Language Modeling

Natalie Parde, Ph.D.

Department of Computer Science

University of Illinois at Chicago

CS 521: Statistical Natural Language Processing

Spring 2020

Many slides adapted from Jurafsky and Martin (https://web.stanford.edu/~jurafsky/slp3/).

What is language modeling?

- The process of building statistical models that predict the likelihood of different word or character sequences in a language.

I'm so excited to be taking CS 521 this _______!

fall

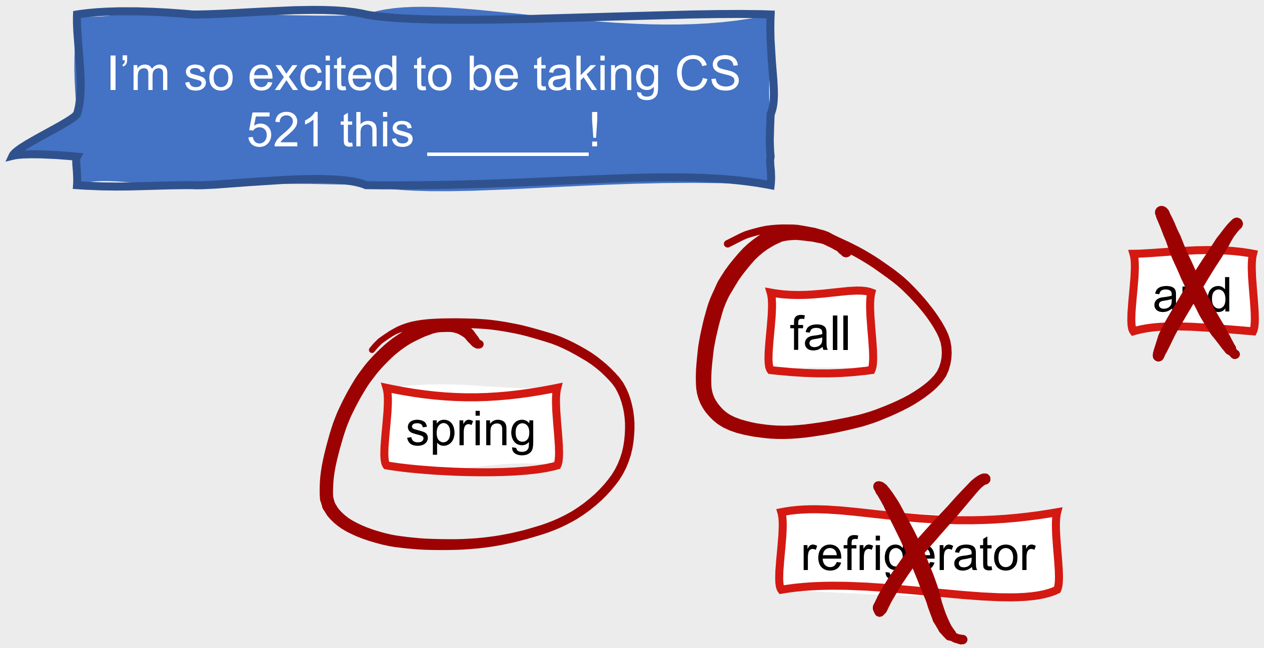
and

spring

refrigerator

What is language modeling?

- The process of building statistical models that predict the likelihood of different word or character sequences in a language.

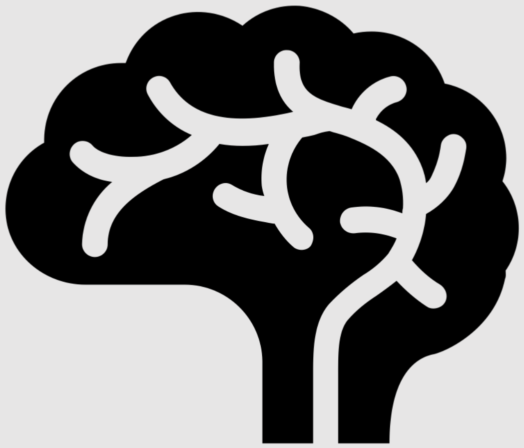


Why is language modeling useful?

- Many reasons!
- Helps in tasks that require words to be identified from noisy, ambiguous input
  - Speech recognition
  - Autocorrect
- Helps in tasks that require sequences of text to be generated
  - Machine translation
  - Image captioning

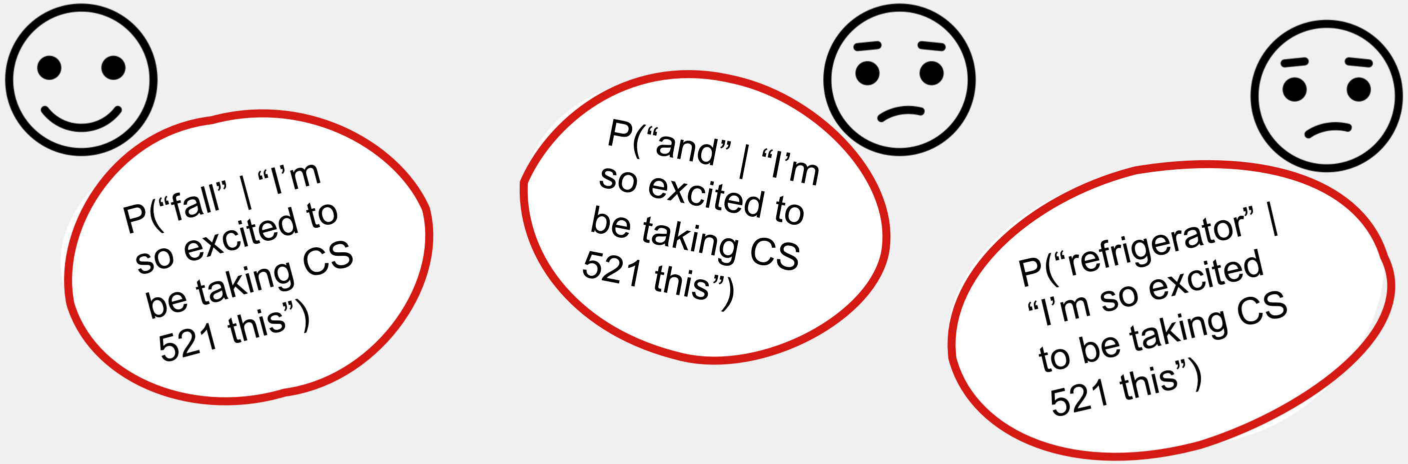
Language models come in many forms.

