

Few-shot learning, tokeny i dalsze kwestie wstępne

Paweł Rychlikowski

Instytut Informatyki UWr

15 października 2025

Przypomnienie: 4 poziomy modelu językowego

- **Poziom 0:** aplikacja
- **Poziom 1:** API generujące teksty
- **Poziom 2:** rozkład prawdopodobieństwa na tokenach
- **Poziom 3:** sieć neuronowa

Przypomnienie: 4 poziomy modelu językowego

- **Poziom 0:** aplikacja
- **Poziom 1:** API generujące teksty
- **Poziom 2:** rozkład prawdopodobieństwa na tokenach
- **Poziom 3:** sieć neuronowa

Jeszcze o few-shots learning

(ważna technika z poziomu 1)

Few shots-learning in LMs

Główna idea:

- Wybrać kilka **demonstracji**
- i wkleić je do prompta.

Jeszcze o few-shots learning

(ważna technika z poziomu 1)

Few shots-learning in LMs

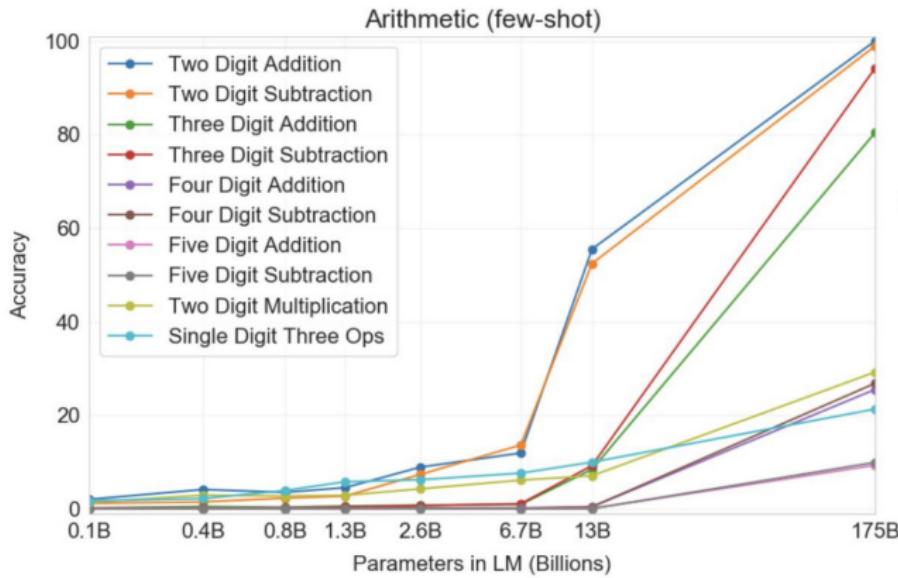
Główna idea:

- Wybrać kilka **demonstracji**
- i wkleić je do prompta.

Kilka pytań:

- Jak wybrać przykłady do promptu (czy wszystkie?)
- Czy warto oprócz przykładów podać opis zadania?
- Jaka kolejność przykładów?
- Czy formatowanie ma znaczenie?

