IMPERIAL

How Language Modelling Started

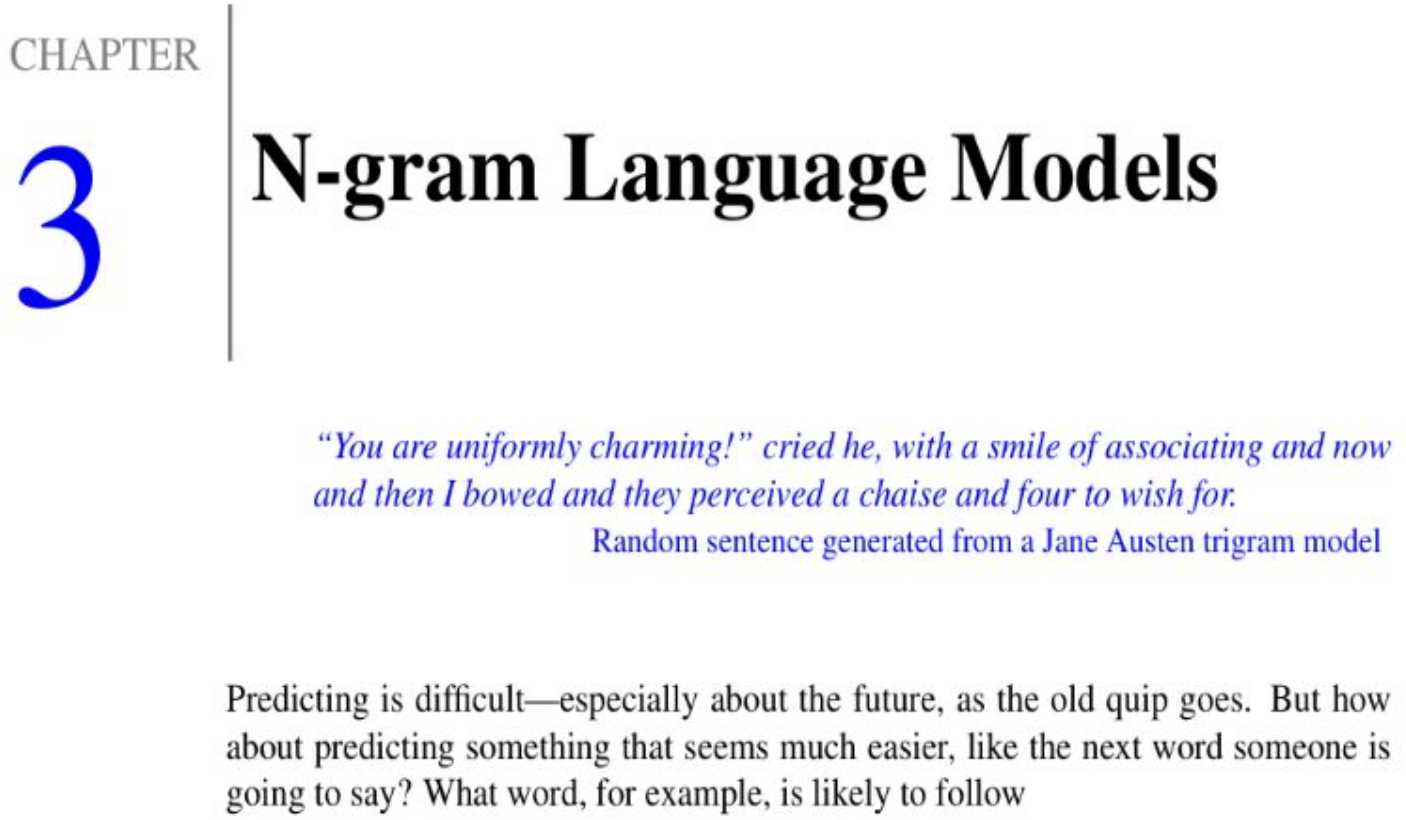
N-grams and the Statistical Era

26/11/2025

Shamsuddeen Muhammad
Google DeepMind Academic Fellow,
Imperial College London
https://shmuhammadd.github.io/

