N-Gram Language Models

Natalie Parde

UIC CS 421

Language is inherently contextual.

- Words or characters in language are dependent upon one another!
- **Sequence modeling** allows us to make use of sequential information in language
- One way to model sequential information in language is with **language models**

This Week's Topics

Tuesday

N-gram language modeling

Evaluating LMs

Improving n-gram LMs

Thursday

Text classification

Naïve Bayes

Evaluating text classifiers

This Week's Topics

Tuesday

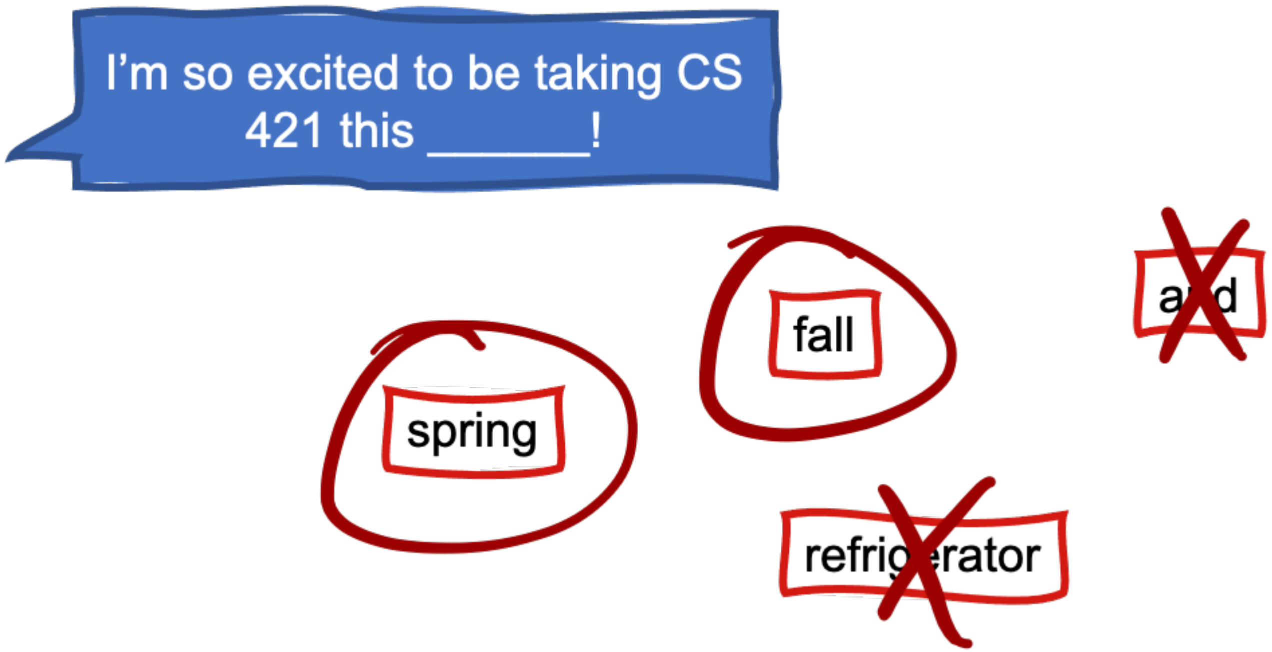
- N-gram language modeling
- Evaluating LMs
- Improving n-gram LMs

