Language model: N-gram

Lecture-6, 7, 8 (n-gram model)

Types of Language Models in NLP

Language modeling is a fundamental concept in Natural Language Processing (NLP). It involves building **statistical or machine learning models** that can predict the probability of a sequence of words. In essence, these models learn the patterns and structures of language, enabling machines to understand and generate human-like text.

Here's a breakdown of the key types:

1. **Statistical Language Models:**
   These models use statistical techniques to determine the probability of word sequences. They rely on counting word occurrences in large text corpora.
   - **N-gram Models:**
     Traditional and simpler types of language models. They calculate the probability of a word based on the preceding *n* words.
     Examples:
     - *Unigrams:* Consider each word independently.
     - *Bigrams:* Consider the previous word.
     - *Trigrams:* Consider the previous two words.
       Advantages: Easy to implement, efficient for small datasets.
       Limitations: Fail to capture long-term dependencies (cannot model meaning beyond a few words).
   - **Probabilistic & Bayesian Language Models:**
     These models use probability theory to generate sequences.
     - **Markov Models:** Predict sequences based on visible state transitions (e.g., word-to-word). Simpler than HMMs, used in early text generation.
     - **Hidden Markov Model (HMM):** Assumes words depend only on the previous state (Markov assumption). Used in speech

recognition and part-of-speech tagging. Simple and interpretable, but not effective for long-range dependencies.

2. **Neural Language Models:**
   These models utilize neural networks to learn representations of words and their relationships, capturing complex patterns and semantic information. Examples include Recurrent Neural Networks (RNNs), with subtypes like LSTM (Long Short-Term Memory) for better memory, and Transformer networks.
3. **Large Language Models (LLMs):**
   These are a subset of neural language models, separated due to their massive scale and usage. Characterized by a large number of parameters and enormous datasets, they're based on the Transformer architecture. They excel in NLP tasks like text generation, translation, and question-answering. Examples include GPT models and Google's PaLM models.
    - **Applications:** HMMs powered traditional POS tagging; LLMs now handle it with deeper context.

Applications of language models-

Language models are used in a wide range of NLP applications, including:

a. Machine translation
b. Speech recognition
c. Text generation
d. Question answering
e. Text summarization

Spell correction-

Language models help in **spell correction** by predicting the most **likely correct word** based on context and probability.

- A user types a word with a spelling mistake (e.g., "teh" instead of "the")
- The system needs to **correct it** based on **context and word probability**.

How Does a Language Model Help?

1Uses a probability-based model to determine the most likely correct word.
2 Considers the surrounding words (context) to predict the best correction.
3Ranks possible corrections based on how often they appear in real-world text.

Let's say a user types:

"I am goign to the market"

Possible corrections for "goign" are:

- "going"
- "gogin"
- "gonig"

A **bigram language model** estimates the probability of each word based on context:

- **P(going | I am) = 0.9**
- **P(gogin | I am) = 0.02**
- **P(gonig | I am) = 0.08**

The model **chooses "going"** because it has the **highest probability** in natural text.

N-Gram Models

A **language model (LM)** is a statistical model that predicts the **next word** in a sequence given the previous words. It is essential in applications like **speech recognition, text generation, machine translation, and spell correction**.

An **N-Gram model** is a type of **probabilistic language model** that **predicts the next word** based on the **previous (N-1) words** in a sequence.

- **Unigram (1-Gram)** → Predicts a word independently (no context).
- **Bigram (2-Gram)** → Predicts a word based on **1 previous word**.
- **Trigram (3-Gram)** → Predicts a word based on **2 previous words**.
- **N-Gram (N ≥ 4)** → Predicts a word based on **N-1 previous words**.

Unigram: *"The"*, *"dog"*, *"runs"* → No dependency
Bigram: *P(dog | The)*
Trigram: *P(runs | The dog)*

Unigram probability

Corpus: I am happy because I am learning

Size of corpus m = 7

$$P(I) = \frac{2}{7} \qquad P(happy) = \frac{1}{7}$$

Probability of unigram: $$P(w) = \frac{C(w)}{m}$$

Bigram probability

Corpus: I am happy because I am learning

$$P(am|I) = \frac{C(I\ am)}{C(I)} = \frac{2}{2} = 1 \qquad P(happy|I) = \frac{C(I\ happy)}{C(I)} = \frac{0}{2} = 0$$

✖ I happy

$$P(learning|am) = \frac{C(am\ learning)}{C(am)} = \frac{1}{2}$$

Probability of a bigram: $$P(y|x) = \frac{C(x\ y)}{\sum_w C(x\ w)} = \frac{C(x\ y)}{C(x)}$$

Corpus: “In every place of great resort the monster was the fashion. They sang of it in the cafes, ridiculed it in the papers, and represented it on the stage ” (Jules Verne, Twenty Thousand Leagues under the Sea)

In the context of our corpus, what is the probability of word “papers” following the phrase “it in the”?