Today

Chapter 3: N-gram Language Models



CHAPTER

3

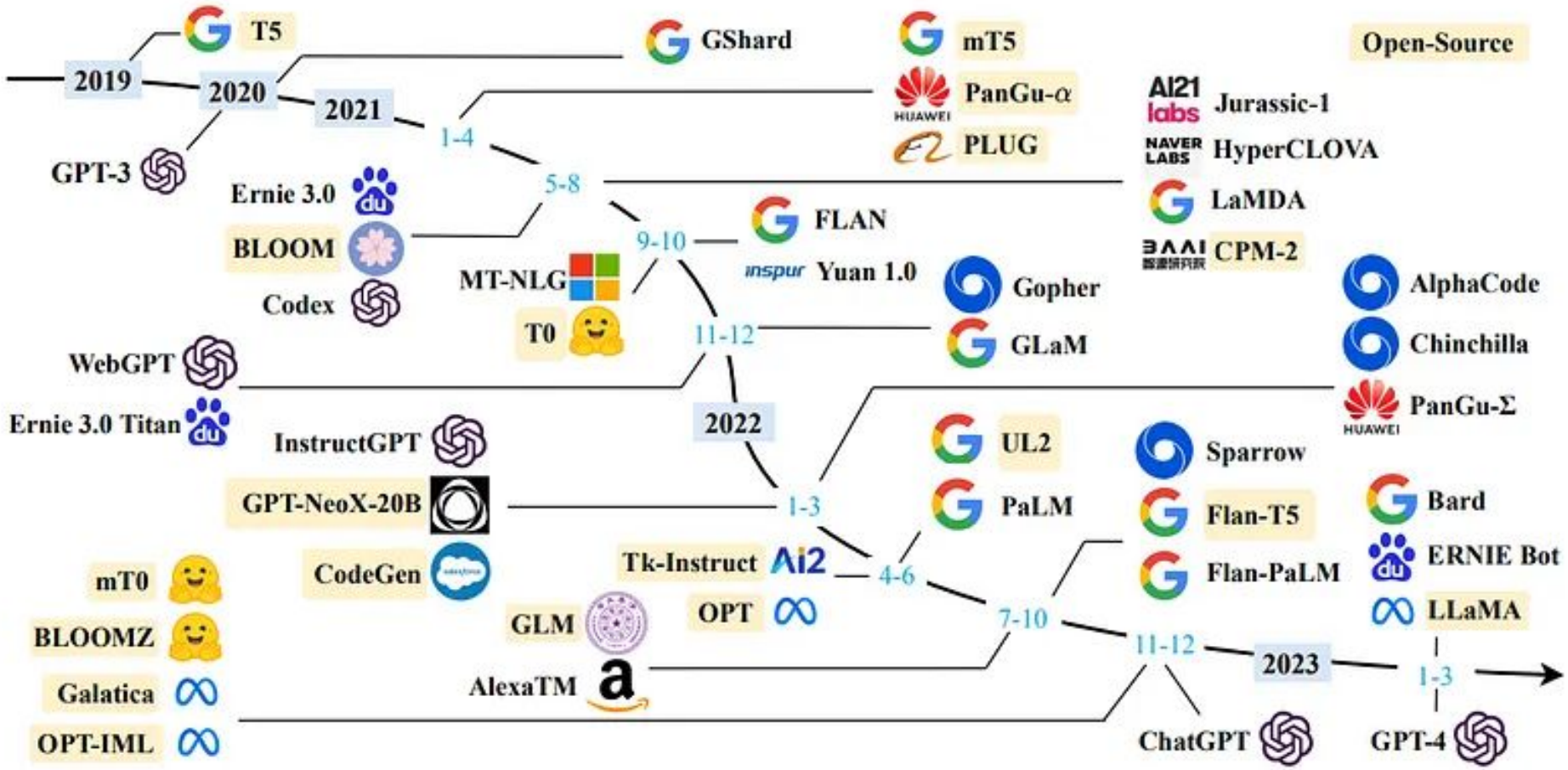
N-gram Language Models

"You are uniformly charming!" cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

Random sentence generated from a Jane Austen trigram model

Predicting is difficult—especially about the future, as the old quip goes. But how about predicting something that seems much easier, like the next word someone is going to say? What word, for example, is likely to follow

Language Model Evolution



Before LLMs: The Statistical Roots of Language Models

- Modern language models did not emerge suddenly — they evolved from *simple statistical foundations.*

- **N-gram models** introduced the first systematic way to estimate probabilities of word sequences using real data.

- Understanding ***N-grams provides the historical and mathematical basis for understanding how today's LLMs work.***

N-gram Language Models

Predicting word

Prediction is difficult, especially about the future.

But how about predicting something that seems much easier, like the next word?

AIMS South Africa is located in …

Cape Town	*Addis Ababa
Limbe	*Lagos

This is what language models do: they predict the next word.

What is a language model?

A **language model (LM)** is a system that **predicts the next word** (or sequence of words) in a sentence.

Assign probabilities to possible next words

Example: "I am going to the ___" → {market, store, mosque, …}

Assign a probability to an entire sentence

(How likely or fluent a sentence is)

*A language model is a machine learning model that learns patterns in text so it can produce a **probability distribution** over possible next words.*

Why predict the next word?

Predicting the next word helps models learn:

- Grammar: learns correct structures
- Word order: what words naturally follow each other
- Meaning: understands context
- Coherence: keeps sentences logically connected
- Fluency: produces natural-sounding text etc

This ability becomes the foundation for tasks like *translation, summarization, autocomplete, speech recognition, and modern LLMs.*

Why word prediction?

It's a helpful part of language tasks

Grammar or spell checking

Their are two midterms	~~Their~~ There are two midterms
Everything has improve	Everything has ~~improve~~ improved

Language Model

Language models don't just predict the next word — they also judge how *likely* an entire sentence is.

