N-grams, a fundamental concept in NLP, play a pivotal role in capturing patterns and relationships within a sequence of words. In this blog post, we'll delve into the world of N-grams, exploring their significance, applications, and how they contribute to enhancing language processing tasks.

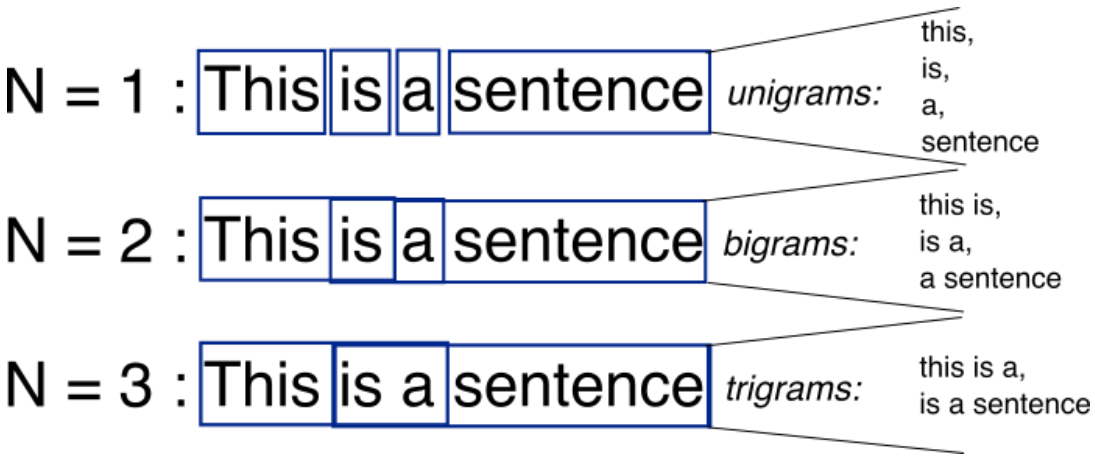
Understanding N-grams:

Definition:

N-grams are contiguous sequences of 'n' items, typically words in the context of NLP. These items can be characters, words, or even syllables, depending on the granularity desired. The value of 'n' determines the order of the N-gram.

Examples:

- **Unigrams (1-grams):** Single words, e.g., "cat," "dog."
- **Bigrams (2-grams):** Pairs of consecutive words, e.g., "natural language," "deep learning."
- **Trigrams (3-grams):** Triplets of consecutive words, e.g., "machine learning model," "data science approach."
- 4-grams, 5-grams, etc.: Sequences of four, five, or more consecutive words.



Significance of N-grams in NLP:

1. Capturing Context and Semantics:

- N-grams help capture the contextual information and semantics within a sequence of words, providing a more nuanced understanding of language.

2. Improving Language Models:

- In language modeling tasks, N-grams contribute to building more accurate and context-aware models, enhancing the performance of applications such as machine translation and speech recognition.

3. Enhancing Text Prediction:

- N-grams are essential for predictive text applications, aiding in the prediction of the next word or sequence of words based on the context provided by the preceding N-gram.

4. Information Retrieval:

- In information retrieval tasks, N-grams assist in matching and ranking documents based on the relevance of N-gram patterns.

5. Feature Extraction:

- N-grams serve as powerful features in text classification and sentiment analysis, capturing meaningful patterns that contribute to the characterization of different classes or sentiments.

Applications of N-grams in NLP:

1. Speech Recognition:

- N-grams play a crucial role in modeling and recognizing spoken language patterns, improving the accuracy of speech recognition systems.

2. Machine Translation:

- In machine translation, N-grams contribute to understanding and translating phrases within a broader context, enhancing the overall translation quality.

3. Predictive Text Input:

- Predictive text input on keyboards and mobile devices relies on N-grams to suggest the next word based on the context of the input sequence.

4. Named Entity Recognition (NER):

- N-grams aid in identifying and extracting named entities from text, such as names of people, organizations, and locations.