Thursday

- Text classification
- Naïve Bayes
- Evaluating text classifiers

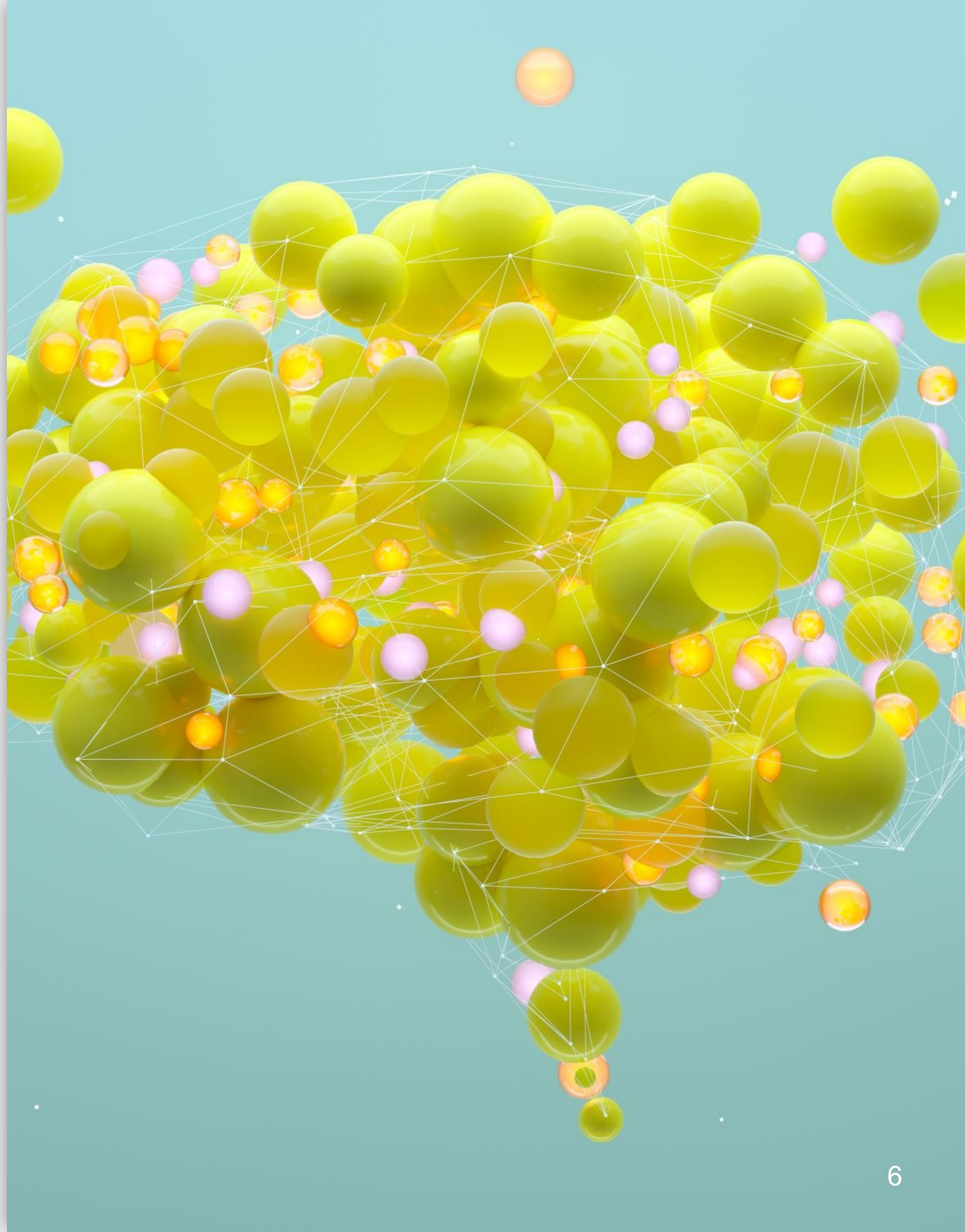
Language Modeling

- Learning how to effectively predict the likelihood of word or character sequences in a language



Why is language modeling useful?

- Helps identify words in noisy, ambiguous input
  - Speech recognition or autocorrect
- Helps generate natural-sounding language
  - Machine translation or image captioning
- In contemporary NLP, language modeling forms the basis of most approaches
  - Language representation



Language models come in many forms!

- More straightforward:
  - **N-gram language models**
- More sophisticated
  - Neural language models

N-Grams

- Sequences of a predefined item type within a language
  - N → Size of the sequence
  - -gram → Greek-derived suffix meaning "what is written"
- First use of the term appears to be in the late 1940s
  - *A Mathematical Theory of Communication*, by Claude Shannon: https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