Pushing GPT-3 Further: Arithmetic

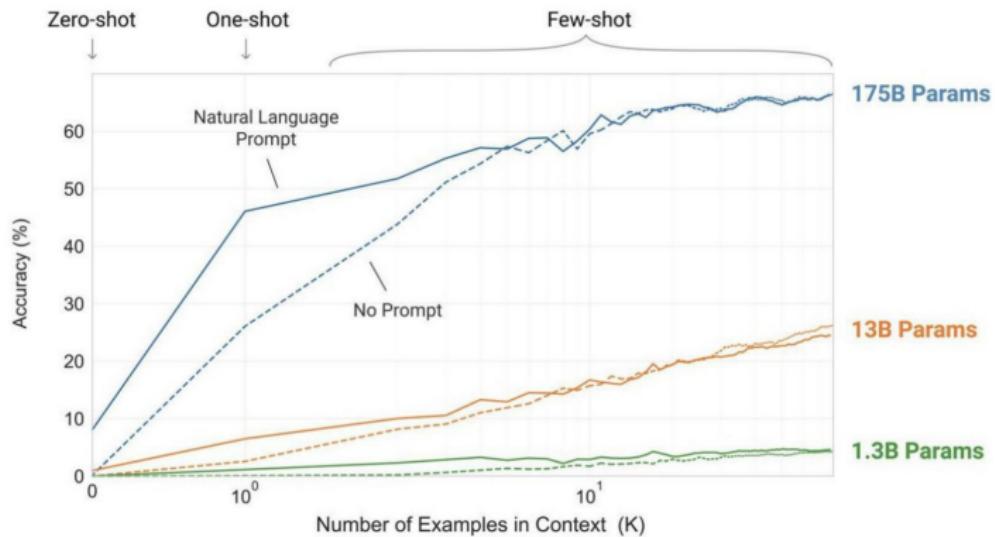


Observations:

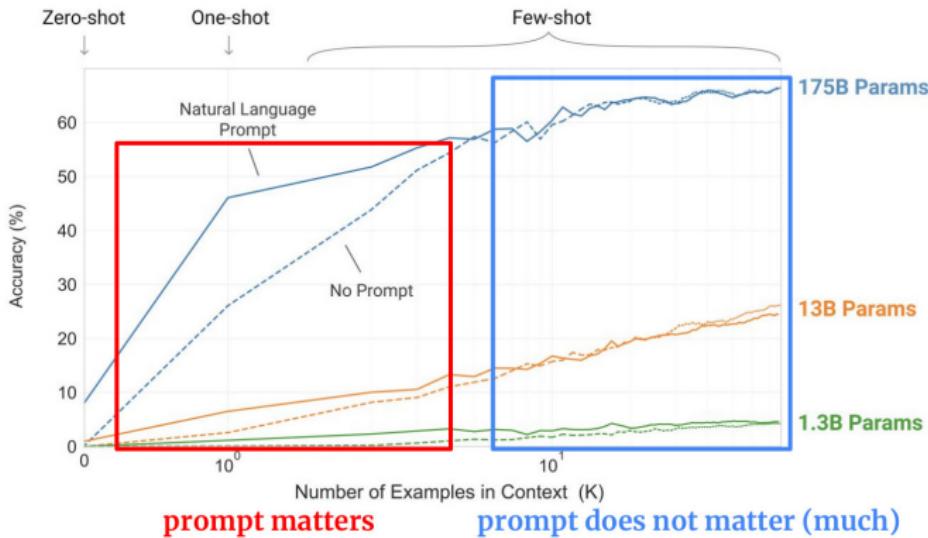
1. Scale is important!
2. >1 operation or >3 digit numbers are much harder

Q: What is 48 plus 76?
A: 124.

Emergent Capability - In-Context Learning



Larger Models Learn Better In-Context



Demonstracje pasujące do konkretnego przypadku wejściowego

- Prosta intuicja: przykłady powinny być podobne *wejścia*

Demonstracje pasujące do konkretnego przypadku wejściowego

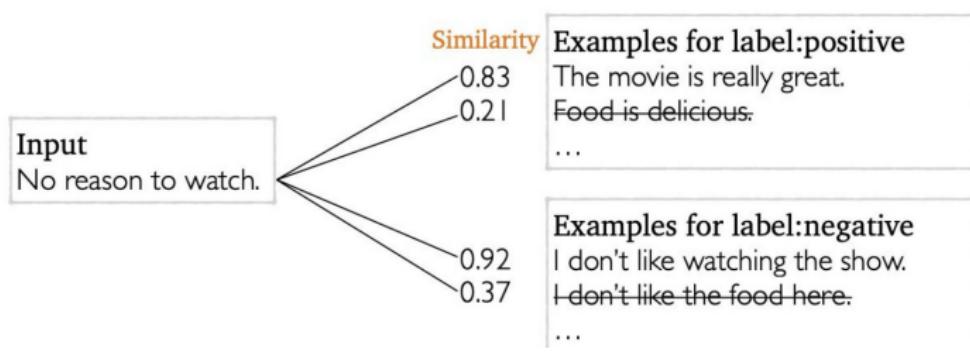
- Prosta intuicja: przykłady powinny być podobne *wejścia*
- Jak mierzyć podobieństwo – o tym będzie cały wykład, na razie możemy przyjąć, że jest to dowolna heurystyczna procedura, zliczająca powtarzające się wyrazy, mierząca odległość edycyjną, patrzącą na początek pytania ([W którym roku](#)), etc

