



# N-gram Language model

Capturing Language Patterns with Statistical Sequences

Quick recap of previous lecture

# Why text preprocessing is important?

- **Noise Reduction:** Remove irrelevant characters (punctuation, HTML tags).
- **Consistency:** Lowercasing, standardizing formats (e.g., dates).
- **Efficiency:** Smaller vocabulary size = faster model training.
- **Accuracy:** Improves NLP task performance (e.g., sentiment analysis).

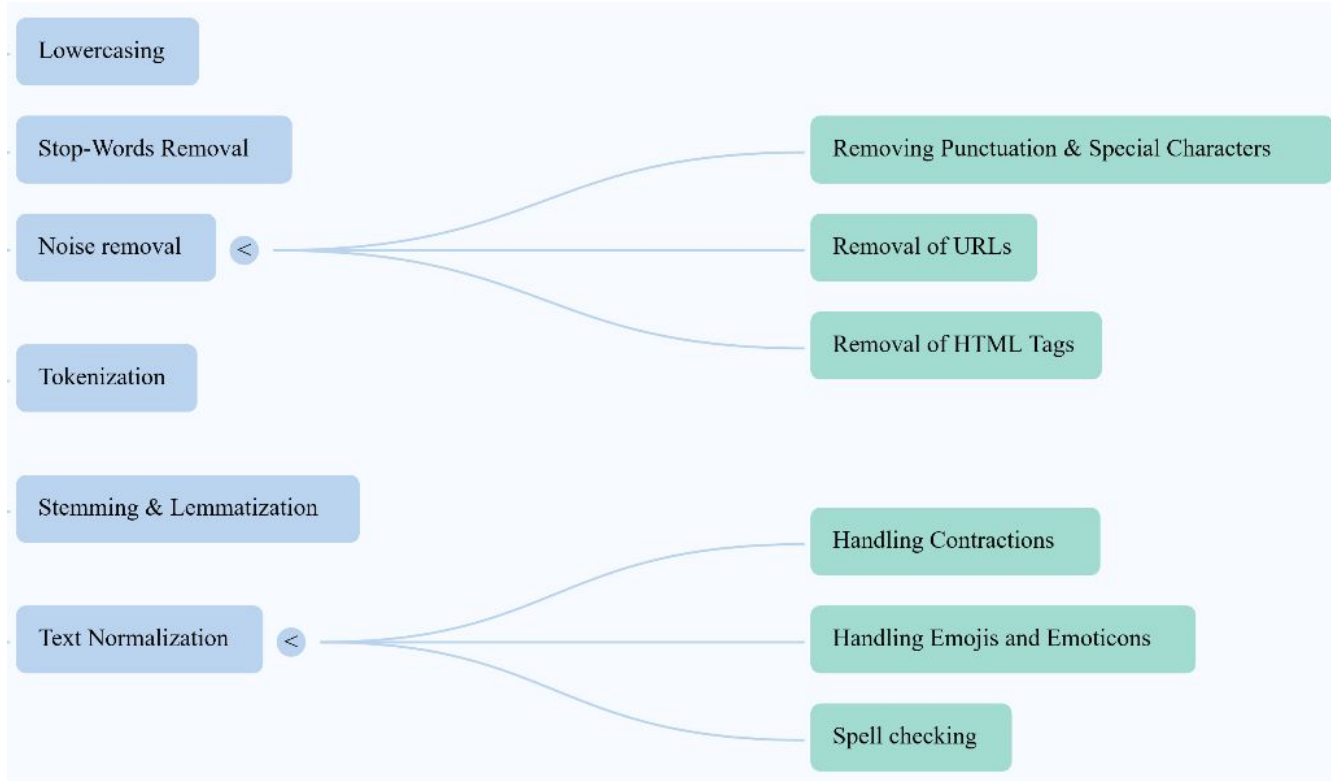
## Challenges

- **Ambiguity:** “Apple” (fruit vs. company).
- **Language Differences:** Morphology in Arabic vs. English.
- **Resource Limits:** Lemmatization requires heavy dictionaries.