- ○ P(papers|it in the) = 1/2
- ○ P(papers|it in the) = 2/3
- ○ P(papers|it in the) = 0

Probability of a sequence

- Given a sentence, what is its probability?

$$P(the\ teacher\ drinks\ tea) = ?$$

- Conditional probability and chain rule reminder

$$P(B|A) = \frac{P(A,B)}{P(A)} \implies P(A,B) = P(A)P(B|A)$$

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

Probability of a sequence

$$P(the\ teacher\ drinks\ tea) =$$

$$P(the)P(teacher|the)P(drinks|the\ teacher)$$
$$P(tea|the\ teacher\ drinks)$$

Sentence not in corpus

- Problem: Corpus almost never contains the exact sentence we're interested in or even its longer subsequences!

Input: the teacher drinks tea

$$P(the\ teacher\ drinks\ tea) = P(the)P(teacher|the)P(drinks|the\ teacher)P(tea|the\ teacher\ drinks)$$

$$P(tea|the\ teacher\ drinks) = \frac{C(the\ teacher\ drinks\ tea)}{C(the\ teacher\ drinks)}$$

← Both
← likely 0

Approximation of sequence probability

the teacher drinks tea

$$P(tea|the\ teacher\ drinks) \approx P(tea|drinks)$$

$P(teacher|the)$
$P(drinks|teacher)$
$P(tea|drinks)$

$$P(the\ teacher\ drinks\ tea) =$$

$$P(the)P(teacher|the)P(drinks|the\ teacher)P(tea|the\ teacher\ drinks)$$

$$\Downarrow$$

$$P(the)P(teacher|the)P(drinks|teacher)P(tea|drinks)$$

Question

Given these conditional probabilities:

- P(Mary)=0.1
- P(likes)=0.2
- P(cats)=0.3
- P(Mary|likes) =0.2
- P(likes|Mary) =0.3
- P(cats|likes)=0.1
- P(likes|cats)=0.4

Approximate the probability of the following sentence with bigrams: “Mary likes cats”

Start of sentence token <s>

the teacher drinks tea

$$P(the\ teacher\ drinks\ tea) \approx P(the)P(teacher|the)P(drinks|teacher)P(tea|drinks)$$

<s> the teacher drinks tea

$$P(<s>\ the\ teacher\ drinks\ tea) \approx P(the|<s>)P(teacher|the)P(drinks|teacher)P(tea|drinks)$$

- N-gram model: add N-1 start tokens <s>

Start of sentence token <s> for N-grams

- Trigram:

$$P(the\ teacher\ drinks\ tea) \approx P(the)P(teacher|the)P(drinks|the\ teacher)P(tea|teacher\ drinks)$$

the teacher drinks tea => <s> <s> the teacher drinks tea

$$P(w_1^n) \approx P(w_1|<s>\ <s>)P(w_2|<s>\ w_1)...P(w_n|w_{n-2}\ w_{n-1})$$

- N-gram model: add N-1 start tokens <s>

End of sentence token </s> - motivation

$$P(y|x) = \frac{C(x\ y)}{\sum_w C(x\ w)} = \frac{C(x\ y)}{C(x)}$$

Corpus:
<s> Lyn drinks chocolate
<s> John drinks

$$\sum_w C(drinks\ w) = 1$$

$$C(drinks) = 2$$

End of sentence token </s> - solution

- Bigram

  <s> the teacher drinks tea => <s> the teacher drinks tea </s>

$$P(the|<s>)P(teacher|the)P(drinks|teacher)P(tea|drinks)P(</s>|tea)$$

Corpus:
<s> Lyn drinks chocolate </s>
<s> John drinks </s>

$$\sum_w C(drinks\ w) = 2$$

$$C(drinks) = 2$$

- N-gram => just one </s>

Example - bigram

Corpus

<s> Lyn drinks chocolate </s>

<s> John drinks tea </s>

<s> Lyn eats chocolate </s>

$$P(sentence) = \frac{2}{3} * \frac{1}{2} * \frac{1}{2} * \frac{2}{2} = \frac{1}{6}$$

