Language Modelling with n-grams

Marcel Bollmann

Department of Computer Science (IDA)

LINKÖPING UNIVERSITY

Language modelling in a nutshell

- What is the **probability of a sentence**?

$$P(\text{`I like horror movies'}) > P(\text{`like horror movies I'})$$

$$P(\text{`Sweden is a country in Europe'}) > P(\text{`Sweden is a country in Asia'})$$

$$P(\text{`He is drinking coffee'}) > P(\text{`He is drinking corn flakes'})$$

- What is the **conditional probability** of a word given context?

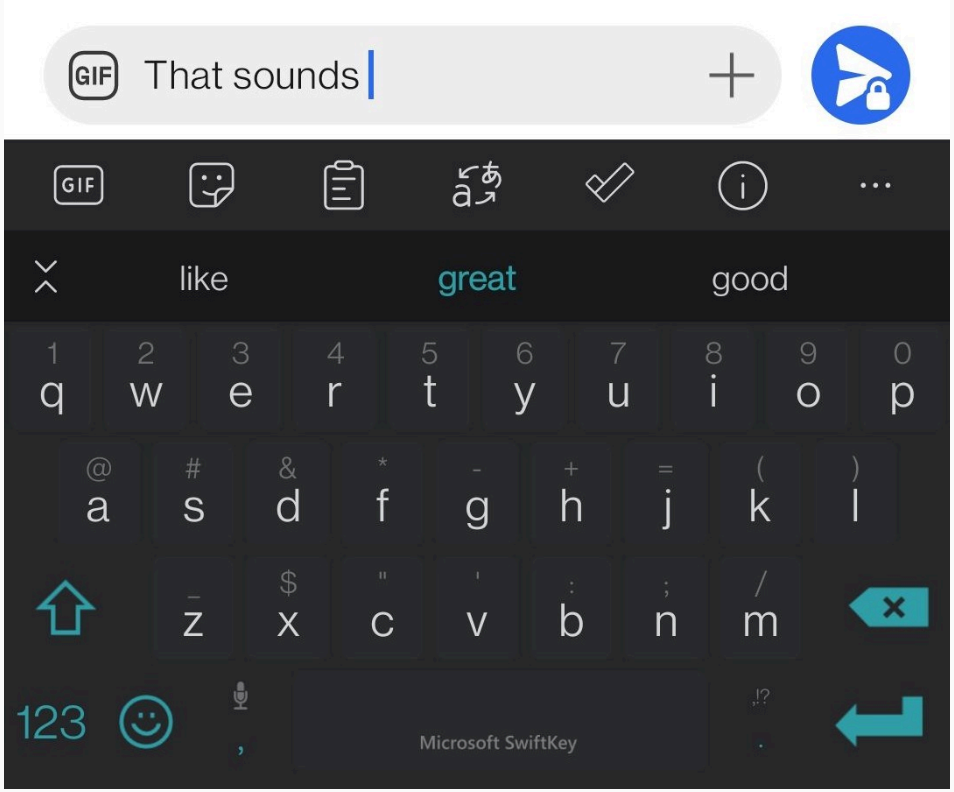
$$P(\text{`Europe'} \mid \text{`Sweden is a country in'})$$

These two formulations are equivalent!

Example: Predictive typing

- **Predict the next word** given the words that were already typed.

