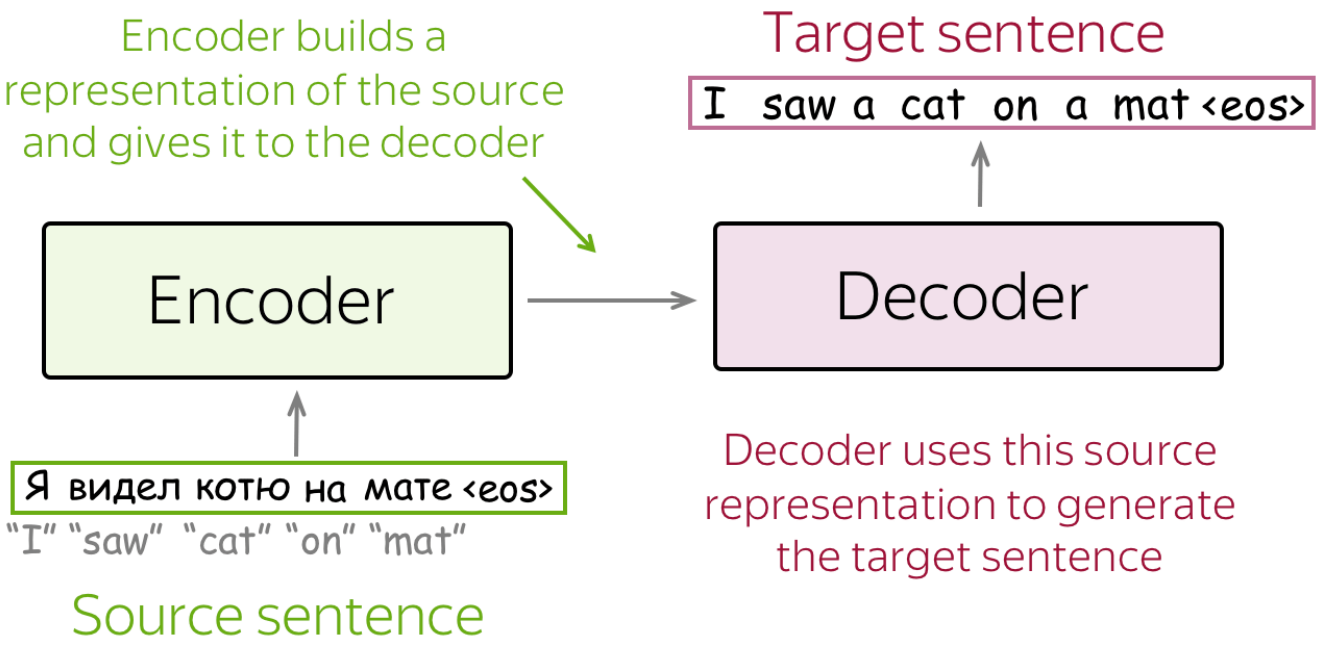
- learning

How to find θ?

