Probabilistic Language Modeling in Natural Language Processing

1. The Basics of Language Modelling

1.1. What is a Language Model?

A **Language Model (LM)** is a statistical or probabilistic model that computes a probability distribution over a sequence of words. Given a sequence of words of length m, a language model assigns a probability $P(w_1, w_2, \ldots, w_m)$ to the entire sequence. In essence, it captures the "likelihood" of a sequence of words occurring in a given language. A model that has learned the patterns of a language can predict the next word in a sequence or score how "natural" a given sentence sounds.

For example, a well-trained language model would assign a much higher probability to the sentence "the cat sat on the mat" than to "mat the on sat cat the". This ability to quantify the plausibility of text is fundamental to many NLP tasks.

1.2. Why is Language Modeling Required?

Language models are the backbone of modern NLP, enabling machines to understand and generate human language. Their core function is to model the likelihood of word sequences, which is crucial for a wide range of applications:

- **Machine Translation:** An LM helps select the most fluent and grammatically correct translation from multiple candidates. For example, it would prefer "the big red house" over "the house red big" in an English translation.
- **Speech Recognition:** LMs resolve ambiguity between phonetically similar phrases. For instance, an LM would assign a higher probability to "recognize speech" than "wreck a nice beach," guiding the system to the correct transcription.
- **Text Generation:** In dialogue systems, summarization, and story writing, an LM predicts the most probable next word at each step to generate coherent and contextually relevant text.
- **Part-of-Speech (POS) Tagging:** An LM can help determine if a word like "book" is a noun ("I read the book") or a verb ("We need to book a flight"). A bigram POS tagger, for instance, knows that a determiner ("the") is more likely to be followed by a noun than a verb.
- **Word Sense Disambiguation:** The context provided by a language model helps identify the correct meaning of a word. For example, it can distinguish between "a river bank" (geographical feature) and "a savings bank" (financial institution) based

on surrounding words.

1.3. Prerequisites

To understand probabilistic language models, a foundational knowledge of the following concepts is essential:

- **Basic Probability Theory:** Concepts like conditional probability, the chain rule of probability, and Bayes' theorem are fundamental.
- **Corpus Linguistics:** Understanding what a text corpus is and how it's used to train models is necessary. A corpus is simply a large, structured collection of texts.

The core idea of probabilistic LMs is to learn the statistical patterns from a large text corpus and use those patterns to make predictions.

2. n-gram Language Modelling

2.1. What is an n-gram Language Model?

An **n-gram** is a contiguous sequence of n items (in this case, words) from a given sample of text. An n-gram language model predicts the probability of a given word based on the $n-1$ preceding words. It is based on a simplifying assumption known as the **Markov assumption**, which states that the probability of the next word depends only on a fixed, finite number of previous words (specifically, $n-1$).

The probability of a word w_i given its history $w_1, \ldots, w_{i-1}$ is given by the chain rule:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1})$$

However, calculating the probability based on the entire history is computationally expensive and suffers from data sparsity (most long sequences will never appear in the training data). The Markov assumption simplifies this to:

$$P(w_i | w_1, \ldots, w_{i-1}) \approx P(w_i | w_{i-n+1}, \ldots, w_{i-1})$$

2.2. Unigram Language Modelling (n=1)

Basics and Mathematical Details

The **unigram model** is the a-z simplest n-gram model. It makes the strongest, most simplifying assumption: the probability of a word is completely independent of any other word in the text. This is often called a "bag-of-words" model. The position of the word does not matter, and neither does the context.

The probability of a word w_i is calculated as its relative frequency in the training corpus:

$$P(w_i) = \frac{\text{count}(w_i)}{\sum_{j=1}^{V} \text{count}(w_j)} = \frac{\text{count}(w_i)}{N}$$

Where:

- $\text{count}(w_i)$ is the number of times word w_i appears in the corpus.
- V is the size of the vocabulary (total number of unique words).
- N is the total number of words (tokens) in the corpus.

The probability of an entire sentence $W = (w_1, w_2, \ldots, w_k)$ is the product of the individual unigram probabilities:

$$P(W) = \prod_{i=1}^{k} P(w_i)$$

Example:

Consider the corpus: `"the cat sat on the mat. the dog sat."`

Total words (N) = 9.

