N-Gram Language Models

Detailed Notes — N-Gram Language Models (Part I: Foundations)

Koorosh Asil Ghrehbaghi
K.N. Toosi University of Technology (KNTU) — B.Sc. Computer Science
Instructor: Dr. Maryam Abdolali

Compiled on: September 9, 2025

Abstract

These notes present a complete, step-by-step introduction to n-gram language models. We start with probability basics and the chain rule, introduce the Markov assumption, show how n-grams (unigrams, bigrams, trigrams) are estimated from data, discuss smoothing (why it is needed and how it works), and finish with evaluation using perplexity. The presentation uses concrete examples, worked calculations, pseudocode, and practical recommendations.

Contents

1 Overview

Language models assign probabilities to sequences of words. They are the backbone of many NLP systems: speech recognition, machine translation, autocorrect, and text generation. The purpose of this chapter is to explain how simple statistical language models (n-grams) work and why they are useful.

Definition

A *language model* is a probability distribution over sequences of words $w_1^T = w_1, w_2, \ldots, w_T$. It answers questions such as: How likely is the sentence "I like sunflowers"? or What is the probability of the next word given the previous words?

Importance & Use Cases

N-gram models are simple, efficient, and still useful. They teach key probabilistic ideas (chain rule, conditional probability, smoothing) and remain a baseline for modern models.

2 Probability basics: chain rule and conditional probability

The fundamental identity for language modeling is the **chain rule**:

$$P(w_1^T) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2) \cdots P(w_T \mid w_1^{T-1}). \tag{1}$$

This is just repeated application of conditional probability: $P(a, b) = P(a)P(b \mid a)$.

Example

For a three-word sequence $w_1 w_2 w_3$:

$$P(w_1 w_2 w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2).$$

Computing these full-conditionals directly is infeasible because the conditioning context w_1^{t-1} grows with t. There are too many possible histories in natural language.

3 The Markov assumption and n-grams

To make modeling tractable we make a *limited history* assumption (Markov): the probability of the next word depends only on the previous $n-1$ words, not the entire history.

Definition

An **n-gram model** assumes:

$$P(w_t \mid w_1^{t-1}) \approx P(w_t \mid w_{t-n+1}^{t-1}).$$

For $n = 1$ we get the *unigram* model, $n = 2$ the *bigram* model, $n = 3$ the *trigram* model, etc.

This reduces Eq. (1) to:

$$P(w_1^T) \approx \prod_{t=1}^{T} P(w_t \mid w_{t-n+1}^{t-1}).$$

T is the total length of the sequence

Example

For a bigram ($n = 2$) model we estimate

$$P(w_1^T) \approx P(w_1) \prod_{t=2}^{T} P(w_t \mid w_{t-1}).$$

4 Estimating n-gram probabilities from counts

We estimate probabilities with counts from a training corpus. Let $C(\cdot)$ denote counts.

Definition

Maximum-likelihood estimate (MLE) for an n-gram probability:

$$\hat{P}_{\text{MLE}}(w_t \mid w_{t-n+1}^{t-1}) = \frac{C(w_{t-n+1}^{t})}{C(w_{t-n+1}^{t-1})},$$

where $C(w_{t-n+1}^{t})$ is the count of the full n-gram and $C(w_{t-n+1}^{t-1})$ the count of the preceding context.

Example

Small corpus (tokenized, with sentence boundaries `<s>` and `</s>`):

```
<s> i like sunflowers </s>
<s> i like flowers </s>
<s> i love sunflowers </s>
```

In n-gram language models, the symbol <s> is used to mark the beginning of a sentence, allowing the model to learn which words tend to appear at the start. When we count bigrams, only the word immediately following <s> is considered a starting word. For example, in the sentence "I like sunflowers," we represent it as <s> I like sunflowers </s>. Here, <s> I is counted as a starting bigram, while like sunflowers is treated as a regular bigram within the sentence. Words like "sunflowers" or multi-word sequences like "to be" occur later in the sentence, so they are never immediately after <s> and therefore are not counted as starting words. This approach ensures that the model correctly learns sentence-initial probabilities without mistakenly treating words appearing later as sentence starters.