$$\arg\max_{w \in V} P(w \mid \text{`That sounds'})$$

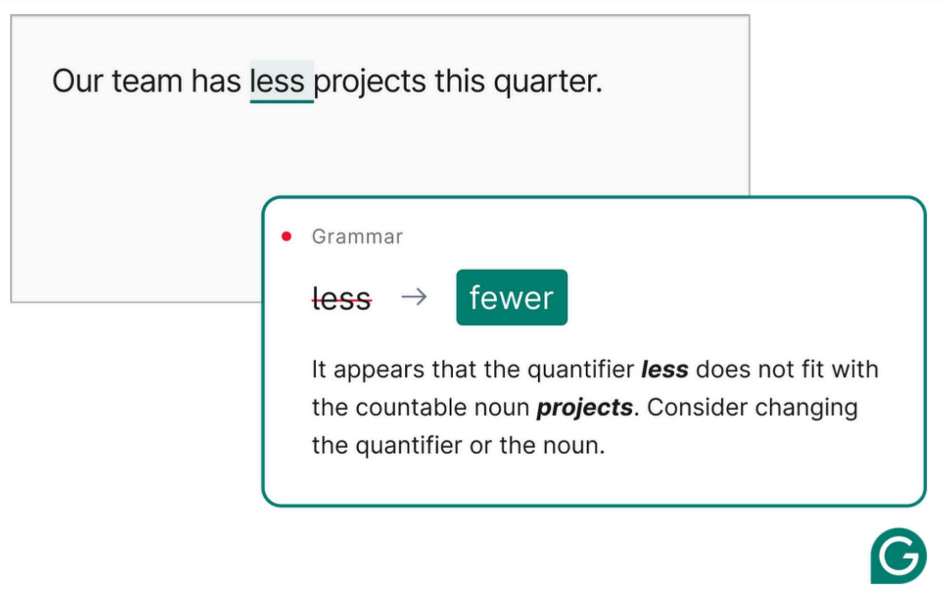


Via Microsoft SwiftKey

Example: Grammar correction

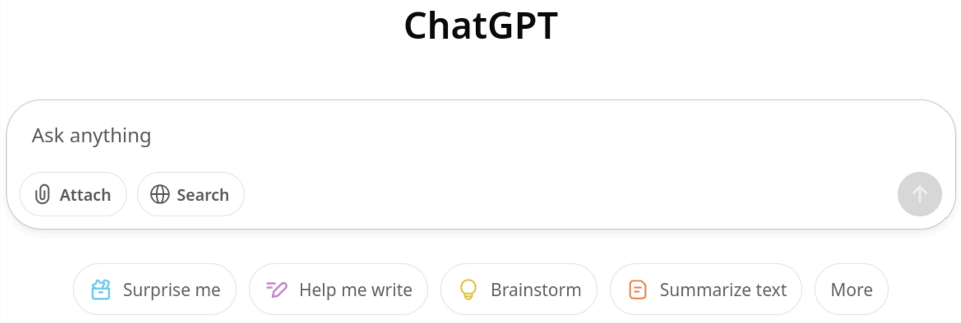
- **Grammatical mistakes** should result in lower probabilities.

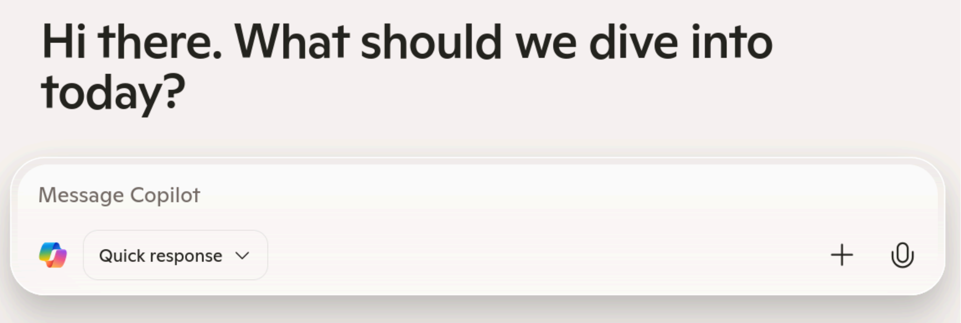
$$P(\text{‘less projects’}) < P(\text{‘fewer projects’})$$