- Simple (today's focus):
  - N-gram language models
- More sophisticated (later this semester):
  - Neural language models



N-Gram Language Models

- Goal: Predict P(word|history)
  - P("spring" | "I'm so excited to be taking CS 521 this")

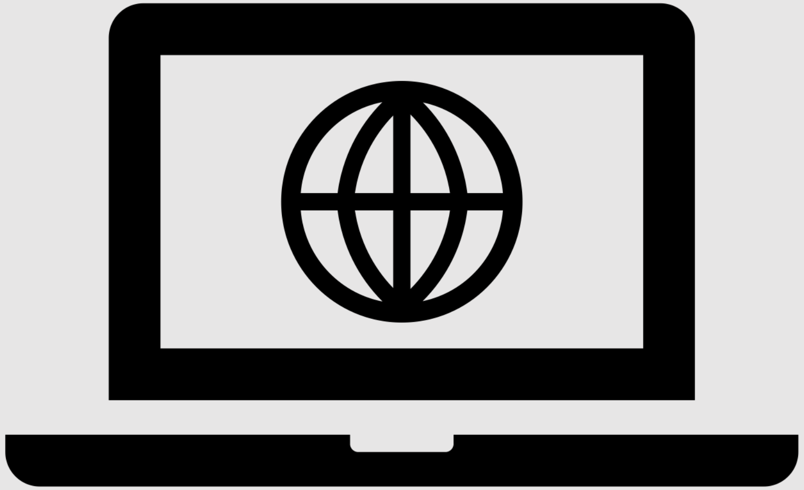


How do we predict these probabilities?

- One method: Estimate it from frequency counts
  - Take a large corpus
  - Count the number of times you see the history
  - Count the number of times the specified word follows the history

P(“spring” | “I’m so excited to be taking CS 521 this”)

= C(“I’m so excited to be taking CS 521 this spring”) / C(“I’m so excited to be taking CS 521 this”)



However, there are a few problems with this method.

- What if our word (or our history) contains uncommon words?
- What if we have limited computing resources?

P("spring" | "I'm so excited to be taking **Natalie Parde's** CS 521 this")

Out of all possible 11-word sequences on the web, how many are "I'm so excited to be taking Natalie Parde's CS 521 this"?

We need a better way to estimate P(word|history)!

- The solution: Instead of computing the probability of a word given its entire history, **approximate the history using the most recent few words**.
- These sequences of words are referred to as **n-grams**, where *n* is the length of the *recent words* + the *current word*

P(“spring” | “taking CS 521 this”)

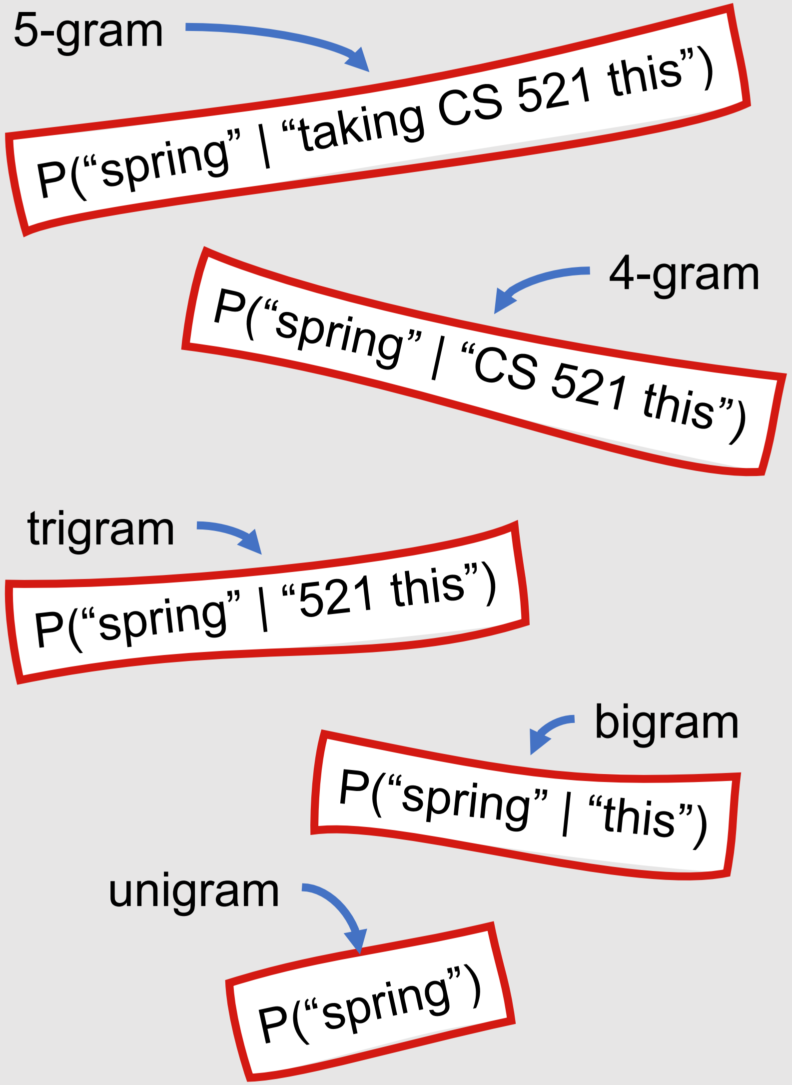
P(“spring” | “CS 521 this”)

P(“spring” | “521 this”)

P(“spring” | “this”)

Special N-Grams

- Most higher-order (n>3) n-grams are simply referred to using the value of *n*
  - 4-gram
  - 5-gram
- However, lower-order n-grams are often referred to using special terms:
  - Unigram (1-gram)
  - Bigram (2-gram)
  - Trigram (3-gram)



N-gram models follow the **Markov assumption**.