Counts (selected):

ngram	count	comment
C(<s>)	3	three sentences
C(i)	3	'i' occurs 3 times
C(i like)	2	bigram 'i like'
C(like sunflowers)	1	
C(love sunflowers)	1	

Using MLE, the bigram estimate is for example:

$$\hat{P}(\text{like} \mid \text{i}) = \frac{C(\text{i like})}{C(\text{i})} = \frac{2}{3}.$$

5 Sentence start and end tokens

To model $P(w_1)$ and short contexts, we usually add special tokens: `<s>` for start-of-sentence and `</s>` for end-of-sentence. This allows the model to represent probability of the first word and to compute probabilities of entire sentences.

6 Normalization and probability distributions

An estimated conditional distribution must satisfy

$$\sum_{w \in V} P(w \mid h) = 1 \quad \text{for every history } h.$$

Under MLE the normalization is automatic because

$$\sum_{w} \frac{C(hw)}{C(h)} = \frac{1}{C(h)} \sum_{w} C(hw) = 1. \qquad \sum_{w \in V} P_{\text{MLE}}(w \mid h) = \sum_{w \in V} \frac{C(hw)}{C(h)} = \frac{\sum_{w \in V} C(hw)}{C(h)} = \frac{C(h)}{C(h)} = 1$$

However smoothing changes counts and we must ensure the smoothed probabilities still sum to one; most smoothing formulas are constructed to preserve normalization.

Intuition

Bigram model learns only what it has seen
If a word pair never appeared, the model thinks it's impossible, hence probability = 0

This is exactly the problem smoothing tries to solve: we pretend each unseen pair happened a little bit, so the model doesn't assign zero probability.

7 Sparsity and the need for smoothing

Even with large corpora many plausible n-grams have zero counts (data sparsity). Assigning zero probability to them leads to problems: entire sentence probabilities become zero. Smoothing redistributes some probability mass from seen to unseen events.

I like sunflowers
I love sunflowers
I like roses

- The bigram `(I, like)` occurs 2 times → $C(I, like) = 2$
- The bigram `(I, love)` occurs 1 time → $C(I, love) = 1$
- The bigram `(I, adore)` **does not appear** anywhere → $C(I, adore) = 0$

So when we compute the **MLE probability**:

$$P(\text{adore} \mid I) = \frac{C(I, \text{adore})}{C(I)} = \frac{0}{3} = 0$$

That's why the model assigns **0 probability** to the unseen bigram `(I, adore)` — it **never occurred in the training data**.

7.1 Add-one (Laplace) smoothing

Add-one smoothing adds one to each n-gram count:

$$\hat{P}_{\text{add-1}}(w \mid h) = \frac{C(hw) + 1}{C(h) + |V|},$$

where $|V|$ is vocabulary size. This guarantees non-zero probabilities but biases estimates (especially for large vocabularies).

Example

If $C(h) = 10$, $C(hw) = 0$ and $|V| = 5000$, then

$$\hat{P}(w \mid h) = \frac{1}{10 + 5000} = 1/5010,$$

which can be very small and distorts higher-frequency events.

For a bigram model:

$$P(w \mid h) = \frac{C(h, w) + k}{C(h) + k \cdot |V|}$$

- $C(h, w)$ = count of the bigram (h, w) in your training data
- $C(h)$ = count of the history word h
- V = vocabulary (all possible words)
- $k > 0$ = smoothing parameter
  - **Add-1 smoothing:** $k = 1$
  - Add-0.5 or other values are also possible (tunable)

7.2 Add-k smoothing

Generalization: add a small constant α $(0 < \alpha < 1)$ to counts.

7.3 Good-Turing smoothing (intuition)

Good-Turing redirects probability mass based on counts of counts N_r (how many n-grams occur exactly r times). The adjusted estimate for items seen r times uses the ratio of N_{r+1}/N_r. The method works well for rare events but needs reliable estimation of N_r.

7.4 Backoff and interpolation

Rather than smoothing each n-gram independently, we can back off to lower-order models when data is sparse.

Definition

Katz backoff assigns probability:

$$P_{\text{katz}}(w \mid h) = \begin{cases} d \;\; d_{hw} & \text{if } C(hw) > 0, \\ \alpha(h) P_{\text{katz}}(w \mid h') & \text{otherwise,} \end{cases}$$

where h' is the shorter history, d_{hw} are discounted counts, and $\alpha(h)$ renormalizes the leftover mass.

