Jeszcze o tokenach. Sampling. Modele N-gramowe

Paweł Rychlikowski

Instytut Informatyki UWr

28 października 2024

Język [C]hiński

Pytanie

Co łączy język C i język chiński?

1. Choć można na oba patrzeć jako na ciągi znaków...
2. to w analizie wygodnie jest wyodrębnić **tokeny**.
3. Nie możemy posiłkować się (tylko) spacjami (bo np. w chińskim ich nie ma)

Pytanie

Jak przeprowadzić tokenizację tekstu?

C vs. Chiński

- Referencyjny algorytm tokenizecji dla chińskiego to algorytm **MaxMatch** (taki sam jak dla C)
- Czyli: pierwszym tokenem tekstu jest najdłuższy jego prefiks, który jest zarazem poprawnym tokenem.
- W *bezspacjowym* angielskim działa źle (w polskim?)
  - wecanonlyseeashortdistanceahead

C vs. Chiński

- Referencyjny algorytm tokenizecji dla chińskiego to algorytm **MaxMatch** (taki sam jak dla C)
- Czyli: pierwszym tokenem tekstu jest najdłuższy jego prefiks, który jest zarazem poprawnym tokenem.
- W *bezspacjowym* angielskim działa źle (w polskim?)
  - we canon l y see ash ort distance ahead

MaxMatch dla polskiego

litwoojczyznomojatyjesteśjakzdrowie

MaxMatch dla polskiego

litwo ojczyznom oj a tyje s te ś jak zdrowie

Jeszcze o tokenach

- Język C ma bardzo ściśle zdefiniowane tokeny (i jednoznaczną tokenizację).
- Dla języków naturalnych nie ma tak dobrze:
  - niebiesko-czarni (1 czy 3 tokeny?)
  - F-16 (podobnie)
  - m.in. (1, 2, czy 4 tokeny?)

Pewne decyzje są arbitralne (i trzeba się z tym pogodzić) – ponadto drobne „błędy” tokenizacji da się naprawić na dalszych etapach analizy tekstu.

Jeszcze o tokenach (2)

Czasem pomija się tokenizację, traktując język np. jako:

- Ciąg znaków (ASCII, Unicode)
- Ciąg bajtów (kodowanie utf-8)

Uwaga

Jak uczymy model od zera, nie należy bać sie własnej tokenizacji, wykorzystującej naszą wiedzę o dziedzinie:

- Jak stokenizować DNA?
- Jak stokenizować partie szachów?
- Czasem wiedza lingwistyczna daje lepszą tokenizację niż generyczne algorytmy.

Bytes Pair Encoding

W wielkim skrócie:

a) Liczymy słowa w korpusie (czyli dużym, reprezentatywnym, zbiorze tekstów)
b) W słowach liczymy częstości par liter
   - Jeżeli abrakadabra występowało 15 razy, to zwiększamy licznik **ra** o 30.
c) Zamieniamy najczęstszą parę na nową (pseudo)literę
d) Czynności powtarzamy aż do otrzymania pożądanej liczby pseudoliter.

Każde słowo reprezentujemy jako ciąg pseudoliter (szczegóły na kolejych sjaldach).

Byte Pair Encoding

- Originally a compression algorithm:
  - Most frequent byte pair $\mapsto$ a new byte.

Replace bytes with character ngrams

(though, actually, some people have done interesting things with bytes)

Rico Sennrich, Barry Haddow, and Alexandra Birch. ***Neural Machine Translation of Rare Words with Subword Units****. ACL 2016.*
https://arxiv.org/abs/1508.07909
https://github.com/rsennrich/subword-nmt
https://github.com/EdinburghNLP/nematus

18

Byte Pair Encoding

- A word segmentation algorithm:
    - Though done as bottom up clusering
  - Start with a unigram vocabulary of all (Unicode) characters in data
  - Most frequent ngram pairs ↦ a new ngram

19

Byte Pair Encoding

- A word segmentation algorithm:
  - Start with a vocabulary of characters
  - Most frequent ngram pairs ↦ a new ngram

Dictionary

5	l o w
2	l o w e r
6	n e w e s t
3	w i d e s t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d

Start with all characters in vocab

20

(Example from Sennrich)

Byte Pair Encoding

- A word segmentation algorithm:
  - Start with a vocabulary of characters
  - Most frequent ngram pairs ↦ a new ngram

Dictionary

5	l o w
2	l o w e r
6	n e w **es** t
3	w i d **es** t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, **es**

Add a pair (e, s) with freq 9

21

(Example from Sennrich)

Byte Pair Encoding

- A word segmentation algorithm:
  - Start with a vocabulary of characters
  - Most frequent ngram pairs ↦ a new ngram

Dictionary

5	l o w
2	l o w e r
6	n e w **est**
3	w i d **est**

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, **est**

Add a pair (es, t) with freq 9

22

(Example from Sennrich)

Byte Pair Encoding

- A word segmentation algorithm:
  - Start with a vocabulary of characters
  - Most frequent ngram pairs ↦ a new ngram