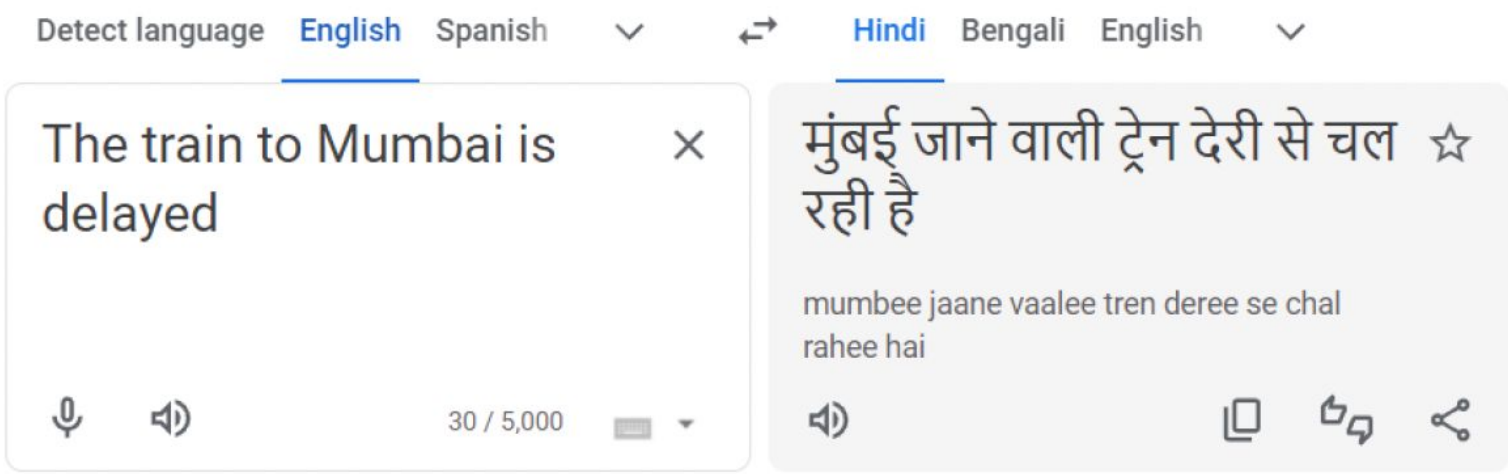
Which sentence sounds more natural?

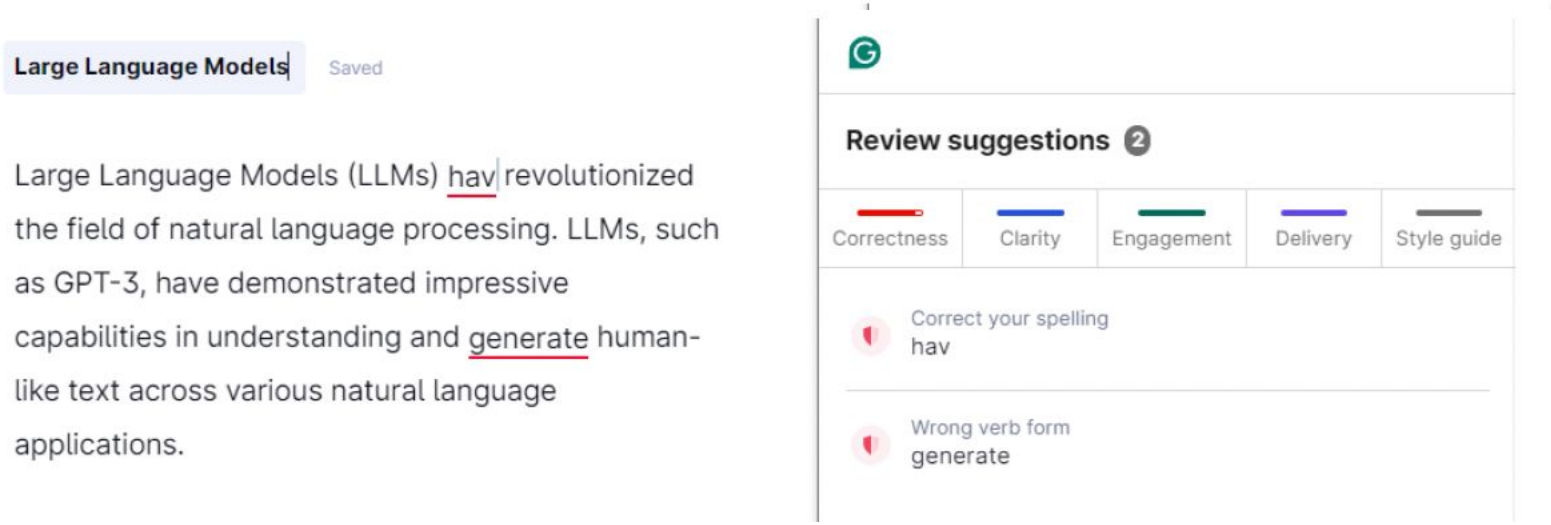
✔️ all of a sudden I notice three guys standing on the sidewalk

❌ on guys all I of notice sidewalk three a sudden standing the

A language model assigns a **higher probability** to the first sentence because it follows natural word order and meaning.

Language Models are Everywhere





Language Models are Everywhere

Model	Open / Closed	Key Strengths
GPT-5	Closed	Frontier reasoning, long-context, multimodal capabilities
Gemini Ultra / Pro	Closed	Strong multimodal + multilingual performance
Claude 3.7	Closed	Excellent alignment, analysis, safety
Llama-4	Open	High-quality open model for fine-tuning
Qwen-3	Open	Strong multilingual + long-context ability
Gemma-2	Open	Efficient, lightweight, high-quality
Mixtral-8x22B	Open	Top MoE open model; efficient inference
Phi-3	Open	Very strong small model for low-resource hardware

Language Models are Statistical

- A language model (whether an n-gram model or a modern neural LM such as GPT or BERT) is fundamentally **statistical** in nature.

