IMPERIAL

Statistical Language Model

Shamsuddeen Muhammad
Advanced Research Fellow, Imperial College London
18/03/2025

Previous Lecture

Introduction to NLP

How does NLP differ from Computational Linguistics?

What are the two major focuses of NLP?

Natural Language Generation (NLG) and Natural Language Understanding (NLU)?

Provide an example of each.

What is a corpus, and why is it essential for NLP research?

What challenges do low-resource languages face in NLP?

Why do some languages lack NLP resources, and how can we address this issue?

What is tokenization, and why is it an essential preprocessing step in NLP?

What is the difference between stemming and lemmatization?

What is one major challenge in NLP?

Suggested Reading

Chapter 3: Speech and Language Processing

https://web.stanford.edu/~jurafsky/sl...

N-gram Language Models

Next word prediction

Guess the next word in the sequence...

I like pizza with loads of _____.

Previous words in the sentence

Word to be predicted

cheese

P(cheese | I like pizza with loads of)

corn

P(corn | I like pizza with loads of)

tree

P(tree | I like pizza with loads of)

P(cheese| I like pizza with loads of) > **P(corn| I like pizza with loads of)** >> **P(tree| I like pizza with loads of)**

Predicting word

Predicting is difficult—especially about the future.

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

AIMS Cameroon is located in …

Doula
Younde
Limbe

*Addis Ababa
*Lagos

Language Models

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

A language model assigns a probability to each possible next word, or equivalently gives a probability distribution over possible next works.

Language Model

Language models can also assign a probability to an entire sentence.

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

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

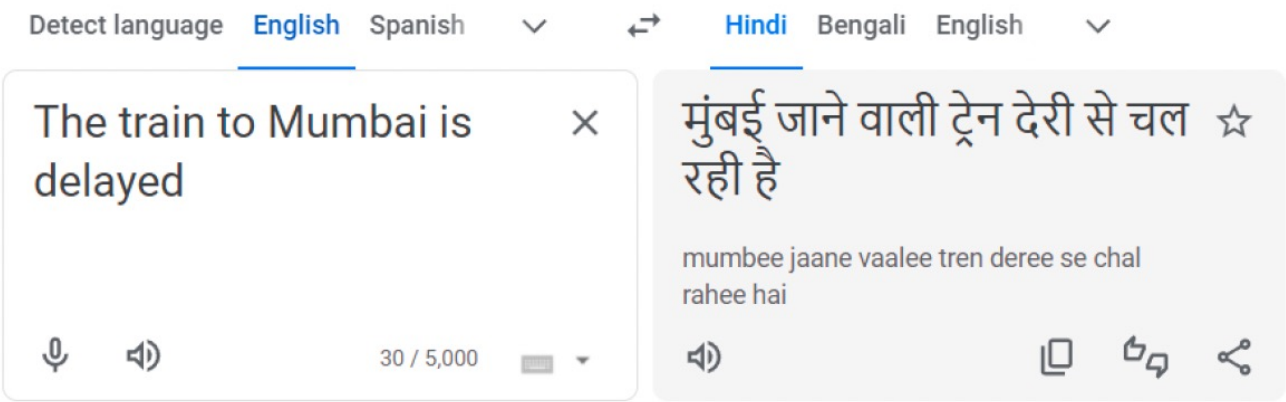
Why do we need to predict next word?