5. Search Engine Algorithms:

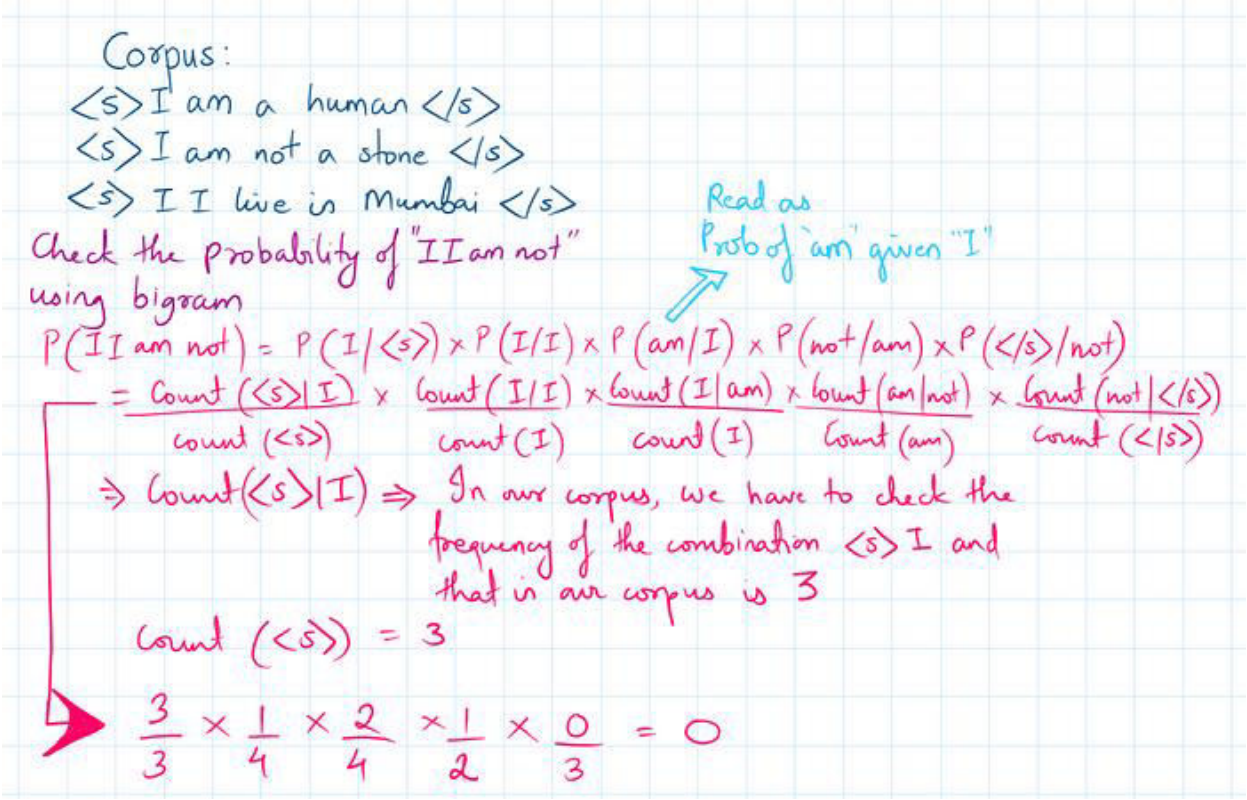
- Search engines use N-grams to index and retrieve relevant documents based on user queries, improving the accuracy of search results.

CODE

```
import nltk
nltk.download('punkt')

from nltk import ngrams
from nltk.tokenize import word_tokenize

# Example sentence
sentence = "N-grams enhance language processing tasks."

# Tokenize the sentence
tokens = word_tokenize(sentence)

# Generate bigrams
bigrams = list(ngrams(tokens, 2))

# Generate trigrams
trigrams = list(ngrams(tokens, 3))
```

```
# Print the results
print("Bigrams:", bigrams)
print("Trigrams:", trigrams)

'''
Output:
Bigrams: [('N-grams', 'enhance'), ('enhance', 'language'), ('language',
'processing'), ('processing', 'tasks'), ('tasks', '.')]
Trigrams: [('N-grams', 'enhance', 'language'), ('enhance', 'language',
'processing'), ('language', 'processing', 'tasks'), ('processing',
'tasks', '.')]
'''
```

Using N Gram to predict the probability of a sentence



Using N grams to predict the next word in the sentence

Consider the following training data

<s> I am Jack </s>
<s> Jack I am </s>
<s> Jack I like </s>
<s> Jack I do like </s>
<s> do I like Jack </s>