- It learns patterns and regularities from *large text corpora and uses these to estimate the probability of word sequences.*

- Rather than applying explicit grammatical rules, *it infers structure implicitly from data, capturing how words and phrases tend to co-occur in real language use*.

Language Modeling (LM) more formally

Goal: compute the probability of a sentence or sequence of words W:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

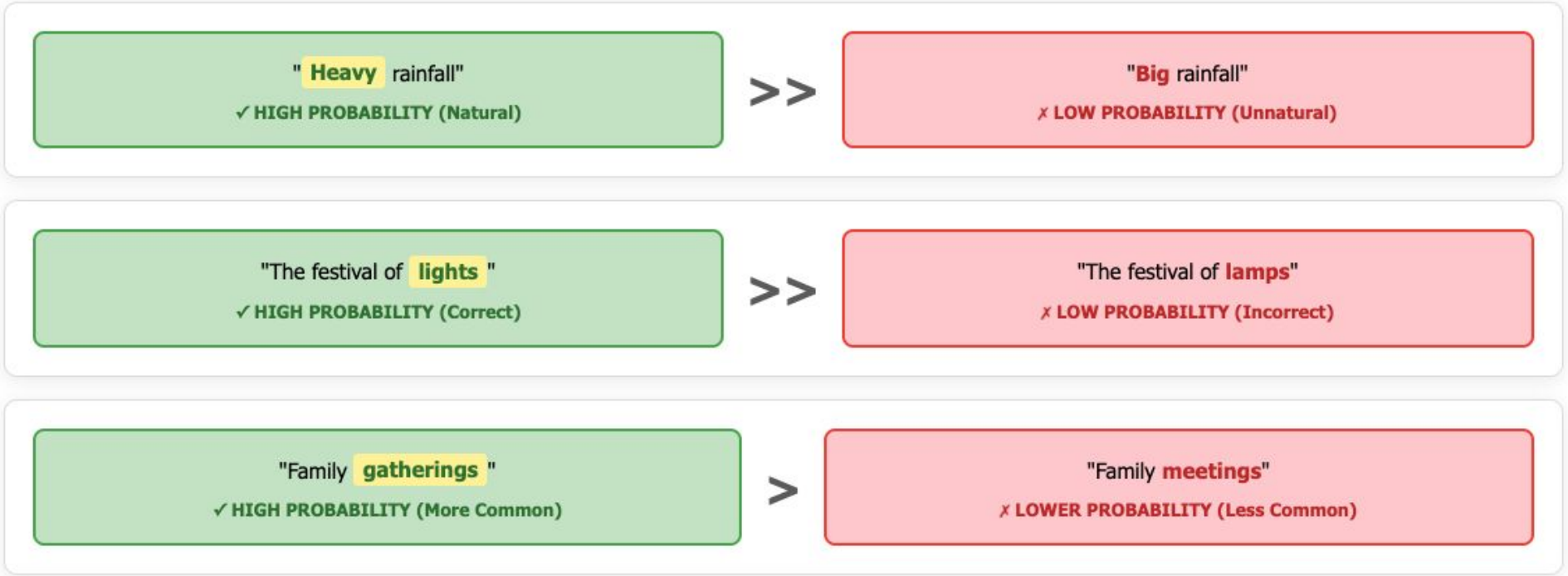
Related task: probability of an upcoming word:

$$P(w_5|w_1,w_2,w_3,w_4) \text{ or } P(w_n|w_1,w_2 \ldots w_{n-1})$$

A language model computes either of these:

$$P(W) \quad \text{or} \quad P(w_n|w_1,w_2 \ldots w_{n-1})$$

How Probabilistic Language Models Work



Language models learn from data which word combinations are more frequent and natural. They assign higher probabilities to sentences that follow common patterns seen in training data, helping systems choose the most plausible interpretation in ambiguous situations.

Language Models in NLP

From Statistical Models to Large Language Models

Categories for Language Models

- Statistical Language Models
- Neural Language Models (NNLMs)
- Transformer-Based Language Models
- Large Language Models (LLMs)
- Multilingual & Low-Resource LMs

Statistical Language Models

N-gram models predict the probability of the **nth** word based on the previous **n−1** words.

They are statistical and rule-free.

N-gram Models (unigram, bigram, trigram...)

> **Key idea:**
> Count frequencies and estimate probabilities from co-occurrences.

Neural Language Models (NNLMs)

Neural-based LMs replace hand-crafted n-gram probabilities with deep learning models.

These are the **first generation** of neural LMs (pre-Transformer).

- Feed-forward neural LMs
- Recurrent Neural Networks (RNNs)
- LSTMs, GRUs
- Word embeddings (Word2Vec, GloVe)

> **Key idea:**
>
> Learn continuous representations and model long-range patterns better than n-grams.

Transformer-Based Language Models

Introduced by *Attention Is All You Need* (Vaswani et al., 2017).

- BERT
- GPT-1/2
- RoBERTa
- XLNet
- T5

> **Key idea:**
>
> Replace recurrence with **self-attention** → scalable, parallelizable.

Large Language Models (LLMs)

These are **very large Transformer models**, not a different architecture.