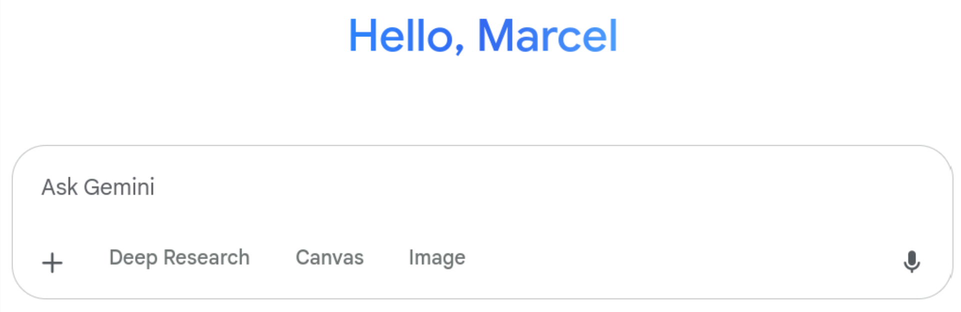


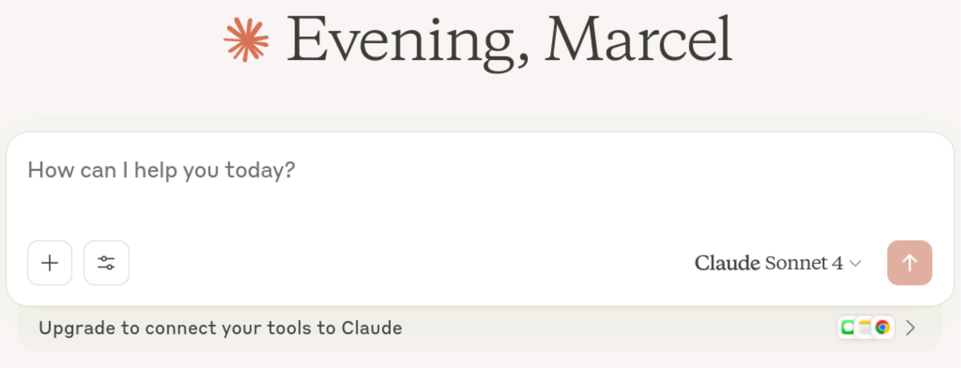
Source: Grammarly

Language modelling is behind all AI assistants









Outline

n-gram Models

- n-grams
- Markov Models
- Applications

Training

- Maximum Likelihood Estimation
- Smoothing
- Out-of-Vocabulary Tokens

Evaluation

- Intrinsic vs. Extrinsic
- Entropy
- Perplexity

n-gram Language Models

What are n-grams?

Definition

An **n-gram** is a sequence of n consecutive tokens.

- We commonly use these terms for $n \in (1, 2, 3)$:

$n = 1$	**Unigram**	`["this"]`
$n = 2$	**Bigram**	`["this", "sounds"]`
$n = 3$	**Trigram**	`["this", "sounds", "great"]`

- For $n > 3$, we would usually speak of 4-grams, 5-grams, etc.

Language modelling with n-grams

- An **n-gram language model** defines a probability distribution over sequences of n tokens.

$$P(w_1 \cdots w_n)$$

- We often look at the **conditional probability** of seeing the **last word** in an n-gram *given* the previous words.

$$P(w_n \mid w_1 \cdots w_{n-1})$$

- n is also called the **order** of the language model.

Unigram models

- A unigram language model is just a **bag-of-words** model.
  - bag-of-words: the order of words doesn't matter

$$P(w_1 \cdots w_m) = \prod_{i=1}^{m} P(w_i)$$

- Here, the probabilities of each word are **mutually independent**.

Markov Assumption

The probability of word w_n only depends on the $n - 1$ previous words.

- For $n = 2$, the probability of "*great*" only depends on "*sounds*":

I think this sounds ***great***

- To compute the probability of a longer sequence, we **multiply the conditional probabilities** of subsequent n-grams:

$$P(\text{'think'} \mid \text{'I'}) \times P(\text{'this'} \mid \text{'think'}) \times P(\text{'sounds'} \mid \text{'this'}) \times P(\text{'great'} \mid \text{'sounds'})$$

Marking sentence boundaries

BOS *I think this sounds great* **EOS**

- For a well-defined model, we also need to **mark the sentence boundaries**.
  - *e.g.* we can't define a probability for the first word otherwise

$$P(\text{‘I’} \mid \text{BOS}) \times P(\text{‘think’} \mid \text{‘I’}) \times \ldots \times P(\text{‘great’} \mid \text{‘sounds’}) \times P(\text{EOS} \mid \text{‘great’})$$

beginning-of-sequence

end-of-sequence

Bigram models

- A bigram language model is a **Markov model** on word sequences.

$$P(w_1 \cdots w_m) = P(w_1 \mid \text{BOS}) \times \left(\prod_{i=2}^{m} P(w_i \mid w_{i-1}) \right) \times P(\text{EOS} \mid w_m)$$

$$P(s_1 \cdots s_k) = \prod_{i=2}^{k} P(w_i \mid w_{i-1})$$

sequence including BOS and EOS

- The **probability of each word** depends only on the **word before it**.
  - That's the Markov property.

n-gram models

BOS BOS BOS *I think this sounds great* EOS

- In general, the input sequence must be **padded with $n-1$ *BOS*** tokens.
  - Note that it's always enough to have one *EOS* token!