$$P(John|<s>) = \frac{1}{3}$$

$$P(</s>|tea) = \frac{1}{1}$$

$$P(chocolate|eats) = \frac{1}{1}$$

$$P(Lyn|<s>) =? = \frac{2}{3}$$

Question

<s> John drinks tea </s>

<s> She prefers tea with sugar </s>

Compute the Bigram Probability of sentence <s> John drinks tea </s>

Why use Log Prob?

We use **log probabilities** in **n-gram calculations** primarily to **prevent numerical underflow** and **simplify probability computations**.

1. **Avoiding Numerical Underflow:**
    - **N-gram models** compute the probability of a sentence by multiplying the probabilities of individual words.
    - Since all probabilities lie between **0 and 1**, multiplying many small probabilities results in **extremely tiny numbers**, which can **cause numerical underflow** (i.e., values becoming too small for the computer to represent accurately).
    - **Using logarithms** transforms the product into a **sum**, which avoids this issue.
2. **Mathematical Simplification:**
    - A **logarithmic identity** states:
      log(a×b×c)=loga+logb+logc= log a + log b + log c

- Instead of **multiplying** many small numbers, we can **add their log values**, making computations simpler and more stable.

3. **Efficient Computation in Machine Learning & NLP:**
   - Log probabilities allow for more **efficient storage and processing** in **large corpora**.
   - Many **machine learning models** work better with **log values** rather than raw probabilities.

Example in N-Gram Probability Calculation

Suppose we have a **trigram** probability:

$$P(w_1, w_2, w_3) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1, w_2)$$

If each probability is very small, e.g., 0.001, then:

$$P(w_1, w_2, w_3) = 0.001 \times 0.001 \times 0.001 = 10^{-9}$$

This value is **too small** and could lead to computational errors.

Instead, we use log probabilities:

$$\log P(w_1, w_2, w_3) = \log P(w_1) + \log P(w_2|w_1) + \log P(w_3|w_1, w_2)$$

If $\log 0.001 = -3$, then:

$$\log P(w_1, w_2, w_3) = -3 + (-3) + (-3) = -9$$

which is **much easier to handle** in computations.

The **log probability** in an **n-gram model** represents the likelihood of a sequence of words occurring in a corpus, but expressed in the **logarithmic domain** instead of the standard probability domain.

What Does Log Probability Represent?

If we calculate the probability of a sentence $S = w_1, w_2, ..., w_n$ using an **n-gram model**, we compute:

$$P(S) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1, w_2) \times \cdots \times P(w_n|w_{n-2}, w_{n-1})$$

Instead of working with this **very small probability**, we take the logarithm:

$$\log P(S) = \log P(w_1) + \log P(w_2|w_1) + \log P(w_3|w_1, w_2) + \cdots + \log P(w_n|w_{n-2}, w_{n-1})$$

Now, this **log probability** tells us **how likely the sequence of words is**, but on a logarithmic scale. Since logarithms of values between 0 and 1 are always negative, the **log probability is always ≤ 0**.

Interpreting Log Probability:

- **Higher (closer to 0) log probability → more likely sequence.**
- **Lower (more negative) log probability → less likely sequence.**

Example Interpretation

If we have two sentences:

1. **Sentence A:** "The dog chased the cat."
    - Log probability: **-5.2**
2. **Sentence B:** "Cat the chased dog the."
    - Log probability: **-12.8**

The log probability of Sentence A is **higher** (less negative), meaning it is **more likely** based on the n-gram model.

Perplexity?

Perplexity is a measure of how well a language model predicts a sequence of words. Think of it as the model's "confusion level"—lower perplexity means the model is less confused, better at guessing the next word. It's widely used to evaluate N-gram models (and others) by quantifying their predictive power on test data.

- **Intuition:** Imagine you're guessing the next word in "The cat ___." If your model strongly predicts "runs" (high probability), it's less perplexed. If it's unsure (low probability spread across many words), perplexity shoots up.
- **Goal:** Lower perplexity = better model. It's like a score—aim for the lowest you can get!

Why Perplexity in N-gram Models?