Dictionary

5	**lo** w
2	**lo** w e r
6	n e w est
3	w i d est

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, **lo**

Add a pair (l, o) with freq 7

23

(Example from Sennrich)

Byte Pair Encoding

- Have a target vocabulary size and stop when you reach it
- Do deterministic longest piece segmentation of words
- Segmentation is only within words identified by some prior tokenizer (commonly Moses tokenizer for MT)
- Automatically decides vocab for system
  - No longer strongly "word" based in conventional way

Top places in WMT 2016!
Still widely used in WMT 2018

24

https://github.com/rsennrich/nematus

Modele N-gramowe

Definicja

N-gramem nazywamy ciąg kolejnych słów o długości N. 1-gramy to unigramy, 2-gramy to bigramy, 3-gramy to trigramy.

Za pomocą N-gramów tworzymy model języka, w którym staramy się przewidzieć kolejne słowo (N-te) na podstawie $N-1$ słów poprzednich.

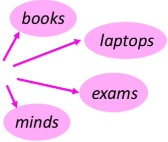
Uwaga

Na kolejnych slajdach (z wykładu na Stanfordzie, CS/...) powiemy krótko o modelach n-gramowych i próbkowaniu.

Language Modeling

- **Language Modeling** is the task of predicting what word comes next

the students opened their ______



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**

19

Language Modeling

- You can also think of a Language Model as a system that assigns a probability to a piece of text
- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) &= P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} \mid \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} \mid \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)}) \\ &= \prod_{t=1}^{T} \underbrace{P(\boldsymbol{x}^{(t)} \mid \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}} \end{aligned}$$

n-gram Language Models

- First we make a Markov assumption: $x^{(t+1)}$ depends only on the preceding n-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$ (assumption)

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}}$$ (definition of conditional prob)

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\dots,\boldsymbol{x}^{(t-n+2)})}$$ (statistical approximation)

24

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the~~ *students opened their* ______

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count(students opened their } \boldsymbol{w})}{\text{count(students opened their)}}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

25

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then w has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w})}}{\boxed{\text{count}(\text{students opened their})}}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* w!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing n makes sparsity problems *worse*. Typically, we can't have n bigger than 5.

26

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then w has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count(students opened their } \boldsymbol{w})}}{\boxed{\text{count(students opened their)}}}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* w!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse*. Typically, we can't have *n* bigger than 5.

26

Storage Problems with n-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w})}}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

27

n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

today the _______

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

Sparsity problem: not much granularity in the probability distribution

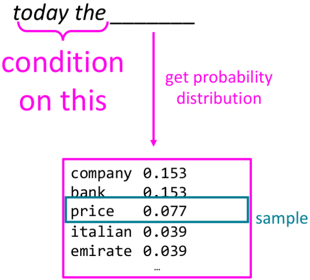
Otherwise, seems reasonable!

*** Try for yourself:** https://nlpforhackers.io/language-models/

28

Generating text with a n-gram Language Model

You can also use a Language Model to generate text



29

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today $\underbrace{\textit{the price}}_{\text{condition on this}}$ ________

get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

sample

30

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the $\underbrace{\textit{price of}}_{\text{condition on this}}$ ________

get probability distribution

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
…	

sample

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

…but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing *n* worsens sparsity problem, and increases model size…

32

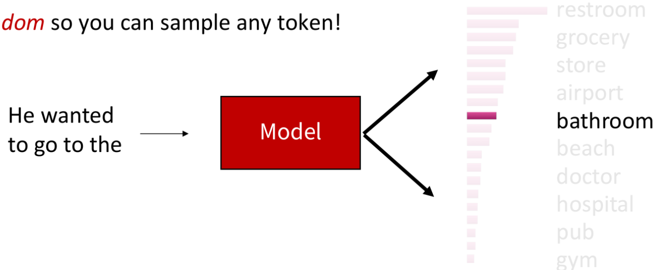
- Popatrzmy na generację w modelu Papuga
- (będziemy pokazywać 10 najbardziej prawdopodobnych opcji)

Time to get random : Sampling!

- Sample a token from the distribution of tokens

$$\hat{y}_t \sim P(y_t = w \mid \{y\}_{<t})$$

- It's *random* so you can sample any token!



30

Decoding: Top-*k* sampling

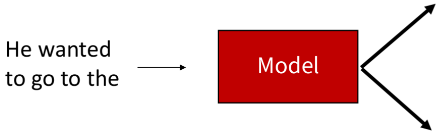
- Problem: Vanilla sampling makes every token in the vocabulary an option
  - Even if most of the probability mass in the distribution is over a limited set of options, the tail of the distribution could be very long and in aggregate have considerable mass (statistics speak: we have "heavy tailed" distributions)
  - Many tokens are probably *really wrong* in the current context
  - Why are we giving them *individually* a tiny chance to be selected?
  - Why are we giving them *as a group* a high chance to be selected?