Vocabulary: {"the", "cat", "sat", "on", "mat", "dog"}

Counts: count("the")=3, count("cat")=1, count("sat")=2, count("on")=1, count("mat")=1, count("dog")=1.

Unigram probabilities:

$P(\text{"the"}) = 3/9$

$P(\text{"cat"}) = 1/9$

$P(\text{"sat"}) = 2/9$

$P(\text{"dog"}) = 1/9$

Probability of the sentence "the dog sat":

$P(\text{"the dog sat"}) = P(\text{"the"}) \times P(\text{"dog"}) \times P(\text{"sat"}) = (3/9) \times (1/9) \times (2/9) = 6/729 \approx 0.0082$

2.3. Bigram Language Modelling (n=2)

Basics and Mathematical Details

A **bigram model** takes one step up in complexity by considering the previous word as context. It approximates the probability of a word w_i based only on the identity of the preceding word w_{i-1}.

The probability of a word w_i given the previous word w_{i-1} is calculated using conditional probability, estimated via counts from the corpus:

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

The probability of a sentence $W = (w_1, w_2, \ldots, w_k)$ is the product of its bigram probabilities. We often prepend a special start-of-sentence token (e.g., `<s>`) to model the first word.

$$P(W) = P(w_1|\langle s\rangle) \times \prod_{i=2}^{k} P(w_i|w_{i-1})$$

Example:

Corpus: `<s> the cat sat </s> <s> the dog sat </s>`

Unigram Counts: count(<s>)=2, count("the")=2, count("cat")=1, count("dog")=1.

Bigram Counts: count(<s>, the)=2, count(the, cat)=1, count(cat, sat)=1, count(the, dog)=1, count(dog, sat)=1.

Let's calculate the probability of "the cat sat":

$$P(\text{"the"} | \langle s \rangle) = \frac{\text{count}(\langle s \rangle,\text{the})}{\text{count}(\langle s \rangle)} = 2/2 = 1.0$$

$$P(\text{"cat"} | \text{"the"}) = \frac{\text{count(the, cat)}}{\text{count(the)}} = 1/2 = 0.5$$

$$P(\text{"sat"} | \text{"cat"}) = \frac{\text{count(cat, sat)}}{\text{count(cat)}} = 1/1 = 1.0$$

Total sentence probability: $P(\text{"the cat sat"}) = 1.0 \times 0.5 \times 1.0 = 0.5$

Strengths

Context-Aware: Captures local word dependencies and word order, providing a better model of language fluency than unigrams.

Drawbacks

Data Sparsity: Many valid bigrams will not appear in the training data, leading to zero probabilities for unseen sequences.

Strengths

Improved Performance: Significantly better than unigrams for tasks requiring some sense of grammar.

Drawbacks

Limited Context: Only considers the immediately preceding word, failing to capture long-range dependencies.

The Problem of Data Sparsity: An Example

Data sparsity is the most significant weakness of bigram (and higher-order) models. It occurs when a valid sequence of words does not appear in the training corpus. The model, therefore, assigns it a probability of zero, which is problematic.

Example: Using the same corpus as above, let's calculate the probability of the new, perfectly grammatical sentence: "the cat sat on the mat".

We need to calculate: $P(\text{”the”}|\langle s\rangle) \times P(\text{”cat”}|\text{”the”}) \times P(\text{”sat”}|\text{”cat”}) \times P(\text{”on”}|\text{”sat”}) \times \ldots$

Let's find the probability of the bigram "sat on":

$$P(\text{”on”}|\text{”sat”}) = \frac{\text{count}(\text{sat, on})}{\text{count}(\text{sat})}$$

Our training corpus is `<s> the cat sat </s> <s> the dog sat </s>`. The bigram "sat on" **never occurs**. Therefore, $\text{count}(\text{sat}, \text{on}) = 0$.

This means $P(\text{"on"}|\text{"sat"}) = 0/2 = 0$.

Because the probability of the entire sentence is a product of its bigram probabilities, the presence of a single zero-probability bigram makes the entire sentence probability zero. This is clearly wrong, as "the cat sat on" is a plausible start to a sentence. This problem necessitates smoothing.

2.4. Trigram and Higher-Order n-gram Models

The General n-gram Model

We can generalize beyond bigrams to create models that consider even more context. A general **n-gram model** is based on the Markov assumption that the probability of a word depends only on the previous **n-1** words. This includes unigrams (n=1, context=0 words), bigrams (n=2, context=1 word), trigrams (n=3, context=2 words), 4-grams (n=4, context=3 words), and so on.

