NLP preprocessing steps

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

Statistical Language Models:

- These models use statistical techniques to determine the probability of word sequences.
- A prominent example is the N-gram model, which calculates the probability of a word based on the preceding N-1 words.
- These models rely on counting word occurrences in large text corpora.

Neural Language Models:

- These models utilize neural networks to learn representations of words and their relationships.
- They address limitations of statistical models by capturing more complex patterns and semantic information.
- Examples include models based on Recurrent Neural Networks (RNNs) and Transformer networks.

Large Language Models (LLMs):

- These are a subset of neural language models, but are worth separating because of their current massive usage.
- LLMs are characterized by their very large number of parameters, and the massive data sets they are trained on.
- These models, based primarily on the Transformer architecture, have shown remarkable capabilities in various NLP tasks, including text generation, translation, and question answering.
- Examples include GPT models, and Google's Palm models.

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.

Applications

1. Next word prediction (Autocomplete)

2. Spell correction

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

Problem:

- 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.

Here's a breakdown of the key types:

1. Statistical Language Models:

- **N-gram Models:**
    - These are the 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 one word.
        - **Trigrams:** Consider the previous two words.
    - **Advantages:** Easy to implement, efficient for small datasets.
      Limitations: Fails for long-term dependencies (Cannot capture meaning beyond a few words).
- **Exponential Models:**
    - These models, like Maximum Entropy models, offer more flexibility by considering various features beyond just the preceding words.
    - They aim to maximize entropy, leading to more accurate probability estimations.

2. Neural Language Models:

- **Recurrent Neural Networks (RNNs):**
    - Designed to process sequential data, RNNs can capture contextual information by maintaining a hidden state.
    - Variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) address the vanishing gradient problem, improving the ability to learn long-range dependencies.
- **Transformer-based Models:**
    - These models have revolutionized NLP with their self-attention mechanism, which allows them to weigh the importance of different words in a sequence.
    - They excel at capturing complex relationships and have led to the development of powerful models like:
        - **BERT (Bidirectional Encoder Representations from Transformers):** Focuses on understanding the context of words in both directions.
        - **GPT (Generative Pre-trained Transformer):** Excels at generating coherent and contextually relevant text.
        - **T5 (Text-to-Text Transfer Transformer):** Frames all NLP tasks as text-to-text problems.
- **Large Language Models (LLMs):**
    - These are advanced neural language models with billions of parameters, trained on massive datasets.
    - They demonstrate remarkable capabilities in understanding and generating human-like text.

Key Concepts:

- **Applications:**
    - Language models are used in a wide range of NLP applications, including:
        - Machine translation

- Speech recognition
- Text generation
- Sentiment analysis
- Question answering
- Text summarization

Statistical Language Models (SLMs)

These models use **probability distributions** over sequences of words to predict the likelihood of the next word.

N-Gram Models

Neural Language Models (Deep Learning-Based)

Neural-based language models use **deep learning architectures** to learn **word representations and context** more effectively than statistical models.

Feedforward Neural Network Language Model (NNLM)

- Uses a simple multi-layer perceptron (MLP) to predict words.
- Example: Bengio et al. (2003) proposed one of the first NNLMs.
  ✔ Better than N-Grams as it learns distributed word representations.
  ❌ Computationally expensive for large vocabularies.

Recurrent Neural Network Language Models (RNN-LM)

- **Uses Recurrent Neural Networks (RNNs) to process sequences** and predict words based on previous context.
- **Example:**

  Simple RNN → Long Short-Term Memory (LSTM) → Gated Recurrent Unit (GRU)

  Applications: Speech Recognition, Text Generation (e.g., Chatbots).

✔ Captures sequential dependencies better than N-Grams.
❌ Still struggles with long-term dependencies due to vanishing gradients.

Transformer-Based Language Models (State-of-the-Art)

Transformers are currently the best language models, **outperforming RNNs** in almost all NLP tasks.

1. Attention-Based Models (Self-Attention Mechanism)

- **Example: Transformer (Vaswani et al., 2017)**
- Uses **self-attention** to weigh the importance of words in a sentence.

✔ Captures long-range dependencies in text.
✔ Highly parallelizable (Faster training than RNNs).

---

2. Pretrained Transformer Models

These models are trained on massive datasets and fine-tuned for specific tasks.

Model	Architecture	Purpose
BERT (2018)	Bidirectional Transformer	Contextual embeddings, NLP tasks
GPT-3 (2020)	Decoder-only Transformer	Text generation, Chatbots
T5 (2019)	Encoder-Decoder	Text summarization, Question answering
XLNet (2019)	Transformer with autoregressive modeling	Improved text generation

✔ State-of-the-art performance in NLP tasks.
✔ Used in AI chatbots, text generation (ChatGPT), and translation.
❌ High computational cost, requires GPUs for training.

Probabilistic & Bayesian Language Models

These models use **probability theory** to generate sequences.