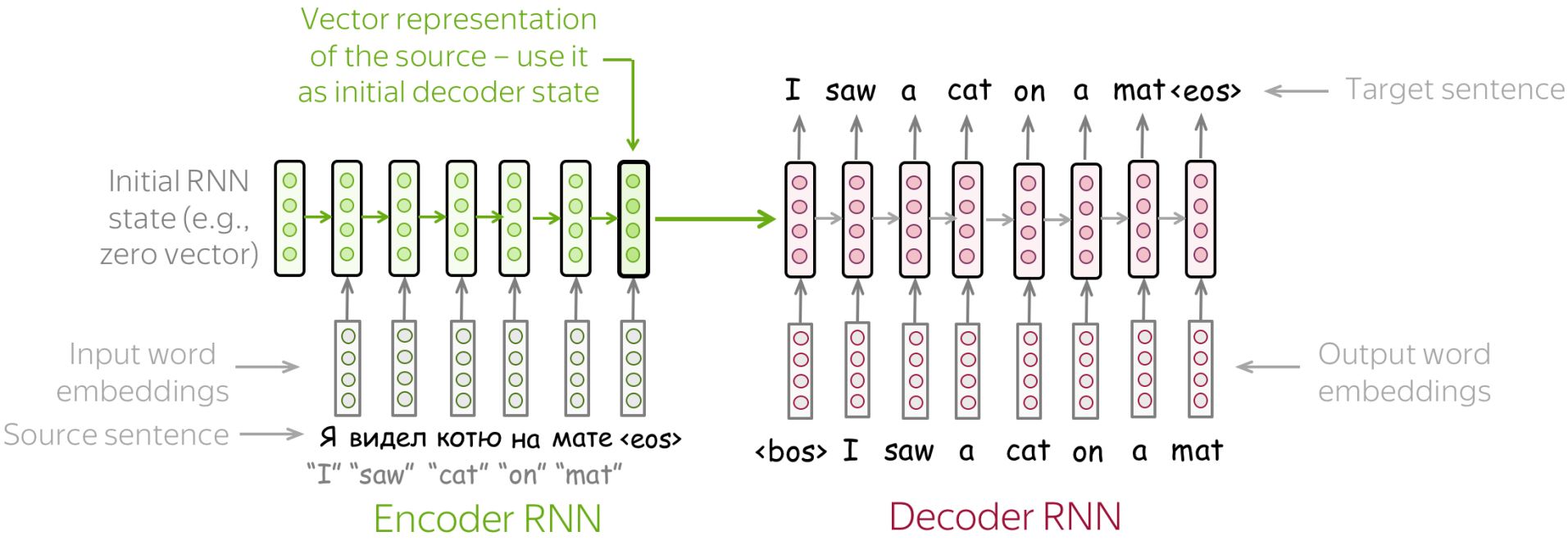
- search

How to find the argmax?

Recap: Encoder-decoder framework



Recap: simplest RNN-based Model:



Inference (aka decoding)

$$y' = \arg\max_y p(y|x) = \arg\max_y \prod_{t=1}^{n} p(y_t|y_{<t}, x)$$

How to find the argmax?

The simplest idea – greedy decoding, at each step, pick the most likely token, but note:

$$\arg\max_y \prod_{t=1}^{n} p(y_t|y_{<t}, x) \neq \prod_{t=1}^{n} \arg\max_{y_t} p(y_t|y_{<t}, x)$$

We can also do sampling (e.g., nucleus), but this is not argmax, how can approximate the global argmax better: beam-search

Beam search

Maintaining top hypotheses as you go

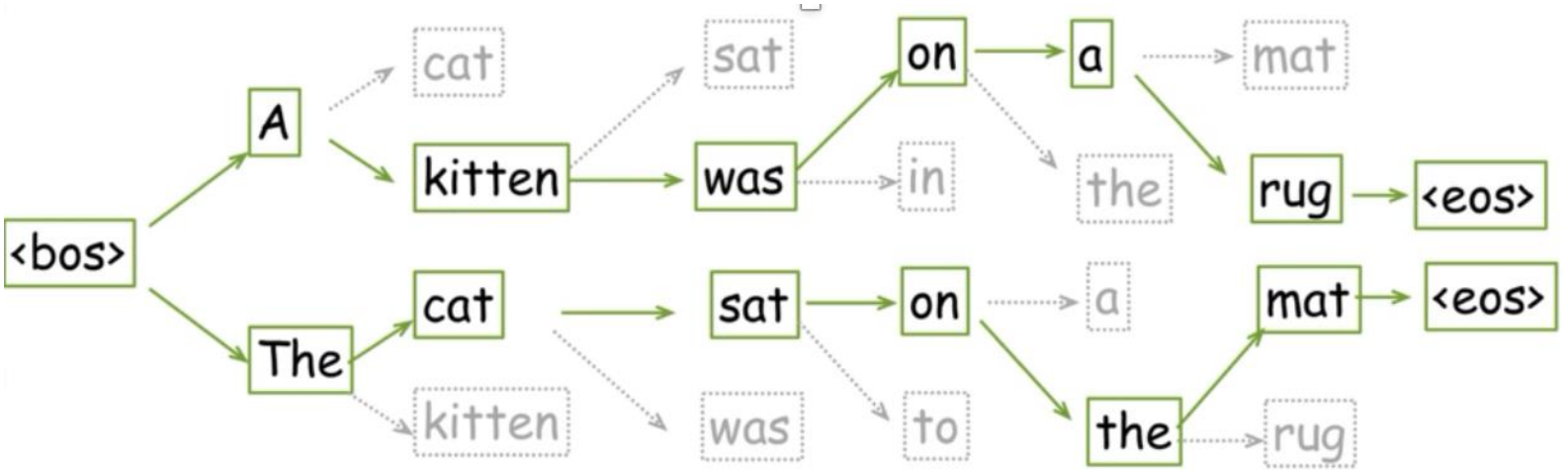
<bos>

Start with the begin of sentence token or with an empty sequence

(video, not visible in pdf)

Beam search

Maintaining top hypotheses as you go



All hypotheses are complete - generation ended

Why not sampling?