Demonstracje pasujące do konkretnego przypadku wejściowego

- Prosta intuicja: przykłady powinny być podobne *wejścia*
- Jak mierzyć podobieństwo – o tym będzie cały wykład, na razie możemy przyjąć, że jest to dowolna heurystyczna procedura, zliczająca powtarzające się wyrazy, mierząca odległość edycyjną, patrzącą na początek pytania ([W którym roku](#)), etc
- Dla zaawansowanych: podobieństwo cosinusowe osadzeń wyliczonych przez pretrenowany model typu BERT

Demonstracje pasujące do konkretnego przypadku wejściowego

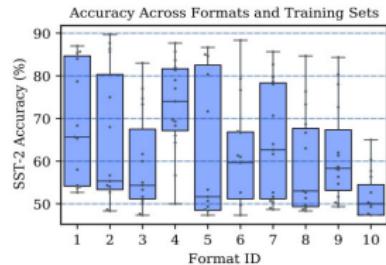
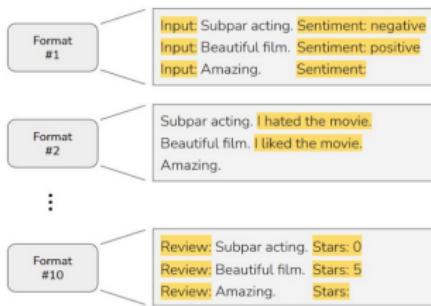
- Prosta intuicja: przykłady powinny być podobne *wejścia*
- Jak mierzyć podobieństwo – o tym będzie cały wykład, na razie możemy przyjąć, że jest to dowolna heurystyczna procedura, zliczająca powtarzające się wyrazy, mierząca odległość edycyjną, patrzącą na początek pytania ([W którym roku](#)), etc
- Dla zaawansowanych: podobieństwo cosinusowe osadzeń wyliczonych przez pretrenowany model typu BERT



How important is the structure of the prompt for in-context learning?

Components of a prompt

1. **Prompt format**
2. Training example selection
3. Training example permutation

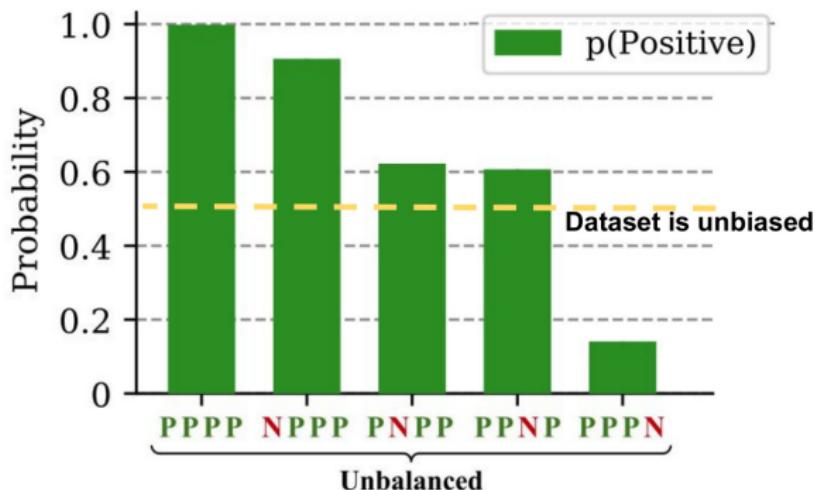


In-context learning is highly sensitive to prompt format

What causes this sensitivity?

