

NLP and the Web – WS 2024/2025



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Lecture 8 Language Modeling Foundations

Dr. Thomas Arnold
Hovhannes Tamoyan
Kexin Wang

Ubiquitous Knowledge Processing Lab
Technische Universität Darmstadt



Syllabus (tentative)

<u>Nr.</u>	<u>Lecture</u>
01	Introduction / NLP basics
02	Foundations of Text Classification
03	IR – Introduction, Evaluation
04	IR – Word Representation
05	IR – Transformer/BERT
06	IR – Dense Retrieval
07	IR – Neural Re-Ranking
08	LLM – Language Modeling Foundations
09	LLM – Neural LLM, Tokenization
10	LLM – Adaption, LoRa, Prompting
11	LLM – Alignment, Instruction Tuning
12	LLM – Long Contexts, RAG
13	LLM – Scaling, Computation Cost
14	Review & Preparation for the Exam

N-Gram Models

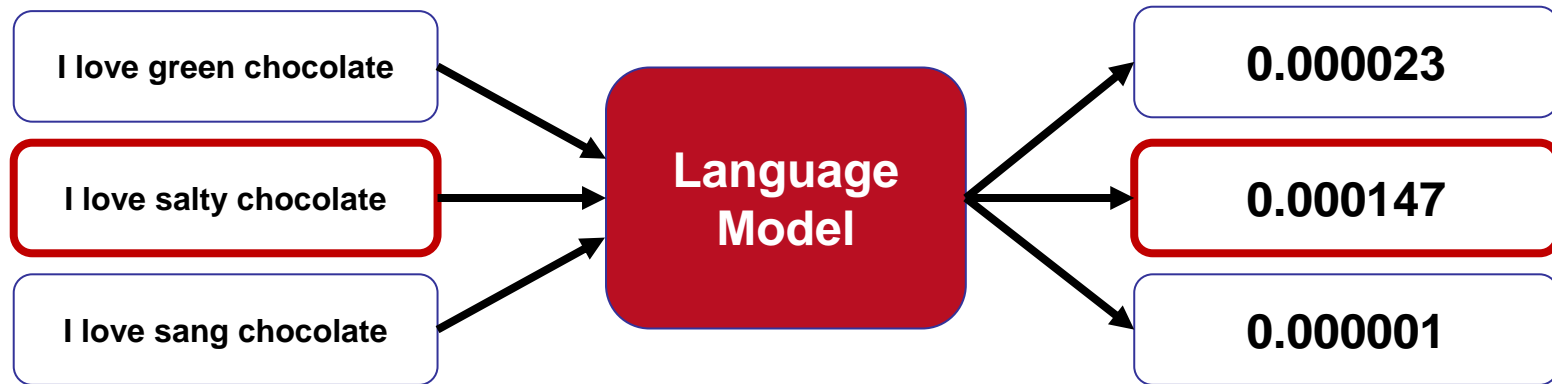
Generation with N-Gram Models

Out-Of-Vocabulary Problem

Language Model (LM) = Model that assigns probabilities to sequences of words



Language Model (LM) = Model that assigns probabilities to sequences of words



Why is this useful?

Speech recognition:

$P(\text{"I want it two letter go"}) < P(\text{"I wanted to let her go"})$

Spelling correction:

$P(\text{Their are three topics today}) < P(\text{There are three topics today})$

Language Generation:

Will be discussed later

Also: Machine Translation, Summarization, Question Answering...

Simplest LM based on N-Grams

N-Gram: Sequence of n tokens (words or punctuation)

Unigram (1-Gram): “My”

“!”

Bigram (2-Gram): “My name”

“mouth!”

Trigram (3-Gram): “My name is”

“your mouth!”

4-Gram: “My name is Thomas”

“Shut your mouth!”

...

N-Grams Language Models

Based on conditional probabilities

How likely is the sentence “Sir Arnold is the” followed by the token “worst”?

$$P(\textit{worst}|\textit{Sir Arnold is the})$$

We can estimate this probability by counting occurrences:

$$P(\textit{worst}|\textit{Sir Arnold is the}) \sim \frac{C(\textit{Sir Arnold is the worst})}{C(\textit{Sir Arnold is the})}$$

N-Grams Language Models



TECHNISCHE
UNIVERSITÄT
DARMSTADT

"Sir Thomas is the" × 🔊 📷 🔍

Bilder Videos Websites zu Orten News Maps Bücher Flüge Finanzen

Ungefähr 57.800 Ergebnisse (0,52 Sekunden)

Tipp: Suche auf Ergebnisse auf **Englisch** beschränken. Weitere Informationen zum Filtern nach Sprache