- We can predict the probability of some future unit without looking too far into the past
  - **Bigram language model:** Probability of a word depends only on the previous word
  - **Trigram language model:** Probability of a word depends only on the two previous words
  - **N-gram language model:** Probability of a word depends only on the n-1 previous words

More formally….

- $P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$
- We can then multiply these individual word probabilities together to get the probability of a word sequence
  - $P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-N+1}^{k-1})$

P("Winter break is already over?")

P("over?" | "already") * P("already" | "is") * P("is" | "break") * P("break" | "Winter")

To compute n-gram probabilities, maximum likelihood estimation is often used.

- **Maximum Likelihood Estimation (MLE):**
  - Get the requisite n-gram frequency counts from a corpus
  - Normalize them to a 0-1 range
    - $P(w_n \mid w_{n-1})$ = # of occurrences of the bigram $w_{n-1}\ w_n$ / # of occurrences of the unigram w_{n-1}

Example: Maximum Likelihood Estimation

I am cold.

You are cold.

Everyone is cold.

This is Chicago.

Example: Maximum Likelihood Estimation

I am cold. → <s> I am cold. </s>

You are cold. → <s> You are cold. </s>

Everyone is cold. → <s> Everyone is cold. </s>

This is Chicago. → <s> This is Chicago. </s>

Example: Maximum Likelihood Estimation

I am cold. → <s> I am cold. </s>

You are cold. → <s> You are cold. </s>

Everyone is cold. → <s> Everyone is cold. </s>

This is Chicago. → <s> This is Chicago. </s>

Bigram	Frequency
<s> I	1
I am	1
am cold.	1
cold. </s>	3
…	…
is Chicago.	1
Chicago. </s>	1

Example: Maximum Likelihood Estimation

I am cold. → <s> I am cold. </s>

You are cold. → <s> You are cold. </s>

Everyone is cold. → <s> Everyone is cold. </s>

This is Chicago. → <s> This is Chicago. </s>

Bigram	Freq.
<s> I	1
I am	1
am cold.	1
cold. </s>	3
…	…
is Chicago.	1
Chicago. </s>	1

Unigram	Freq.
<s>	4
I	1
am	1
cold.	3
…	…
Chicago.	1
</s>	4

Example: Maximum Likelihood Estimation

I am cold. → <s> I am cold. </s>

You are cold. → <s> You are cold. </s>

Everyone is cold. → <s> Everyone is cold. </s>

This is Chicago. → <s> This is Chicago. </s>

Bigram	Freq.
<s> I	1
I am	1
am cold.	1
cold. </s>	3
…	…
is Chicago.	1
Chicago. </s>	1

Unigram	Freq.
<s>	4
I	1
am	1
cold.	3
…	…
Chicago.	1
</s>	4

P("I" | "<s>") = C("<s> I") / C("<s>") = 1 / 4 = 0.25

Example: Maximum Likelihood Estimation

I am cold. → <s> I am cold. </s>

You are cold. → <s> You are cold. </s>

Everyone is cold. → <s> Everyone is cold. </s>

This is Chicago. → <s> This is Chicago. </s>

Bigram	Freq.
<s> I	1
I am	1
am cold.	1
cold. </s>	3
…	…
is Chicago.	1
Chicago. </s>	1

Unigram	Freq.
<s>	4
I	1
am	1
cold.	3
…	…
Chicago.	1
</s>	4

P(“I” | “<s>”) = C(“<s> I”) / C(“<s>”) = 1 / 4 = 0.25

P(“</s>” | “cold.”) = C(“cold. </s>”) / C(“cold.”) = 3 / 3 = 1.00

Example: Maximum Likelihood Estimation

I am cold.	<s> I am cold. </s>
You are cold.	<s> You are cold. </s>
Everyone is cold.	<s> Everyone is cold. </s>
This is Chicago.	<s> This is Chicago. </s>

Bigram	Freq.
<s> I	1
I am	1
am cold.	1
cold. </s>	3
…	…
is Chicago.	1
Chicago. </s>	1

Unigram	Freq.
<s>	4
I	1
am	1
cold.	3
…	…
Chicago.	1
</s>	4

P("I" | "<s>") = C("<s> I") / C("<s>") = 1 / 4 = 0.25

P("</s>" | "cold.") = C("cold. </s>") / C("cold.") = 3 / 3 = 1.00 🤷‍♀️

What do bigram counts from larger corpora look like?

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

What do bigram probabilities from larger corpora look like?

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

What can we learn from n-gram statistics?

- Syntactic information
  - “to” is usually followed by a verb
  - Nouns often follow verbs
- Task information
  - Virtual assistants are likely to hear the word “I”
- Cultural/sociological information
  - People like some cuisines more than others

What type of n-gram is best?