Three main reasons

1. Majority label bias
2. Common token bias
3. Recency bias

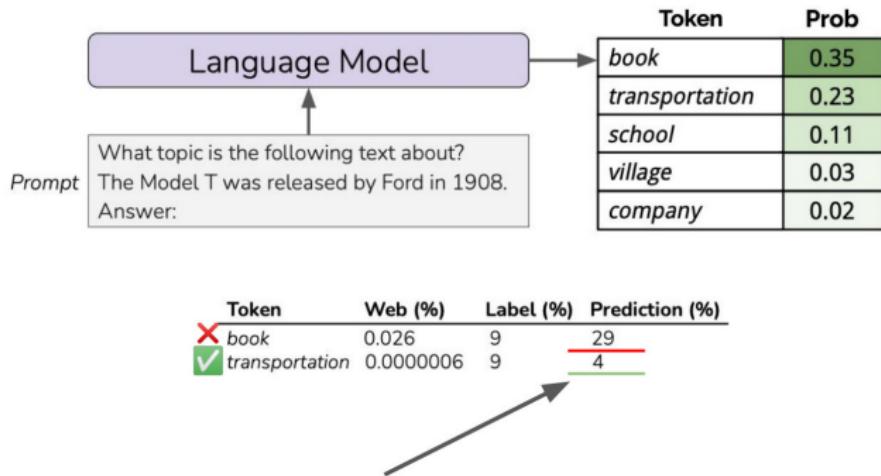


1. Model prefers to predict positive when the majority labels is "P/Positive"
2. Surprising because the validation dataset is balanced!

What causes this sensitivity?

Three main reasons

1. Majority label bias
2. **Common token bias**
3. Recency bias

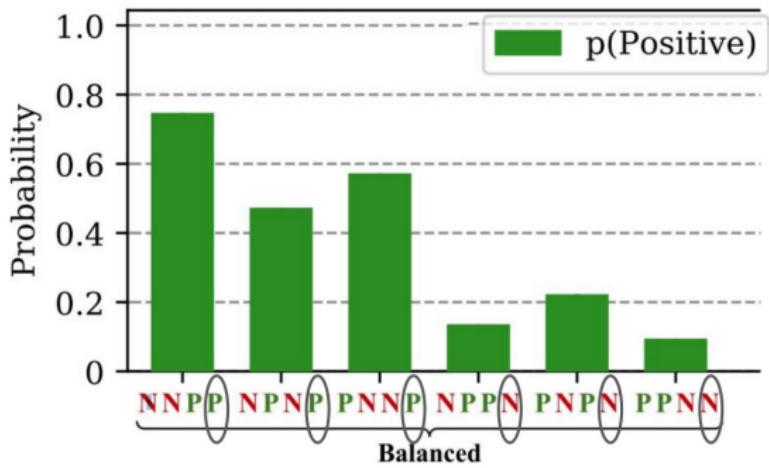


Model is biased towards predicting the incorrect frequent token "book" even when both "book" and "transportation" are equally likely labels in the dataset

What causes this sensitivity?

Three main reasons

1. Majority label bias
2. Common token bias
3. Recency bias



1. Model is heavily biased towards the most recent label
2. Again, dataset is balanced!

Przypomnienie: 4 poziomy modelu językowego

- **Poziom 0:** aplikacja
- **Poziom 1:** API generujące teksty
- **Poziom 2:** rozkład prawdopodobieństwa na tokenach
- **Poziom 3:** sieć neuronowa

Tokeny

- Prawdopodobieństwo sekwencji tokenów można obliczyć następująco:

$$P(w_1 \dots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \dots P(w_n|w_1 \dots w_{n-1})$$

Tokeny

- Prawdopodobieństwo sekwencji tokenów można obliczyć następująco:

$$P(w_1 \dots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \dots P(w_n|w_1 \dots w_{n-1})$$

Ale czym są te tokeny?

Tokenizacja

Definicja

Tokenizacja jest zamianą ciągu znaków na odpowiadający mu ciąg tokenów.

Decyzja, czy dany串 jest jednym tokenem, czy wymaga podziału nie zawsze jest oczywista!

Tradycyjna tokenizacja w NLP

Wariant 1

Wykonujemy operację `split` na każdym wierszu

Wada: Przyklejona interpunkcja!

Tradycyjna tokenizacja w NLP

Wariant 1

Wykonujemy operację `split` na każdym wierszu

Wada: Przyklejona interpunkcja!

Wariant 2

Każdy znak interpunkcyjny otaczamy (wirtualnie) spacjami, następnie wykonujemy operację `split`.

- For every punctuation character c, do
`s = s.replace(c, ' ' + c + ' ')`
- Return `s.split()` or `s.lower().split()`

Tradycyjna tokenizacja w NLP

Wariant 1

Wykonujemy operację `split` na każdym wierszu

Wada: Przyklejona interpunkcja!

Wariant 2

Każdy znak interpunkcyjny otaczamy (wirtualnie) spacjami, następnie wykonujemy operację `split`.