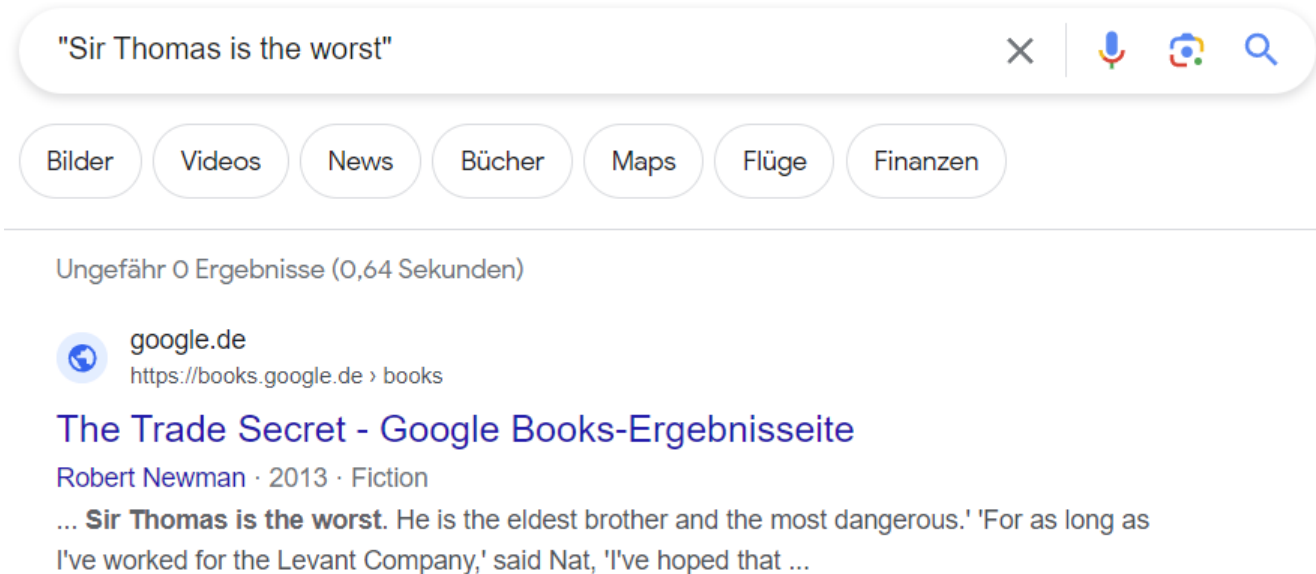






Glendon Place








<https://glendonplace.net> > ... > Fall ⋮

Sir Thomas


N-Grams Language Models



"Sir Thomas is the worst"    

Ungefähr 0 Ergebnisse (0,64 Sekunden)

 google.de
<https://books.google.de> › books

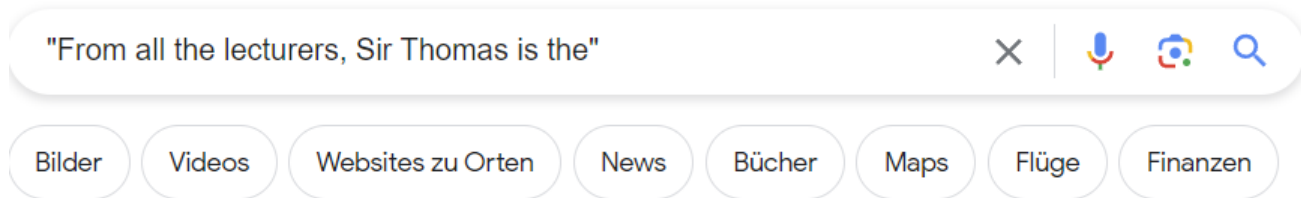
The Trade Secret - Google Books-Ergebnisseite

Robert Newman · 2013 · Fiction

... **Sir Thomas is the worst.** He is the eldest brother and the most dangerous.' 'For as long as I've worked for the Levant Company,' said Nat, 'I've hoped that ...

N-Grams Language Models

Problem: New sentences are created all the time – we won't be able to count entire sentences!



A screenshot of a Google search interface. The search bar contains the text "From all the lecturers, Sir Thomas is the". To the right of the search bar are icons for clearing the search, voice search, image search, and a magnifying glass. Below the search bar is a row of filters: Bilder, Videos, Websites zu Orten, News, Bücher, Maps, Flüge, and Finanzen.