- In general, the highest-order value of *n* that your data can handle!
- Higher order → sparser
- Note: Because n-gram probabilities tend to be small, it is most common to perform operations in log space
  - Multiplying in linear space = adding in log space
  - Less likely to run into numerical underflow when representing sequences

Evaluating Language Models

- Two types of evaluation paradigms:
  - Extrinsic
  - Intrinsic
- **Extrinsic evaluation:** Embed the language model in an application, and compute changes in task performance
- **Intrinsic evaluation:** Measure the quality of the model, independent of any application

Perplexity

- Intrinsic evaluation metric for language models
- Perplexity (PP) of a language model on a test set is the **inverse probability of the test set**, normalized by the number of words in the test set

More formally….

- $PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \ldots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$
    - Where *W* is a test set containing words w_1, w_2, …, w_n
- Higher conditional probability of a word sequence → lower perplexity
    - Minimizing perplexity = maximizing test set probability according to the language model

Example: Perplexity

Training Set

Word	Frequency
CS	10
521	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Example: Perplexity

Training Set

Word	Frequency
CS	10
521	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Test String

CS 521 Statistical Natural Language Processing University of Illinois Chicago

Example: Perplexity

Training Set

Word	Frequency
CS	10
521	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Test String

CS 521 Statistical Natural Language Processing University of Illinois Chicago

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \ldots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Example: Perplexity

Training Set

Word	Frequency
CS	10
521	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Test String

CS 521 Statistical Natural Language Processing University of Illinois Chicago

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \ldots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

P("CS") = C("CS") / C(<all unigrams>) = 10/100 = 0.1

Example: Perplexity

Training Set

Word	Frequency
CS	10
521	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Test String

CS 521 Statistical Natural Language Processing University of Illinois Chicago

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \dots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

P(“CS”) = C(“CS”) / C(<all unigrams>) = 10/100 = 0.1

P(“521”) = C(“521”) / C(<all unigrams>) = 10/100 = 0.1

Example: Perplexity

Training Set

Word	Frequency	P(Word)
CS	10	0.1
521	10	0.1
Statistical	10	0.1
Natural	10	0.1
Language	10	0.1
Processing	10	0.1
University	10	0.1
of	10	0.1
Illinois	10	0.1
Chicago	10	0.1

Test String

CS 521 Statistical Natural Language Processing University of Illinois Chicago

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \ldots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Example: Perplexity

Training Set

Word	Frequency	P(Word)
CS	10	0.1
521	10	0.1
Statistical	10	0.1
Natural	10	0.1
Language	10	0.1
Processing	10	0.1
University	10	0.1
of	10	0.1
Illinois	10	0.1
Chicago	10	0.1

Test String

CS 521 Statistical Natural Language Processing University of Illinois Chicago

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \dots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

PP("CS 521 Statistical Natural Language Processing University of Illinois Chicago")

$$= \sqrt[10]{\frac{1}{0.1*0.1*0.1*0.1*0.1*0.1*0.1*0.1*0.1*0.1}} = 10$$

Example: Perplexity

Training Set

Word	Frequency	P(Word)
CS	1	
521	1	
Statistical	1	
Natural	1	
Language	1	
Processing	1	
University	1	
of	1	
Illinois	1	
Chicago	91	

Test String

Illinois Chicago Chicago Chicago Chicago Chicago Chicago Chicago Chicago Chicago

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \ldots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Example: Perplexity

Training Set

Word	Frequency	P(Word)
CS	1	0.01
521	1	0.01
Statistical	1	0.01
Natural	1	0.01
Language	1	0.01
Processing	1	0.01
University	1	0.01
of	1	0.01
Illinois	1	0.01
Chicago	91	0.91

Test String

Illinois Chicago Chicago Chicago Chicago Chicago Chicago Chicago Chicago Chicago

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \ldots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Example: Perplexity

Training Set

Word	Frequency	P(Word)
CS	1	0.01
521	1	0.01
Statistical	1	0.01
Natural	1	0.01
Language	1	0.01
Processing	1	0.01
University	1	0.01
of	1	0.01
Illinois	1	0.01
Chicago	91	0.91