- For every punctuation character c, do
`s = s.replace(c, ' ' + c + ' ')`
- Return `s.split()` or `s.lower().split()`

Wada (?): yahoo! albo F-16

Tokenizacja (cd)

Wariant 3

Uznajemy, że ktoś to rozwiązał i znajdujemy bibliotekę (np. [NLTK](#), [spaCy](#), również frameworki neuronowe jak Pytorch i Keras), czy biblioteka [transformers](#) i korzystamy z bibliotecznego tokenizatora

Poziom 2 (przypomnienie)

- Prawdopodobieństwo sekwencji tokenów można obliczyć następująco:

$$P(w_1 \dots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\dots P(w_n|w_1 \dots w_{n-1})$$

Popatrzmy na kod obliczający prawdopodobieństwo tekstu.

Ważna uwaga

Ten kod **jest deterministyczny!**

Wykorzystanie prawdopodobieństwa zdania

Gdzie może być wykorzystane:

- Zadania klasyfikacji: co jest bardziej prawdopodobne?
[tekst-opinii-klienta] **Polecam!**
[tekst-opinii-klienta] **Nie polecam!**

Wykorzystanie prawdopodobieństwa zdania

Gdzie może być wykorzystane:

- Zadania klasyfikacji: co jest bardziej prawdopodobne?
[tekst-opinii-klienta] **Polecam!**
[tekst-opinii-klienta] **Nie polecam!**
- Problemy do rozwiązania:
 - ▶ dłuższe teksty mają 'pod górkę'.
 - ▶ Nawet jak teksty są równej długości, to któryś z nich jest bardziej prawdopodobny

Wykorzystanie prawdopodobieństwa zdania

Gdzie może być wykorzystane:

- Zadania klasyfikacji: co jest bardziej prawdopodobne?
[tekst-opinii-klienta] **Polecam!**
[tekst-opinii-klienta] **Nie polecam!**
- Problemy do rozwiązania:
 - ▶ dłuższe teksty mają 'pod górkę'.
 - ▶ Nawet jak teksty są równej długości, to któryś z nich jest bardziej prawdopodobny
- Przykładowe rozwiązanie: ustalić optymalny próg na różnice log-prawdopodobieństw

Jeszcze o tokenach

Czasem pomija się tokenizację, traktując język np. jako:

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

Jeszcze o tokenach

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 się własnej tokenizacji, wykorzystującej naszą wiedzę o dziedzinie:

- Jak stokenizować DNA?
- Jak stokenizować wzory chemiczne?
- Jak stokenizować partie szachów?

Jeszcze o tokenach

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 się własnej tokenizacji, wykorzystującej naszą wiedzę o dziedzinie:

- Jak stokenizować DNA?
- Jak stokenizować wzory chemiczne?
- Jak stokenizować partie szachów?
- Czasem wiedza lingwistyczna daje lepszą tokenizację niż generyczne algorytmy (dla standardowych tekstów, nie dla DNA).

Bytes Pair Encoding

W wielkim skrócie:

Bytes Pair Encoding

W wielkim skrócie:

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

Każde słowo reprezentujemy jako ciąg pseudoliter (szczegóły na kolejnych slideach).

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>

Byte Pair Encoding

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

Byte Pair Encoding

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

Dictionary

5 low
2 lower
6 newest
3 widest

Vocabulary

I, 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 \mapsto a new ngram

Dictionary

5 low
2 lower
6 newes t
3 wid es t

Vocabulary

I, o, w, e, r, n, w, s, t, i, d, es

Add a pair (e, s) with freq 9

(Example from Sennrich)

Byte Pair Encoding

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

Dictionary

5 low
2 lower
6 new est
3 wid est

Vocabulary

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

Add a pair (es, t) with freq 9

(Example from Sennrich)

Byte Pair Encoding

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

Dictionary

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

Vocabulary

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

Add a pair (l, o) with freq 7

(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.

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.

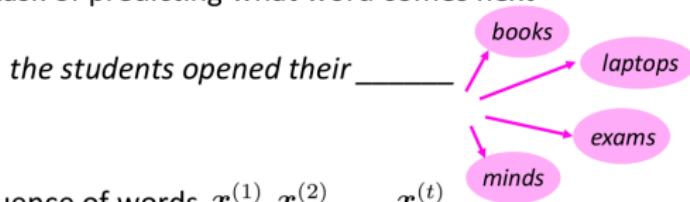
Za pomocą N-gramów tworzymy model języka, w którym staramy się przewidzieć kolejne słowo (N -te) na podstawie $N - 1$ słów poprzednich.

