# UNIT II
# Language Modelling

By
Dr. S. S. Gharde
Dept. of Information Technology/ AIML
Government Polytechnic Nagpur

## Contents..

# 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

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

# 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

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

## 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:

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

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

# Bigram Model contd…

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

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

- 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

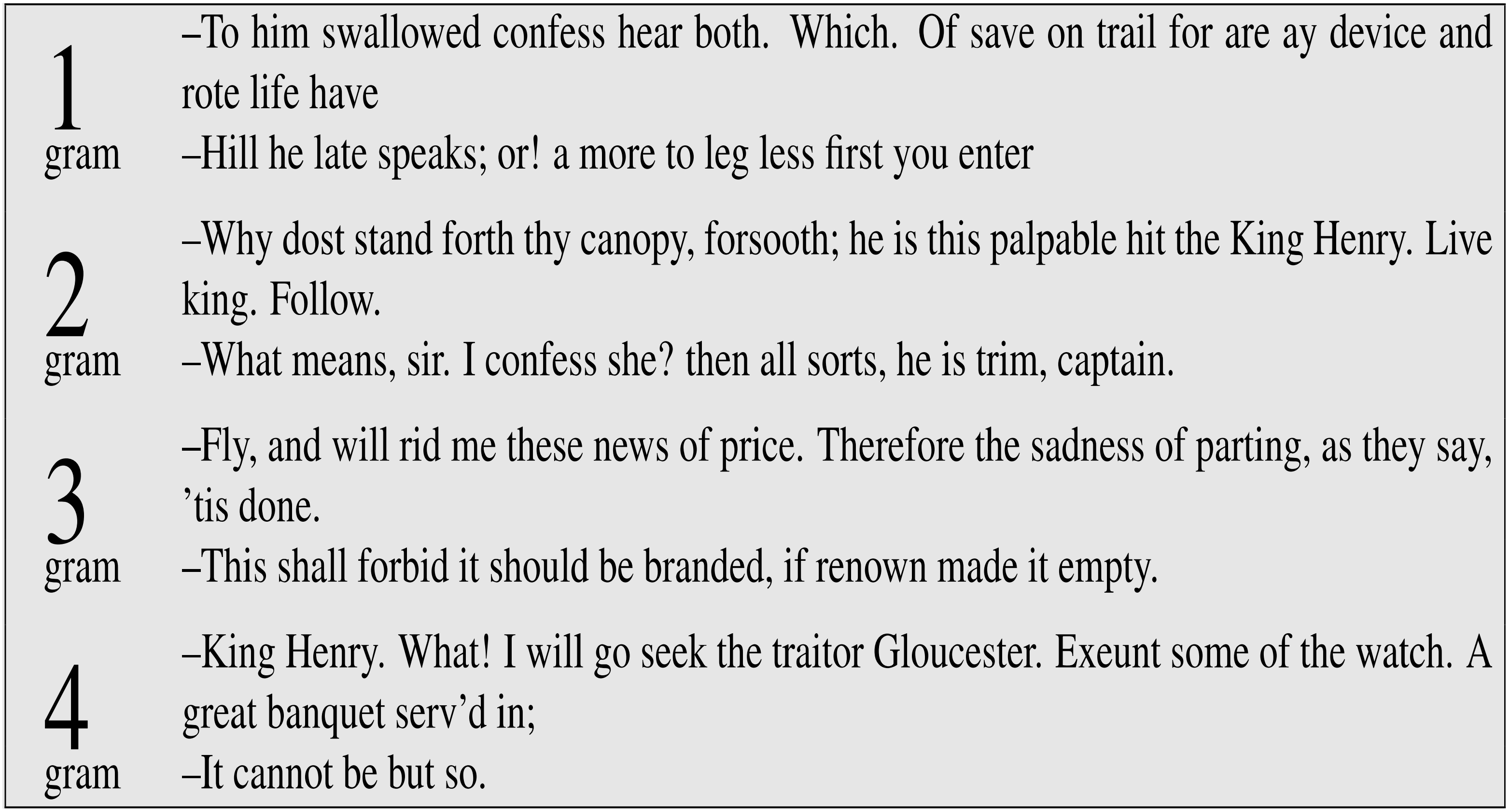
- Choose a random bigram

 (<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<br>–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.<br>–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.<br>–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;<br>–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…

- **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…

- **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…

- 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.

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

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:

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

## 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.

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

**Laplace / Add-1 Smoothing**

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

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

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**

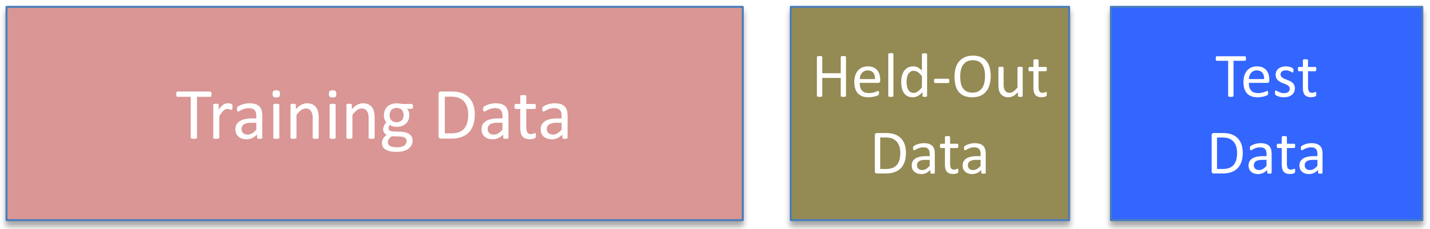
- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

# 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

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

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 glasses ____________?
  - "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

$$\lambda(w_{i-1}) = \underbrace{\frac{d}{c(w_{i-1})}}_{\text{the normalized discount}} \underbrace{\left|\{w : c(w_{i-1}, w) > 0\}\right|}_{\text{The number of word types that can follow } w_{i-1}}$$

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