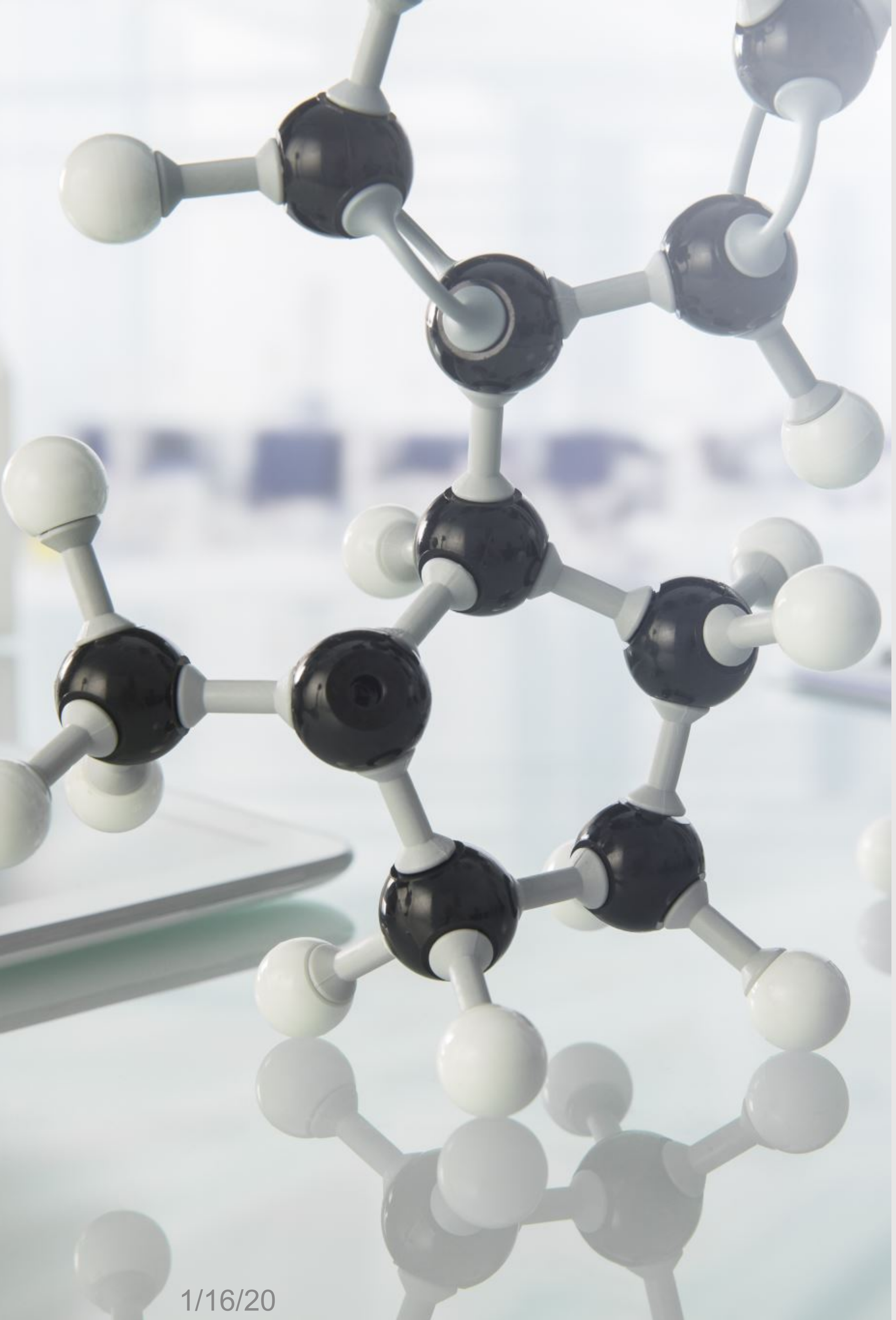
Test String

Illinois Chicago Chicago Chicago Chicago Chicago Chicago Chicago Chicago Chicago

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1 w_2 \ldots w_n)}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

PP("CS 521 Statistical Natural Language Processing University of Illinois Chicago")

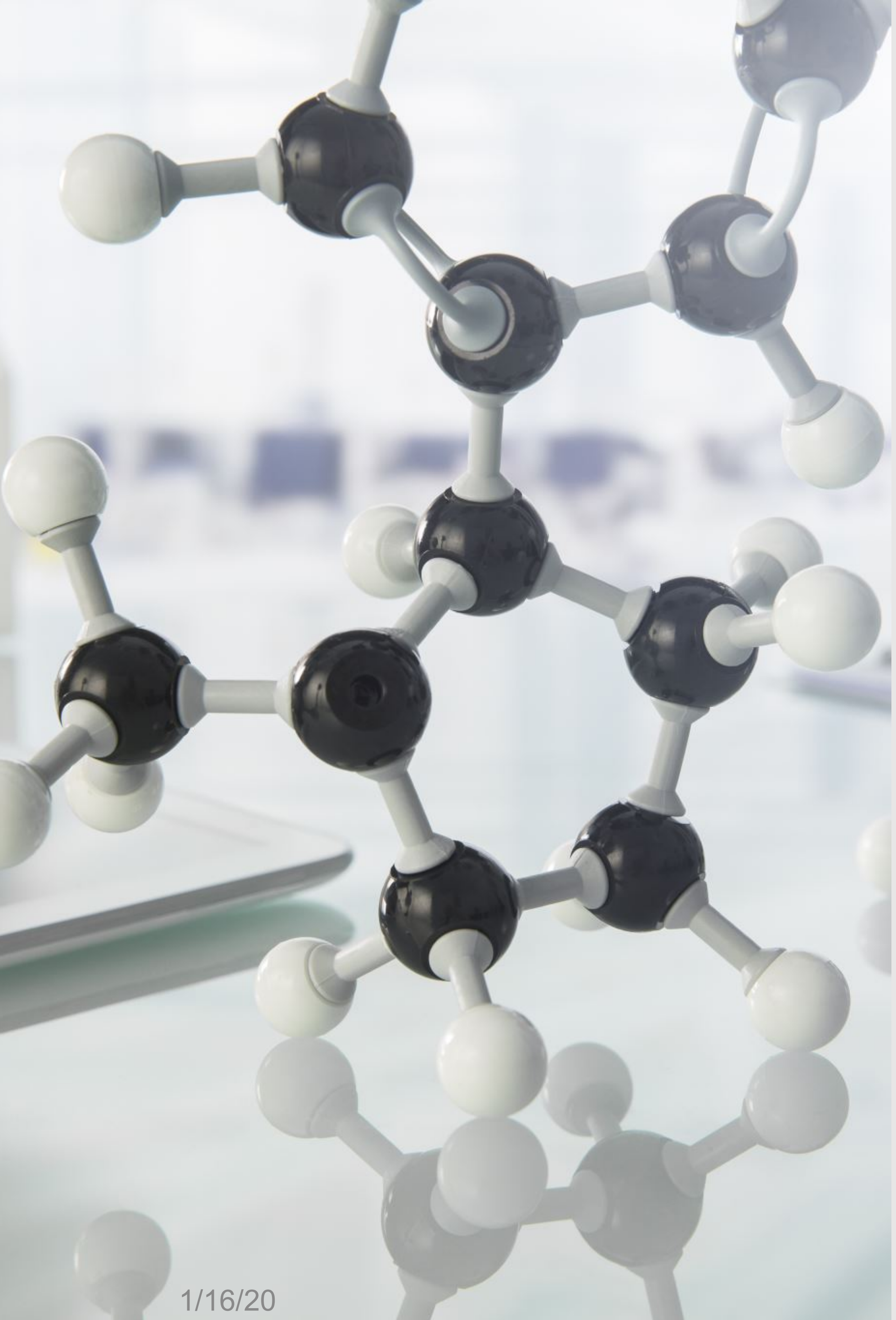
$$= \sqrt[10]{\frac{1}{0.01*0.91*0.91*0.91*0.91*0.91*0.91*0.91*0.91*0.91}} = 1.73$$



Perplexity can be used to compare different language models.