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



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

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$$

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

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

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 $x^{(1)}, \dots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(x^{(1)}, \dots, x^{(T)}) &= P(x^{(1)}) \times P(x^{(2)} | x^{(1)}) \times \dots \times P(x^{(T)} | x^{(T-1)}, \dots, x^{(1)}) \\ &= \prod_{t=1}^T P(x^{(t)} | x^{(t-1)}, \dots, x^{(1)}) \\ &\quad \underbrace{\qquad\qquad\qquad}_{\text{This is what our LM provides}} \end{aligned}$$

n-gram Language Models

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

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)}) = P(\mathbf{x}^{(t+1)} | \underbrace{\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}}_{n-1 \text{ words}}) \quad (\text{assumption})$$

$$\begin{aligned} & \xrightarrow{\text{prob of a n-gram}} P(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}) \\ & = \frac{P(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})}{P(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})} \quad (\text{definition of conditional prob}) \\ & \xrightarrow{\text{prob of a (n-1)-gram}} P(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}) \end{aligned}$$

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

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

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

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

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
- “students opened their books” occurred 400 times
 - $P(\text{books} | \text{students opened their}) = 0.4$
- “students opened their exams” occurred 100 times
 - $P(\text{exams} | \text{students opened their}) = 0.1$

 Should we have discarded
the “proctor” context?

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*.

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

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.

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*.

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

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.

Storage Problems with n-gram Language Models

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

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

Increasing n or increasing corpus increases model size!

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*

today the _____

Business and financial news

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
...	

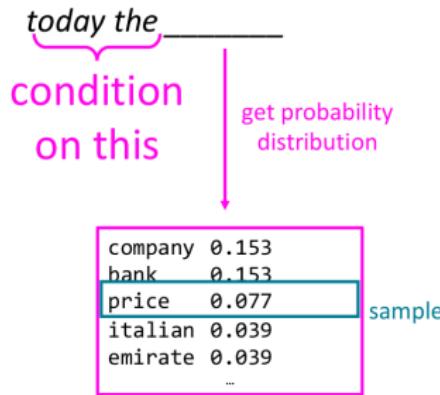
Sparsity problem:
not much granularity
in the probability
distribution

Otherwise, seems reasonable!

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

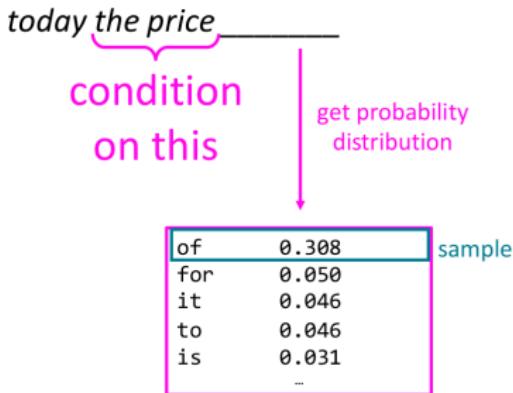
Generating text with a n-gram Language Model

You can also use a Language Model to generate text



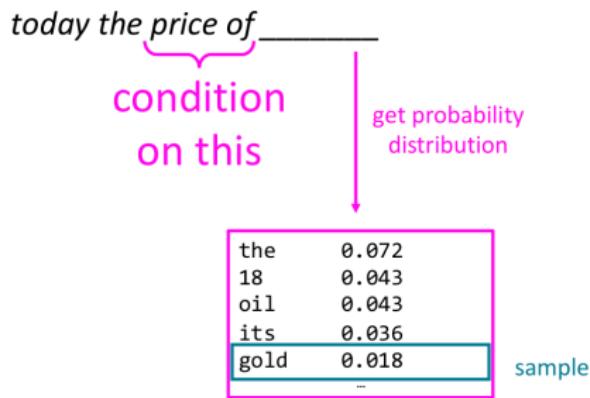
Generating text with a n-gram Language Model

You can also use a Language Model to generate text



Generating text with a n-gram Language Model

You can also use a Language Model to generate text



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...