# Domain-Specific Preprocessing

- **Social Media:** Emoji handling, slang normalization.
- **Scientific Texts:** Retain equations/symbols.
- **Medical Texts:** Protect sensitive terms (e.g., patient names).

# Common preprocessing steps:



# Classes of tokenization algorithms

- Top-down tokenization
  - Rule-based (Penn Treebank Tokenizer)
- Bottom-up tokenization
  - Data-driven (Byte-Pair Encoding)
- Speed matters:
  - Tokenization is performed on every input before other language processing steps.
  - Large datasets and real-time systems need tokenizers that are highly optimized for speed.
  - Word tokenizers generally use deterministic (rule-based or algorithmic) logic for reliability and efficiency

# Word normalization

Word normalization is the task of putting words or tokens in a standard format.

## Lemmatization

was → (to) be

better → good

meeting → meeting

## Stemming

adjustable → adjust

formality → formalit

formality → formal

airliner → airlin

# Minimum edit distance

Minimum Edit Distance is the **minimum number of operations** required to convert one string into another.

## Edit Operations:

- **Insertion** (add a character)
- **Deletion** (remove a character)
- **Substitution** (replace one character with another)

## Applications:

- Spell checking
- Machine translation evaluation
- Plagiarism detection
- Speech recognition

## Why does it matter?:

- MED quantifies how “similar” two pieces of text are [critical in many NLP tasks].

The gap between **intention** and **execution**

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s			i	s			



## How to calculate MED? (Dynamic programming approach)

$$D[i, j] = \min \begin{cases} D[i-1, j] + \text{del-cost}(\text{source}[i]) \\ D[i, j-1] + \text{ins-cost}(\text{target}[j]) \\ D[i-1, j-1] + \text{sub-cost}(\text{source}[i], \text{target}[j]) \end{cases}$$

### Levenshtein Distance

$$D[i, j] = \min \begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \begin{cases} 2; & \text{if } \text{source}[i] \neq \text{target}[j] \\ 0; & \text{if } \text{source}[i] = \text{target}[j] \end{cases} \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
Src\Tar		#	e	x	e	c	u	t	i	o	n
0	#	0	1	2	3	4	5	6	7	8	9
1	i	1	2	3	4	5	6	7	6	7	8
2	n	2	3	4	5	6	7	8	7	8	7
3	t	3	4	5	6	7	8	7	8	9	8
4	e	4	3	4	5	6	7	8	9	10	9
5	n	5	4	5	6	7	8	9	10	11	10
6	t	6	5	6	7	8	9	8	9	10	11
7	i	7	6	7	8	9	10	9	8	9	10
8	o	8	7	8	9	10	11	10	9	8	9
9	n	9	8	9	10	11	12	11	10	9	8

**Figure 2.18** Computation of minimum edit distance between *intention* and *execution* with the algorithm of Fig. 2.17, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions.

	#	e	x	e	c	u	t	i	o	n
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ 1	↖←↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↑ 7	↖←↑ 8	↖ 7
t	↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖ 7	←↑ 8	↖←↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← 5	← 6	← 7	←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9
n	↑ 5	↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖ 8	← 9	← 10	←↑ 11
i	↑ 7	↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9	↖ 8	← 9	← 10
o	↑ 8	↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↑ 10	↑ 9	↖ 8	← 9
n	↑ 9	↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖←↑ 12	↑ 11	↑ 10	↑ 9	↖ 8

At each step:

- If you follow ↖ (**diagonal**), you either substitute (if letters differ) or match (if same).
- If you follow ↑ (**top**), you've deleted a character from the source.
- If you follow ← (**left**), you've inserted a character into the source.

I N T E \* N T I O N

| | | | | | | | |

\* E X E C U T I O N

d s s i s

# N-gram Language model

# Word prediction

*“I am Indian. I grew up in Paris. I can speak ..... fluently.”*

*“The water of the Pangong Lake in Leh is beautifully .....”*

French

Hindi

Bengali

English

...