Assume that we use a bigram language model based on the above data

What is the most probable next word predicted by model

1) <s> Jack ____
2) <s> Jack I do ____
3) <s> Jack I am Jack ____
4) <s> do I like ____

$P(I|<s>) = \frac{Count(<s>|I)}{Count(<s>)} = \frac{1}{5}$

$P(am|I) = \frac{Count(I|am)}{Count(I)} = \frac{2}{5}$

$P(Jack|am) = \frac{Count(am|Jack)}{Count(am)} = \frac{1}{2}$

$P(</s>|Jack) = \frac{Count(Jack|</s>)}{Count(Jack)} = \frac{2}{5}$

$P(Jack|<s>) = \frac{Count(<s>|Jack)}{Count<s>} = \frac{2}{5}$

$P(I|Jack) = \frac{Count(Jack|I)}{Count(Jack)} = \frac{3}{5}$

$P(</s>|am) = \frac{Count(am|</s>)}{Count(am)} = \frac{1}{2}$

$P(like|I) = \frac{Count(I|like)}{Count(I)} = \frac{2}{5}$

$P(</s>|like) = \frac{Count(like|</s>)}{Count(like)} = \frac{2}{3}$

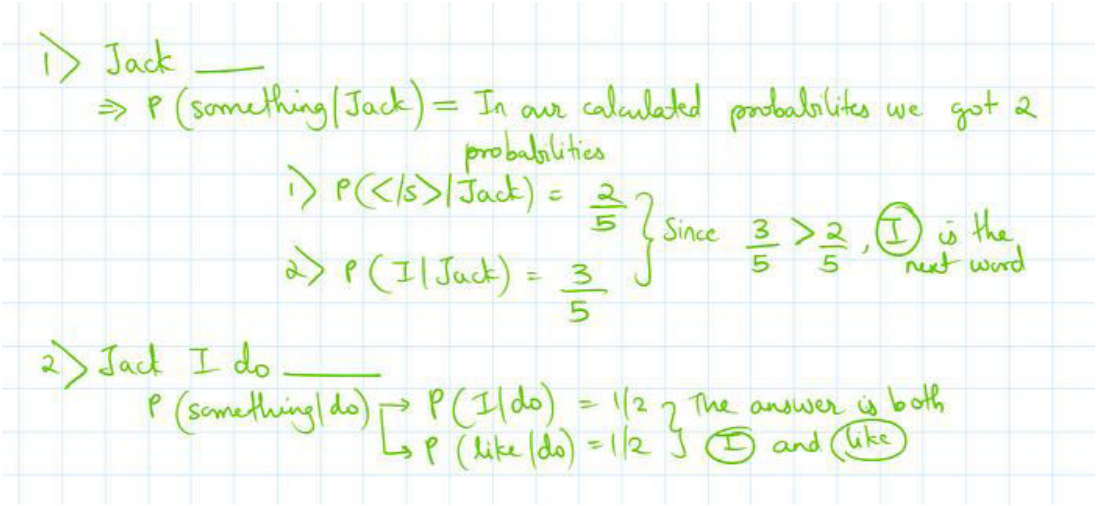
$P(do|I) = \frac{Count(I|do)}{Count(I)} = \frac{1}{5}$

$P(like|do) = \frac{Count(do|like)}{Count(do)} = \frac{1}{2}$

$P(do|<s>) = \frac{Count(<s>|do)}{Count(<s>} = \frac{1}{5}$

$P(I|do) = \frac{Count(do|I)}{Count(do)} = \frac{1}{2}$

$P(Jack|like) = \frac{Count(like|Jack)}{Count(like)} = \frac{1}{3}$



N-Grams

Let's begin with the task of computing P(w | h), the probability of a word w given some history h. Suppose the history h is "its water is so transparent that" and we want to know the probability that the next word is the:

P(the | its water is so transparent that): (3.1)

One way to estimate this probability is from relative frequency counts: take a very large corpus, count the number of times we see its water is so transparent that, and count the number of times this is followed by the. This would be answering the question "Out of the times we saw the history h, how many times was it followed by the word w", as follows:

P(the | its water is so transparent that) =
C(its water is so transparent that the) /
C(its water is so transparent that)

Evaluating Language Models

The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called extrinsic evaluation.