Actually, we can also sample in machine translation too (as with language modelling)

The risk is that a sample translation can deviate from the source sentence in meaning (i.e. hallucinate)

As discussed last time, sampling may be preferable to beam-search for other seq2seq task, where there is more freedom in the choice of generations (e.g., generate a story given keywords)

Evaluating text generation models

How to evaluate text generation?

Consider French to English machine translation

Source sentence: Le chat est assis sur le tapis

Human translation into English: The cat is on the carpet

How to evaluate text generation?

Consider French to English machine translation

Source sentence: Le chat est assis sur le tapis

Human translation into English: The cat is on the carpet

How can we design a metric which would score MT1 > MT2?

MT1: The cat is seated on the mat

MT2: The chat is assassinated on the tape

(We are looking into *automatic extrinsic evaluation*)

Idea: count overlapping ngrams - BLEU

Typically, we need more than 1 human (aka reference) translation per sentence to have reliable evaluation.

Let's focus on unigrams (individual tokens) for now

MT: The the the the the the the a

Reference 1: The cat is on the mat

Reference 2: There is a cat on the mat

(ignore capitalization for evaluation, i.e. treat 'The' and 'the' as the same word)

Idea: count overlapping ngrams - BLEU

MT: The the the the the the a — 'the' appears 7 times

Reference 1: The cat is on the mat — 'the' appears 2 times

Reference 2: There is a cat on the mat — 'the' appears 1 time

Modified unigram precision: 2 / 7

Idea: count overlapping ngrams - BLEU

MT: The the the the the the the a — 'a' appears 1 time

Reference 1: The cat is on the mat — 'a' appears 0 times

Reference 2: There is a cat on the mat — 'a' appears 1 time

Modified unigram precision: (2 + 1) / (7 + 1) = 3 / 8

Aggregate over all unigrams in the MT ('candidate')

BLEU metric

Actual BLEU is considerably more complicated, as needs to

- aggregate over the entire test set
- aggregate over ngrams of different order (unigrams, bigrams, …)
- penalize short translation (remember from parsing: *precision* favors models producing short outputs)
- …

There are other ngram overlap metrics which can be more suitable for other text generation problems (e.g., ROUGE for summarization)

Ngram overlap metrics - weaknesses

- do not account for lexical paraphrases (e.g., substituting words with their synonyms)
- even more problematic for long text generation (e.g., document machine translation)
- unreliable for tasks with less restricted outputs (e.g., generate "a scary novel about Edinburgh")
- do not sufficiently penalize hallucinations

