Language Models

A Deep Dive into N-Gram Language Models

NLP Students

Contents

1 Introduction: The Autocorrect Brain

We use Language Models every day. When you type a message and your phone suggests the next word, or when you start a search query and Google finishes the sentence—that's a Language Model (LM) in action.

The Big Idea: What is a Language Model?

A Language Model is a probabilistic machine that performs two fundamental tasks:

1. **Prediction:** Calculating the probability of the next word given the preceding words. $P(w_{next}|w_1, w_2, ..., w_{n-1})$.
2. **Assessment:** Calculating the probability of an entire sequence of words. $P(W)$.

The Intuition: The Fluency Judge

A language model acts as a "Fluency Judge." It internalizes the patterns, grammar, and idioms of a language by analyzing vast amounts of text. When presented with a sentence, it determines how "natural" or "likely" that sentence is.

- P("I saw a white swan") $\rightarrow$ High Probability (Sounds natural)
- P("Swan a white saw I") $\rightarrow$ Low Probability (Sounds strange)

1.1 Why Do We Need Them?

Language models are the backbone of modern NLP, crucial for tasks where fluency and prediction are essential:

- **Machine Translation (MT):** When translating, an MT system might generate candidates like "large winds tonight" and "high winds tonight." The LM assigns a higher probability to "high winds," selecting the more idiomatic phrase.
- **Spell Correction:** The model sees "The office is about fifteen minuets from my house." It recognizes that P("fifteen minutes") is vastly higher than P("fifteen minuets") in this context.
- **Speech Recognition (ASR):** ASR systems use LMs to disambiguate acoustically similar phrases (e.g., "I saw a van" vs. "Eyes awe of an").
- **Predictive Text/Autocomplete:** The most direct application—predicting what you want to type next.

2 The Mechanics of Prediction: The Chain Rule and Its Crisis

How do we calculate the probability of a sentence like $W = (w_1, w_2, w_3, w_4)$? We need a fundamental tool from probability theory: The Chain Rule.

2.1 The Chain Rule of Probability

The chain rule allows us to break down the joint probability of a sequence into a product of conditional probabilities.

The Mathematics: The Chain Rule

For a sequence of events A, B, C, D:

$$P(A, B, C, D) = P(A) \times P(B|A) \times P(C|A, B) \times P(D|A, B, C)$$

Applied to words:

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

The Intuition: Building the Story Step-by-Step

The Chain Rule models language generation as a sequential process.

1. Start with the probability of the first word.
2. Multiply by the probability of the second word, given the first.
3. Multiply by the probability of the third word, given the first two.
4. And so on...

2.2 The Crisis: Data Sparsity and Computational Impossibility

The Chain Rule is mathematically sound. So why can't we just use it directly? We face two insurmountable obstacles.

The Challenge: The Sparsity Crisis

Let's try to calculate $P(\text{transparent}|\text{Its water is so})$.
To estimate this using counts (Maximum Likelihood Estimation), we would need:

$$P(\text{transparent}|\text{Its water is so}) = \frac{\text{Count}(\text{Its water is so transparent})}{\text{Count}(\text{Its water is so})}$$

The Problem: Language is creative. Most specific long sequences of words have never occurred before.

- The numerator ("Its water is so transparent") might have a count of 0, even on the entire web. This gives a probability of 0.
- The denominator ("Its water is so") might also be 0. This results in division by zero (undefined).

The Challenge: The Computational Impossibility

Even if we had infinite data, storing the counts is impossible. With a vocabulary of 50,000 words, the number of possible 5-word sequences (5-grams) is 50000^5. This number is astronomical. We cannot store counts for every conceivable sequence.

We need a way to approximate the probability without relying on the full history.

3 The N-Gram Approximation: The Markov Assumption

To solve the computational crisis, we introduce a simplifying assumption known as the **Markov Assumption**.

The Intuition: The "Short Attention Span" Model

Instead of assuming the next word depends on the *entire history* of the sentence, we assume it only depends on a fixed, small window of preceding words. We deliberately limit the model's "attention span."

The Mathematics: The Markov Assumption

We approximate the full history with a limited context of length k:

$$P(w_i|w_1, ..., w_{i-1}) \approx P(w_i|w_{i-k}, ..., w_{i-1})$$

This assumption forms the basis of **N-Gram Models**. "N" refers to the total size of the sequence being considered.