The mathematical formulation for the probability of a word w_i given its n-1 history is:

$$P(w_i | w_{i-n+1}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-n+1}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-n+1}, \ldots, w_{i-1})}$$

For example, in a **trigram model (n=3)**, we approximate the probability of a word based on the two words that precede it. The formula becomes:

$$P(w_i | w_{i-2}, w_{i-1}) = \frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})}$$

The Trade-off of Increasing 'n'

Choosing the value of 'n' involves a critical trade-off between context and sparsity.

- **Benefit of Larger 'n':** A larger 'n' (e.g., a 4-gram or 5-gram model) allows the model to capture more context and longer-range word dependencies. This can lead to more accurate and fluent predictions, as the model has a better understanding of the linguistic patterns. For example, a 4-gram model might learn the pattern "tickets are now on..." is often followed by "sale".
- **Cost of Larger 'n':** This benefit comes at a steep price. As 'n' increases, the **data sparsity problem** becomes exponentially worse. The number of possible 4-grams is vastly larger than the number of possible bigrams. This means that even in a massive corpus, the overwhelming majority of valid 4-grams will never have been seen, leading to the zero-probability issue. Furthermore, the memory and computational power required to store and process the counts for all seen n-grams grow rapidly, making higher-order models very expensive.

In practice, trigram models are often a sweet spot, though 4-grams and 5-grams are used in large-scale systems like machine translation, always in conjunction with sophisticated smoothing techniques.

Smoothing Techniques

Smoothing addresses the zero-probability issue by redistributing some of the probability mass from seen n-grams to unseen n-grams, ensuring no sequence has a probability of exactly zero.

Laplace (Add-One) Smoothing

This is the simplest smoothing method. It works by adding one to the count of every observed n-gram, including those that were not seen. The formula for a bigram model is adjusted as follows:

$$P_{Laplace}(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + V}$$

Where V is the size of the vocabulary (the total number of unique words).

Example (Add-One): Let's revisit the problem of calculating $P(\text{"on"}|\text{"sat"})$.

Corpus: `<s> the cat sat </s> <s> the dog sat </s>`

Vocabulary (V): {the, cat, sat, dog, on, mat} = 6 unique words (excluding sentence markers for simplicity in V count).

From before: $\text{count}(\text{sat}, \text{on}) = 0$ and $\text{count}(\text{sat}) = 2$.

Applying the Add-One formula:

$P_{Laplace}(\text{"on"}|\text{"sat"}) = \frac{\text{count(sat, on)}+1}{\text{count(sat)}+V} = \frac{0+1}{2+6} = \frac{1}{8} = 0.125$

The probability is no longer zero, solving the immediate problem.

Add-k Smoothing

Add-One smoothing can sometimes be too blunt and assign too much probability to unseen events. **Add-k smoothing** (also known as Lidstone smoothing) generalizes this by adding a small fractional value k (where $0 < k < 1$) instead of 1.

$$P_{Add-k}(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + k}{\text{count}(w_{i-1}) + kV}$$

Example (Add-k): Let's use the same scenario with $k = 0.1$.

$P_{Add-0.1}(\text{"on"}|\text{"sat"}) = \frac{\text{count(sat, on)}+0.1}{\text{count(sat)}+(0.1\times V)} = \frac{0+0.1}{2+(0.1\times 6)} = \frac{0.1}{2.6} \approx 0.038$

This provides a non-zero probability that is smaller and often more realistic than the result from Add-One smoothing.

4. Evaluating Language Models

After training a language model, we need a way to measure its quality. The goal is to determine how well the model generalizes to new, unseen text. A good model should assign a high probability to sentences that are grammatically correct and semantically plausible, and a low probability to nonsensical ones. The most common metric for this is **perplexity**.

Perplexity

Conceptual Understanding

Perplexity is a measurement of how well a probability model predicts a sample. In the context of language models, it can be thought of as a measure of the model's "surprise" when encountering a piece of text. A low perplexity indicates that the model is not very surprised by the test data, meaning it correctly predicted the sequence of words. A high perplexity indicates the model was very surprised, meaning the test data was unpredictable to it.