Blue

Green

Red

...

Pumpkin

Elephant

Horse

...

# Language models

A language model is a machine learning model that predicts upcoming words.

- An LM assigns a probability to each potential next word.
- An LM assigns a probability to a whole sentence.

Two paradigms:

- N-gram language models
- Large language models [will discuss after mid sem]

# Word prediction application

## Grammar or spell checking

- “Everything has improve”
- “Their are two midterms”

# Word prediction application

## Grammar or spell checking

- “Everything has improved~~d~~”
- ~~“Their~~ **There** are two midterms”

## Speech recognition

- *“Hello dear” or “Hello there”*
- *“I like this test” or “I like dish taste”*
- *“Dirty Mouse House” or “Thirty miles us”*



# Word prediction

Word prediction is a key to various NLP tasks.

LLMs are trained to predict words.

- Left-to-right (autoregressive) LMs learn to predict next word

LLMs generate text by predicting words.

- Predict the next word over and over again

# n-gram

- **Unigram**: an individual word in a sequence of words, e.g. **The, water, of, ...**
- **Bigram**: two-word sequence of words, e.g. **The water, water of, ...**
- **Trigram**: three-word sequence of words, e.g. **The water of, water of Walden,**
- ...
- and so on.

*The water of Walden Pond is so beautifully blue.*

# N-gram language model

- An n-gram LM is a probabilistic model that can estimate the probability of a next word given the previous words.
- Thereby assign probabilities to entire sequences.
- Eg. We have “**The water of Walden Pond is so beautifully**”, and we want to know if **blue** is the next probable word:

$$P(\text{blue} | \text{The water of Walden Pond is so beautifully}) = \frac{C(\text{The water of Walden Pond is so beautifully blue})}{C(\text{The water of Walden Pond is so beautifully})}$$

# Chain rule of probability

Bayes' Theorem:  $P(A | B) = \frac{P(A \cap B)}{P(B)}$

$$\begin{aligned} P(X_1 \dots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \dots P(X_n|X_{1:n-1}) \\ &= \prod_{k=1}^n P(X_k|X_{1:k-1}) \end{aligned}$$

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

# Challenge in counting long sequences

- Language is creative
- Data Sparsity as new sentences are invented all the time
- Zero Counts for Valid Sequences

$$P(\text{blue}|\text{The water of Walden Pond is so beautifully}) = \frac{C(\text{The water of Walden Pond is so beautifully blue})}{C(\text{The water of Walden Pond is so beautifully})}$$

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

# Markov assumption

## Simplifying assumption:



Andrei Markov

$P(\text{blue} | \text{The water of Walden Pond is so beautifully})$

$\approx P(\text{blue} | \text{beautifully})$

$$P(w_n | w_{1:n-1}) \approx P(w_n | w_{n-1})$$

## Bigram Markov Assumption

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

instead of:

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

## N-gram Markov assumption

$$P(w_n | w_{1:n-1}) \approx P(w_n | w_{n-N+1:n-1})$$

We can predict a word using Trigram model as:

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-2} w_{i-1})$$



# How to estimate probabilities?

- Maximum Likelihood Estimation (MLE)

- Bigram model: 
$$P(w_n | w_{n-1}) = \frac{C(w_{n-1} w_n)}{\sum_w C(w_{n-1} w)}$$

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

## Example

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = 0.67$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = 0.33$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = 0.67$$

$$P(\text{</s>} | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = 0.5$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = 0.33$$

## More examples

Berkeley Restaurant Project (a sample of user queries [normalized text] on the website)

can you tell me about any good cantonese restaurants close by

tell me about chez panisse

i'm looking for a good place to eat breakfast

when is caffe venezia open during the day

	<b>i</b>	<b>want</b>	<b>to</b>	<b>eat</b>	<b>chinese</b>	<b>food</b>	<b>lunch</b>	<b>spend</b>
<b>i</b>	5	827	0	9	0	0	0	2
<b>want</b>	2	0	608	1	6	6	5	1
<b>to</b>	2	0	4	686	2	0	6	211
<b>eat</b>	0	0	2	0	16	2	42	0
<b>chinese</b>	1	0	0	0	0	82	1	0
<b>food</b>	15	0	15	0	1	4	0	0
<b>lunch</b>	2	0	0	0	0	1	0	0
<b>spend</b>	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. Each cell shows the count of the column label word following the row label word. Thus the cell in row **i** and column **want** means that **want** followed **i** 827 times in the corpus.