Ungefähr 305.000.000 Ergebnisse (0,33 Sekunden)

Keine Ergebnisse für **"From all the lecturers, Sir Thomas is the"** gefunden

Ergebnisse für **From all the lecturers, Sir Thomas is the** (ohne Anführungszeichen):

Also, joint probabilities of longer sequences are hard to compute:

$$\begin{aligned} P(w_1, w_2, \dots, w_n) &= P(w_1) * P(w_2|w_1) * P(w_3|w_{1:2}) \dots * P(w_n|w_{1:n-1}) \\ &= P(w_{1:n}) = \prod_{k=1}^n P(w_k|w_{1:k-1}) \end{aligned}$$

N-Gram models approximate this probability by reducing the “history” (Markov assumption):

Bigram model: $P(w_{1:n}) \sim \prod_{k=1}^n P(w_k|w_{k-1})$

With bigram probabilities: $P(w_n|w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$

Example: Bigram LM

There once was a ship that put to sea
The name of the ship was the Billy O' Tea
The winds blew up, her bow dipped down
Oh blow, my bully boys, blow (huh)

Soon may the Wellerman come
To bring us sugar and tea and rum
One day, when the tonguing is done
We'll take our leave and go

(Lyrics of the song “Wellerman”)

Word 1	Word 2	Count

Example: Bigram LM

<S>: Beginning of Sentence

There once was a ship that put to sea
The name of the ship was the Billy O' Tea
The winds blew up, her bow dipped down
Oh blow, my bully boys, blow (huh)

Soon may the Wellerman come
To bring us sugar and tea and rum
One day, when the tonguing is done
We'll take our leave and go

(Lyrics of the song "Wellerman")

Word 1	Word 2	Count
<S>	There	1

Example: Bigram LM

There once was a ship that put to sea
The name of the ship was the Billy O' Tea
The winds blew up, her bow dipped down
Oh blow, my bully boys, blow (huh)

Soon may the Wellerman come
To bring us sugar and tea and rum
One day, when the tonguing is done
We'll take our leave and go

(Lyrics of the song “Wellerman”)

Word 1	Word 2	Count
<S>	There	1
There	once	1

Example: Bigram LM

There **once was** a ship that put to sea
The name of the ship was the Billy O' Tea
The winds blew up, her bow dipped down
Oh blow, my bully boys, blow (huh)

Soon may the Wellerman come
To bring us sugar and tea and rum
One day, when the tonguing is done
We'll take our leave and go

(Lyrics of the song “Wellerman”)

Word 1	Word 2	Count
<S>	There	1
There	once	1
once	was	1

Example: Bigram LM

<Large amounts of
text>

Word 1	Word 2	Count
<S>	There	256
<S>	The	321
<S>	Oh	17
...
There	once	123
There	must	29
There	has	69
...

Example: Bigram LM

Word 1	Word 2	Count	$P(W2 W1)$
<S>	There	256	?
<S>	The	321	?
<S>	Oh	17	?
<S>	I	69	?
<S>	They	169	?
<S>	Also	123	?
<S>	However	45	?
Anything	else	...	

Example: Bigram LM

Word 1	Word 2	Count	P(W2 W1)
<S>	There	256	?
<S>	The	321	?
<S>	Oh	17	?
<S>	I	69	?
<S>	They	169	?
<S>	Also	123	?
<S>	However	45	?
Anything	else	...	

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$

Example: Bigram LM

Word 1	Word 2	Count	P(W2 W1)
<S>	There	256	?
<S>	The	321	?
<S>	Oh	17	?
<S>	I	69	?
<S>	They	169	?
<S>	Also	123	?
<S>	However	45	?
Anything	else	...	

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$
$$C(<S>) = 1000$$

Example: Bigram LM

Word 1	Word 2	Count	P(W2 W1)
<S>	There	256	0.256
<S>	The	321	?
<S>	Oh	17	?
<S>	I	69	?
<S>	They	169	?
<S>	Also	123	?
<S>	However	45	?
Anything	else	...	

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$
$$C(<S>) = 1000$$

$$\frac{C(<S> \text{There})}{C(<S>)} = \frac{256}{1000}$$
$$= 0.256$$

Example: Bigram LM

Word 1	Word 2	Count	P(W2 W1)
<S>	There	256	0.256
<S>	The	321	0.321
<S>	Oh	17	0.017
<S>	I	69	0.069
<S>	They	169	0.169
<S>	Also	123	0.123
<S>	However	45	0.045
Anything	else	...	