- GPT-3 → GPT-5
- Llama-2, Llama-3, Llama-4
- Gemini
- Claude
- Mixtral, Qwen, Gemma

example usage:

- GPT-4 (OpenAI) – Used in ChatGPT, Microsoft Copilot
- Claude (Anthropic AI) – AI safety-focused model
- PaLM 2 (Google) – Used in Google Bard
- LLaMA 2 (Meta) – Open-source LLM
- Mistral & Falcon – Open-source powerful alternatives

LLMs are a ***subset*** of Transformer models, not a separate type of LM.

Multilingual & Low-Resource LMs

These are **specialized** or **multilingual transformer models**, but still part of Category 3/4.

- mBERT
- XLM-R
- Aya
- NLLB
- AfriBERTa
- Mistral multilingual
- Many African language-focused models (AfriLMs)

These are ***application-focused***, not a fundamentally different architecture.

Statistical Language Models

N-gram Language Models

Probabilistic Language Model

- **Goal:** Calculate the probability of a sentence or sequence consisting of *n* words

$$P(W) = P(w_1, w_2, w_3, \dots, w_n)$$

or

- **Related Task:** Calculate the probability of the next word conditioned on the preceding words

$$P(w_6 \mid w_1, w_2, w_3, w_4, w_5)$$

A model that calculates either of these is referred to as a **Language Model (LM).**

Chain Rule of Probability

- Definition of **conditional probability**:

$$P(A \mid B) = P(A, B) / P(B)$$

Rewriting: $P(A, B) = P(A \mid B) P(B)$

- More variables: $P(A,B,C,D) = P(A) \cdot P(B \mid A) \cdot P(C \mid A,B) \cdot P(D \mid A,B,C)$

- The **Chain Rule** in general:

$$P(x_1 , x_2 , x_3 , \dots , x_n) = P(x_1) P(x_2 \mid x_1) P(x_3 \mid x_1,x_2) \dots P(x_n \mid x_1,\dots,x_{n-1})$$

Example

Suppose we want to model the probability of a sentence:

”The cat sat on the mat."

$$P(\text{"The cat sat on the mat"}) = P(\text{"The"})P(\text{"cat"} \mid \text{"The"})P(\text{"sat"} \mid \text{"The cat"}) \ldots P(\text{"mat"} \mid \text{"The cat sat on the"})$$

I love machine learning

$$P(\text{"I love machine learning"}) = P(\text{"I"})P(\text{"love"} \mid \text{"I"})P(\text{"machine"} \mid \text{"I love"})P(\text{"learning"} \mid \text{"I love machine"})$$

How do we estimate these probabilities?

The Maximum Likelihood Estimate (MLE)

$$P(w_i \mid w_{i-1}) = count(w_{i-1}, w_i) / count(_{wi-1})$$

$$P(w_i \mid w_{i-1}) = c(w_{i-1}, w_i) / c(w_{i-1})$$

Estimate probability by counting co-occurrences in training data.

Example

$$P(w_i \mid w_{i-1}) = c(w_{i-1}, w_i) / c(w_{i-1})$$

Training Corpus (3 sentences):

<s> I am Sam </s>

<s> Sam I am </s>

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

How can we compute language model probabilities?

Example: Computing Probabilities

$$P(w_i \mid w_{i-1}) = c(w_{i-1}, w_i) / c(w_{i-1})$$

<s> I am Sam </s>

<s> Sam I am </s>

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

Step 1: Count occurrences

P(I | <s>) = 2/3 = 0.67

P(</s> | Sam) = 1/2 = 0.5

<s> appears 3 times

I follows **<s>** 2 times

Sam appears 2 times

</s> follows **Sam** 1 time

Example: All Bigram Probabilities

<s> I am Sam </s>

<s> Sam I am </s>

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

P(I | <s>) = 2/3 = 0.67	P(Sam | <s>) = 1/3 = 0.33	P(am | I) = 2/3 = 0.67
P(</s> | Sam) = 1/2 = 0.5	P(Sam | am) = 1/2 = 0.5	P(do | I) = 1/3 = 0.33
P(I | Sam) = 1/2 = 0.5	P(not | do) = 1/1 = 1.0	P(like | not) = 1/1 = 1.0
P(green | like) = 1/1 = 1.0	P(eggs | green) = 1/1 = 1.0	P(and | eggs) = 1/1 = 1.0
P(ham | and) = 1/1 = 1.0	P(</s> | ham) = 1/1 = 1.0	P(</s> | am) = 1/2 = 0.5

Why these probabilities

- They are **counts from real text** (our training corpus).
- High probability = this bigram appears frequently.
- Low probability = the sequence is uncommon.
- Zero probability = the sequence never appears in training text.

These numbers help the model **learn the patterns of a language** by counting how words follow each other.

What Do These Bigram Probabilities Tell Us?

- Each probability answers the question:

  "Given the previous word, how likely is the next word?"

- Example:
  P(I | <s>) = 0.67
  In our tiny corpus, two out of three sentences start with *I*.

- Another example:
  P(am | I) = 0.67
  Whenever the word *I* appears, it is followed by *am* two-thirds of the time.