Unigram counts:

<b>i</b>	<b>want</b>	<b>to</b>	<b>eat</b>	<b>chinese</b>	<b>food</b>	<b>lunch</b>	<b>spend</b>
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.

$$P(\text{want}|\text{i}) = 827/2533 = 0.32649$$

$$P(\text{eat}|\text{to}) = 686/2417 = 0.284$$

$$P(\text{want}|\text{i}) = 0.32649$$

$$P(\text{eat}|\text{to}) = 0.284$$

$$P(\text{i}|\text{<s>}) = 0.25$$

$$P(\text{english}|\text{want}) = 0.0011$$

$$P(\text{food}|\text{english}) = 0.5$$

$$P(\text{</s>}|\text{food}) = 0.68$$

Compute probabilities of:

- *"I want English food"*

$$\begin{aligned} P(\text{<s> i want english food </s>}) &= P(\text{i}|\text{<s>})P(\text{want}|\text{i})P(\text{english}|\text{want}) \\ &\quad P(\text{food}|\text{english})P(\text{</s>}|\text{food}) \\ &= 0.25 \times 0.33 \times 0.0011 \times 0.5 \times 0.68 \\ &= 0.000031 \end{aligned}$$

- **Longer sequences can lead to numerical underflow**

$$P(\text{want}|\text{i}) = 0.32649$$

$$P(\text{eat}|\text{to}) = 0.284$$

$$P(\text{i}|\text{<s>}) = 0.25$$

$$P(\text{english}|\text{want}) = 0.0011$$

$$P(\text{food}|\text{english}) = 0.5$$

$$P(\text{</s>}|\text{food}) = 0.68$$

Compute probabilities of:

- *"I want English food"*

$$\begin{aligned} P(\text{<s> i want english food </s>}) &= P(\text{i}|\text{<s>})P(\text{want}|\text{i})P(\text{english}|\text{want}) \\ &\quad P(\text{food}|\text{english})P(\text{</s>}|\text{food}) \\ &= 0.25 \times 0.33 \times 0.0011 \times 0.5 \times 0.68 \\ &= 0.000031 \end{aligned}$$

Adding in log space is equivalent to multiplying in linear space

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

Dataset: <https://stressosaurus.github.io/raw-data-google-ngram/>

Toolkits:

- <http://www.speech.sri.com/projects/srilm/>
- <https://kheafield.com/code/kenlm/>

# Example (Bigram LM generation)

<s> I  
I want  
want to  
to eat  
eat Chinese  
Chinese food  
food </s>

I want to eat Chinese food



# Sentence generation using four N-gram language models, trained on a dataset of Shakespeare's text.

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

# How to evaluate LMs?

**Extrinsic evaluation** (in-vivo evaluation):

Compare models A and B:

- Put each model in a real task (Machine Translation, Speech recognition, etc.)
- Run the task, get a score for A and for B
- Compare accuracy for A and B

Challenges:

- Expensive and time consuming
- Cannot generalize to other applications

# How to evaluate LMs (N-gram models)

- A good LM is one that assigns a higher probability to the next word that actually occurs.
- The best language model is one that best predicts the entire unseen test set
- Probability depends on size of test set
  - Longer the text smaller the probability score
- **Perplexity** is the inverse probability of the test set, normalized by the number of words.

$$P(w_1 w_2 \dots w_N)$$

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

# How to evaluate LMs (N-gram models)

## Intrinsic evaluation (Perplexity):

- Directly measure LM performance at predicting words.
- Doesn't necessarily correspond with real application performance.
- Gives a general metric for LMs performances.
- Useful for LLMs evaluation as well.