N-gram models predict the probability of a word based on the previous *N-1* words (e.g., bigrams use 1 prior word, trigrams use 2). Perplexity tests how well these probabilities hold up on unseen text:

- **Unigram:** Guesses each word independently—high perplexity (lots of uncertainty).
- **Bigram:** Uses one prior word—better, but still limited.
- **Trigram:** Uses two prior words—lower perplexity if trained well.

Formula for Perplexity

Perplexity is derived from the model's probability of a test sequence. For an N-gram model, it's th average "surprise" (inverse probability) per word, raised to a power. Here's the formal definition:

For a sequence of words $W = w_1, w_2, ..., w_M$ (length M), perplexity (PP) is:

$$PP(W) = P(w_1, w_2, ..., w_M)^{-1/M}$$

Where:

- $P(w_1, w_2, ..., w_M)$ is the joint probability of the sequence under the N-gram model.
- $-1/M$ is the exponent, normalizing by the number of words (like an average).

In N-gram Terms

N-grams approximate the joint probability using conditional probabilities:

- **Unigram:** $P(w_1, w_2, ..., w_M) = P(w_1) \cdot P(w_2) \cdot ... \cdot P(w_M)$
- **Bigram:** $P(w_1, w_2, ..., w_M) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot ... \cdot P(w_M|w_{M-1})$
- **Trigram:** $P(w_1, w_2, ..., w_M) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot ... \cdot P(w_M|w_{M-2}, w_{M-1})$

So, perplexity becomes:

$$PP(W) = \left(\prod_{i=1}^{M} P(w_i|w_{i-N+1}, ..., w_{i-1}) \right)^{-1/M}$$

Log Form (Practical Version)

Since probabilities are tiny and products get messy, we use logarithms to avoid underflow:

$$PP(W) = 2^{-\frac{1}{M} \sum_{i=1}^{M} \log_2 P(w_i|w_{i-N+1}, ..., w_{i-1})}$$

- $\log_2$ is common (base 2), but any base works (e.g., e or 10)—it's just a scaling factor.
- Negative log probabilities sum up the "surprise"—higher surprise = higher perplexity.

How to Calculate It: Example

Let's compute perplexity for a bigram model ($N = 2$) on a tiny test sentence: "The cat runs."

Step 1: Model Probabilities

Assume a simple bigram model trained on some corpus:

- $P(\text{The}) = 0.1$ (initial probability—could be unigram start).
- $P(\text{cat}|\text{The}) = 0.8$ ("cat" often follows "The").
- $P(\text{runs}|\text{cat}) = 0.6$ ("runs" follows "cat" decently).

Step 2: Joint Probability

$$P(\text{The cat runs}) = P(\text{The}) \cdot P(\text{cat}|\text{The}) \cdot P(\text{runs}|\text{cat})$$

$$= 0.1 \cdot 0.8 \cdot 0.6 = 0.048$$

Step 3: Perplexity

- $M = 3$ (three words).
- $PP = P(\text{The cat runs})^{-1/M} = 0.048^{-1/3}$
- Compute:
  - $0.048^{-1} = 1/0.048 \approx 20.833$
  - $20.833^{1/3} \approx 2.752$ (cube root).
- **Result:** Perplexity ≈ 2.75.

Log Version (Easier)

- $\log_2(0.1) \approx -3.32$, $\log_2(0.8) \approx -0.32$, $\log_2(0.6) \approx -0.74$.
- Sum: $-3.32 + (-0.32) + (-0.74) = -4.38$.
- $-\frac{1}{M} \cdot -4.38 = \frac{4.38}{3} \approx 1.46$.
- $PP = 2^{1.46} \approx 2.75$.
- **Matches!** Same answer, cleaner math.

What Does Perplexity Mean?

- **Perplexity = 2.75:** The model's "effective vocabulary" is ~2.75 words—it's like it's choosing between ~3 options per word on average. Lower is better—means higher confidence.

- **High Perplexity (e.g., 100):** Model's guessing from 100 possibilities—terrible predictions.
- **Perfect Model:** Perplexity = 1 (probability = 1 for every word—impossible in practice).

Why Use Perplexity?

- **Evaluation:** Compare models—bigram (e.g., 2.75) vs. trigram (maybe 2.5)—lower wins.
- **N-gram Limits:** High perplexity on long sentences shows N-grams miss distant context (e.g., "The cat... [20 words]... runs").