Which language model is best?

- Model A: Perplexity = 962
- Model B: Perplexity = 170
- Model C: Perplexity = 109



Perplexity can be used to compare different language models.

Which language model is best?

- Model A: Perplexity = 962
- Model B: Perplexity = 170
- Model C: Perplexity = 109

A cautionary note….

- Improvements in perplexity do not guarantee improvements in task performance!
- However, the two are often correlated (and perplexity is quicker and easier to check)
- Strong language model evaluations also include an extrinsic evaluation component

Generalization and Sparsity

- Probabilities in n-gram models often encode specific characteristics of the training corpus
  - These characteristics are encoded more strongly in higher-order n-grams
- We can see this when generating text from different n-gram models
  - Select an n-gram randomly from the distribution of all n-grams in the training corpus
  - Randomly select an n-gram from the same distribution, dependent on the previous n-gram
    - In a bigram model, if the previous bigram was “CS 521” then the next bigram has to start with “521”
  - Repeat until the sentence-final token is reached

Sample Sentences Generated from Shakespearean N-Gram Models

Unigram

- To him swallowed confess hear both. 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

Bigram

- 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.

Trigram

- 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.

Sample Sentences Generated from Shakespearean N-Gram Models

No coherence between words

Unigram

- To him swallowed confess hear both. 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

Bigram

- 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.

Trigram

- 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.

Sample Sentences Generated from Shakespearean N-Gram Models

Unigram

No coherence between words

- To him swallowed confess hear both. 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

Bigram

Minimal local coherence between words

- 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.

Trigram

- 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.

Sample Sentences Generated from Shakespearean N-Gram Models

Unigram

No coherence between words

- To him swallowed confess hear both. 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

Bigram

Minimal local coherence between words

- 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.

Trigram

More coherence….

- 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.

Sample Sentences Generated from Shakespearean N-Gram Models

Unigram

No coherence between words

- To him swallowed confess hear both. 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

Bigram

Minimal local coherence between words

- 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.

Trigram

More coherence….

- 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

Direct quote from Shakespeare

- 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.

Why did we end up with a direct Shakespearean quote?

- The corpus of all Shakespearean text is relatively small
  - N=884,647
  - V=29,066
- This means the higher-order n-gram matrices are *very* sparse!
- Only five possible continuations (*that*, *I*, *he*, *thou*, and *so*) for the sequence *It cannot be but*

Sparse n-gram models assume a probability of zero for a large number of n-grams.

Training

Bigram	Frequency
CS 421	8
CS 590	5
CS 594	2

Test

CS 521

P("521" | "CS") = 0

Why is this problematic?

- We're underestimating the probability of lots of potential n-grams
- If the probability of any n-gram in the test set is 0, the probability of the entire test set will be 0
  - Perplexity is the inverse probability of the test set
  - It's impossible to divide by 0
  - We can't compute perplexity!

Handling Unknown Words

- Out of vocabulary (OOV)
- Model potential OOV words by adding a pseudoword, <UNK>
- How to assign a probability to <UNK>?
  - Option A:
    - Choose a fixed word list
    - Convert any words not in that list to <UNK>
    - Estimate the probabilities for <UNK> like any other word
  - Option B:
    - Replace all words occurring fewer than n times with <UNK>
    - Estimate the probabilities for <UNK> like any other word
- Beware of “gaming” perplexity!!
  - If you choose a small vocabulary and thus assign <UNK> a high probability, your language model will probably have lower perplexity (make sure to only compare to other language models using the exact same vocabulary)

Handling Words in Unseen Contexts

- Smoothing: Taking a bit of the probability mass from more frequent events and giving it to unseen events.
  - Sometimes also called "discounting"
- Many different smoothing techniques:
  - Laplace (add-one)
  - Add-k
  - Stupid backoff
  - Kneser-Ney

Bigram	Frequency
CS 421	8
CS 590	5
CS 594	2
CS 521	0 😢

Bigram	Frequency
CS 421	7
CS 590	5
CS 594	2
CS 521	1 🥰

Laplace Smoothing

- Add one to all n-gram counts before they are normalized into probabilities
- Not the highest-performing technique for language modeling, but a useful baseline
  - Practical method for other text classification tasks
- $P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Laplace}}(w_i) = \frac{c_i+1}{N+V}$

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4
is	8
cold	6
hot	0

Bigram	Frequency
Chicago is	2
is cold	4
is hot	0

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4
is	8
cold	6
hot	0

Bigram	Frequency
Chicago is	2
is cold	4
is hot	0

$$P(w_i) = \frac{c_i}{N}$$

Unigram	Probability
Chicago	$\frac{4}{18} = 0.22$
is	$\frac{8}{18} = 0.44$
cold	$\frac{6}{18} = 0.33$
hot	$\frac{0}{18} = 0.00$

Bigram	Probability
Chicago is	
is cold	
is hot	

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4
is	8
cold	6
hot	0

Bigram	Frequency
Chicago is	2
is cold	4
is hot	0

$$P(w_i) = \frac{c_i}{N}$$

Unigram	Probability
Chicago	$\frac{4}{18} = 0.22$
is	$\frac{8}{18} = 0.44$
cold	$\frac{6}{18} = 0.33$
hot	$\frac{0}{18} = 0.00$

Bigram	Probability
Chicago is	$\frac{2}{4} = 0.50$
is cold	$\frac{4}{8} = 0.50$
is hot	$\frac{0}{8} = 0.00$