3.1 Types of N-Gram Models

3.1.1 The Unigram Model (N=1)

The most radical simplification. We assume words are completely independent of each other. The context size is 0.

$$P(w_1, ..., w_n) \approx \prod_i P(w_i)$$

Example: P(I like food) ≈ P(I)P(like)P(food).

Flaw: P(I like food) = P(food like I). It ignores word order entirely.

3.1.2 The Bigram Model (N=2)

We assume the next word depends only on the immediately preceding word (context size 1).

$$P(w_i|\text{History}) \approx P(w_i|w_{i-1})$$

Example: P(I like food) ≈ P(I|<s>) P(like|I) P(food|like). (Where <s> is a start token).

3.1.3 The Trigram Model (N=3)

We assume the next word depends on the two preceding words (context size 2).

$$P(w_i|\text{History}) \approx P(w_i|w_{i-2}, w_{i-1})$$

3.2 The Trade-off

- **Higher N (e.g., 5-gram):** Captures more context and grammar (more fluent). However, it dramatically increases sparsity and computational cost.
- **Lower N (e.g., Bigram):** Very robust and easy to compute (less sparse), but captures limited context (less fluent).

The Challenge: Long-Distance Dependencies

The main limitation of N-gram models is their inability to capture long-distance dependencies.

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

To know whether to use "is" or "are", the model needs to remember the subject ("computer" or "computers") from much earlier in the sentence. A typical N-gram model (N<15) cannot do this. (This limitation motivated the development of RNNs and Transformers).

4 Estimating Probabilities: Maximum Likelihood Estimation (MLE)

Now that we have the N-gram structure, how do we estimate the actual probabilities (e.g., $P(\text{food}|\text{like})$)?

The standard approach is **Maximum Likelihood Estimation (MLE)**.

The Intuition: Just Count What You See

MLE is the most intuitive approach: the probability of an event is simply its relative frequency in the training data. We estimate the probabilities such that they maximize the likelihood of the observed training data.

4.1 The MLE Formula

For a Bigram model:

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

The numerator is the count of the bigram. The denominator is the count of the context (the unigram).

4.2 A Worked Example

Let's train a Bigram model on a tiny corpus. We first pad the sentences with start <s> and end </s> tokens.

Corpus:

1. <s> I am Sam </s>
2. <s> Sam I am </s>
3. <s> I do not like green eggs and ham </s>

Walkthrough: Bigram MLE Walkthrough

Calculating Probabilities:

1. $P(\text{I}|\text{<s>})$: How often does "I" follow <s>? 2 times. How often does <s> occur? 3 times.

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

2. $P(\text{Sam}|\text{am})$: How often does "Sam" follow "am"? 1 time. How often does "am" occur?

2 times.

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

3. $P(\text{am}|\text{I})$: How often does "am" follow "I"? 2 times. How often does "I" occur? 3 times.

$$P(am|I) = 2/3 \approx 0.67$$

4.3 The Berkeley Restaurant Project (BeRP) Example

Let's look at a more realistic (though still small) example from the BeRP corpus (9222 sentences).

Selected Unigram Counts (Denominators):

I	want	to	eat	food
2533	927	2417	746	1093

Selected Bigram Counts (Numerators):

	I	want	to	eat	food
I	5	827	0	9	0
want	2	0	608	1	6
to	2	0	4	686	0
eat	0	0	2	0	2

Calculating Probabilities (MLE):

- P(want | I) = Count(I want) / Count(I) = 827 / 2533 ≈ 0.33
- P(to | want) = Count(want to) / Count(want) = 608 / 927 ≈ 0.66
- P(food | I) = Count(I food) / Count(I) = 0 / 2533 = 0

This highlights the knowledge captured by the model: grammar (P(to|want) is high), but also the inherent sparsity (P(food|I) = 0).

5 The "Zero Problem" and the Need for Smoothing

The MLE approach has a critical flaw, illustrated perfectly by $P(\text{food}|I) = 0$ in the previous example.

5.1 The Catastrophe of Zero Probability

In any realistic corpus, most possible word combinations will never occur. The training data is just a sample of the language.