The intuition is that perplexity is the **weighted average branching factor** of a language. A perplexity of 100 means that whenever the model is trying to predict the next word, it is as confused as if it had to choose between 100 different words with equal probability. Therefore, a **lower perplexity is better**.

Mathematical Formulation

Perplexity is the inverse probability of the test set, normalized by the number of words. For a test sentence $W = (w_1, w_2, \ldots, w_N)$, the perplexity is calculated as:

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

Using the chain rule for n-grams, this can be expanded. For a bigram model, it is:

$$PP(W) = \left(\prod_{i=1}^{N} P(w_i|w_{i-1})\right)^{-\frac{1}{N}}$$

Because the probabilities are very small numbers, we often work with log probabilities to avoid numerical underflow. The perplexity formula using logs (it is the exponential of the cross-entropy) is:

$$PP(W) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_{i-1})\right)$$

Numerical Example

Let's calculate the perplexity of a test sentence using our bigram model from before, equipped with **Add-One smoothing** to handle unseen bigrams.

Model: Bigram model trained on `<s> the cat sat </s> <s> the dog sat </s>`.

Vocabulary Size (V): 6 (the, cat, sat, dog, on, mat).

Test Sentence (W): `<s> the cat on the mat </s>`

Number of words (N): 5 (we count the words in the sequence, including the end token if modeled, but here we will count the main words for simplicity in N). So, N=4 for "the cat on the mat".

We need to calculate the joint probability $P(W)$ first, using smoothed probabilities:

1. $P("\text{the}"|\langle s\rangle) = \frac{\text{count}(\langle s\rangle,\text{the})+1}{\text{count}(\langle s\rangle)+V} = \frac{2+1}{2+6} = \frac{3}{8}$
2. $P("\text{cat}"|"\text{the}") = \frac{\text{count}(\text{the, cat})+1}{\text{count}(\text{the})+V} = \frac{1+1}{2+6} = \frac{2}{8}$

3. $P(\text{"on"}|\text{"cat"}) = \frac{\text{count(cat, on)}+1}{\text{count(cat)}+V} = \frac{0+1}{1+6} = \frac{1}{7}$ (This is an unseen bigram)
4. $P(\text{"the"}|\text{"on"}) = \frac{\text{count(on, the)}+1}{\text{count(on)}+V} = \frac{0+1}{0+6} = \frac{1}{6}$ (Unigram "on" was not in training data, so its count is 0)

Joint Probability:

$$P(W) = \frac{3}{8} \times \frac{2}{8} \times \frac{1}{7} \times \frac{1}{6} = \frac{6}{2688} \approx 0.00223$$

Perplexity Calculation:

$$PP(W) = (0.00223)^{-\frac{1}{4}} = \sqrt[4]{\frac{1}{0.00223}} \approx \sqrt[4]{448} \approx 4.60$$

The perplexity of our model on this sentence is approximately **4.60**. This means the model is as confused as if it had to choose between roughly 4-5 words at each step. For a real model, this would be a very good (low) perplexity, but it's artificially low here due to the tiny vocabulary and test set.

5. Other Probabilistic Language Models

While n-gram models were foundational, modern NLP has largely shifted towards models that can handle long-range dependencies more effectively without suffering from the same sparsity issues. These models often learn a dense, continuous representation of words (embeddings) rather than relying on discrete counts.

- **Neural Network Language Models:** Feedforward neural networks were an early step, learning to predict the next word from a fixed-size window of previous word embeddings.
- **Recurrent Neural Network (RNN) Language Models:** RNNs, including LSTMs and GRUs, process sequences word-by-word, maintaining a hidden state that theoretically captures the entire history of the sequence, thus overcoming the fixed context window of n-gram models.
- **Transformer-based Language Models:** Models like GPT (Generative Pre-trained Transformer) and BERT use attention mechanisms to weigh the importance of all words in the context, regardless of their distance. These are the current state-of-the-art for most NLP tasks.

6. Supplementary Reads

- **Jurafsky, D., & Martin, J. H. (2023).** ***Speech and Language Processing*** **(3rd ed. draft).** The definitive textbook on NLP, with excellent chapters on n-gram language models.
- **Manning, C. D., & Schütze, H. (1999).** ***Foundations of Statistical Natural Language Processing*.** A classic text with a deep dive into the mathematical foundations of probabilistic models.
- **Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model.** ***Journal of Machine Learning Research*.** The seminal paper that introduced the idea of using neural networks for language modeling.