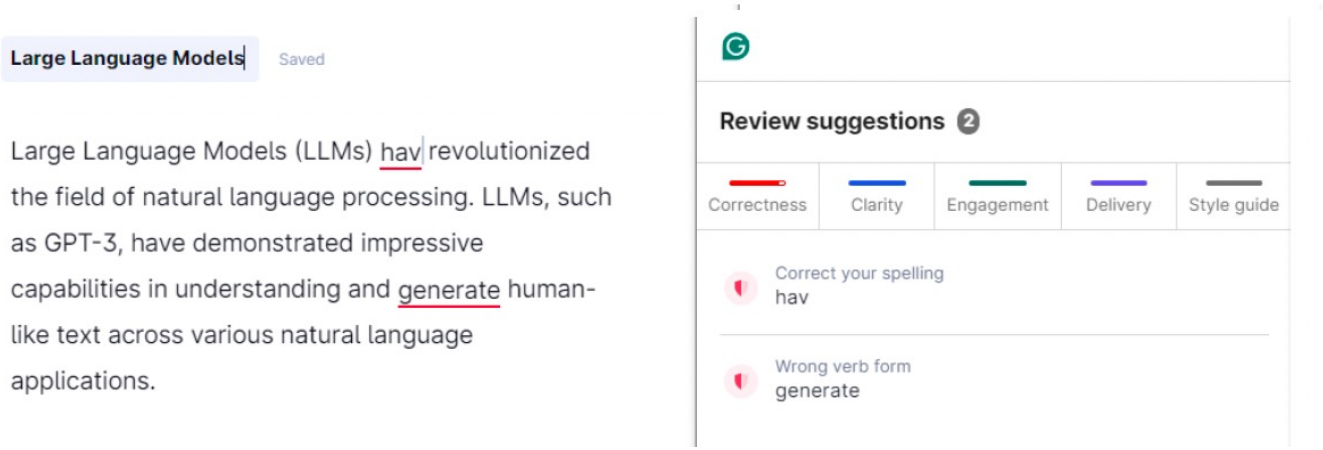
Language Model

Why do we need to predict next word?

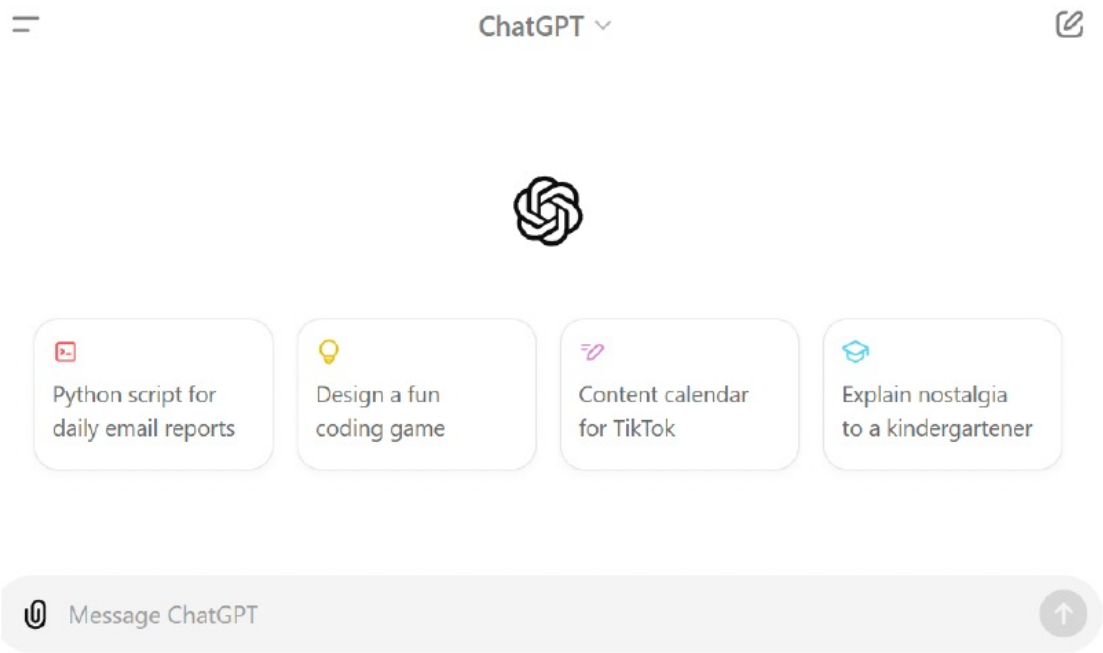
Generation: choosing contextually better words

Language Models are Everywhere





Language Model?