Extrinsic evaluation is the only way to know if a particular improvement in a component is really going to help the task at hand. Thus, for speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and seeing which gives the more accurate transcription.

Unfortunately, running big NLP systems end-to-end is often very expensive. Instead, it would be nice to have a metric that can be used to quickly evaluate potential improvements in a language model. An intrinsic evaluation metric is one that measures the quality of a model independent of any application. For an intrinsic evaluation of a language model we need a test set. As with many of the statistical models in our field, the probabilities of an n-gram model come from the corpus it is trained on, the training set or training corpus. We can then measure the quality of an n-gram model by its performance on some unseen data called the test set or test corpus.
So if we are given

For N-1 words, the N-gram modeling predicts most occurred words that can follow the sequences. The model is the probabilistic language model which is trained on the collection of the text. This model is useful in applications i.e. speech recognition, and machine translations. A simple model has some limitations that can be improved by smoothing, interpolations, and back off. So, the N-gram language model is about finding probability distributions over the sequences of the word. Consider the sentences i.e. "There was heavy rain" and "There was heavy flood". By using experience, it can be said that the first statement is good. The N-gram language model tells that the "heavy rain" occurs more frequently than the "heavy flood". So, the first statement is more likely to occur and it will be then selected by this model. In the one-gram model, the model usually relies on that which word occurs often without pondering the previous words. In 2-gram, only the previous word is considered for predicting the current word. In 3-gram, two previous words are considered. In the N-gram language model the following probabilities are calculated:
P (“There was heavy rain”) = P (“There”, “was”, “heavy”, “rain”) = P (“There”) P (“was” |“There”) P (“heavy”| “There was”) P (“rain” |“There was heavy”).

As it is not practical to calculate the conditional probability but by using the “Markov Assumptions”, this is approximated to the bi-gram model as [4]:
P (“There was heavy rain”) ~ P (“There”) P (“was” |“'There”) P (“heavy” |“was”) P (“rain” |“heavy”)

Applications of the N-gram Model in NLP

In speech recognition, the input can be noisy. This noise can make a wrong speech to the text conversion. The N-gram language model corrects the noise by using probability knowledge. Likewise, this model is used in machine translations for producing more natural statements in target and specified languages. For spelling error corrections, the dictionary is useless sometimes. For instance, "in about fifteen minutes" 'minuets' is a valid word according to the dictionary but it is incorrect in the phrase. The N-gram language model can rectify this type of error.
The N-gram language model is generally at the word levels. It is also used at the character levels for doing the stemming i.e. for separating the root words from a suffix. By looking at the N-gram model, the languages can be classified or differentiated between the US and UK spellings. Many applications get benefit from the N-gram model including tagging of part of the speech, natural language generations, word similarities, and sentiments extraction. [4].

Limitations of N-gram Model in NLP

The N-gram language model has also some limitations. There is a problem with the out of vocabulary words. These words are during the testing but not in the training. One solution is to use the fixed vocabulary and then convert out vocabulary words in the training to pseudowords. When implemented in the sentiment analysis, the bi-gram model outperformed the uni-gram model but the number of the features is then doubled. So, the scaling of the N-gram model to the larger data sets or moving to the higher-order needs better feature selection approaches. The N-gram model captures the long-distance context poorly. It has been shown after every 6-grams, the gain of performance is limited.

N-gram Language Modeling in Natural Language Processing

Introduction

Language modeling is used to determine the probability of the word's sequence. This modeling has a large number of applications i.e. recognition of speech, filtering of spam, etc. [1].

Natural Language Processing (NLP)

Natural language processing (NLP) is the convergence of artificial intelligence (AI) and linguistics. It is used to make the computers understand the words or statements that are written in human languages. NLP has been developed for making the work and communication with the computer easy and satisfying. As all the computer users cannot be well known by the specific languages of machines so NLP works better with the users who cannot have time for learning the new languages of machines. We can define language as a set of rules or symbols. Symbols are combined to convey the information. They are tyrannized by the set of rules. NLP is classified into two portions that are natural language understanding and natural language generation which evolves the tasks for understanding and generating the text. The classifications of NLP are shown in Figure 1 [2].