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$
$$C(<S>) = 1000$$

$$\frac{C(<S> \text{There})}{C(<S>)} = \frac{256}{1000}$$
$$= 0.256$$

Example: Bigram LM



Examples from the Berkeley Restaurant Project corpus:

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

Example: Bigram LM



	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

Example: Bigram LM



i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

Example: Bigram LM

We can use these probabilities to calculate:

$$P(< S > I \text{ want english food } </S >) =$$

$$P(I | < S >)$$

$$* P(\text{want} | I)$$

$$* P(\text{english} | \text{want})$$

$$* P(\text{food} | \text{english})$$

$$* P(</S > | \text{food})$$

$$= .000031$$

(Note: Not all probabilities are listed on the previous slides)

N-Grams Language Models



Unigram model: $P(w_{1:n}) \sim \prod_{k=1}^n P(w_k)$

Bigram model: $P(w_{1:n}) \sim \prod_{k=1}^n P(w_k | w_{k-1})$

In practice it was more common to use trigram, 4-Gram or even 5-Gram models

Trigram model: $P(w_{1:n}) \sim \prod_{k=1}^n P(w_k | w_{k-2:k-1})$

4-Gram model: $P(w_{1:n}) \sim \prod_{k=1}^n P(w_k | w_{k-3:k-1})$

5-Gram model: $P(w_{1:n}) \sim \prod_{k=1}^n P(w_k | w_{k-4:k-1})$

Side note: For larger N-Grams, we use additional pseudo-words for sentence beginning:

Example for 4-Grams: $P(I | < S > < S > < S >)$

Problem:

- Probabilities are (by design) ≤ 1
- Multiplications of probabilities results in very small numbers
- Very small numbers might result in numerical underflow

Useful implementation trick:

- Store log probabilities instead of raw probabilities
- Adding in log space is equivalent to multiplication in linear space
- Relative order stays the same
- Only convert back to probabilities if we really need it:

$$P_1 * P_2 * P_3 * P_4 = \exp(\log P_1 + \log P_2 + \log P_3 + \log P_4)$$

N-Grams Language Models

Example: $P_1 = 0.001, P_2 = 0.003, P_3 = 0.00015, P_4 = 0.0132$

Multiplication: $P_1 * P_2 * P_3 * P_4 = 5.94E - 12$
= Danger of float precision problems

Logarithmic Sum: $\log P_1 + \log P_2 + \log P_3 + \log P_4 = -25.849312$
= Very easy to store and compare

In many situations, we are not interested in the exact probabilities, only in max/min
min/max(multiplicative probabilities) is the same as min/max(log probabilities)!

Extrinsic versus Intrinsic evaluation:

Extrinsic: Use the model in an actual application and measure the improvement of the end task

Example: Use new LM in speech recognition and measure accuracy of transcriptions

Usually, these evaluations are very expensive

Intrinsic: Evaluate on a gold standard test set

Example: Use part of a corpus to train a LM, use an unseen part of the corpus as test set

Important! Don't let the test sentences into the training set! (Bad science!)

Evaluating Language Models

What if we want to try different approaches or fine-tune parameters?

We use a third, completely separate “development set” (also called “validation set”)

Training Set
(60%)

Dev Set
(20%)

Test Set
(20%)

These numbers can vary based on task and data set size.

How to evaluate a LM on a test set?

Compute probabilities that the model assigns to the sentences
The model with the highest probabilities wins!

Instead of raw probabilities, we usually use **Perplexity (PPL)**

For $W = w_1, w_2, w_3 \dots w_n$:

$$\text{Perplexity}(W) = \sqrt[N]{\frac{1}{P(W)}}, \text{ for example in a bigram model: } \text{Perplexity}(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^N P(w_i|w_{i-1})}}$$

Minimizing perplexity = maximizing probability

Example: Perplexity

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

```
inputs = tokenizer("ABC is a startup based in New York City and Paris", return_tensors = "pt")  
loss = model(input_ids = inputs["input_ids"], labels = inputs["input_ids"]).loss  
ppl = torch.exp(loss)  
print(ppl)
```

Output: 29.48 # Model was trained on this sentence = lower PPL

```
inputs_wiki_text = tokenizer("Generative Pretrained Transformer is an opensource artificial  
intelligence created by OpenAI in February 2019", return_tensors = "pt")  
loss = model(input_ids = inputs_wiki_text["input_ids"], labels = inputs_wiki_text["input_ids"]).loss  
ppl = torch.exp(loss)  
print(ppl)
```