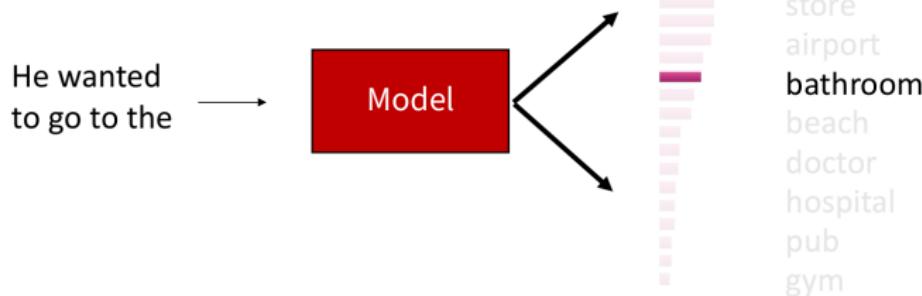
- Popatrzmy na generację w modelu Papuga
- (będziemy pokazywać 10 najbardziej prawdopodobnych opcji)

Time to get random : Sampling!

- Sample a token from the distribution of tokens

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

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

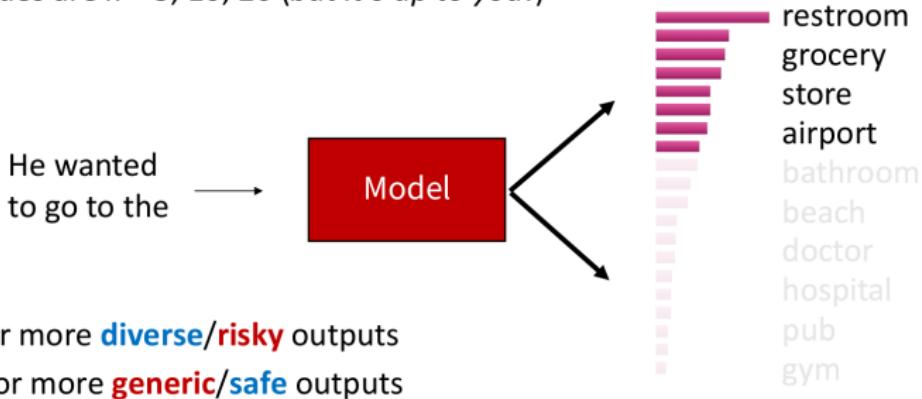


Decoding: Top- k sampling

- 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

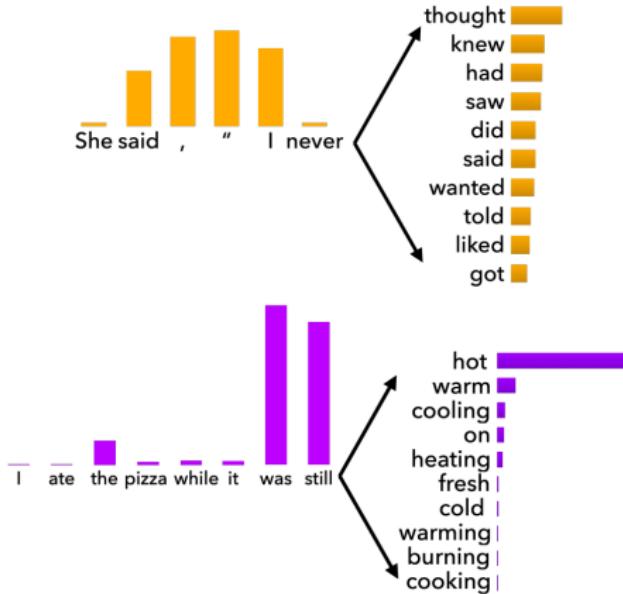
Decoding: Top- k sampling

- Solution: 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

Issues with Top- k sampling



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

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

(Holtzman et. al., ICLR 2020)

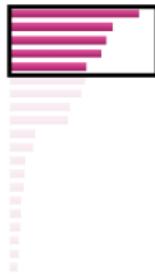
Decoding: Top- p (nucleus) sampling

- 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

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

$$P_t^1(y_t = w | \{y\}_{<t})$$



$$P_t^2(y_t = w | \{y\}_{<t})$$



$$P_t^3(y_t = w | \{y\}_{<t})$$



(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)