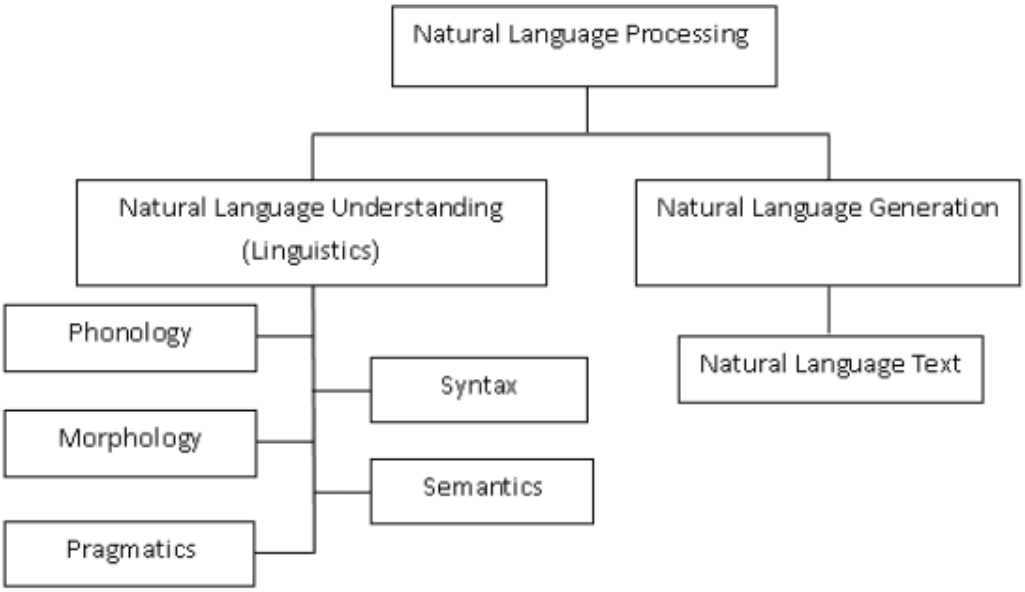


Figure 1 Classifications of NLP

Methods of the Language Modeling

Language modelings are classified as follows:

Statistical language modelings: In this modeling, there is the development of probabilistic models. This probabilistic model predicts the next word in a sequence. For example N-gram language modeling. This modeling can be used for disambiguating the input. They can be used for selecting a probable solution. This modeling depends on the theory of probability. Probability is to predict how likely something will occur. **Neural language modelings:** Neural language modeling gives better results than the classical methods both for the standalone models and when the models are incorporated into the larger models on the challenging tasks i.e. speech recognitions and machine translations. One method of performing neural language modeling is by word embedding [1].

UNIT II
Language Modelling

Contents..

- Introduction
- N-gram models
- Evaluating language models
- Sampling sentences
- Generalization and Zeros
- Smoothing
- Kneser-Ney Smoothing
- RNN

Introduction

- Need: Predicting upcoming word/s.
- Applications: Speech Recognition, Spelling correction, Machine Translation
- Language Model: Models that assign probabilities to sequences of words are called language models or LMs.

- Goal: To compute the probability of a sentence or sequence of words:

  $P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$

- Related task: probability of an upcoming word:

  $P(w_5 | w_1,w_2,w_3,w_4)$

- A model that computes either of these:

  $P(W)$ or $P(w_n | w_1,w_2 \ldots w_{n-1})$ is called a language model.

N-gram Language Models

- An n-gram is a sequence of n words:
  - 2-gram (which we'll call **bigram**) is a two-word sequence of words
    - Ex. "please turn", "turn your", or "your homework"
  - 3-gram (a trigram) is a three-word sequence of words
    - Ex. "please turn your", or "turn your homework"

N-gram Language Models contd…

How to compute this joint probability P(W) :

– P(its, water, is, so, transparent, that)

- The Chain Rule of Probability

p(B|A) = P(A,B)/P(A) Rewriting: **P(A,B) = P(A)P(B|A)**

- More variables:

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

- The Chain Rule in General

$P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$

N-gram Language Models contd…

The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

N-gram Language Models

- How to estimate these probabilities

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{\textit{Count}(\text{its water is so transparent that the})}{\textit{Count}(\text{its water is so transparent that})}$$

Bigram Model

Andrei Markov

- It is also called Markov Assumption
- Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past
- Simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \quad P(\text{the} \mid \text{that})$$

$$P(w_n \mid w_{1..n-1}) \approx P(w_n \mid w_{n-1})$$

Bigram Model contd…

- A bigram model to predict the conditional probability of the next word.
- Making the following approximation

$$P(w_n \mid w_{1..n-1}) \approx P(w_n \mid w_{n-1})$$

- the bigram assumption for the probability of an individual word
- The assumption that the probability of a word depends only on the previous word is called a Markov assumption.