Output: 211.81 # Model did not see such sentences before = higher PPL

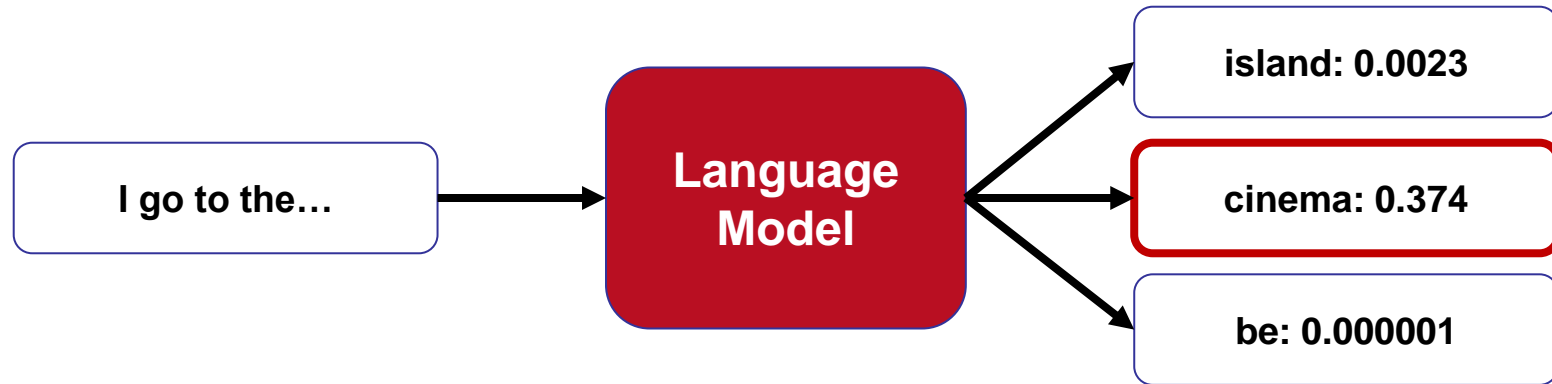
N-Gram Models

Generation with N-Gram Models

Out-Of-Vocabulary Problem

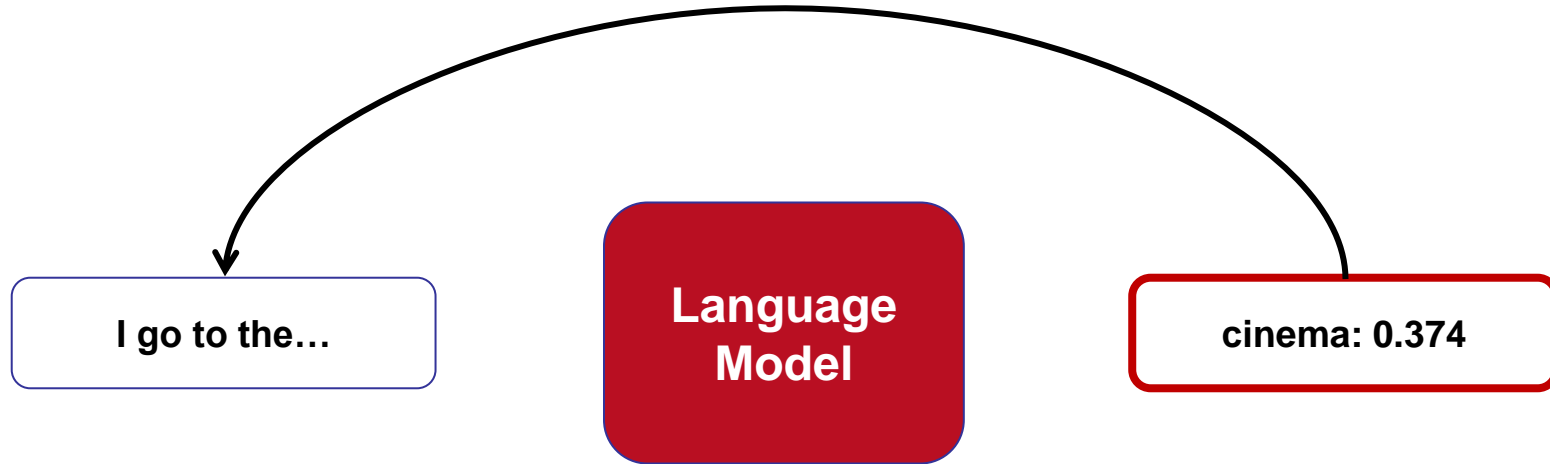
Generation with Language Models

Use case: Text completion / generation



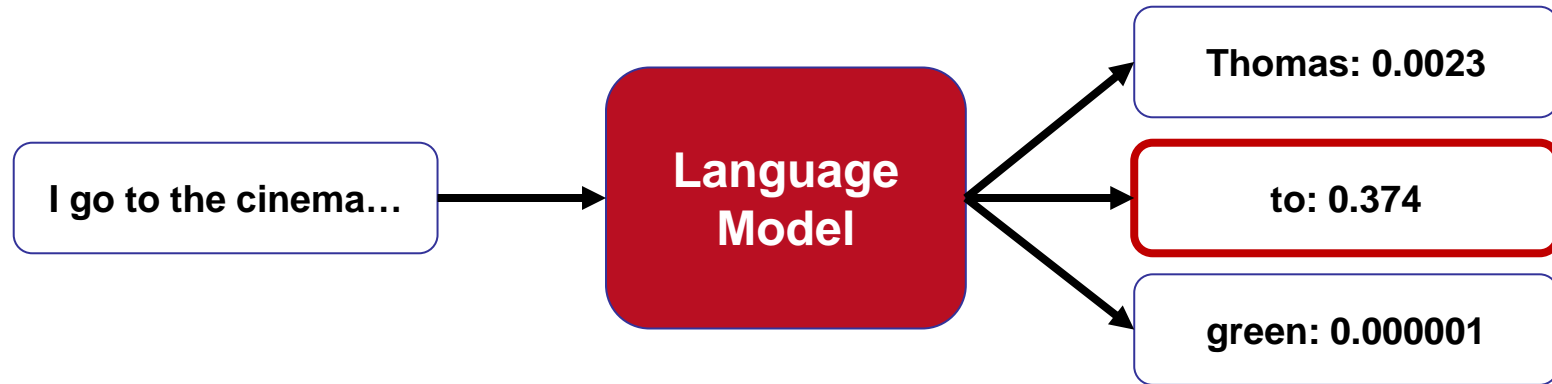
Generation with Language Models

Use case: Text completion / generation



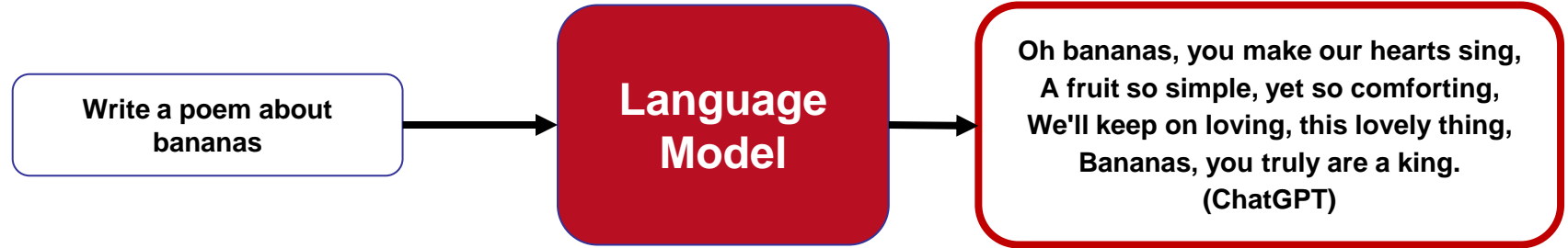