📌 (A) Hidden Markov Model (HMM)

- Assumes words depend **only on the previous state** (Markov assumption).
- Used in **speech recognition, part-of-speech tagging**.

✔ **Simple & interpretable.**
❌ **Not effective for long-range dependencies.**

Language Modeling - Basics of 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

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

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

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

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

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

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>

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

$$P(John|<s>) = \frac{1}{3}$$
$$P(chocolate|eats) = \frac{1}{1}$$
$$P(</s>|tea) = \frac{1}{1}$$
$$P(Lyn|<s>) =? = \frac{2}{3}$$

Question
<s> John drinks tea </s>
<s> She prefers tea with sugar </s>

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

Log Prob

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

Why Use Log Probabilities in N-Gram Models?

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** (e.g., 10−5010^{-50}10−50), 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 \times b \times c) = \log a + \log b + \log clog(a×b×c)=loga+logb+logc
    - 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.

Comparison with Regular Probability

- **Probability:** P(S)=0.0001P(S) = 0.0001P(S)=0.0001 (very small number)
- **Log Probability:** logP(S)=−4.0\log P(S) = -4.0logP(S)=−4.0 (easier to manage)

Instead of saying *"This sentence has a probability of 0.0001,"* we say *"This sentence has a log probability of -4.0,"* which is easier for computations.

Evaluate a language model (Perplexitiy)

Probability Estimation in N-Gram Models

The probability of a sentence **W = (w_1, w_2, ..., w_n)** can be estimated using the **chain rule of probability**:

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots P(w_n|w_1, w_2, ..., w_{n-1})$$

Since this is computationally expensive, **N-Gram approximation** simplifies this:

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

For a **Bigram Model (N=2)**, the probability is:

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

For a **Trigram Model (N=3)**:

$$P(w_n|w_1, w_2, ..., w_{n-1}) \approx P(w_n|w_{n-2}, w_{n-1})$$

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

Maximum Likelihood Estimation (MLE), N-Gram Probability Estimation, and Smoothing Techniques in Language Modeling

1. Maximum Likelihood Estimation (MLE)

Definition

Maximum Likelihood Estimation (MLE) is a fundamental statistical method used to estimate the parameters of a probability distribution by maximizing the likelihood of observed data. In the context of language modeling, MLE is used to compute the probability of a word or sequence of words based on their occurrences in a given corpus.

Formula for MLE

Formula for MLE

For a unigram model (where each word's probability is estimated independently), the probability of a word w is given by:

$$P(w) = \frac{\text{Count}(w)}{N}$$

where:

- $\text{Count}(w)$ is the number of times word w appears in the corpus.
- N is the total number of words in the corpus.

For a bigram model, which considers the probability of a word given its previous word, the MLE formula is:

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

where:

- $\text{Count}(w_{i-1}, w_i)$ represents the number of times the bigram (w_{i-1}, w_i) appears in the corpus.
- $\text{Count}(w_{i-1})$ is the number of times the word w_{i-1} appears in the corpus.

Limitations of MLE

1. **Zero Probability Problem**: If a word sequence is missing in the training corpus, MLE assigns it a probability of zero, which is problematic in real-world language processing.
2. **Overfitting to Training Data**: MLE estimates probabilities directly from observed counts, which may not generalize well to unseen text.
3. **Poor Handling of Rare Words**: Less frequent words tend to have very low probabilities, which can negatively impact model performance.

2. N-Gram Probability Estimation

Definition

An n-gram model is a probability-based approach for predicting the next word in a sequence, considering the previous words. These models rely on MLE to estimate conditional probabilities but suffer from data sparsity issues, requiring additional techniques like smoothing.

N-Gram Probability Formula

For an n-gram model, the probability of a word given its previous $n-1$ words is:

$$P(w_n|w_1, w_2, ..., w_{n-1}) = \frac{\text{Count}(w_1, w_2, ..., w_n)}{\text{Count}(w_1, w_2, ..., w_{n-1})}$$

For example, in a trigram model:

$$P(w_3|w_1, w_2) = \frac{\text{Count}(w_1, w_2, w_3)}{\text{Count}(w_1, w_2)}$$

This approach enables **better context modeling** but faces challenges due to **data sparsity**.

3. Need for Smoothing

Why is Smoothing Necessary?

Smoothing is essential in language modeling to address the limitations of MLE and improve the generalizability of n-gram models. The main reasons for applying smoothing techniques are:

1. **Handling Zero Probabilities:** In MLE, unseen n-grams are assigned a probability of zero, making it impossible to generate or process new sentences that contain these n-grams.
2. **Data Sparsity:** Even large corpora cannot contain all possible n-grams, and many word sequences may appear rarely or not at all.
3. **Better Generalization:** Smoothing helps redistribute probability mass to unseen n-grams, making models more robust in real-world applications.
4. **Avoiding Overfitting:** Without smoothing, probabilities are overly reliant on the training corpus and may not perform well on new text.

4. Smoothing Techniques

Definition

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.