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**

- Speech recognition

  I will be back soonish **I will be bassoon dish**

Why word prediction?

It's how *large language models (LLMs)* work!

LLMs are *trained* to predict words

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

LLMs *generate* text by predicting words

- By predicting the next word over and over again

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:

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

Language Modeling (LM) more formally

A language model computes either of these:

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

Do we call this Grammar or Language Model?

Language Models Are Statistical, While Grammar Is Rule-Based

Grammar refers to a set of explicit syntactic and morphological rules that define how words and phrases should be structured in a language (e.g., subject-verb agreement, tense rules).

A **language model** (e.g., n-gram models, neural LMs like GPT or BERT) is statistical or probabilistic. It learns patterns from large corpora and assigns probabilities to word sequences, without explicitly following grammatical rules.

Grammar rule: *"A singular subject takes a singular verb."*

Language model: *"The cat sleeps" (high probability), "The cat sleep" (low probability), even though it never explicitly learns grammatical rules.*

language models in NL

N-Gram Language Models e.g., Bigram Mode

Neural Language Models (NNLM) e.g., Word2Vec

Transformer-Based Language Models e.g., BERT

Large Language Models (LLMs) e.g., GTP

N-Gram Language Models

N-gram models predict the probability of a word based on the previous words. They are statistical and rule-free.

Examples:

- Unigram Model
- Bigram Model:
- Trigram Model:

Autocomplete & Spell Checking (Google Search, MS Word)
Machine Translation (before neural models like Transformer)

Neural Language Models (NNLM)

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

Examples:

- Feedforward Neural Language Model (NNLM)
- Recurrent Neural Network Language Model (RNN-LM)
- Long Short-Term Memory (LSTM-LM)
- Gated Recurrent Unit (GRU-LM)

Transformer-Based Language Models

Transformers introduced the **self-attention mechanism**, replacing RNNs for superior contextual understanding.

Examples:

- BERT (Bidirectional Encoder Representations from Transformers)
- GPT (Generative Pre-trained Transformer)
- T5 (Text-to-Text Transfer Transformer)
- XLNet (Permutation-based Transformer)
- BLOOM (Multilingual Open Science Model)

Example Usage:

- ChatGPT (GPT-3, GPT-4)
- Google Search (BERT-powered search ranking)
- Code Completion (GitHub Copilot, OpenAI Codex

Large Language Models (LLMs)

These models, built on billions of parameters, generate human-like text.

Examples:

- 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

Probabilistic Language Model (N-GRam)

- **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).**

Probability of a Sentence

Let's consider the following sentence:

The monsoon season has begun

- How to compute the probability of the sentence?

$$P(W) = P(\text{“The monsoon season has begun”})$$
$$= P(\text{The, monsoon, season, has, begun})$$

We compute the above joint probability by using the principles of **Chain Rule of Probability**.

Chain Rule of Probability

- Definition of **conditional probability**:

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

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

*This states that the **joint probability** of A and B occurring together is the probability of B, multiplied by the probability of A given that B has already occurred.*

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

- The **Chain Rule** in general:

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

Example

Suppose we want to model the probability of a sentence:

"The cat sat on the mat." Using the chain rule:

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

I love machine learning

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

Probability of a Sequence

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P(W) = P("The monsoon season has begun")

= P(The, monsoon, season, has, begun)

= P(The) x P(monsoon | The) x P(season | The monsoon) x P(has | The monsoon season) x P(begun | The monsoon season has)

How do we count the probability of sequence? e.g., P (season | The monosoon)

How to estimate these probabilities

One way to estimate this probability is directly from relative frequency counts

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.
- **Solution:** Markov Assumption

When the number of text is large?