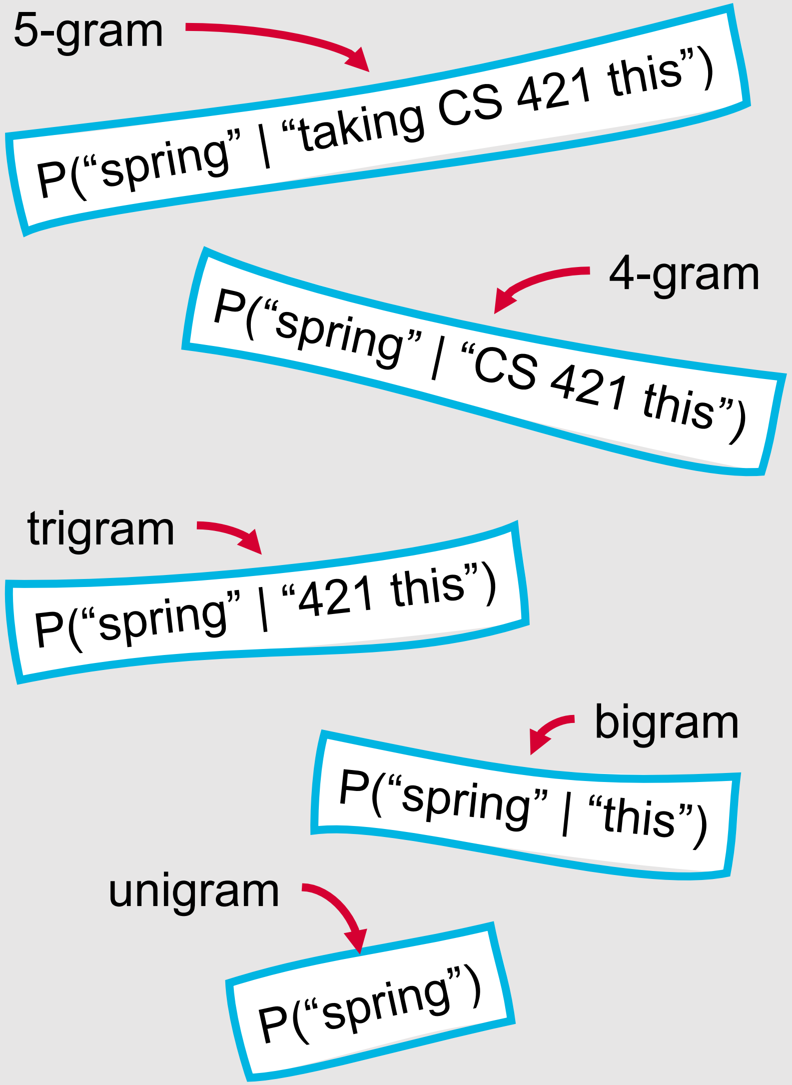
N-grams can be words, characters, or any other type of item in your language.

N-grams are interesting!

N-grams are interesting!

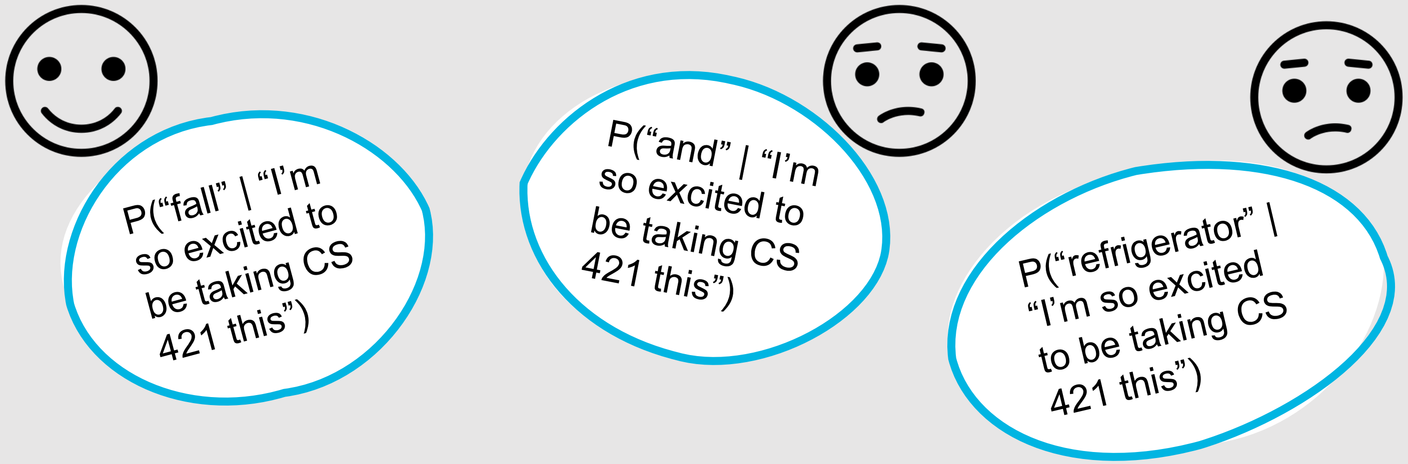
Special N-Grams

- 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 Language Models

- Goal: Predict P(word|history)
  - P(“spring” | “I’m so excited to be taking CS 421 this”)