P(I | <s>) = 2/3 = 0.67	P(Sam | <s>) = 1/3 = 0.33	P(am | I) = 2/3 = 0.67
P(</s> | Sam) = 1/2 = 0.5	P(Sam | am) = 1/2 = 0.5	P(do | I) = 1/3 = 0.33
P(I | Sam) = 1/2 = 0.5	P(not | do) = 1/1 = 1.0	P(like | not) = 1/1 = 1.0
P(green | like) = 1/1 = 1.0	P(eggs | green) = 1/1 = 1.0	P(and | eggs) = 1/1 = 1.0
P(ham | and) = 1/1 = 1.0	P(</s> | ham) = 1/1 = 1.0	P(</s> | am) = 1/2 = 0.5

These numbers help the model **learn the patterns of a language** by counting how words follow each other. foundation for autocomplete, machine translation etc

Probability of a Sequence

How do we count the probability of sequence? e.g., P (*“Learning” | “I love machine” *)

When the number of text is large?

$$P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)\, P(x_2 \mid x_1)\, P(x_3 \mid x_1, x_2) \ldots P(x_n \mid x_1, \ldots, x_{n-1})$$

Estimate Conditional Probabilities

$$P(\text{begun} \mid \text{The monsoon season has}) = \frac{\text{Count (The monsoon season has begun)}}{\text{Count (The monsoon season has)}}$$

- **Problem:** Enough data is not available to get an accurate estimate of the above quantities.

We'll never see enough data for estimating these The number of possible histories grows exponentially with context length.

Solution: Markov assumption

The Markov assumption says: the next word depends only on a limited recent history, not the entire prefix.

instead of needing counts for every possible long prefix, you only need counts for short ***n-grams***

Markov Chain Assumption

A **Markov Chain** is a stochastic model that assumes the probability of a future state (word) depends only on a fixed number of previous states (words).

This is the foundation for **n-gram language models**, where different assumptions define how many past words influence the next word.

1. Unigram Assumption (Zero-Order Markov Model)
2. Bigram Assumption (First-Order Markov Model)
3. Ngram Assumption (N^{th}-Order Markov Model)

Andrei Markov

Unigram Assumption (Zero-Order Markov Model)

A **unigram model** assumes that each word is **independent** of previous words, meaning it does not rely on any history.

$$P(w_n|w_1, w_2, ..., w_{n-1}) = P(w_n)$$

Thus, the probability of a sentence $S = w_1, w_2, ..., w_n$ is calculated as:

$$P(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} P(w_i)$$

For the sentence **"The cat sat on the mat"**, the unigram model estimates:

$$P(\text{"The cat sat on the mat"}) = P(\text{"The"})P(\text{"cat"})P(\text{"sat"})P(\text{"on"})P(\text{"the"})P(\text{"mat"})$$

Since the model assumes **no dependency** between words, it treats each word as occurring independently.

Bigram Assumption (First-Order Markov Model)

In a bigram model, the probability of each word depends only on the **previous word**. This follows the first-order Markov assumption, meaning:

$$P(w_n|w_1, w_2, ..., w_{n-1}) \approx P(w_n|w_{n-1})$$

Thus, the probability of a sentence $S = w_1, w_2, ..., w_n$ is calculated as:

$$P(w_1, w_2, ..., w_n) = P(w_1) \prod_{i=2}^{n} P(w_i|w_{i-1})$$

For the sentence **"The cat sat on the mat"**, the bigram model estimates:

$$P(\text{"The cat sat on the mat"}) = P(\text{"The"})P(\text{"cat"}|\text{"The"})P(\text{"sat"}|\text{"cat"})$$
$$P(\text{"on"}|\text{"sat"})P(\text{"the"}|\text{"on"})P(\text{"mat"}|\text{"the"})$$

The bigram model captures word dependencies and improves over the unigram model by considering the previous word in sequence prediction.

N-gram Assumption

Bigram Model (1st Order Markov)

Each word depends only on the previous word. This follows the first-order Markov assumption:

$$P(w_n|w_1, w_2, ..., w_{n-1}) \approx P(w_n|w_{n-1})$$

Trigram Model (2nd Order Markov)

Each word depends only on the previous two words. This follows the second-order Markov assumption:

$$P(w_n|w_1, w_2, ..., w_{n-1}) \approx P(w_n|w_{n-2}, w_{n-1})$$

General N-gram Model (kth Order Markov)

Each word depends only on the last k words:

$$P(w_n|w_1, w_2, ..., w_{n-1}) \approx P(w_n|w_{n-k}, ..., w_{n-1})$$

In many probabilistic models, including language models, Hidden Markov Models (HMMs), and Bayesian networks, calculations often involve multiplying many small probabilities. This can lead to numerical underflow and computational inefficiency.

To address this, we perform calculations in log space.

Practical Issues

Language models output very small probabilities; for example:

$$P(w_i \mid w_{i-1}) \approx 10^{-5}.$$

Multiplying many such values produces numbers so tiny that the computer rounds them to 0 (**underflow**):

$$P = P_1 \times P_2 \times P_3 \times \cdots \approx 0.$$

Logs Prevent Underflow

- Logarithms convert multiplication into addition:

$$\log(P_1 P_2 P_3 P_4) = \log P_1 + \log P_2 + \log P_3 + \log P_4.$$

- This keeps numbers in a safe range for the computer.

- Computers add faster than they multiply. Using log-probabilities speeds up N-gram models, and modern neural networks.