Probability range is  $[0,1]$ , perplexity range is  $[1,\infty]$

Minimizing perplexity is the same as maximizing probability

## How to evaluate LMs (N-gram models)

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$
$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

	Unigram	Bigram	Trigram
Perplexity	962	170	109

\* The lower the perplexity, the better the language model.

# Weighted average branching factor (Perplexity)

- Perplexity is also the **weighted average branching factor** of a language.
- Branching factor: **number of possible next words** that can follow any word
- Eg. If a deterministic language (no probabilities) with vocabulary consist of three colors **{red,blue,green}**.
  - Then, the branching factor of this language is **3**.

$$\begin{aligned}\text{perplexity}_A(T) &= P_A(\text{red red red red blue})^{-\frac{1}{5}} \\ &= \left( \left( \frac{1}{3} \right)^5 \right)^{-\frac{1}{5}} \\ &= \left( \frac{1}{3} \right)^{-1} = 3\end{aligned}$$

## Weighted average branching factor (Perplexity)

$$P(\text{red}) = 0.8 \quad P(\text{green}) = 0.1 \quad P(\text{blue}) = 0.1$$

$$\begin{aligned} \text{perplexity}_B(T) &= P_B(\text{red red red red blue})^{-1/5} \\ &= 0.04096^{-\frac{1}{5}} \\ &= 0.527^{-1} = 1.89 \end{aligned}$$

The weighted average branching factor of this language is 1.89

# Sampling

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.



# Example

Choose a random bigram ( $\langle s \rangle$ ,  $w$ )

according to its probability  $p(w|\langle s \rangle)$

$\langle s \rangle$  I

I want

Now choose a random bigram ( $w$ ,  $x$ )