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4
is	8
cold	6
hot	0

Bigram	Frequency
Chicago is	2
is cold	4
is hot	0

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

Unigram	Probability
Chicago	
is	
cold	
hot	

Bigram	Probability
Chicago is	
is cold	
is hot	

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4+1
is	8+1
cold	6+1
hot	0+1

Bigram	Frequency
Chicago is	2+1
is cold	4+1
is hot	0+1

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

Unigram	Probability
Chicago	
is	
cold	
hot	

Bigram	Probability
Chicago is	
is cold	
is hot	

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4+1
is	8+1
cold	6+1
hot	0+1

Bigram	Frequency
Chicago is	2+1
is cold	4+1
is hot	0+1

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

Unigram	Probability
Chicago	$\frac{5}{22} = 0.23$
is	$\frac{9}{22} = 0.41$
cold	$\frac{7}{22} = 0.32$
hot	$\frac{1}{22} = 0.05$

Bigram	Probability
Chicago is	
is cold	
is hot	

Example: Laplace Smoothing

Corpus Statistics:

Unigram	Frequency
Chicago	4+1
is	8+1
cold	6+1
hot	0+1

Bigram	Frequency
Chicago is	2+1
is cold	4+1
is hot	0+1

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

Unigram	Probability
Chicago	$\frac{5}{22} = 0.23$
is	$\frac{9}{22} = 0.41$
cold	$\frac{7}{22} = 0.32$
hot	$\frac{1}{22} = 0.05$

Bigram	Probability
Chicago is	$\frac{3}{4+4} = \frac{3}{8} = 0.38$
is cold	$\frac{5}{8+4} = \frac{5}{12} = 0.42$
is hot	$\frac{1}{8+4} = \frac{1}{12} = 0.08$

This results in a sharp change in probabilities!

Bigram	Probability
Chicago is	$\frac{2}{4} = 0.50$
is cold	$\frac{4}{8} = 0.50$
is hot	$\frac{0}{8} = 0.00$

Bigram	Probability
Chicago is	$\frac{3}{8} = 0.38$
is cold	$\frac{5}{12} = 0.42$
is hot	$\frac{1}{12} = 0.08$

Add-K Smoothing

- Moves a bit less of the probability mass from seen to unseen events
- Rather than adding one to each count, add a fractional count
  - 0.5
  - 0.05
  - 0.01
- The value *k* can be optimized on a validation set
- $P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Add-K}}(w_i) = \frac{c_i + k}{N + kV}$
- $P(w_n|w_{n-1}) = \frac{c(w_{n-1}w_n)}{c(w_{n-1})} \rightarrow P_{\text{Add-K}}(w_n|w_{n-1}) = \frac{c(w_{n-1}w_n)+k}{c(w_{n-1})+kV}$

Add-K smoothing is useful for some tasks, but still tends to be suboptimal for language modeling.

- Other smoothing techniques?
  - **Backoff:** Use the specified n-gram size to estimate probability if its count is greater than 0; otherwise, *backoff* to a lower-order n-gram
  - **Interpolation:** Mix the probability estimates from multiple n-gram sizes, weighing and combining the n-gram counts

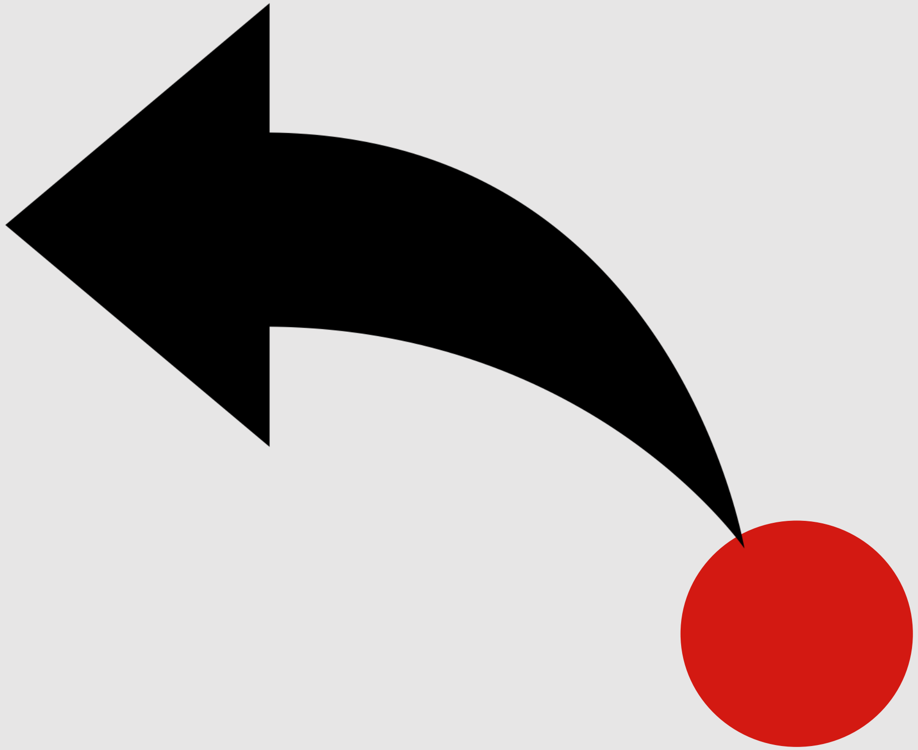
Interpolation

- **Linear interpolation**
  - $P'(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$
    - Where $\sum_i \lambda_i = 1$
- **Conditional interpolation**
  - $P'(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1}) P(w_n)$

Context-conditioned weights

Backoff

- If the n-gram we need has zero counts, approximate it by backing off to the (n-1)-gram
- Continue backing off until we reach a size that has non-zero counts
- Just like with smoothing, some probability mass from higher-order n-grams needs to be redistributed to lower-order n-grams