Need for Smoothing

What's the Problem Without Smoothing?

N-gram models predict the probability of a word based on the previous $N-1$ words by counting occurrences in a training corpus. For example, in a bigram model ($N=2$):

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

- **Issue:** If a bigram (w_{i-1}, w_i) never appears in the training data, $\text{Count}(w_{i-1}, w_i) = 0$, so $P = 0$.

Why This Breaks Things

1. **Zero Probability Ruins Joint Probability:**
   - For "The cat runs," if $P(\text{runs}|\text{cat}) = 0$ (unseen in training), then:
   $$P(\text{The cat runs}) = P(\text{The}) \cdot P(\text{cat}|\text{The}) \cdot 0 = 0$$
   - The whole sequence gets $P = 0$—even if it's valid in real life.
2. **Perplexity Explodes:**
   - Perplexity: $PP = P^{-1/M}$.
   - If $P = 0$, $\frac{1}{0} = \infty$, $PP = \infty$—model's useless, can't evaluate it (your perplexity question!).
3. **Real Language is Sparse:**
   - Training data (e.g., 1 million words) can't cover all possible N-grams—English has tons of valid combinations.
   - E.g., "cat dances" might not appear, but it's plausible—$P = 0$ is too harsh.

The Zero Problem Without Smoothing

Take “I like runs” from a bigram model (no smoothing):

- Corpus: “I like to, I like cats.”
- $P(\text{I}) = \frac{2}{7} \approx 0.286$, $P(\text{like}|\text{I}) = \frac{2}{2} = 1.0$, $P(\text{runs}|\text{like}) = \frac{0}{2} = 0$.

No Logs, Straight Calc

- $P = 0.286 \cdot 1.0 \cdot 0 = 0$.
- $PP = 0^{-1/3} = \infty$.

With Logs

- $\log_2(0.286) \approx -1.807$, $\log_2(1.0) = 0$, $\log_2(0) = ?$.
- **Problem:** $\log(0)$ is undefined (approaches $-\infty$)—math breaks down.
- Sum: $-1.807 + 0 + (-\infty) = -\infty$.
- $-\frac{-\infty}{3} = \infty$, $2^{\infty} = \infty$.
- **Result:** Perplexity is still infinite—logs don’t fix the zero.

Why Smoothing is Needed

Smoothing adjusts these probabilities to avoid zeros by giving a tiny chance to unseen N-grams. It’s like saying, “I haven’t seen this, but it’s not impossible.” This:

- Keeps $P > 0$ for all sequences.
- Makes the model generalize to unseen data.
- Keeps perplexity finite and meaningful.

Smoothing Techniques-

Smoothing techniques address the issue of zero probabilities in MLE-based n-gram models by redistributing probability mass across observed and unseen word sequences.

(a) Laplace Smoothing (Add-One Smoothing)

Laplace Smoothing avoids zero probabilities by adding 1 to all observed counts:

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

where:

- V is the vocabulary size.

Pros:

- Prevents zero probabilities.

Cons:

- Overestimates probabilities of unseen events.

(b) Add-K Smoothing (Generalized Laplace Smoothing)

Instead of adding 1, a small constant k (0 < k < 1) is added:

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

Pros:

- More flexible than Laplace smoothing.

Advantages & Limitations of N-Gram Models

✅ Advantages:

✔ **Simple & Efficient:** Easy to train and use for text generation.
✔ **Works Well for Small Datasets:** Performs decently for moderate text corpora.

❌ Limitations:

❌ **Data Sparsity:** Large N-Grams require huge amounts of training data. As N increases, the number of possible N-grams grows exponentially, leading to sparse data and increased computational demands.
❌ **Lack of Long-Range Context:** Cannot capture dependencies beyond N-words.
❌ **High Computational Cost:** Higher N-Gram models require more memory.

Applications of N-Gram Models in NLP

- **Text Prediction (Smartphones, Keyboards - T9, SwiftKey)**
- **Speech Recognition (Google Speech, Siri, Alexa)**
- **Machine Translation (Statistical MT before Deep Learning)**
- **Spell Checking & Auto-Correction (Grammarly, MS Word)**
- **Plagiarism Detection & Text Summarization**

Code

https://colab.research.google.com/drive/1g5hVdk8hd6WF1LA-suTN1nOC7KWCaFPE