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

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)

Andrei Markov

N-ram Language Models

Simplest kind of Language Models

An n-gram is a sequence of **n-words** : a 2-gram (which we'll call bigram) is a two-word sequence of words like The water, or water of

3 gram (a trigram) is a three-word sequence of words like The water of, or water of Walden.

'ngram' to mean a probabilistic model that can estimate the probability of a word given the n-1 previous words, and thereby also to assign probabilities to entire sequences.

Markov Assumption

Every next state depends only the previous k states

- Chain Rule:

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i | w_1 w_2 \ldots w_{i-1})$$

- Applying Markov Assumption we condition on only the preceding k words:

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i | w_{i-k} \ldots w_{i-1})$$

- Probabilistic Language Models exploit the **Chain Rule of Probability** and **Markov Assumption** to build a probability distribution over sequences of words.

Markov Assumption (n-gram Approximation)

Computing the full chain rule probability is often impractical because it requires considering all previous words.

Instead, we apply **the Markov assumption**, which simplifies the dependency by assuming that a word depends only on a fixed number of preceding words.

The **Markov assumption** states that:

$$P(w_1, w_2, \ldots, w_n) = \prod_i P(w_i | w_{i-k}, \ldots, w_{i-1})$$

where k is a fixed window size (e.g., in a bigram model, $k = 1$; in a trigram model, $k = 2$).

Example: Bigram Model ($k = 1$)

$$P(\text{"The cat sat down"}) \approx P(\text{"The"})P(\text{"cat"}|\text{"The"})P(\text{"sat"}|\text{"cat"})P(\text{"down"}|\text{"sat"})$$

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:

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

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:

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

This means that we do not need to remember everything, just the last k words!

N-gram Language Model

Relation between Markov model and Language Model:

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

N-gram Models

- We can extend to trigrams, 4-grams, 5-grams
- In general, this is an insufficient model of language
- because language has long-distance dependencies:

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

Which Model Generates Better English?

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

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

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

Why Neural is this better?

- Uses full sentence context.
- Captures grammar, meaning, and coherence.
- Can generate creative and human-like responses.

How do we estimate these probabilities?

The Maximum Likelihood Estimate

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

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

Example

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

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

How can we compute language model probabilities?

Example

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

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

$$P(\texttt{I} \mid \texttt{<s>}) = \tfrac{2}{3} = .67$$
$$P(\texttt{</s>} \mid \texttt{Sam}) = \tfrac{1}{2} = 0.5$$

Example

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

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

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

More Examples

Berkeley Restaurant Project sentences

- 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

Raw bigram counts

- Out of 9222 sentences

	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

Raw bigram probabilities

Normalize by unigrams:

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

Result:

	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

Bigram estimates of sentence probabilities

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

What kinds of knowledge?

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

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

Why Use Log Space in Probability Calculations?

Probabilities are often very small numbers, for example:

$$P(w_i \mid w_{i-1}) \approx 0.00001$$

When multiplying many small probabilities, the result can become so small that it rounds to zero due to the limits of floating-point precision.

Taking the logarithm of probabilities converts multiplication into addition, reducing the risk of underflow:

$$P = P_1 \times P_2 \times P_3 \times P_4$$

$$\log P = \log P_1 + \log P_2 + \log P_3 + \log P_4$$

This transformation ensures that even very small probabilities remain numerically stable.

Why Use Log Space?

In log space, we replace expensive multiplications with faster additions:

$$\log(P_1 \times P_2 \times P_3) = \log P_1 + \log P_2 + \log P_3$$

Computational efficiency is especially important in **large-scale language models and neural networks**.

Google Ngram

https://books.google.com/ngrams/

Limitation of N-Gram Language Model

- An insufficient model of language since they are **not effective in capturing long-range dependencies present in language**.

  - Example:

    The **project**, which he had been working on for months, was finally **approved** by the committee.