according to its probability  $p(x|w)$

want to

to eat

And so on until we choose  $\langle /s \rangle$

eat Chinese

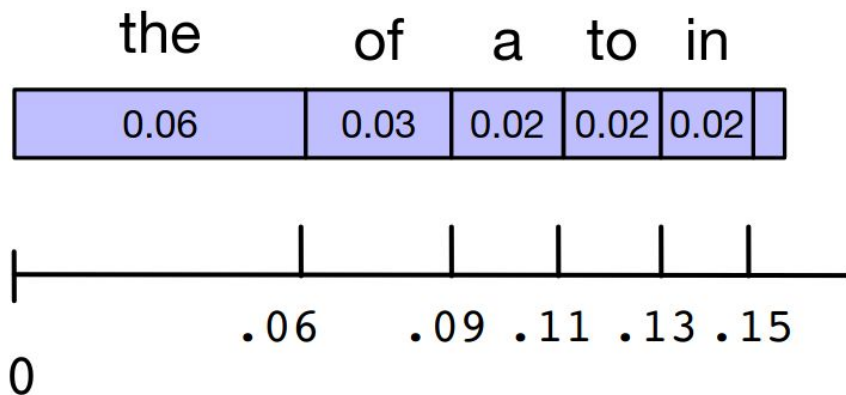
Then string the words together

Chinese food

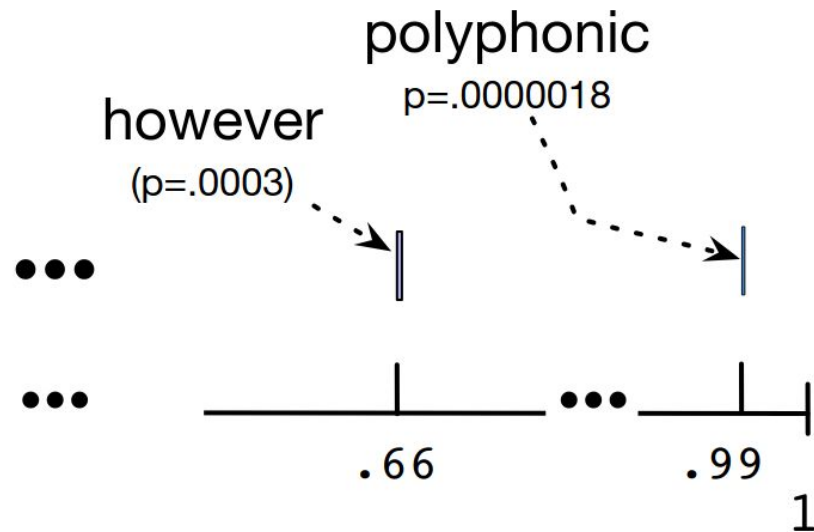
food  $\langle /s \rangle$

I want to eat Chinese food

# Sampling unigram distribution



- Random sampling
- Top-k sampling
- Top-p sampling



# Smoothing

	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.

## Maximum likelihood estimation has a problem

- **Language is creative:** Many valid word sequences may not appear in training data.
- **Assigns zero probability** to unseen n-grams (data sparsity problem).
- **Zero probabilities** break language models and lead to severe errors in various applications.

## Smoothing

- **Goal:** Redistribute some probability mass from seen to unseen events to avoid zeros.
- **Laplace (Add-1) Smoothing:**  
Adds 1 to all counts, ensuring every possible n-gram has a nonzero probability.

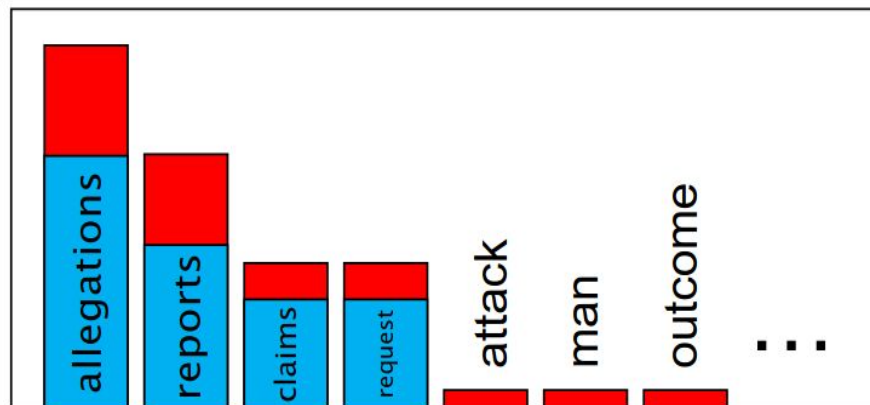
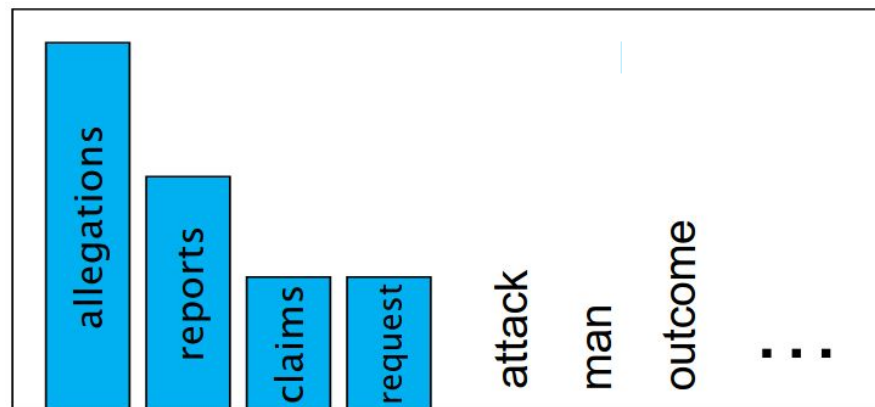
## Interpolation

- **Goal:** Combine multiple models of different orders (e.g., trigram, bigram, unigram) to get more reliable probability estimates.