Katz Backoff

- Incorporate a function α to distribute probability mass to lower-order n-grams
- Rely on a discounted probability P* if the n-gram has non-zero counts
- Otherwise, recursively back off to the Katz probability for the (n-1)-gram

- $P_{BO}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } c(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{BO}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise} \end{cases}$

Kneser-Ney Smoothing

- One of the most commonly used and best-performing n-gram smoothing methods
- Incorporates absolute discounting
  - Subtracts an absolute discount *d* from each count
- Simple absolute discounting:
  - $P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)-d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i)$

Kneser-Ney Smoothing

- One of the most commonly used and best-performing n-gram smoothing methods
- Incorporates absolute discounting
  - Subtracts an absolute discount *d* from each count
- Simple absolute discounting:
  - $P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)-d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i)$
- Kneser-Ney smoothing comes up with a more sophisticated way to handle the lower-order n-gram distribution

Kneser-Ney Smoothing

- Objective: Capture the intuition that although some lower-order n-grams are frequent, they are mainly only frequent in specific contexts
  - tall nonfat decaf peppermint ________
    - "york" is a more frequent unigram than "mocha" (7.4 billion results vs. 135 million results on Google), but it's mainly frequent when it follows the word "new"
- Creates a unigram model that estimates the probability of seeing the word *w* as a novel continuation, in a new unseen context
  - Based on the number of different contexts in which *w* has already appeared
  - $P_{\text{Continuation}}(w) = \frac{|\{v : C(vw) > 0\}|}{|\{(u', w') : C(u'w') > 0\}|}$

Kneser-Ney Smoothing

$$P_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\left(c_{KN}\left(w_{i-n+1}^{i}\right) - d,\ \ 0\right)}{\sum_v c_{KN}\left(w_{i-n+1}^{i-1}v\right)} + \lambda(w_{i-n+1}^{i-1})P_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1})$$

Kneser-Ney Smoothing

$$P_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\left(c_{KN}\left(w_{i-n+1}^{i}\right) - d,\ \ 0\right)}{\sum_v c_{KN}\left(w_{i-n+1}^{i-1} v\right)} + \lambda(w_{i-n+1}^{i-1}) P_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1})$$

Normalizing constant to distribute the probability mass that's been discounted

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : c(w_{i-1}w) > 0\}|$$

Kneser-Ney Smoothing

$$P_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\left(c_{KN}\left(w_{i-n+1}^{i}\right) - d,\ \ 0\right)}{\sum_v c_{KN}\left(w_{i-n+1}^{i-1}v\right)} + \lambda(w_{i-n+1}^{i-1})P_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1})$$

Normalizing constant to distribute the probability mass that's been discounted

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)}|\{w : c(w_{i-1}w) > 0\}|$$

Normalized discount

Number of word types that can follow w_{i-1}

Kneser-Ney Smoothing

$$P_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d,\ \ 0)}{\sum_v c_{KN}(w_{i-n+1}^{i-1}v)} + \lambda(w_{i-n+1}^{i-1}) P_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1})$$

Normalizing constant to distribute the probability mass that's been discounted

Regular count for the highest-order n-gram, or the number of unique single word contexts for lower-order n-grams

Kneser-Ney Smoothing

$$P_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d,\ \ 0)}{\sum_v c_{KN}(w_{i-n+1}^{i-1}v)} + \lambda(w_{i-n+1}^{i-1})P_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1})$$

Normalizing constant to distribute the probability mass that's been discounted

Regular count for the highest-order n-gram, or the number of unique single word contexts for lower-order n-grams

At termination of recursion, unigrams are interpolated with the uniform distribution (ε = empty string)

$$P_{KN}(w) = \frac{\max(c_{KN}(w) - d, 0)}{\sum_{w'} c_{KN}(w')} + \lambda(\varepsilon)\frac{1}{V}$$

Stupid Backoff

- Gives up the idea of trying to make the language model a true probability distribution 😌
- No discounting of higher-order probabilities
- If a higher-order n-gram has a zero count, simply backoff to a lower-order n-gram, weighted by a fixed weight
- $S\left(w_i \middle| w_{i-k+1}^{i-1}\right) = \begin{cases} \frac{c(w_{i-k+1}^{i})}{c(w_{i-k+1}^{i-1})} & \text{if } c\left(w_{i-k+1}^{i}\right) > 0 \\ \lambda S\left(w_i \middle| w_{i-k+2}^{i-1}\right) & \text{otherwise} \end{cases}$
    - Terminates in the unigram, which has the probability:
        - $S(w) = \frac{c(w)}{N}$

Stupid Backoff

- Gives up the idea of trying to make the language model a true probability distribution 😌
- No discounting of higher-order probabilities
- If a higher-order n-gram has a zero count, simply backoff to a lower-order n-gram, weighted by a fixed weight
- $S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} \frac{c(w_{i-k+1}^{i})}{c(w_{i-k+1}^{i-1})} & \text{if } c(w_{i-k+1}^{i}) > 0 \\ \lambda S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$
  - Terminates in the unigram, which has the probability:
    - $S(w) = \frac{c(w)}{N}$

Generally, 0.4 works well (Brants et al., 2007)

Summary: Language Modeling

- **Language models** are statistical models that predict the likelihood of word or character sequences in a language
- **N-gram language models** are based on n-gram frequencies
  - **N-Gram:** An *n*-length sequence of words or characters
- **Maximum likelihood estimation** is often used to compute n-gram probabilities
- Language models can be evaluated intrinsically using **perplexity**
- Unknown words and words in unseen contexts need to be handled to avoid issues stemming from n-gram **sparsity**
- N-gram language models can be improved using a variety of **smoothing techniques**
  - **Laplace smoothing**
  - **Add-K smoothing**
  - **Interpolation**
  - **Katz backoff**
  - **Kneser-Ney smoothing**
  - **Stupid backoff**