Generation with Language Models

Use case: Text completion / generation



Generation with Language Models

Use case: Text completion / generation

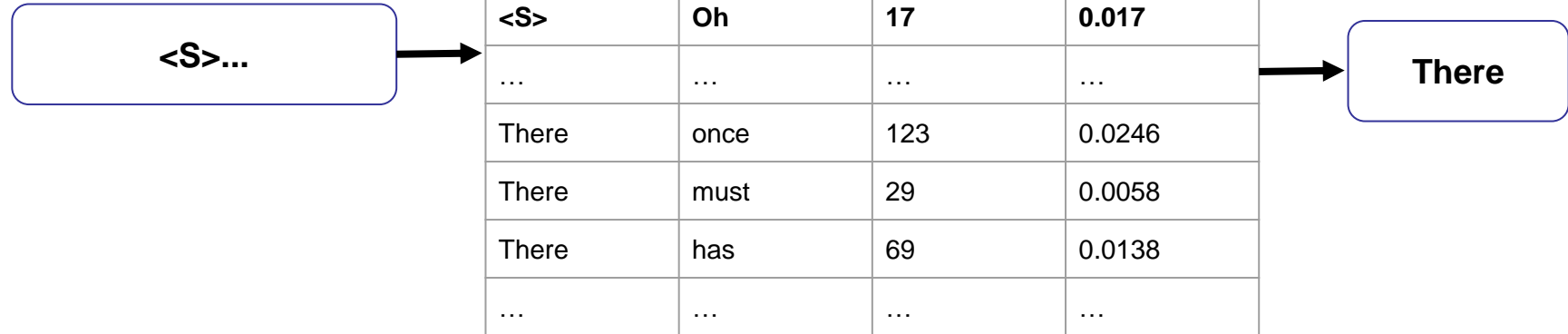
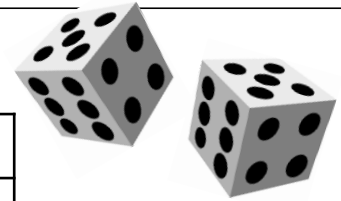


Example: Generation with a Bigram LM

Word 1	Word 2	Count	$P(W2 W1)$
<S>	There	256	0.256
<S>	The	321	0.321
<S>	Oh	17	0.017
<S>	I	69	0.069
<S>	They	169	0.169
<S>	Also	123	0.123
<S>	However	45	0.045
Anything	else	...	