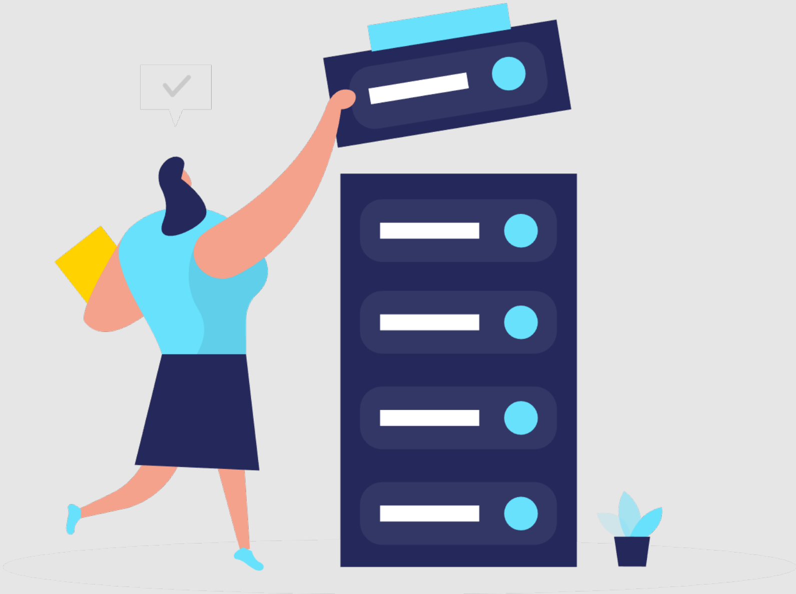


Probabilities for n-gram language models come from corpus frequencies.

- Intuition:
  1. Take a large corpus
  2. Count the number of times you see the history
  3. Count the number of times the specified word follows the history

P("spring" | "I'm so excited to be taking CS 421 this")

= C("I'm so excited to be taking CS 421 this spring") / C("I'm so excited to be taking CS 421 this")



However, we don’t necessarily want to consider our *entire* history.

- What if 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 421 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 421 this”?

Better way of estimating P(word|history)

- Instead of computing the probability of a word given its entire history, **approximate the history using the most recent few words**.
- We do this using fixed-length **n-grams**.

P("spring" | "taking CS 421 this")

P("spring" | "CS 421 this")

P("spring" | "421 this")

P("spring" | "this")

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....

- $P\left(w_k \middle| w_1^{k-1}\right) \approx P(w_k | w_{k-N+1}^{k-1})$
- 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("Summer break is already over?")

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

To compute n-gram probabilities, we can use maximum likelihood estimation.

- **Maximum Likelihood Estimation (MLE):**
  - Get the required 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 $w_{n-1}\ w_n$, divided by
      - # 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 🤷‍♀️

We can learn a lot of useful things from n-gram statistics!

- Syntactic information
  - Do nouns often follow verbs?
  - Do verbs usually follow specific unigrams?
- Task-relevant information
  - Is it likely that virtual assistants will hear the word “I” in a user’s input?
- Cultural or sociological information
  - Are people likelier to want quesadillas than haggis?

Which type of n-gram is best?

- In general, the highest-order value of *n* that your data can support
- Sparsity increases with order, and sparse feature vectors are not very useful when training statistical models
- Make sure that your dataset is large enough to handle your selected n-gram size
- We can usually determine this by running experiments on the same data with different n-gram sizes and figuring out which size leads to the best results
- For a deep dive into statistical power in NLP experiments, check out the following paper:
  - *With Little Power Comes Great Responsibility*, by Dallas Card et al.: https://aclanthology.org/2020.emnlp-main.745/

This Week's Topics

Tuesday

- N-gram language modeling
- Evaluating LMs
- Improving n-gram LMs

Thursday

- Text classification
- Naïve Bayes
- Evaluating text classifiers

We’ve learned how to build n-gram language models, but how do we evaluate them?