- We can then define an ***n*-gram language model** for arbitrary n:

$$P(s_1 \cdots s_k) = \prod_{i=n}^{k} P\left(s_i \mid s_{i-(n-1)} \cdots s_{i-1}\right)$$

sequence including BOS and EOS

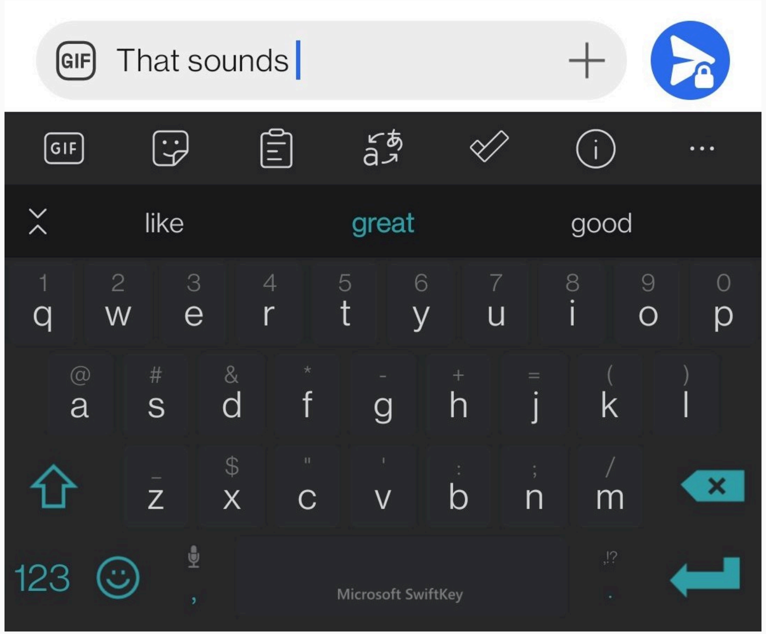
$n-1$ preceding words

Predictive typing with n-gram models

- To predict the next word, we can **choose the word with the highest probability** from our vocabulary:

$$\arg\max_{w \in V} P(w \mid \text{'That sounds'})$$

vocabulary =
set of all possible words



Via Microsoft SwiftKey

Generating text from n-gram models

- We can **generate new text** by **sampling** from the vocabulary.
  - "sampling": picking words based on their probability in the language model
  - Without further conditioning, this does not produce meaningful text…

> *to him at the chamber at his best, and my words brought a newspaper i expect, will reallocate resources, and which analysis alone can bring one of those categories.*

Text sampled from a small English trigram model

Limitations of n-gram models

- n-gram models are **easy to understand and train**, and can be useful tools!
- They are also very limited when it comes to **long-range dependencies**.

$$P(\text{`Sweden is a country in Europe'})$$

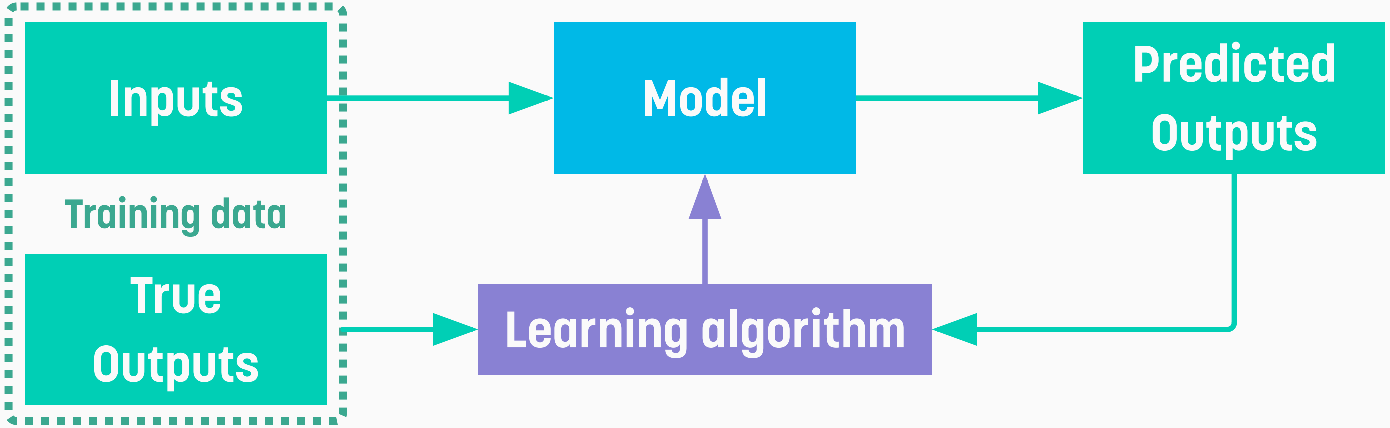
$$P(\text{`Sweden has a population of 10.6 million and its capital is Stockholm'})$$