Bigram Model contd…

- Given the bigram assumption for the probability of an individual word, we can compute the probability of a complete word sequence:

$$P(w_{1..n}) \approx \prod_{K=1}^{n} P(w_k \mid w_{k-1})$$

Maximum Likelihood Estimation (MLE)

- Use: To estimate a bigram or n-gram probabilities
- How: By getting **counts** from a corpus, and **normalizing** the counts so that they lie between 0 and 1.

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

An example

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\texttt{I} \mid \texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \tfrac{1}{3} = .33$$

Evaluating Language Models cont…

- Extrinsic evaluation
  - The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves.
  - Such end-to-end evaluation is called extrinsic evaluation.
  - Speech Recognition

- Intrinsic evaluation
  - An intrinsic evaluation metric is one that measures the quality of a model independent of any application.
  - Need test set and training set/training corpus.

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - Sometimes use **intrinsic** evaluation: **perplexity**
  - Bad approximation
    - unless the test data looks **just** like the training data
    - So **generally only useful in pilot experiments**
  - But is helpful to think about.

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Definition: Perplexity is the inverse probability of the test set, normalized by the number of words:

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

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

Evaluating Language Models cont…

Test set and Training set

- Training set
  - Training corpus
  - The probabilities of an n-gram model come **from the corpus** it is trained on.
  - We train parameters of our model on a training set.
- Test set
  - Test corpus
  - The quality of an n-gram model is measured by its performance on some **unseen data** called the test set or test corpus.

- An **evaluation metric** tells us how well our model does on the test set.

Evaluating Language Models cont…

- Development test set or, devset
  - Fresh test set that is truly unseen which is the initial test set is called as the development test set or, devset.

- Training on the test set
  - If test sentence is part of the training corpus, we will mistakenly assign it an artificially high probability when it occurs in the test set. This situation is called as **training on the test set.**
  - The probabilities all look too high, and causes huge inaccuracies in perplexity, the probability-based metric

Evaluating Language Models cont…

- Division of Data
- Training set - 80%
- Development set - 10%
- Test set – 10%

Evaluating Language Models cont…

- Steps for Evaluation of Language Model
  - Given data is Corpus and then Compare two n-gram models
  - Divide the data into training and test sets
  - Train the parameters of both models on the training set
  - Compare how well the two trained models fit the test set
  - Fit the test set: model assigns a higher probability to the test set

Sampling sentences from a language model

- Sampling from a distribution means to choose random points according to their likelihood.
- Thus sampling from a language model to generate some sentences, choosing each sentence according to its likelihood as defined by the model.
- More likely to generate sentences that the model thinks have a high probability and less likely to generate sentences that the model thinks have a low probability.
- This technique of visualizing a language model by sampling was first suggested very early on by Shannon (1951) and Miller and Selfridge (1950).
- It's simplest to visualize how this works for the unigram case.

Generalization and Zeros

- Generalization in the field of natural language processing (NLP) is the ability of models to efficiently make predictions on previously unseen data based on what it has learned from the training data.

- The concept of zeros in NLP refers to the presence of zero-valued words in the corpus which are words that do not appear in a given input text. The presence of zeros has a significant impact on the performance of models in NLP generalization.

Generalization and Zeros

- Generalization is typically most closely related to two main factors, either overfitting or underfitting conditions.
  - There can be models that have learned the training dataset too well and know nothing else, it performs on the training dataset but does not perform well on any other new inputs. This is the case of **overfitting**
  - There will also be the case of models that are designed such that they do not understand the underlying problem statement and acts poorly on a training dataset and do not perform on new inputs. This is the case of **underfitted** models.

- **Good Fit Model** is the one we need to target as the desired scenario as the model appropriately learns the training dataset and generalizes it to new inputs.