If we encounter a bigram in the test set that never appeared in the training set, the MLE assigns it a probability of 0.

The Challenge: The Multiplication Killer

Because the probability of a sentence is calculated by multiplying the probabilities of its N-grams, a single zero probability N-gram will zero out the probability of the entire sentence.
$P(S) = P(w_1) \times P(w_2|w_1) \times \cdots \times \mathbf{0} \times \cdots = 0$

This means our model incorrectly believes the test sentence is impossible. This is a form of **overfitting** to the training data.

The Intuition: The Black Swan Fallacy

Just because you haven't seen a black swan doesn't mean they don't exist. Similarly, just because a specific N-gram wasn't in the training data doesn't mean it's impossible in the real world. We must assign some small probability to unseen events.

This process is called **Smoothing** or **Discounting**.

6 Smoothing Techniques

Smoothing techniques adjust the probabilities calculated by MLE by stealing probability mass from frequent events and redistributing it to the rare or unseen events.

6.1 Laplace Smoothing (Add-1 Smoothing)

The simplest smoothing technique is Laplace Smoothing, often called "Add-1 Smoothing".

The Big Idea: The "Benefit of the Doubt" Approach

Pretend we have seen every possible N-gram one more time than we actually did.

We add 1 to every count in the numerator. To ensure the probabilities still sum to 1, we must adjust the denominator. For every word type we added 1 to in the numerator (which is the size of the vocabulary, V), we must add 1 to the denominator.

The Mathematics: Laplace Smoothing for Bigrams

$$P_{Laplace}(w_n|w_{n-1}) = \frac{\text{Count}(w_{n-1}, w_n) + 1}{\text{Count}(w_{n-1}) + V}$$

Where V is the size of the vocabulary.

6.1.1 Laplace Example (BeRP Corpus)

Let's revisit the BeRP example. Assume the vocabulary size V = 1446.

Original MLE:

- P(want | I) = 827 / 2533 ≈ 0.33
- P(food | I) = 0 / 2533 = 0

Laplace Smoothed: The new denominator for context "I" is Count(I) + V = 2533 + 1446 = 3979.

- $P_{Laplace}$(want | I) = (827 + 1) / 3979 = 828 / 3979 ≈ 0.21
- $P_{Laplace}$(food | I) = (0 + 1) / 3979 = 1 / 3979 ≈ 0.00025

Analysis: We solved the zero problem! P(food|I) is now a small, non-zero number. However, notice that P(want|I) decreased significantly (from 0.33 to 0.21).

6.1.2 The Flaw of Laplace Smoothing

Laplace smoothing is a blunt instrument. It often steals too much probability mass from frequent events, especially when the vocabulary V is large. It is generally poor for language modeling, though useful in other domains like text classification.

6.2 Advanced Techniques: Interpolation and Backoff

We need more sophisticated ways to handle sparsity, particularly for higher-order N-grams (Trigrams, 4-grams). The intuition is to combine information from different N-gram orders.

6.2.1 Backoff (e.g., Katz Backoff)

Idea: Use the highest-order N-gram if we have sufficient evidence; otherwise, "back off" to a lower-order N-gram.

The Intuition: Consulting the Experts Hierarchically

1. Ask the Trigram expert for $P(w_3|w_1, w_2)$.
2. If the Trigram expert hasn't seen it, we "back off".
3. Ask the Bigram expert for $P(w_3|w_2)$.
4. If the Bigram expert hasn't seen it, back off again and ask the Unigram expert for $P(w_3)$.

(Note: Proper backoff models use "discounting" to reserve probability mass for the backoff steps, ensuring the distribution sums to 1).

6.2.2 Interpolation (e.g., Jelinek-Mercer)

Idea: Always mix the estimates from all N-gram orders (Trigram, Bigram, Unigram) using a weighted average.

Interpolation generally performs better than Backoff.

The Mathematics: Linear Interpolation (Trigram)

$$P_{Interp}(w_n|w_{n-2}, w_{n-1}) = \lambda_1 P(w_n|w_{n-2}, w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

Where the weights (lambdas) must sum to 1: $\sum \lambda_i = 1$.