Linear interpolation (simpler) mixes orders:

$$P_{\text{interp}}(w \mid h) = \lambda_n \hat{P}(w \mid h) + \lambda_{n-1} \hat{P}(w \mid h') + \cdots + \lambda_1 \hat{P}(w),$$

with $\sum \lambda_i = 1$ and $\lambda_i \geq 0$.

Example

Trigram interpolation example:

$$P(w_t \mid w_{t-2} w_{t-1}) = \lambda_3 \hat{P}(w_t \mid w_{t-2} w_{t-1}) + \lambda_2 \hat{P}(w_t \mid w_{t-1}) + \lambda_1 \hat{P}(w_t).$$

Interpolation weights can be learned on held-out data.

7.5 Kneser-Ney smoothing (intuition)

Kneser-Ney is a state-of-the-art method which discounts counts and uses a special backoff distribution computed from continuation counts (how often a word appears as a novel continuation). Intuitively it favours words that appear in many different contexts.

8 Evaluation: Cross-entropy and Perplexity

For a test corpus $W = w_1^N$, the average negative log-likelihood (cross-entropy) under model M is:

$$H(p, M) = -\frac{1}{N} \sum_{i=1}^{N} \log_2 P_M(w_i \mid w_{i-n+1}^{i-1}).$$

Perplexity is defined as:

$$\text{Perplexity}(W) = 2^{H(p,M)} = 2^{-\frac{1}{N} \sum \log_2 P_M(\cdot)}.$$

Lower perplexity indicates a better model.

Example

If model assigns probability 0.25 to each of 4 equally likely words, cross-entropy is $-\log_2 0.25 = 2$ bits and perplexity $= 2^2 = 4$.

9 Worked numerical example

Corpus (same small corpus as above). Compute bigram probabilities (MLE) and the probability of sentence "<s> i like sunflowers </s>".

Counts recap:

ngram	count
C(<s> i)	3
C(i like)	2
C(like sunflowers)	1
C(sunflowers </s>)	2
C(i)	3

Bigram MLE probabilities:

$$P(i \mid < s >) = \frac{C(< s > i)}{C(< s >)} = 3/3 = 1.$$

$$P(like \mid i) = 2/3, \quad P(sunflowers \mid like) = 1/2, \quad P(< /s > \mid sunflowers) = 2/2 = 1.$$

Sentence probability (product):

$$P = 1 \times \frac{2}{3} \times \frac{1}{2} \times 1 = \frac{1}{3}.$$

10 Practical pseudocode

```
# Train n-gram model (counts + MLE)
Input: tokenized_corpus, n
counts = defaultdict(int)
context_counts = defaultdict(int)
for sentence in tokenized_corpus:
    tokens = ['<s>'] + sentence + ['</s>']
    for i in range(len(tokens)):
        for k in range(1, n+1):
            if i-k+1 >= 0:
                ngram = tuple(tokens[i-k+1:i+1])
                counts[ngram] += 1
                context = ngram[:-1]
                context_counts[context] += 1
# MLE probs
probs = {ngram: counts[ngram]/context_counts[ngram[:-1]] for ngram in counts if len(
    ngram)>1}

def score_sentence(sentence, probs, n):
    tokens = ['<s>'] + sentence + ['</s>']
    score = 1.0
    for i in range(1, len(tokens)):
        context = tuple(tokens[max(0, i-n+1):i])
        ngram = context + (tokens[i],)
        p = probs.get(ngram, 0.0) # backoff or smoothing needed
        score *= p
    return score
```

11 Practical recommendations

Importance & Use Cases

Use at least trigrams for moderate corpora (tens of millions tokens) and backoff + Kneser-Ney smoothing for best performance. Always reserve a held-out set to tune interpolation weights or discount parameters. Evaluate with perplexity and also check downstream task performance.

12 Summary

N-gram models are simple probabilistic models that approximate full conditional distributions using fixed-length histories. Key ideas: chain rule, Markov assumption, maximum-likelihood estimation from counts, and smoothing to handle sparsity. Though neural models now dominate, n-grams remain essential for understanding the foundations of language modeling and for many practical applications.

Prepared by: Koorosh Asil Ghrehbaghi
Course: KNTU Computer Science (Instructor: Dr. Maryam Abdolali)