Small Numerical Example

Raw probabilities:

$$P_1 = 10^{-5},\ P_2 = 10^{-4},\ P_3 = 10^{-3}.$$

Multiplying:

$$P = 10^{-5} \times 10^{-4} \times 10^{-3} = 10^{-12} \approx 0.$$

In log space:

$$\log P = -5 + (-4) + (-3) = -12.$$

Working in log space makes probability computation, stable, accurate, and efficient

N-gram Language Model

An **N-gram model** considers only the preceding **N −1 words**.

Relation between Markov model and Language Model:

An N-gram Language Model ≡ (N −1) order Markov Model

N-gram Models

- N-grams can be extended to trigrams, 4-grams, or even 5-grams.
- However, simply increasing *n* does **not** fully capture the complexity of natural language.
- Human language often contains **long-distance dependencies** that n-grams cannot model well.

*"The **computer** which I had just put into the machine room on the fifth floor **crashed**."*

The subject **computer** must be connected to the verb **crashed**, but many words intervene, far beyond the reach of typical n-gram windows.

Which Model Generates Better English?

Unigram:

> *"The cat on sat mat the."*

Bigram/Trigram:

> *"The cat sat on the mat near the fireplace and purred softly."*

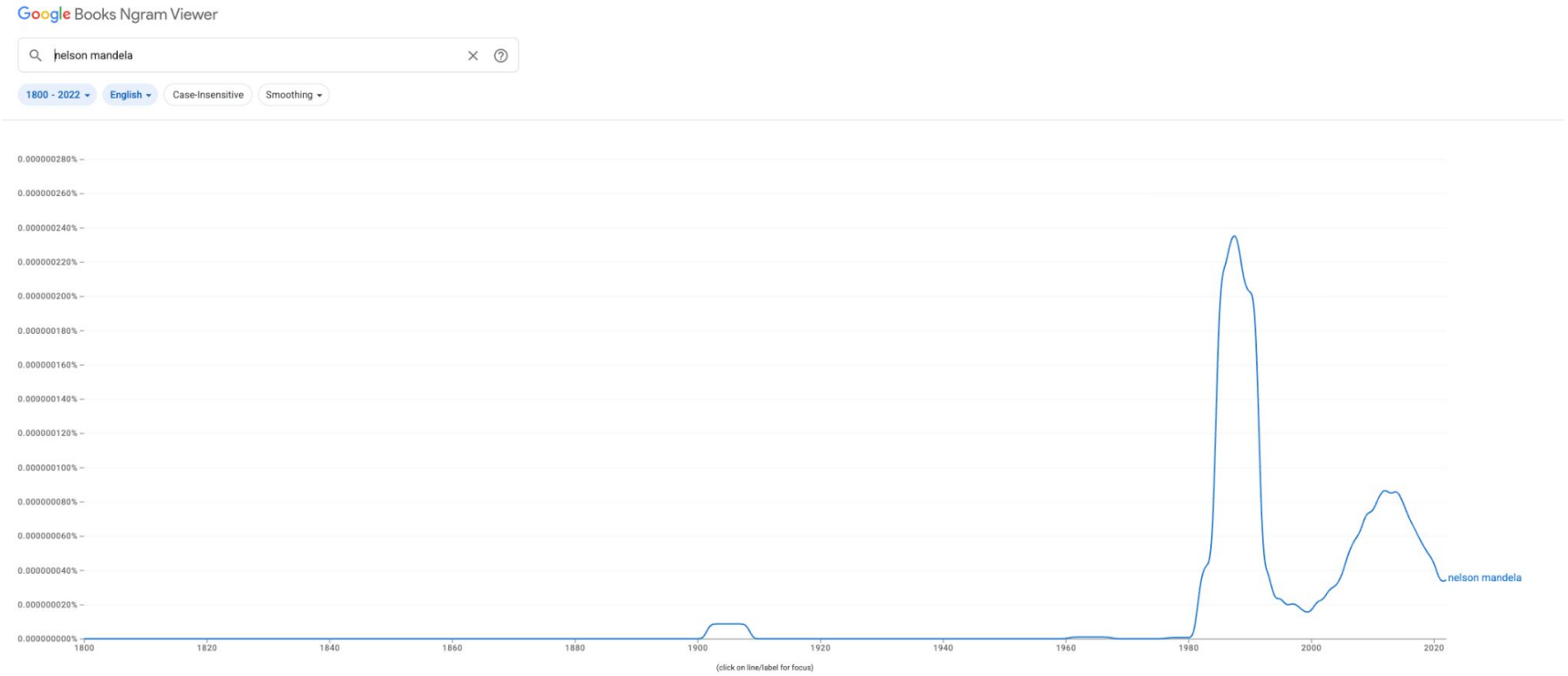
Neural LM / LLM:

> *"The cat sat on the mat near the warm fireplace, purring softly as the evening light faded."*