The Intuition: Averaging the Experts

We ask all three experts (Trigram, Bigram, Unigram) for their opinion and then take a weighted average. We might set $\lambda_1 = 0.7, \lambda_2 = 0.2, \lambda_3 = 0.1$, indicating we trust the trigram most, but still incorporate the others.

The λ values are learned using a **held-out dataset** (validation set) to optimize the model's predictive power.

6.2.3 Stupid Backoff (Web-Scale Solution)

For massive datasets, standard backoff and interpolation can be too computationally expensive. "Stupid Backoff" (Brants et al., 2007) is a simplification used at scale.

It simply backs off to the lower order if the higher order count is zero, multiplying by a fixed penalty (α, e.g., 0.4) at each step, without complex discounting. It doesn't produce a true probability distribution, but it works very well and is very fast on huge datasets.

7 Practical Considerations

7.1 Working in Log Space

In practice, we never multiply probabilities directly. Probabilities are small numbers (between 0 and 1). Multiplying many small numbers together leads to **numerical underflow**—the result becomes so small that the computer can no longer represent it and rounds it to zero.

The Solution: Perform calculations in logarithmic space.

The Mathematics: Log Probabilities

We use the property: $\log(A \times B) = \log(A) + \log(B)$.

$$\log(P(W)) = \sum_{i=1}^{n} \log(P(w_i|w_{i-1}))$$

Multiplication becomes addition, which is numerically stable and faster.

7.2 Handling Unknown Words (OOV)

What happens if the test set contains a word that was never seen during training? This is the Out-Of-Vocabulary (OOV) problem.

- **Closed Vocabulary:** Assumes we know all possible words in advance. (Rarely true).
- **Open Vocabulary:** Assumes new words can appear.

The Strategy: The <UNK> Token

1. Define a fixed vocabulary V during training (often based on frequency).
2. Replace any word in the training data that is not in V with a special token **<UNK>** (Unknown).
3. Train the N-gram probabilities for <UNK> just like any other word.
4. At test time, any OOV word is replaced by <UNK>, and the model uses the learned probability for it.

8 Evaluating Language Models: Perplexity

How do we know if Language Model A is better than Language Model B?

8.1 Extrinsic Evaluation (In-vivo)

The best way is to embed the models into a real-world task (like Machine Translation) and measure the improvement in that task (e.g., BLEU score, Word Error Rate). This is accurate but time-consuming.

8.2 Intrinsic Evaluation: Perplexity (PP)

We often need a faster, approximate measure of quality during model development. The standard intrinsic metric is **Perplexity**.

The Big Idea: Perplexity: The Measure of Surprise

Perplexity measures how well the probability distribution predicts a sample (the test set). A lower perplexity indicates a better model, meaning the model is less "surprised" by the test data.

The Mathematics: Definition of Perplexity

Perplexity (PP) of a model P on a test set W $(w_1, \ldots, w_N)$ is the inverse probability of the test set, normalized by the number of words (N).

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

The Intuition: The Branching Factor Analogy (The Dice Roll)

Perplexity can be interpreted as the weighted average branching factor of the language model.
Imagine a model predicting digits (0-9), where each digit is equally likely (P=1/10). The perplexity is 10. This means that at any point, the model is as confused when predicting the next item as if it were choosing randomly among 10 possibilities. It's like rolling a 10-sided die to predict the next word.
If a language model has a perplexity of 109 on the Wall Street Journal, it means the model is as confused by the text as if it were choosing randomly among 109 words at each step.

8.3 Comparing Models

Training on the Wall Street Journal (WSJ) corpus:

N-gram Order	Perplexity
Unigram	962 (High surprise)
Bigram	170
Trigram	109 (Lower surprise)

Lower perplexity confirms that models incorporating more context (Trigram) are significantly better predictors of the language than models with less or no context (Unigram).

9 Conclusion

N-gram models, while simple, are foundational to Natural Language Processing. They provide a robust statistical method for predicting the next word and assessing the fluency of text. By leveraging the Markov Assumption, we overcome the computational impossibility of the full Chain Rule. However, the inherent data sparsity requires sophisticated smoothing techniques like Interpolation and Backoff to prevent the catastrophic "Zero Problem" and allow the models to generalize. While modern neural models have surpassed N-grams in performance, the core concepts of probabilistic language modeling remain essential.