# Laplace (Add-1) Smoothing

$$P_{\text{MLE}}(w_n | w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

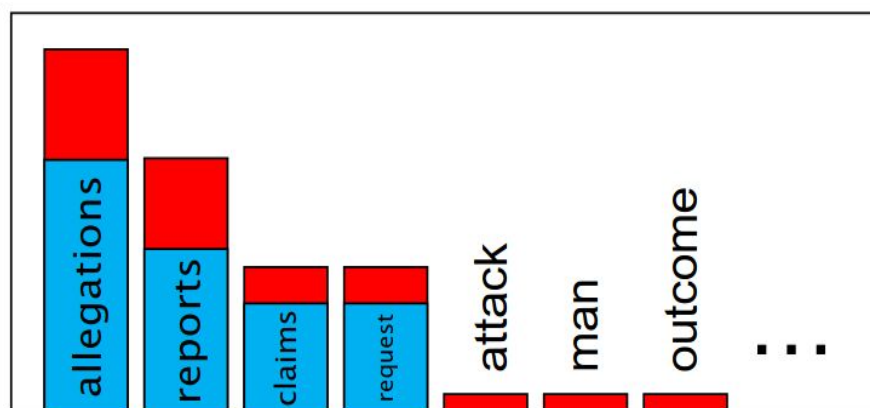
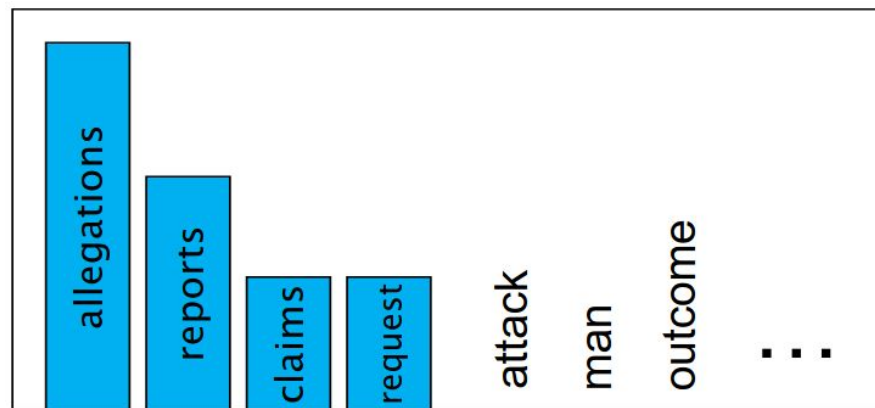
$$P_{\text{Laplace}}(w_n | w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$



## Laplace (Add-k) Smoothing

$$P_{\text{Add-k}}^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$




# Interpolation

- Combine multiple models of different orders (e.g., trigram, bigram, unigram) to get more reliable probability estimates.

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) = & \lambda_1 P(w_n) \\ & + \lambda_2 P(w_n|w_{n-1}) \\ & + \lambda_3 P(w_n|w_{n-2}w_{n-1})\end{aligned}$$

Find the  $\lambda$  values (weights for n-gram models) that maximize the likelihood of the held-out (validation) data.


$$\begin{aligned} L(\lambda_1, \lambda_2, \lambda_3) &= \sum_{u,v,w} c'(u, v, w) \log q(w|u, v) \\ &= \sum_{u,v,w} c'(u, v, w) \log (\lambda_1 \times q_{ML}(w|u, v) + \lambda_2 \times q_{ML}(w|v) + \lambda_3 \times q_{ML}(w)) \end{aligned}$$