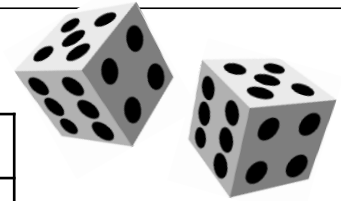
Example: Generation with a Bigram LM

Generate the next token based on conditional probabilities



Example: Generation with a Bigram LM

Generate the next token based on conditional probabilities



There...

Word 1	Word 2	Count	$P(W2 W1)$
<S>	There	256	0.256
<S>	The	321	0.321
<S>	Oh	17	0.017
...
There	once	123	0.0246
There	must	29	0.0058
There	has	69	0.0138
...

There once

Generation with N-Gram Models

Choose random points from a distribution based on likelihood = Sampling
Bigger N-Grams can create more cohesive results

1 gram	-To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have -Hill he late speaks; or! a more to leg less first you enter
2 gram	-Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. -What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	-Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. -This shall forbid it should be branded, if renown made it empty.
4 gram	-King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; -It cannot be but so.

Figure 3.4 Eight sentences randomly generated from four n-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

Generation with N-Gram Models

Of course, these models are highly dependent on their training material!

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Figure 3.5 Three sentences randomly generated from three n-gram models computed from 40 million words of the *Wall Street Journal*, lower-casing all characters and treating punctuation as words. Output was then hand-corrected for capitalization to improve readability.

N-Gram Models

Generation with N-Gram Models

Out-Of-Vocabulary Problem

Big problem: zero probability n-grams

In Evaluation: Test set might contain N-Grams that did never appear in training

= the entire probability of the test set is zero! (multiplication with zero)

= Perplexity is undefined! (division by zero)

In Generation: LM has never seen the words of my prompt before

= No way to generate the next token!

First approach: Use unknown words <UNK>

Two common approaches:

- Choose a vocabulary in advance, and convert every out-of-vocabulary word to <UNK>
- Replace words in the training data with <UNK> based on frequency
 - Choose top V words, replace the rest with <UNK>
 - Replace all words that occur fewer than n times with <UNK>

Word 1	Word 2	Count	$P(W2 W1)$
<S>	There	256	0.256
<S>	The	321	0.321
<S>	Oh	17	0.017
<S>	<UNK>	69	0.069

Second approach: Smoothing

Idea: Distribute some of the probability mass to unseen events

Simplest way: Laplace / add-one Smoothing

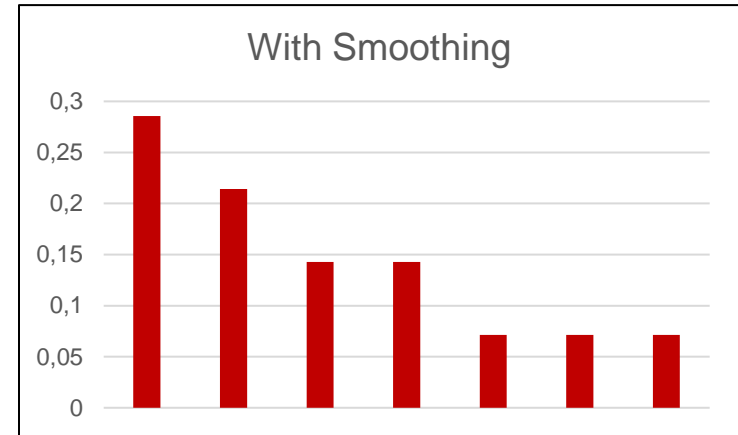
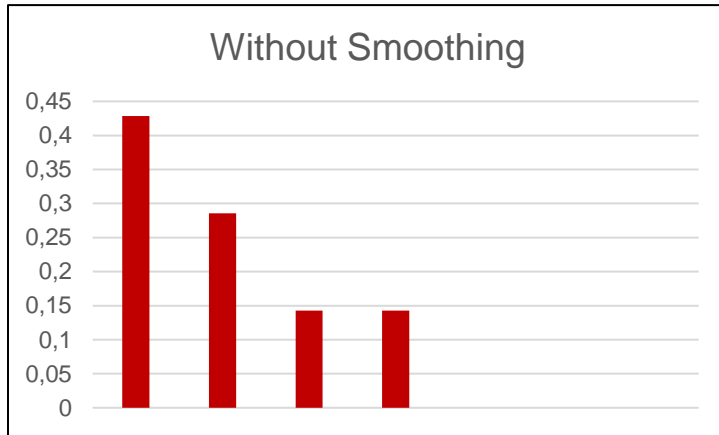
Pretend we saw each word one more time than we did (add one to each count)

Example for Unigrams: $P(w_i) = \frac{c(w_i)}{N}$ and $P_{Laplace}(w_i) = \frac{c(w_i)+1}{N+V}$

With N = total number of tokens and V = Size of Vocabulary

Generation with N-Gram Models

Example of Laplace Smoothing:



Generation with N-Gram Models


Unsmoothed: 608

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figure 3.6 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

Generation with N-Gram Models

Unsmoothed: 0.66



	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00005	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.66	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figure 3.7 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero probabilities are in gray.