Important Concepts

- language modelling
- n-grams, unigrams, bigrams, trigrams
- Markov assumption
- sentence boundary markers (BOS, EOS)

Training n-gram Language Models

Reminder: General machine learning methodology



Maximum likelihood estimation

- We can use **maximum likelihood estimation (MLE)** to obtain probabilities for n-gram language models.
- For **unigram** models, this simply means counting the tokens.

$$P(\text{‘sounds’}) = \frac{\#(\text{‘sounds’})}{N}$$

$$N = \sum_{w \in V} \#(w)$$

total number of tokens

⋮	⋮
sound	49
soundcard	1
sounding	2
sounds	22
soup	4
sour	1
source	62
⋮	⋮

MLE for bigram models

- For **bigram** models, we need **conditional probabilities**.

$$P(\text{`good'} \mid \text{`sounds'}) = \frac{\#(\text{`sounds good'})}{\sum_{w \in V} \#(\text{`sounds'} + w)}$$

total number of bigrams starting with "sounds"

sounds familiar	1
sounds good	3
sounds like	14
sounds off	1
sounds on	1
sounds so	1

MLE for n-gram models

- We can **simplify the equation** a bit:

$$P(\text{'good'} \mid \text{'sounds'}) = \frac{\#(\text{'sounds good'})}{\sum_{w \in V} \#(\text{'sounds'} + w)} = \frac{\#(\text{'sounds good'})}{\#(\text{'sounds'})}$$

do you see why?

- We can perform MLE for **any value of n**:

$$P(w_n \mid w_1 \cdots w_{n-1}) = \frac{\#(w_1 \cdots w_n)}{\#(w_1 \cdots w_{n-1})}$$

A problem with maximum likelihood estimation

- If an n-gram never occurred in the training data, we get a **probability of zero**.

$$P(\text{`amazing'} \mid \text{`sounds'}) = \frac{\#(\text{`sounds amazing'})}{\#(\text{`sounds'})} = \frac{0}{22} = 0$$

- Under a Markov model, each sentence containing this n-gram will receive a **total probability of zero**, regardless of the rest of the sentence!

$$P(\text{`I would love to see this happen because it sounds amazing'}) = 0$$

Unseen n-grams in practice

- Shakespeare's collected works contain ca. 31,000 unique words (= *types*).
- There are **961 million possible bigrams** with these words.

$$31000^2 = 961000000$$

- Shakespeare's collected works contain ca. 300,000 unique bigrams.
- This means that **99.97%** of all possible bigrams **occur zero times**.

$$1 - \frac{300000}{961000000} \approx 0.99968$$

Additive (add-k) smoothing

- We can use **add-k smoothing** to ensure the probability is never zero.

$$P(w \mid u) = \frac{\#(uw) + k}{\#(u) + k \cdot |V|}$$

the size of the vocabulary

  - For arbitrary n:

$$P(w_n \mid w_1 \cdots w_{n-1}) = \frac{\#(w_1 \cdots w_n) + k}{\#(w_1 \cdots w_{n-1}) + k \cdot |V|}$$

  - k can be any positive number, i.e. $k \in \mathbb{R}_{>0}$
  - If $k = 1$, this is also called **Laplace smoothing**.

Additive smoothing and the probability mass

- We only have a constant amount of **probability mass** to distribute.
  - Probabilities must always sum up to 1.

- Additive smoothing…
  - **subtracts** probability from actually ***observed*** n-grams, then
  - **redistributes** it equally among all ***possible*** n-grams.

- There are more sophisticated smoothing techniques for language modelling *(but we won't look at them)*.
  - Witten–Bell smoothing, Kneser–Ney smoothing

Redistributing the probability mass

- Let's consider a toy example.
  - vocabulary: $\{awesome, great, sounds, that\}$
  - training data: [*that sounds great*]

awesome	great	sounds	that
$\frac{0}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
0.00	0.33	0.33	0.33

- After smoothing (with $k = 1$), each observation **loses ca. 14%** of its original probability.

awesome	great	sounds	that
$\frac{1}{7}$	$\frac{2}{7}$	$\frac{2}{7}$	$\frac{2}{7}$
0.14	0.29	0.29	0.29

A problem that smoothing cannot solve...