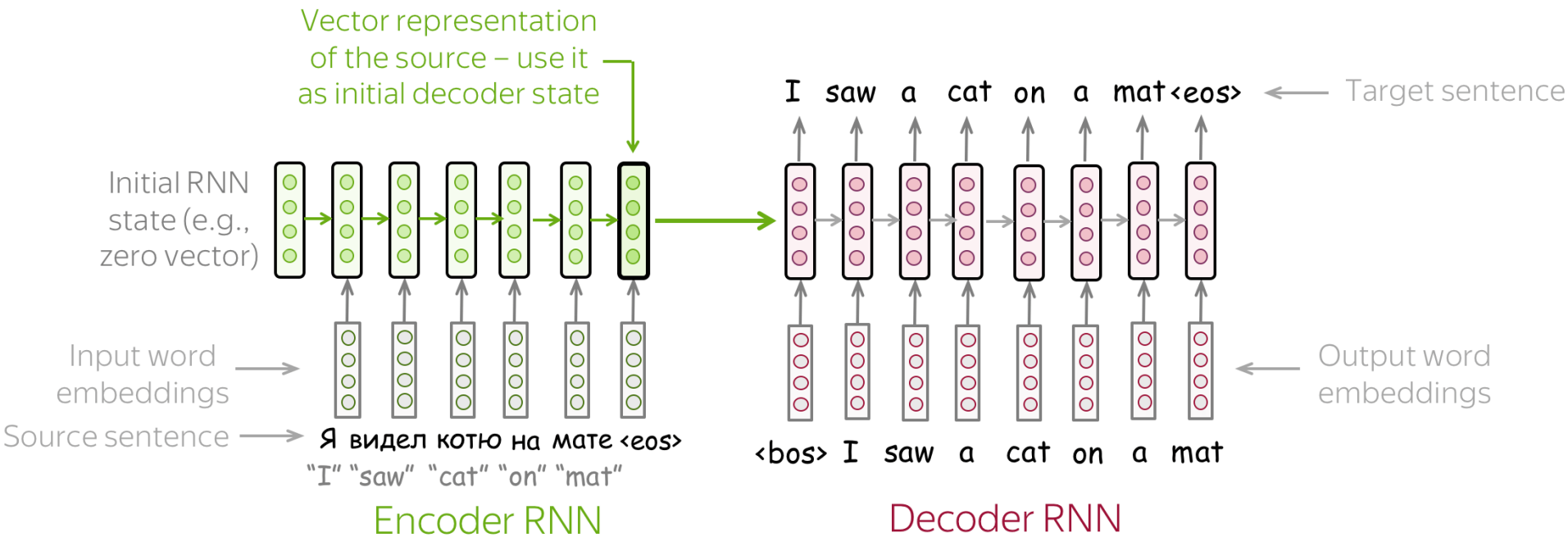
..

What can we do if they are so unreliable?

- Human evaluation (expensive, hard to relate to results of older experiments)
- Neural model-based metrics (e.g., BERTScore, GPTScore)
- Specialized metrics (e.g., FActScore for hallucinations)

Tokenization

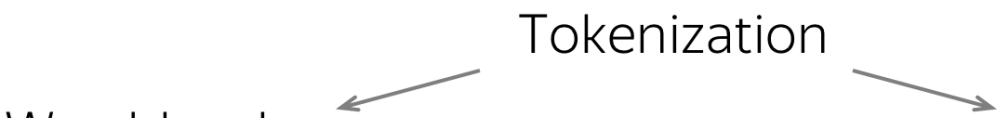
Tokenization



We considered tokenization of sentences into 'words'

(whatever we mean by a 'word')

Tokenization



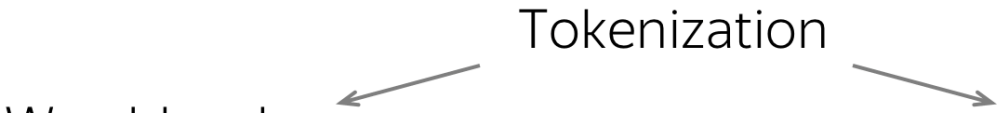
- fixed vocabulary
- can process only a fixed number of words

- open vocabulary
- rare and unknown tokens are encoded as sequences of subword units

Instead of '*unrelated*', we get two tokens 'un@@' '*related*'

why subword tokenization?

Tokenization



- fixed vocabulary
- can process only a fixed number of words

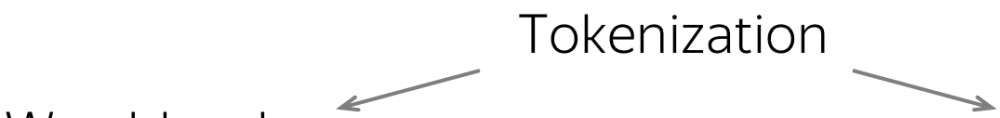
- open vocabulary
- rare and unknown tokens are encoded as sequences of subword units

Instead of '*unrelated*', we get two tokens 'un@@' '*related*'

why subword tokenization?

- reduces sparsity
- memory requirement
- results in a speeds-up (recall: softmax involves summation over all token types, few token typs -> faster computation)
- may provide an appropriate bias for *compositionality,* e.g., tokens for "unrelated" and "related" may be shared (e.g., include *related*)

Tokenization



- fixed vocabulary
- can process only a fixed number of words

- open vocabulary
- rare and unknown tokens are encoded as sequences of subword units

Instead of '*unrelated*', we get two tokens 'un@@' '*related*'

Especially crucial for morphologically-rich languages

Standard segmentation algorithms rely on character ngram frequency, not on morphology (e.g., Byte-Pair Encoding)

Used in virtually any modern neural model

Byte pair encoding algorithm

Byte Pair Encoding (BPE) algorithm (Gage, 1994) was applied to subword segmentation in machine translation by Sennrich, Haddow, and Birch (ACL 2015)

BPE, two phases

- Training: learn "BPE rules", i.e., which pairs of symbols to merge;
- Inference: apply learned rules to segment a text.