Generation with N-Gram Models

Alternative: Add-k smoothing

Instead of adding 1 to each count, we add a fractional count k (.5, .05, .01...)

This parameter could be optimized on a **devset**

Other strategies: Backoff

Use smaller N-Grams if we have no evidence

$$P(\textit{Thomas Arnold is king}) = 0$$

Backoff from 4-Grams to trigrams:

$$P(\textit{Thomas Arnold is}) = ?$$

$$P(\textit{Arnold is king}) = ?$$

If these are also zero, use bigrams, then unigrams

Other strategies: Interpolation

Always use a weighted combination of different N-Grams

Example for trigrams:

$$P^*(w_n | w_{n-2} w_{n-1}) = \alpha_1 P(w_n) + \alpha_2 P(w_n | w_{n-1}) + \alpha_3 P(w_n | w_{n-2} w_{n-1})$$

unigram

bigram

trigram

With $\alpha_1 + \alpha_2 + \alpha_3 = 1$

Again, these parameters should be fine-tuned on the held-out dev set!



Word-level vs. Sub-word tokenization

The Problem with Word-level Tokenization

- Struggles with out-of-vocabulary (OOV) words.
- Fails to generalize well to unseen or rare terms.
- Requires a larger vocabulary to account for all word variations.

The Sub-word Solution

- Breaks words into smaller units, enabling flexible tokenization.
- Guarantees near-complete coverage, even for unseen text.

Example: Unseenword is amazing!

Word-level: [UNK, "is", "amazing", "!"]

Sub-words: ["Un", "seen", "word", "is", "am", "az", "ing", "!"]

Recap: Byte Pair Encoding (BPE)

Subword tokenization technique

Used for data compression and dealing with unknown words

Initialization:

Vocabulary = set of all individual characters

$V = \{A, B, C, \dots a, b, c, \dots 1, 2, 3, \dots !, \$, \%, \dots\}$

Repeat:

- Choose two symbols that appear as a pair most frequently (say “a” and “t”)
- Add new merged symbol (“at”)
- Replace each occurrence with the new symbol (“t”, “h”, “a”, “t” -> “t”, “h”, “at”)

Until k merges have been done

Recap: Byte Pair Encoding (BPE)

see slides IR2 for full example



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Segments:

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

Vocabulary:

</w>, d, e, I, l, n, o, r, s, t, w

Most frequent symbol pair: er (9 times)

Segments:

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

Vocabulary:

</w>, d, e, I, l, n, o, r, s, t, w, er

Introduced by Google, used in models like BERT and DistilBERT

Similar to BPE:

- Initializes the vocabulary with every character in the training data
- Learns a predefined number of merge rules

Difference from BPE:

- Selects symbol pairs that maximize the **likelihood** of the training data, not the most frequent pair
- Evaluates the impact of merging symbols to avoid reducing overall data probability

Example:

Merges "u" and "g" only if $\frac{P(ug)}{P(u)*P(g)}$ is greater than for any other pair

WordPiece Example

Segments:

5 1 ##o ##w
2 1 ##o ##w ##e ##s ##t
6 n ##e ##w ##e ##r
3 w ##i ##d ##e ##r
2 n ##e ##w

Vocabulary:

1, n, w, ##o, ##w, ##e, ##s, ##t, ##r, ##i, ##d

Note: $P(x) \neq TF(x)$, but has the same max

Most likely symbol pair: [##s, ##t]: $\frac{TF(##s,##t)}{TF(##s)*TF(##t)} = \frac{2}{2*2} = 0.5$, which is the max of all pairs

Segments:

5 1 ##o ##w
2 1 ##o ##w ##e ##st
6 n ##e ##w ##e ##r
3 w ##i ##d ##e ##r
2 n ##e ##w

Vocabulary:

1, n, w, ##o, ##w, ##e, ##s, ##t, ##r, ##i, ##d, ##st

Both BPE and WordPiece assume that words are separated by white spaces

Not all languages do that

SentencePiece treats white spaces like any other character

- No need for pre-tokenization
- “_” (white space) is included in the base vocabulary
- It then uses BPE or similar algorithms to merge the tokens

De-Tokenization does not need any language-specific methods

[Hello] [_wor] [ld] [.] -> “Hello World.”

T. Kudo and J. Richardson (2018). “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing”. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Brussels, Belgium: Association for Computational Linguistics, pp. 66–71

Neural Language Modeling