- Solution: Top-*k* sampling
  - Only sample from the top *k* tokens in the probability distribution

(Fan et al., ACL 2018; Holtzman et al., ACL 2018)

Decoding: Top-*k* sampling

- <u>Solution:</u> Top-*k* sampling
  - Only sample from the top *k* tokens in the probability distribution
  - Common values are *k* = 5, 10, 20 (*but it's up to you!*)

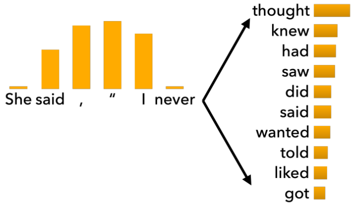


  - Increase *k* for more **diverse**/**risky** outputs
  - Decrease *k* for more **generic**/**safe** outputs

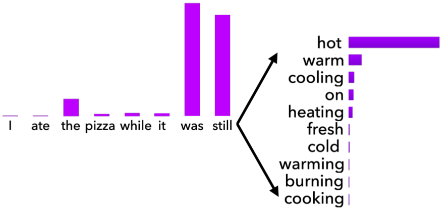
32

(Fan et al., ACL 2018; Holtzman et al., ACL 2018)

Issues with Top-*k* sampling



Top-*k* sampling can cut off too ***quickly***!



Top-*k* sampling can also cut off too ***slowly***!

33

(Holtzman et. al., ICLR 2020)

Decoding: Top-*p* (nucleus) sampling

- Problem: The probability distributions we sample from are dynamic
  - When the distribution P_t is flatter, a limited *k* removes many viable options
  - When the distribution P_t is peakier, a high *k* allows for too many options to have a chance of being selected

- Solution: Top-*p* sampling
  - Sample from all tokens in the top *p* cumulative probability mass (i.e., where mass is concentrated)
  - Varies *k* depending on the uniformity of P_t

34

(Holtzman et. al., ICLR 2020)

Decoding: Top-*p* (nucleus) sampling

- Solution: Top-*p* sampling
  - Sample from all tokens in the top *p* cumulative probability mass (i.e., where mass is concentrated)
  - Varies *k* depending on the uniformity of P_t

$P_t^1(y_t = w \mid \{y\}_{<t})$ $\quad$ $P_t^2(y_t = w \mid \{y\}_{<t})$ $\quad$ $P_t^3(y_t = w \mid \{y\}_{<t})$

35

(Holtzman et. al., ICLR 2020)

Scaling randomness: Softmax temperature

- Recall: On timestep t, the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$

$$P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- You can apply a *temperature hyperparameter* τ to the softmax to rebalance P_t:

$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - **More** diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - **Less** diverse output (probability is concentrated on top words)

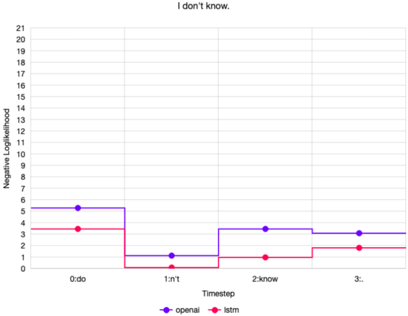
Note: softmax temperature is not a decoding algorithm!

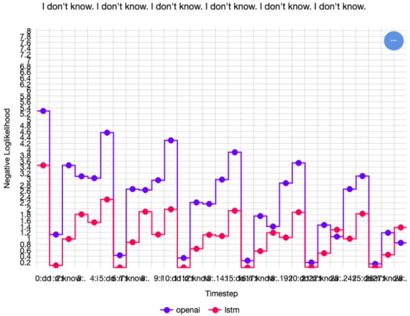
It's a technique you can apply at test time, in conjunction with a decoding algorithm (such as beam search or sampling)

Powtarzalność

- Problemem w generacji jest powtarzalność (to znaczy, że model generuje powtarzające się ciągi)
- Spróbujmy zaobserwować ten fenomen w przypadku papugi.

Why does repetition happen?

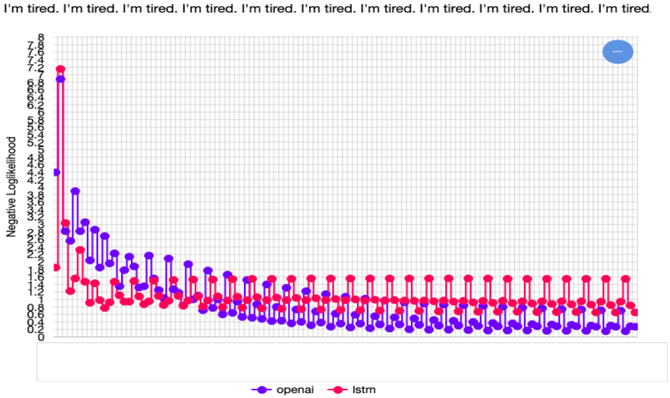




26

(Holtzman et. al., ICLR 2020)

And it keeps going...



27

(Holtzman et. al., ICLR 2020)