BPE: training

Start with individual characters as tokens

Repeat:

- count pairs of tokens: how many times each pair occurs together in the training data;
- find the most frequent pair of tokens;
- merge this pair - add a merge to the merge table, and the new token to the vocabulary.

In practice, the algorithm first counts how many times each word appeared in the data.

=> Using this information, one can count pairs of adjacent tokens more easily.

Note the tokens do not cross word boundary - everything happens within words.

BPE: training

Initial vocabulary:
characters

↓

Split each word
into characters

Words in the data:

word	count
c a t	4
m a t	5
m a t s	2
m a t e	3
a t e	3
e a t	2

Current merge table:

(empty)

BPE: training

Count pairs of tokens:

a + t: 20
m + a: 10
t + e: 7
c + a: 4
t + s: 2
e + a: 2

Words in the data:

word	count
c a t	4
m a t	5
m a t s	2
m a t e	3
a t e	3
e a t	2

Current merge table:

(empty)

BPE: training

Count pairs of tokens:

- a + t: 20
- m + a: 10
- t + e: 7
- c + a: 4
- t + s: 2
- e + a: 2

Add the most frequent pair to the merge table

Words in the data:

word	count
c a t	4
m a t	5
m a t s	2
m a t e	3
a t e	3
e a t	2

Current merge table:

a t -> at

BPE: training

Words in the data:

word	count
c a t	4
m a t	5
m a t s	2
m a t e	3
a t e	3
e a t	2

Apply the merge to the data →

Current merge table:

a t -> at

BPE: training

Words in the data:

word	count
c at	4
m at	5
m at s	2
m at e	3
at e	3
e at	2

Apply the merge to the data →

Current merge table:

a t -> at

BPE: training

Count pairs of new tokens:

m + at: 10
at + e: 7
c + at: 4
at + s: 2
e + at: 2

Words in the data:

word	count
c at	4
m at	5
m at s	2
m at e	3
at e	3
e at	2

Current merge table:

a t -> at

BPE: training

Count pairs of new tokens:

- m + at: 10
- at + e: 7
- c + at: 4
- at + s: 2
- e + at: 2

Add the most frequent pair to the merge table

Words in the data:

word	count
c at	4
m at	5
m at s	2
m at e	3
at e	3
e at	2

Current merge table:

- a t -> at
- m at -> mat

BPE: training

Words in the data:

word	count
c at	4
m at	5
m at s	2
m at e	3
at e	3
e at	2

Apply the merge to the data →

Current merge table:

a t -> at

m at -> mat

BPE: training

Words in the data:

word	count
c at	4
mat	5
mat s	2
mat e	3
at e	3
e at	2

Apply the merge to the data →

Current merge table:

a t -> at

m at -> mat

BPE: training

Count pairs of new tokens:

c + at: 4
at + e: 4
mat + e: 3
mat + s: 2
e + at: 2

Words in the data:

word	count
c at	4
mat	5
mat s	2
mat e	3
at e	3
e at	2

Current merge table:

a t -> at
m at -> mat

BPE: training

Count pairs of new tokens:

c + at: 4
at + e: 4
mat + e: 3
mat + s: 2
e + at: 2

Add the most frequent pair to the merge table

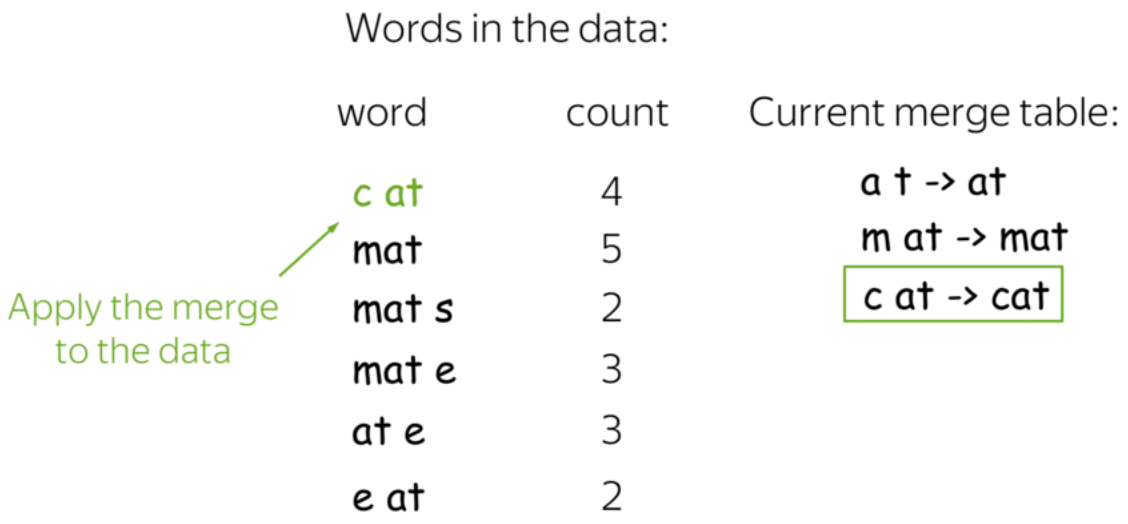
Words in the data:

word	count
c at	4
mat	5
mat s	2
mat e	3
at e	3
e at	2

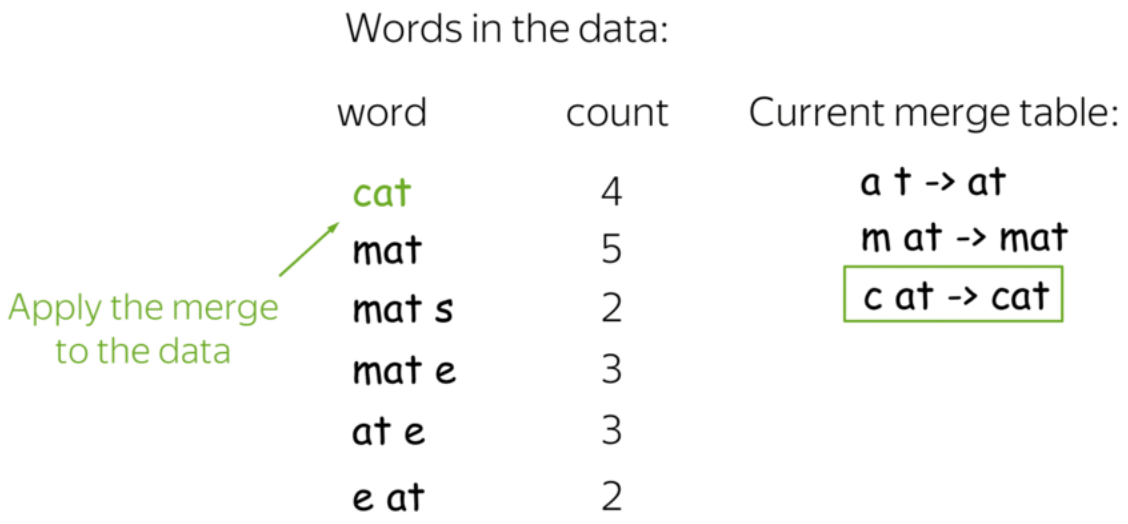
Current merge table:

a t -> at
m at -> mat
c at -> cat

BPE: training



BPE: training



BPE: training

Count pairs of new tokens:

at + e: 4
mat + e: 3
mat + s: 2
e + at: 2

Words in the data:

word	count
cat	4
mat	5
mat s	2
mat e	3
at e	3
e at	2

Current merge table:

a t -> at
m at -> mat
c at -> cat

BPE: training

Count pairs of new tokens:

at + e: 4
mat + e: 3
mat + s: 2
e + at: 2

Add the most frequent pair to the merge table

Words in the data:

word	count
cat	4
mat	5
mat s	2
mat e	3
at e	3
e at	2

Current merge table:

a t -> at
m at -> mat
c at -> cat
at e -> ate

BPE: training

Count pairs of new tokens:

at + e: 4
mat + e: 3
mat + s: 2
e + at: 2

Add the most frequent pair to the merge table

Words in the data:

word	count
cat	4
mat	5
mat s	2
mat e	3
at e	3
e at	2

Current merge table:

a t -> at
m at -> mat
c at -> cat
at e -> ate

BPE: training

Count pairs of new tokens:

mat + e: 3
mat + s: 2
e + at: 2

Words in the data:

word	count
cat	4
mat	5
mat s	2
mat e	3
ate	3
e at	2

Current merge table:

a t -> at
m at -> mat
c at -> cat
at e -> ate

BPE: training

Count pairs of new tokens:

mat + e: 3

mat + s: 2

e + at: 2

Add the most frequent pair to the merge table

Words in the data:

word	count
cat	4
mat	5
mat s	2
mat e	3
ate	3
e at	2

Current merge table:

a t -> at

m at -> mat

c at -> cat

at e -> ate

mat e -> mate

BPE: training

Words in the data:

word	count
cat	4
mat	5
mat s	2
mat e	3
ate	3
e at	2

Apply the merge to the data

Current merge table:

a t -> at
m at -> mat
c at -> cat
at e -> ate
mat e -> mate

BPE: training

Words in the data:

word	count
cat	4
mat	5
mat s	2
mate	3
ate	3
e at	2

Apply the merge to the data

Current merge table:

a t -> at
m at -> mat
c at -> cat
at e -> ate
mat e -> mate

BPE: training

Reached maximum number of merges

↓

stop

Words in the data:

word	count
cat	4
mat	5
mat s	2
mate	3
ate	3
e at	2

Current merge table:

a t -> at
m at -> mat
c at -> cat
at e -> ate
mat e -> mate

BPE: training

Note!

mat and **mat@@** are different tokens!

word	→	segmentation
cat	→	cat
mat	→	mat
mat s	→	mat@@ s
mate	→	mate
ate	→	ate
e at	→	e@@ at

if a word is not "glued" together, add @@

Final merge table:

```
a t -> at
m at -> mat
c at -> cat
at e -> ate
mat e -> mate
```

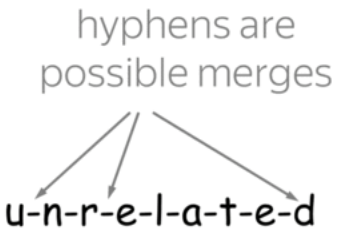
BPE: inference

After training, we are left with a vocabulary and a merge tables (ordered set of merges)

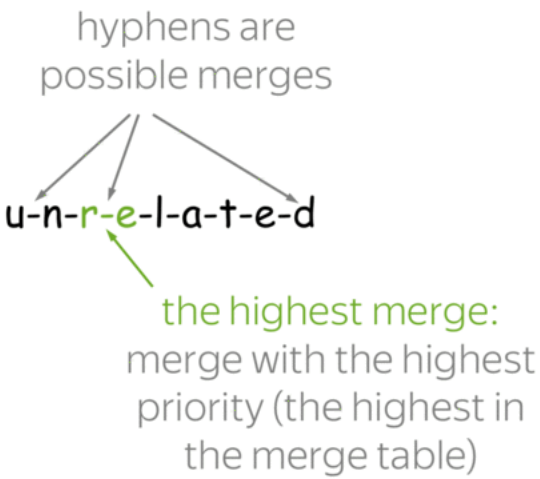
The algorithm starts with segmenting a word into a sequence of characters. After that, it iteratively makes the following two steps until no merge it possible:

- among all possible merges at this step, find the highest merge in the table (highest -> more frequent);
- apply this merge.

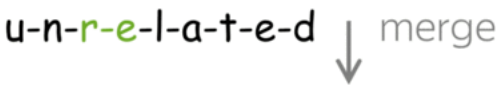
BPE: inference



BPE: inference



BPE: inference

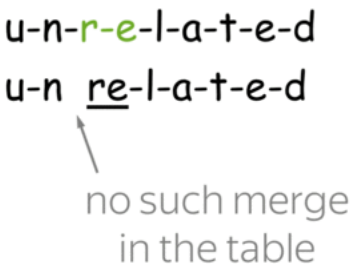


BPE: inference

u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

BPE: inference



BPE: inference

u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

pick the highest merge

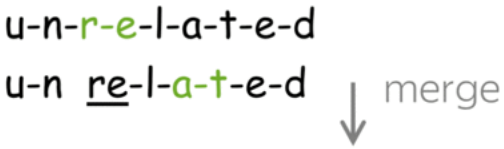
BPE: inference

u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

pick the highest merge

BPE: inference



BPE: inference

u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

u-n re-l-at-e-d

BPE: inference

u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

u-n re-l-at-e-d

pick the highest merge

BPE: inference

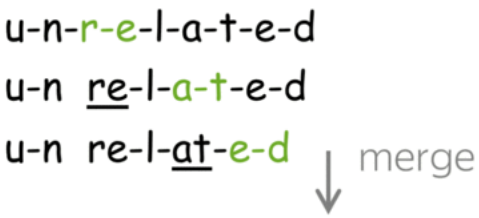
u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