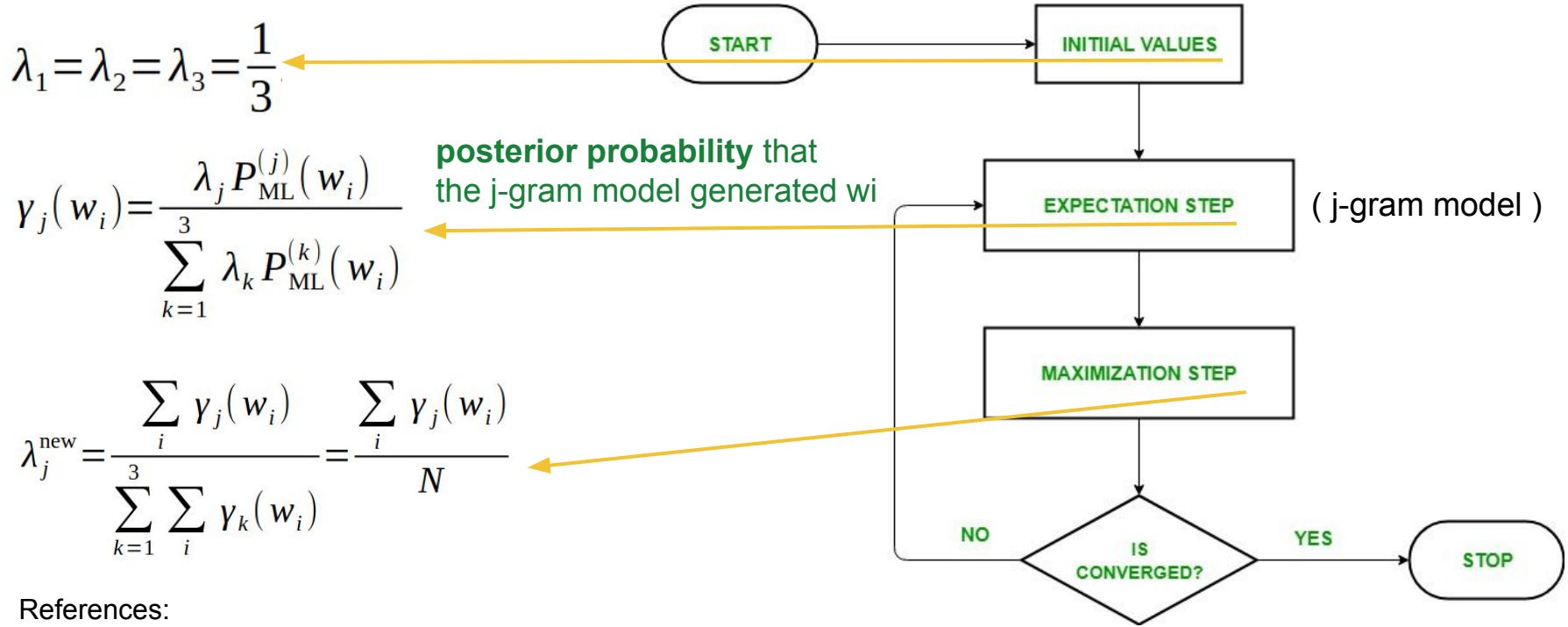
$$\arg \max_{\lambda_1, \lambda_2, \lambda_3} L(\lambda_1, \lambda_2, \lambda_3) \quad \text{s.t.} \quad \lambda_1 + \lambda_2 + \lambda_3 = 1$$

References:

- <https://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf>



# Expectation-Maximization algorithm



## References:

- <https://www.cs.columbia.edu/~mccollins/em.pdf>
- <https://www.geeksforgeeks.org/machine-learning/ml-expectation-maximization-algorithm/>

# Challenges of N-gram models

- Cannot handle long-term dependencies
  - “The key to the cabinets *are* missing.” ❌
    - “The key to the cabinets *is* missing.”
  - “If the dog plays in the yard, *it* will get dirty.” ❌
    - “If the dog plays in the yard, *he* will get dirty.”
- Cannot generalize well to new or rephrased sequences
  - N-gram models rely heavily on exact word patterns (surface-level).
  - N-gram models treat each sequence as a separate, unrelated string.
  - They have no understanding of synonyms, paraphrasing, or semantics.

# References:

- Chapter 3: <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>
- Expectation Maximization algorithm:
  - <https://www.cs.columbia.edu/~mcollins/em.pdf>
  - <https://www.geeksforgeeks.org/machine-learning/ml-expectation-maximization-algorithm/>