The Shannon Visualization Method

- Choose a random bigram (<s>, w) according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

Approximating Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Shakespeare as corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams.
  - So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams worse: What's coming out looks like Shakespeare because it ***is*** Shakespeare

The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!
  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
      - But occur in the test set

Zeros

- Training set:
  … denied the allegations
  … denied the reports
  … denied the claims
  … denied the request

  P(“offer” | denied the) = 0

- Test set
  … denied the offer
  … denied the loan

Zero probability bigrams

- Bigrams with zero probability
  - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!

- Smoothing
- Advanced: Kneser-Ney Smoothing .
- Transformers as Language Model

- Recurrent Neural Networks -RNNs as Language Models

Smoothing contd…

- **Need of Smoothing**
  - To improve the accuracy of our model.
  - To handle data sparsity, out of vocabulary words, words that are absent in the training set.

Source: https://www.exploredatabase.com/2021/02/what-is-smoothing-in-nlp-and-why-do-we-need-smoothing.html

Smoothing contd...

- **Smoothing techniques** are used to determine probability of a sequence of words occurring together when one or more words in the given set have never occurred in the past.

- Smoothing is the process of flattening a probability distribution implied by a language model so that all reasonable word sequences can occur with some probability.

Source: https://vitalflux.com/quick-introduction-smoothing-techniques-language-models/

Smoothing contd…

- In a language model, we use parameter estimation (MLE) on training data.
- We can't actually evaluate our MLE models on unseen test data because both are likely to contain words/n-grams that these models assign zero probability to.
- Need to reserve some probability mass to events that don't occur (unseen events) in the training data.
- Smoothing enhances accuracy in statistical models such as Naïve Bayes when applied to data with high sparsity, by removing the penalty on zero-probability n-grams.
- The problem is with the really zero probabilities or the ones that are really zero.

Source: https://www.exploredatabase.com/2021/02/what-is-smoothing-in-nlp-and-why-do-we-need-smoothing.html

Smoothing contd…

- The smoothing techniques:
  - Laplace smoothing: Another name for Laplace smoothing technique is **add one smoothing**.
  - Add-k smoothing
  - Stupid backoff
  - Kneser-Ney smoothing
  - Good-Turing smoothing
  - Katz smoothing
  - Church and Gale Smoothing

Source: https://vitalflux.com/quick-introduction-smoothing-techniques-language-models/

Smoothing contd...

Example 1:
Training data: *The cow is an animal*.
Test data: *The dog is an animal*.

If we use **unigram model** to train;
P(the) = count(the)/(Total number of words in training set) = 1/5.

Likewise, P(cow) = P(is) = P(an) = P(animal) = 1/5

To evaluate (test) the **unigram model**;
P(the cow is an animal) = P(the) * P(cow) * P(is) * P(an) * P(animal) = 0.00032

While we use unigram model on the test data, it becomes zero because P(dog) = 0.

The term 'dog' never occurred in the training data. Hence, we use smoothing.

Source: https://vitalflux.com/quick-introduction-smoothing-techniques-language-models/

Smoothing

Example 2:

Training data: <S> I like coding </S>

<S> Ayush likes Python</S>

<S> He likes coding</S>

Test data: <S> I like Python </S>

Let's consider bigrams, a group of two words.

$$P(w_i \mid w_{(i-1)}) = count(w_i\ w_{(i-1)}) / count(w_{(i-1)})$$

So, let's find the probability of "I like Python".

P("I like Python")

= P(I | <S>) * P(like | I) * P(Python | like) * P(</S> | Python)

= (count(<S>I) / count(<S>)) * (count(I like) / count(I)) * (count(like Python) / count(like)) * (count(Python </S>) / count(</S>))

= (1/3) * (1/1) * (0/1) * (1/3)

=0

As you can see, P ("I like Python") comes out to be 0, but it can be a proper sentence, but due to limited training data, our model didn't do well.

Now, we'll see how smoothing can solve this issue.

Source: https://www.codingninjas.com/studio/library/smoothing-in-nlp

Laplace Smoothing contd…

- The simplest way to do smoothing is to add one to all the n-gram counts, before we normalize them into probabilities.
- All the counts that used to be zero will now have a count of 1, the counts of 1 will be 2, and so on.
- This algorithm is called Laplace smoothing.