- Smoothing helps with **unseen n-grams**.
  - combinations of tokens that didn't occur in the training data

- Smoothing **does not help** with **out-of-vocabulary** tokens!

> *One ecological change dams bring to rivers is caused by something called* ***hydropeaking***.

Source: New York Times, 02.08.2022

- Remember: We always need a fixed (and finite) vocabulary.

Out-of-vocabulary tokens

- One solution is to **introduce an *UNK* symbol** for "unknown" words.

> *One ecological change dams bring to rivers is caused by something called* **UNK**.

- During training, we replace very rare tokens with *UNK*.
  - *e.g.* all tokens occuring only a single time
  - This makes the model learn probabilities for *UNK*.
- During testing, we replace all out-of-vocabulary tokens with *UNK*.

Important Concepts

- maxixum likelihood estimation
- additive (add-k) smoothing
- out-of-vocabulary words, *UNK* token

Evaluation of Language Models

Intrinsic and extrinsic evaluation

1. **Intrinsic evaluation**: How does the model score on **evaluation metrics**?
   - In classification, we used accuracy, precision, recall, F1-score.
   - In language modelling, these are not very meaningful.
2. **Extrinsic evaluation**: How much does it help on a **downstream task**?
   - "downstream" ≈ anything we want to use the model for
   - *e.g.* How good is a grammatical error detector when using this language model?
   - We use the evaluation metric of the downstream task.

From probabilities to surprisal

- Instead of (raw) probabilities, we often work with **negative log-probabilities**.

$$-\log P(w_1 \cdots w_N)$$

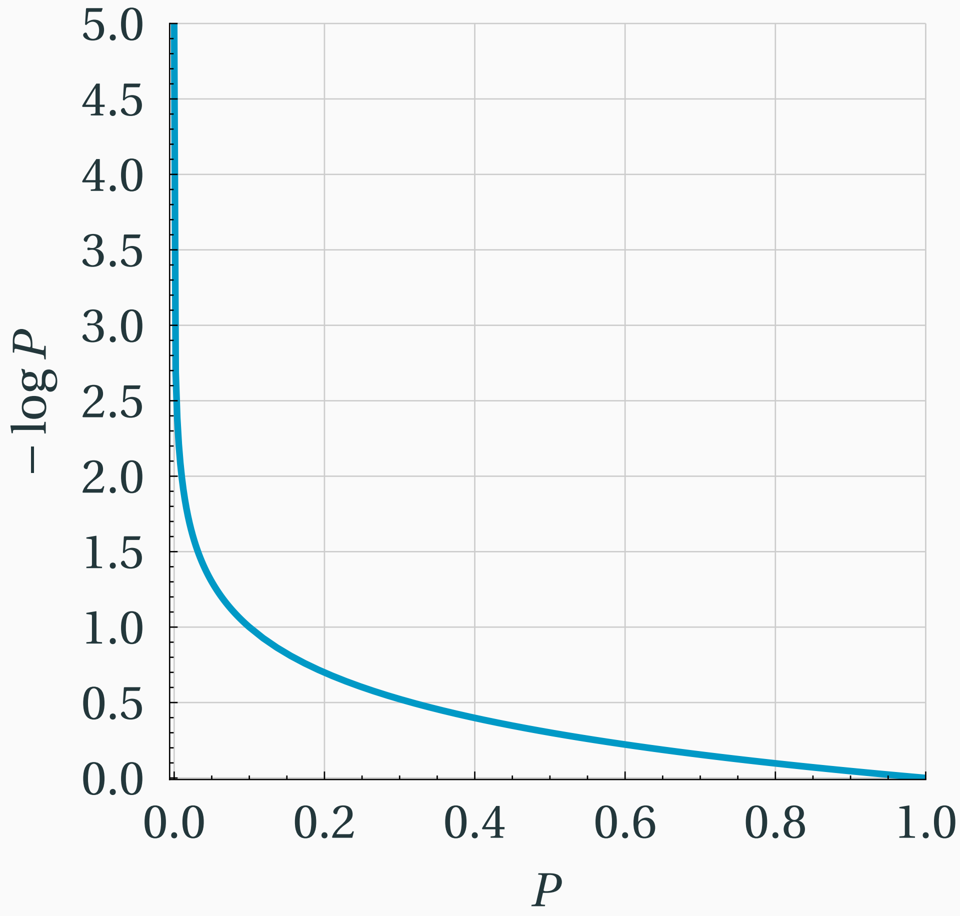
- Intuitively, this measures the **surprisal** of observing the sentence.
  - high probability = low negative log-probability = low surprisal