- 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
  - History size depends on n-gram size
    - $P(w_i | w_{i-1})$ vs $P(w_i | w_{i-2} w_{i-1})$, etc.
- 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
421	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
421	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Test String

CS 421 Statistical Natural Language Processing University of Illinois Chicago

Example: Perplexity

Training Set

Word	Frequency
CS	10
421	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Test String

CS 421 Statistical Natural Language Processing University of Illinois Chicago

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

Example: Perplexity

Training Set

Word	Frequency
CS	10
421	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Test String

CS 421 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

Example: Perplexity

Training Set

Word	Frequency
CS	10
421	10
Statistical	10
Natural	10
Language	10
Processing	10
University	10
of	10
Illinois	10
Chicago	10

Test String

CS 421 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

P("421") = C("421") / C(<all unigrams>) = 10/100 = 0.1

Example: Perplexity

Training Set

Word	Frequency	P(Word)
CS	10	0.1
421	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 421 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
421	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 421 Statistical Natural Language Processing University of Illinois Chicago

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

PP("CS 421 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	
421	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
421	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

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

Example: Perplexity

Training Set

Word	Frequency	P(Word)
CS	1	0.01
421	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

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

PP("Illinois Chicago Chicago Chicago Chicago Chicago Chicago Chicago Chicago 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

What kind of perplexity scores are state-of-the-art language models reaching?

- Depends on the dataset
- Recently, as low as:
  - ~10 on WikiText-103: https://paperswithcode.com/sota/language-modelling-on-wikitext-103
  - ~20 on Penn Treebank (Word Level): https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

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

How can we generate text using an n-gram language model?

1. Select an n-gram randomly from the distribution of all n-grams in the training corpus
2. Randomly select an n-gram from the same distribution, dependent on the previous n-gram
   - If we're using a bigram model and the previous bigram was "CS 421," our next bigram has to start with "421")
3. Repeat until the sentence-final token is reached

N-gram size affects generation output!

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 were we generating verbatim Shakespeare text with a 4-gram language model?

- The corpus of all Shakespearean text is relatively small (by modern NLP standards)
  - 29,066 vocabulary words
  - 884,647 tokens
- This means higher-order n-gram matrices are sparse:
  - Only five possible continuations for “It cannot be but” (“that,” “I,” “he,” “thou,” and “so”)
    - Probability for all other continuations is assumed to be zero

This Week's Topics

Tuesday

- N-gram language modeling
- Evaluating LMs
- Improving n-gram LMs

Thursday

- Text classification
- Naïve Bayes
- Evaluating text classifiers

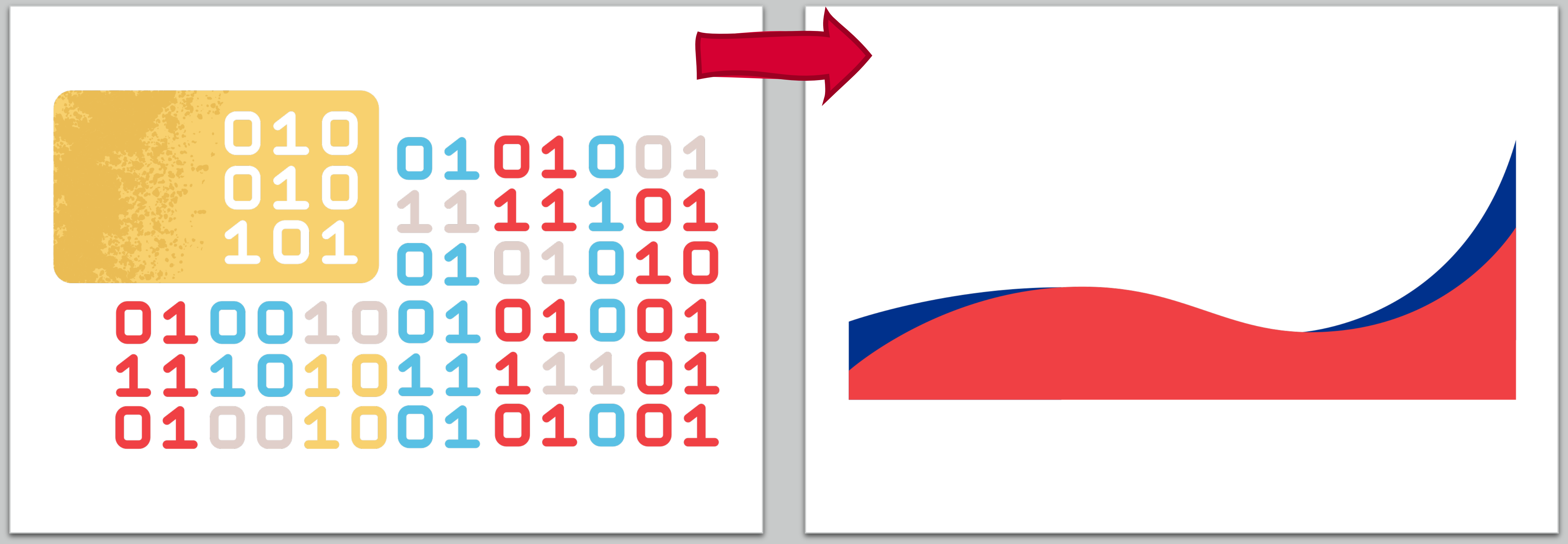
“Zero” probabilities create challenges for language models.