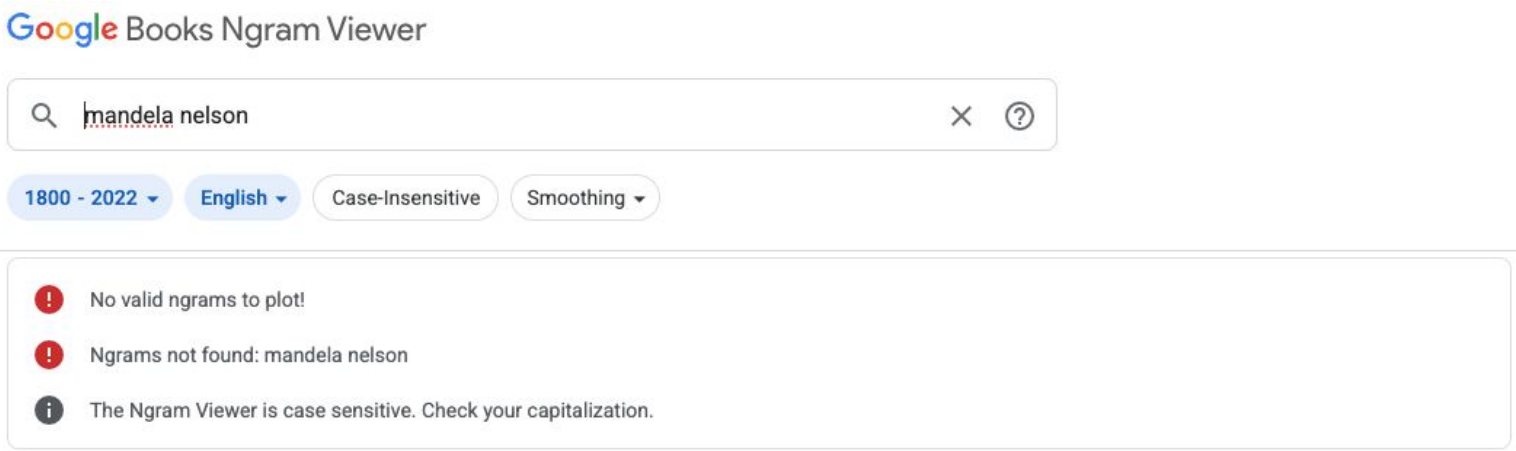
Model	Order	Context Size	Fluency of English Output
Unigram	0th-order	Each word is independent	Poor (Random words, no structure)
Bigram	1st-order	Each word depends on the previous word	Better, but lacks long-term coherence
Trigram	2nd-order	Each word depends on the previous two words	More fluent, better grammar
N-gram (k-order)	k-th order	Each word depends on last k words	Best for structured text, but needs large data
Neural LM	Deep Learning	Full sentence/paragraph context	Superior fluency & coherence

Example: Google Ngram

Google Ngram Viewer (often called *Google NGram*) is a tool created by Google that lets you explore how often words or phrases appear in a very large collection of books over time.
https://books.google.com/ngrams/



Example: Google Ngram



Limitation of N-Gram Language Model

- N-gram models struggle to capture **long-range dependencies** in natural language.
- They only look at the previous *n* words, so important relationships across the sentence are lost.
- They lack global coherence and fail on complex syntax.

Modern Neural Language Models

- **Capture long-range dependencies**
  Self-attention allows every word to attend to all others, regardless of distance.

- **Use deep contextual representations**
  Embeddings encode meaning, synonyms, and semantic relationships.

- **Model full-sentence and paragraph context**
  Enables coherent reasoning and consistent structure.

- **Generate fluent, human-like language**
  Trained on massive datasets with powerful neural architectures.

- **Generalize beyond memorized patterns**
  Can create novel, grammatically correct sentences in different contexts.

Language Model Evaluation

How do we measure the quality of a language model?

Two approaches: extrinsic and intrinsic evaluation.

Extrinsic Evaluation (In-Vivo)

- Best evaluation: test models inside real world (for example, Model A and Model B)
- Examples: spelling correction, speech recognition, MT
- Measure accuracy and compare models

Why Extrinsic Evaluation Is Hard

- Very time-consuming (days or weeks)
- Requires full system integration
- Hard to scale across many models

Intrinsic Evaluation: Perplexity

- Faster alternative to extrinsic evaluation
- Perplexity measures how well a model predicts text
- Lower perplexity → better model

Intrinsic Evaluation: Perplexity

If the model assigns a **high probability** to the full sentence, perplexity is **low**.

If the model assigns a **low probability**, perplexity is **high**.

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

Equivalent form:

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

Lower perplexity
→ better model

What is Perplexity?

- **Perplexity** is a measure of how well a probabilistic model predicts a sample.
- it indicates ***how uncertain the model*** is when generating the next word in a sequence.
- Lower perplexity → Better predictions
- Higher perplexity → Poorer predictions

Laplace Smoothing?

- Laplace Smoothing (also called Add-One Smoothing) is a technique used in **N-gram language models** to handle zero probability issues.

- It ensures that every possible N-gram has a nonzero probability, even if it has never been seen in the training corpus

Why is it needed ?

- In an **N-gram model**, the probability of a word sequence is estimated based on its frequency in the training data.
- If an N-gram (e.g., "rare phrase") does not appear in the training data, its probability becomes **zero**.
- This causes the entire probability of a sentence to be **zero** (multiplicative rule of probabilities).
- Laplace smoothing avoids this issue by assigning a small probability to unseen words.

Limitations of Laplace Smoothing

- It uniformly adds 1 to all word counts, which may overestimate rare words and underestimate frequent ones.

- More advanced smoothing techniques like Add-K Smoothing, Good-Turing, and Kneser-Ney provide better results.

IMPERIAL

Q and A