Definition

The **entropy** of a language model is its **average surprisal** per token.

$$H(w_1 \cdots w_N) = -\frac{1}{N} \log P(w_1 \cdots w_N)$$

Negative log-probabilities



Perplexity

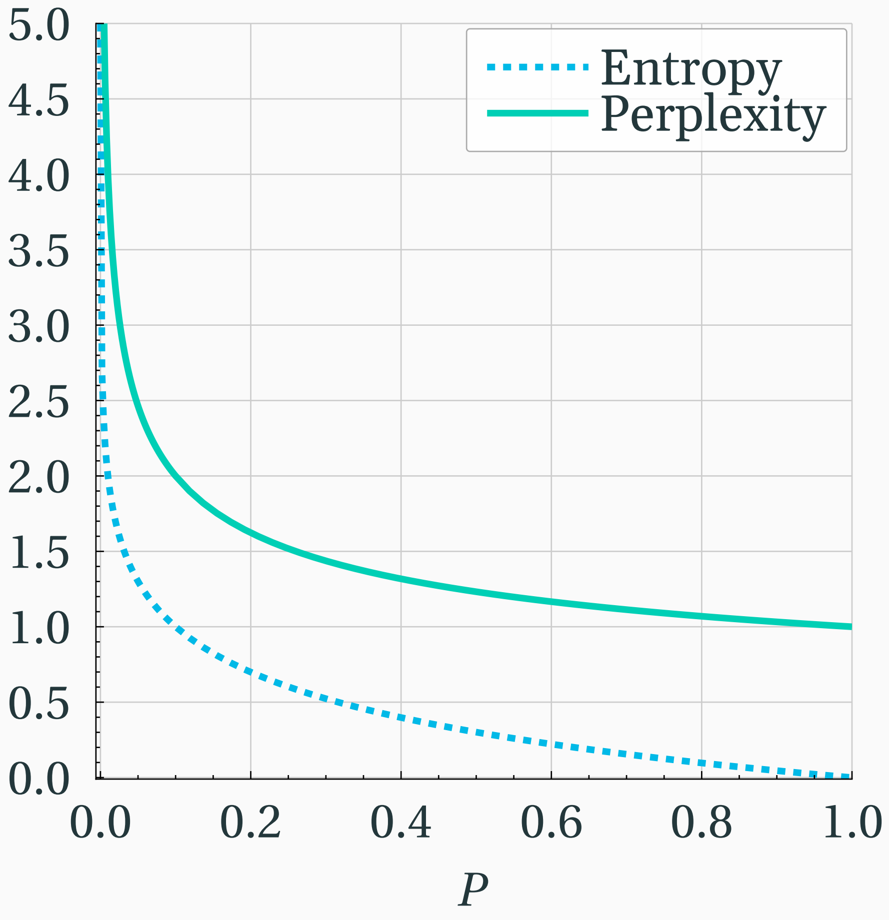
Definition

The **perplexity** of a language model is its **exponentiated entropy**.

$$\mathrm{PPL}(x) = 2^{H(x)} = 2^{-\frac{1}{N}\log_2 P(x)}$$

- Perplexity is one of the **standard metrics** for evaluating language models.
- Intuitively, if the perplexity on a sentence is y, the model is "as surprised" as if it had to, on average, **pick between y tokens with equal probability**.

Entropy vs. perplexity



Interpreting perplexity values

- Typically, perplexity is a **value between 1 and $|V|$**.
  - $\mathrm{PPL}(x) = 1 \longrightarrow$ text can be **predicted perfectly**
  - $\mathrm{PPL}(x) = |V| \longrightarrow$ like **randomly guessing** from the entire vocabulary

- In practice, the absolute perplexity value **depends on the vocabulary**.
  - larger vocabulary = higher uncertainty for each token = higher perplexity

> ⚠ **Caution**
>
> **Comparing two language models** with perplexity only makes sense *if both models use the same vocabulary!*

Important Concepts

- intrinsic vs. extrinsic evaluation
- negative log probabilities
- entropy, perplexity, and how to interpret them