- Zero probabilities occur in two different scenarios:
  - **Unknown words** (**out-of-vocabulary** words)
  - Known words in **unseen contexts**
- However, language is varied and often unpredictable---few combinations are truly impossible
- Zero probabilities also interfere with perplexity calculations

Modeling Unknown Words

- Add a pseudoword **<UNK>** to the vocabulary
- Then….
  - Option A:
    - Choose a fixed words 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
  - Option C:
    - Replace the first occurrence of each word with <UNK>
    - Estimate the probabilities for <UNK> like any other word
- Beware: If <UNK> ends up with a high probability (e.g., because you have a small vocabulary), your language model will have artificially lower perplexity!
  - Make sure to compare to other language models using the same vocabulary to avoid gaming this metric

We can handle known words in previously unseen contexts by applying smoothing techniques.



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, 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
…	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
…	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
…	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
…	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
…	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
…	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:

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

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

Unigram	Frequency
Chicago	4
is	8
cold	6
hot	0

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

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	$\frac{1}{8+4} = \frac{1}{12} = 0.08$

Probabilities: Before and After

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 (e.g., 0.5 or 0.01)
  - $P(w_i) = \frac{c_i}{N} \rightarrow P_{\text{Add-K}}(w_i) = \frac{c_i+k}{N+kV}$
  - $P(w_n|w_{n-1}) = \frac{c(w_{n-1}w_n)}{c(w_{n-1})} \rightarrow$
    $P_{\text{Add-K}}(w_n|w_{n-1}) = \frac{c(w_{n-1}w_n)+k}{c(w_{n-1})+kV}$
- The value *k* can be optimized on a validation set

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 smaller-size n-gram until you reach a size with non-zero counts
  - **Interpolation:** Mix the probability estimates from multiple n-gram sizes, weighing and combining the n-gram counts

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

Interpolation

N	Weight
3	0.5
2	0.4
1	0.1

N-Gram	Probability	Value
I ❤️ 421	P(421 | I ❤️)	0.7
❤️ 421	P(421 | ❤️)	0.5
421	P(421)	0.2

$$\boxed{0.5 * 0.7 + 0.4 * 0.5 + 0.1 * 0.2 = 0.57}$$

- **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

N-Gram	Probability	Value	Weight
I ❤️ 421	P(421 | I ❤️)	0.7	0.5
I 🚕 421	P(421 | I 🚕)	0.7	0.1

Some smoothing techniques incorporate several of these techniques.

Kneser-Ney Smoothing

Stupid Backoff

Kneser-Ney Smoothing

- Commonly used, high-performing technique that incorporates absolute discounting
- 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

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

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

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

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

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

Discounted n-gram probability …when the recursion terminates, 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

- Doesn't even try to make the language model a true probability distribution 😌 (so doesn't discount higher-order probabilities)
- If a higher-order n-gram has a zero count, backs off to a lower-order n-gram, weighted by a fixed weight
- $S\left(w_i \middle| w_{i-k+1}^{i-1}\right) = \begin{cases} \dfrac{c(w_{i-k+1}^{i})}{c(w_{i-k+1}^{i-1})} & \text{if } c\left(w_{i-k+1}^{i}\right) > 0 \\ \lambda S\left(w_i \middle| w_{i-k+2}^{i-1}\right) & \text{otherwise} \end{cases}$
  - Terminates in the unigram, which has the probability:
    - $S(w) = \frac{c(w)}{N}$

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

Summary: Language Modeling with N-Grams

- **N-grams:** Sequences of *n* letters
- **Language models:** Statistical models of language based on observed word or character co-occurrences
- N-gram probabilities can be computed using **maximum likelihood estimation**
- Language models can be **intrinsically evaluated** using **perplexity**
- Unknown words can be handled using **<UNK>** tokens
- Known words in unseen contexts can be handled using **smoothing**