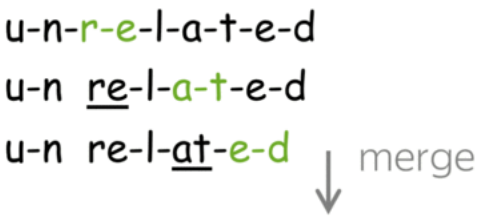
u-n re-l-at-e-d

pick the highest merge

BPE: inference



BPE: inference



BPE: inference

u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

u-n re-l-at-e-d

u-n re-l-at-ed

pick the highest merge

BPE: inference

u-n-r-e-l-a-t-e-d

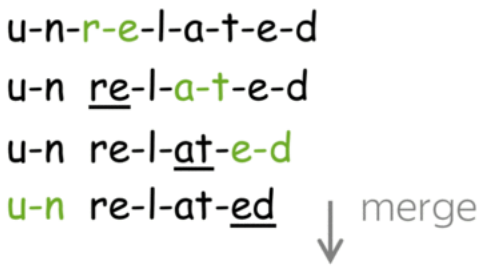
u-n re-l-a-t-e-d

u-n re-l-at-e-d

u-n re-l-at-ed

pick the highest merge

BPE: inference



BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed

BPE: inference

u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

u-n re-l-at-e-d

u-n re-l-at-ed

un re-l-at-ed

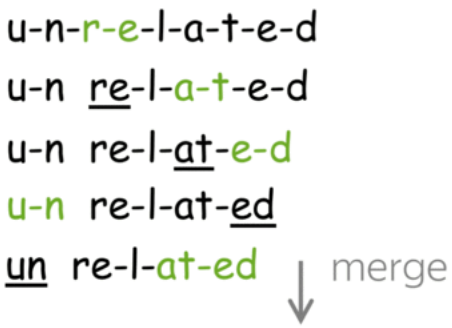
pick the highest merge

BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed

pick the highest merge

BPE: inference



BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed
un re-l-ated

BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed
un re-l-ated

BPE: inference

u-n-r-e-l-a-t-e-d

u-n re-l-a-t-e-d

u-n re-l-at-e-d

u-n re-l-at-ed

un re-l-at-ed

un re-l-ated

pick the highest merge

BPE: inference

BPE: inference

BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed
un re-l-ated
un rel-ated

BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed
un re-l-ated
un rel-ated

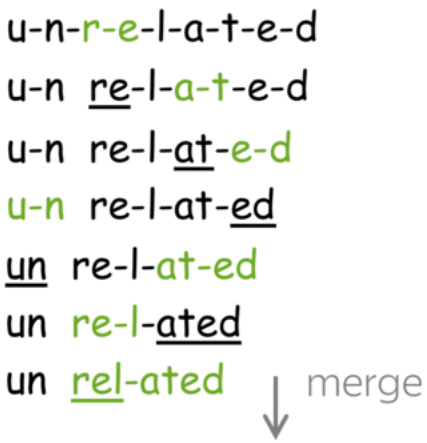
pick the highest merge

BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed
un re-l-ated
un rel-ated

pick the highest merge

BPE: inference



BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed
un re-l-ated
un rel-ated
un related

BPE: inference

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed
un re-l-ated
un rel-ated
un related
un@@ related

No merges possible -> end

BPE – drawbacks / discussion

- The induced segmentation does not align with linguistic morphological boundaries.
- Frequent words that share the same morphemes (e.g., roots) are segmented differently from less frequent words.
- The segmentation does not directly correspond to the information content of a token or the computational effort required in a neural model.

What can we do about it?

Summary

- Done with inference algorithms (greedy, beam-search, sampling, temperature,…)
- Evaluating text generation (e.g., BLEU)
- Subword tokenization