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

$$P_{\text{Laplace}}(w_i) = \frac{c_i+1}{N+V}$$

Source: https://vitalflux.com/quick-introduction-smoothing-techniques-language-models/

Laplace Smoothing contd…

- Also called Add-one estimation
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

- MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Add-1 estimate:

$$P_{Add\ 1}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)+1}{C(w_{i-1})+V}$$

Laplace Smoothing contd…

Example:

Training data: <S> I like coding </S>

<S> Ayush likes Python</S>

<S> He likes coding</S>

Test data: <S> I like Python</S>

Let's consider bigrams, a group of two words.

$$P(w_i \mid w_{(i-1)}) = count(w_i\ w_{(i-1)}) / count(w_{(i-1)})$$

Laplace / Add-1 Smoothing

Here, we simply add 1 to all the counts of words so that we never incur 0 value.

$$P_{Laplace}(w_i \mid w_{(i-1)}) = (count(w_i\ w_{(i-1)}) + 1) / (count(w_{(i-1)}) + V)$$

Where V= total words in the training set, 9 in our example.

So, P("I like Python")

- = P(I | <S>)*P(like | I)*P(Python | like)*P(</S> | Python)
- = ((1+1) / (3+9)) * ((1+1) / (1+9)) * ((0+1) / (1+9)) * ((1+1) / (3+9))
- = 1 / 1800

Source: https://www.codingninjas.com/studio/library/smoothing-in-nlp

Advanced: Kneser-Ney Smoothing contd…

- **Kneser Ney:** Steals from words with higher probability and adds to words with low probability. It also uses interpolation and the notion of fertility

Advanced: Kneser-Ney Smoothing contd…

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram

- Interpolation works better

Advanced: Kneser-Ney Smoothing contd…

Linear Interpolation

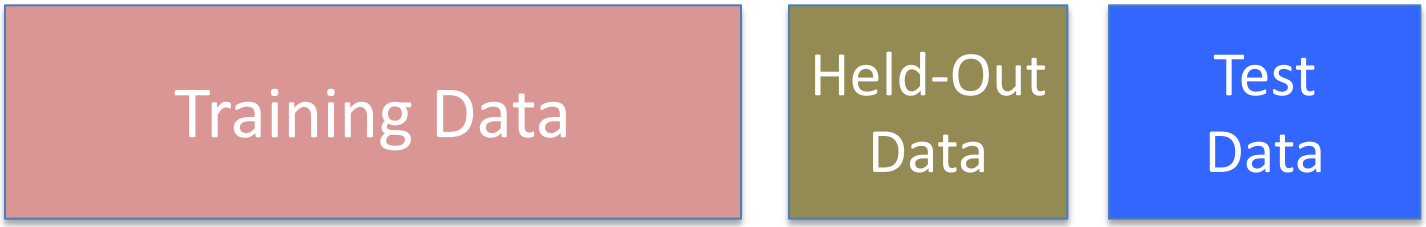
- Simple interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &\quad +\lambda_2 P(w_n|w_{n-1}) \\ &\quad +\lambda_3 P(w_n)\end{aligned}$$

$$\sum_i \lambda_i = 1$$

Advanced: Kneser-Ney Smoothing contd…

- Use a **held-out** corpus



- Choose λs to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held-out set:

Advanced: Kneser-Ney Smoothing contd…

Absolute discounting: just subtract a little from each count

- Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros
- How much to subtract ?
- Church and Gale (1991)'s clever idea
- **It sure looks like c* = (c - 0.75)**

Bigram count in training	Bigram count in held out set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Advanced: Kneser-Ney Smoothing contd…

Absolute Discounting Interpolation

- Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

unigram

– (Maybe keeping a couple extra values of d for counts 1 and 2)

- But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing I

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading* _____ ?
  - "Kong" turns out to be more common than "glasses"
  - … but "Kong" always follows "Hong"
- The unigram is useful exactly when we haven't seen this bigram!
- Instead of P(w): "How likely is w"
- $P_{continuation}$(w): "How likely is w to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \mu \left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|$$

Kneser-Ney Smoothing II

- How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \mu \left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|$$

- Normalized by the total number of word bigram types

$$\left| \{ (w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0 \} \right|$$

$$P_{CONTINUATION}(w) = \frac{\left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|}{\left| \{ (w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0 \} \right|}$$

Kneser-Ney Smoothing III

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

End of UNIT II