    The above example highlights the long-distance dependency between “project” and “approved”, where the context provided by earlier words affects the interpretation of later parts of the sentence.

The solution: **Large language models**

- can handle much longer contexts
- because of using embedding spaces, can model synonymy better, and generate better novel strings

Why N-Gram?

A nice clear paradigm that lets us introduce many of the important issues for **large language models**

- **training** and **test** sets
- the **perplexity** metric
- **sampling** to generate sentences
- ideas like **interpolation** and **backoff**

Evaluation and Perplexity

How to evaluate N-gram models

"Extrinsic (in-vivo) Evaluation"

Best evaluation for comparing models A and B

- Put each model in a task
  - spelling corrector, speech recognizer, MT system
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly
  - How many words translated correctly
- Compare accuracy for A and B

Difficulty of extrinsic evaluation of N--gram models

Extrinsic evaluation

- Time-consuming; can take days or weeks

So

- Sometimes use **intrinsic** evaluation: **perplexity**

Training sets and test sets

We train parameters of our model on a **training set**.

We test the model's performance on data we haven't seen.

- A **test set** is an unseen dataset; different from training set.
  - Intuition: we want to measure generalization to unseen data
- An **evaluation metric** (like **perplexity**) tells us how well our model does on the test set.

Choosing training and test sets

If we're building an LM for a specific task

- The test set should reflect the task language we want to use the model for

If we're building a general-purpose model

- We'll need lots of different kinds of training data
- We don't want the training set or the test set to be just from one domain or author or language.

Training on the test set

We can't allow test sentences into the training set

- Or else the LM will assign that sentence an artificially high probability when we see it in the test set
- And hence assign the whole test set a falsely high probability.
- Making the LM look better than it really is

This is called "Training on the test set"

Bad science!

Dev sets

If we test on the test set many times we might implicitly tune to its characteristics

- Noticing which changes make the model better.

So we run on the test set only once, or a few times

That means we need a third dataset:

- A **development test set** or, **devset**.
- We test our LM on the devset until the very end
- And then test our LM on the **test set** once

Intuition of perplexity as evaluation metric: How good is our language model?

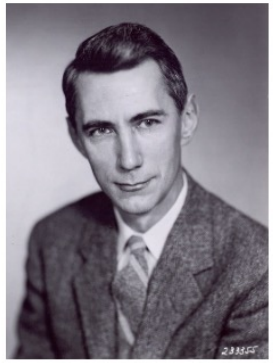
Intuition: A good LM prefers "real" sentences

- Assign higher probability to “real” or “frequently observed” sentences
- Assigns lower probability to “word salad” or “rarely observed” sentences?

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

Intuition of perplexity 2: Predicting upcoming words



Claude Shannon

The Shannon Game: **How well can we predict the next word**?

- Once upon a _____
- That is a picture of a _____
- For breakfast I ate my usual _____

time	0.9
dream	0.03
midnight	0.02
…	
and	1e-100

Unigrams are terrible at this game (Why?)

A good LM is one that assigns a higher probability to the next word that actually occurs

Intuition of perplexity 3: The best language model is one that best predicts the entire unseen test set

We said: a good LM is one that assigns a higher probability to the next word that actually occurs.

Let's generalize to all the words!

- The best LM assigns high probability to the entire test set.

When comparing two LMs, A and B

- We compute P_A(test set) and P_B(test set)
- The better LM will give a higher probability to (=be less surprised by) the test set than the other LM.

Intuition of perplexity 4: Use perplexity instead of raw probability

Probability depends on size of test set

- Probability gets smaller the longer the text
- Better: a metric that is **per-word**, normalized by length

Perplexity is the inverse probability of the test set, normalized by the number of words

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

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

Intuition Behind Perplexity

If a model assigns high probability to a sentence, its perplexity is **low** → **good model**.

If a model assigns low probability, its perplexity is **high** → **poor model**.

What is 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