language modeling: n-gram models

CS 585, Fall 2019
Introduction to Natural Language Processing
http://people.cs.umass.edu/~miyyer/cs585/

Mohit Iyyer
College of Information and Computer Sciences
University of Massachusetts Amherst

some slides from Dan Jurafsky and Richard Socher
questions from last time

- why am i not on gradescope?
  - please fill out the consent poll on piazza!!!!!!!!!!
goal: assign probability to a piece of text

• why would we ever want to do this?

• translation:
  • $P(\text{i flew to the movies}) \lll P(\text{i went to the movies})$

• speech recognition:
  • $P(\text{i saw a van}) \ggggg P(\text{eyes awe of an})$
You use Language Models every day!
You use Language Models every day!

Google Search

what is the

what is the weather
what is the meaning of life
what is the dark web
what is the xfl
what is the doomsday clock
what is the weather today
what is the keto diet
what is the american dream
what is the speed of light
what is the bill of rights
philosophical question! should building a perfect language model be the ultimate goal of NLP?
I tried to get this point across at my #RELNLP talk by having folks imagine they were given the sum total of all Thai literature in a huge library. (All in Thai, no translations.) Assuming you don't already know Thai, you won't learn it from that.
Emily M. Bender
@emilymbender

Replying to @emilymbender @tallinzen and 3 others

I tried to get this point across at my 
#RELNLP talk by having folks imagine they 
were given the sum total of all Thai literature 
in a huge library. (All in Thai, no translations.) 
Assuming you don't already know Thai, you 
won't learn it from that.

2:20 PM - 30 Jul 2018

1 Retweet 7 Likes

why not? what can’t you learn?
Someone came up to me afterwards and asked: Really? If you put a kid in a room with just audiobooks, they won't learn the language. My answer: No, no they won't.

They could learn the sound system, and prosody, and maybe do some word segmentation. But again, if the input is form only, then they only thing you can hope to learn is properties of the forms.

The kid might learn a language model though. That is, be able to produce sentences, even generalize to unseen ones. Maybe even reasonably complete sentences posed by others. (Need to be a really robotic kid though.)
Ok, how's this: Give your NN all well-formed java code that's ever been written but only the surface form of the code. Then ask it to evaluate (i.e. execute) part of it.
Jacob Andreas @jacobandreas · Aug 2

Replying to @emilymbender @gneubig and 5 others

Thanks, this is clarifying. I believe it's the case that: (1) You cannot possibly extract from the LM any bytecode written according to the Java standard; you certainly cannot execute anything extracted on the Java Virtual Machine without cheating.

Jacob Andreas @jacobandreas · Aug 2

But (2) at the same time, there must be an implementation of the JVM inside the weights of the LM: a model can't generate all code with well-formed tests unless it has the ability to execute representations of code.

Jacob Andreas @jacobandreas · Aug 2

In case (2) there will be an isomorphism between compiled Java and parts of the LM state, and an isomorphism between JVM state and different parts of the LM state. (I think this could be made precise---witnesses to non-isomorphism would produce bad Java when decoded.)
Yeah it’s just (one of the many) trivial examples where perfect LM entails solving not only all of language but also all of AI. Which I guess is why all the AGI crowd are so excited about it.
Probabilistic Language Modeling

• Goal: 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:
  \[ P(W) \text{ or } P(w_n | w_1, w_2 \ldots w_{n-1}) \] is called a language model or LM

we have already seen one way to do this… where?
How to compute $P(W)$

• How to compute this joint probability:
  
  • $P(\text{its, water, is, so, transparent, that})$

• Intuition: let’s rely on the Chain Rule of Probability
Reminder: The Chain Rule

• Recall the definition of conditional probabilities
  \[ P(B \mid A) = \frac{P(A,B)}{P(A)} \quad \text{Rewriting: } \quad P(A,B) = P(A)P(B \mid A) \]

• More variables:
  \[ P(A,B,C,D) = P(A)P(B \mid A)P(C \mid A,B)P(D \mid A,B,C) \]

• The Chain Rule in General
  \[ P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1,x_2)\ldots P(x_n \mid x_1,\ldots,x_{n-1}) \]
The Chain Rule applied to compute joint probability of words in sentence

\[ P(w_1w_2\ldots w_n) = \prod_{i} P(w_i \mid w_1w_2\ldots w_{i-1}) \]

\[
P(\text{“its water is so transparent”}) = \]
\[
P(\text{its}) \times P(\text{water} \mid \text{its}) \times P(\text{is} \mid \text{its water}) \times P(\text{so} \mid \text{its water is}) \times P(\text{transparent} \mid \text{its water is so})\]
How to estimate these probabilities

• Could we just count and divide?

\[
P(\text{the} \mid \text{its water is so transparent that}) = \frac{\text{Count}(\text{its water is so transparent that the})}{\text{Count}(\text{its water is so transparent that})}
\]
How to estimate these probabilities

• Could we just count and divide?

\[ P(\text{the | its water is so transparent that}) = \frac{\text{Count(its water is so transparent that the)}}{\text{Count(its water is so transparent that)}} \]

• No! Too many possible sentences!
• We’ll never see enough data for estimating these
Markov Assumption

• Simplifying assumption:

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

• Or maybe

\[ P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that}) \]
Markov Assumption

\[ P(w_1w_2\ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1}) \]

- In other words, we approximate each component in the product

\[ P(w_i \mid w_1w_2\ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1}) \]
Simplest case: Unigram model

\[ P(w_1w_2\ldots w_n) \approx \prod_i P(w_i) \]

Some automatically generated sentences from a unigram model:

fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
Approximating Shakespeare

| 1 gram | –To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have |
| 2 gram | –Hill he late speaks; or! a more to leg less first you enter |
| 3 gram | –Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. |
| 4 gram | –What means, sir. I confess she? then all sorts, he is trim, captain. |
|       | –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. |
|       | –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. |

The longer the context on which we train the model, the more coherent the sentences. In the unigram sentences, there is no coherent relation between words or any sentence-final punctuation. The bigram sentences have some local word-to-word coherence (especially if we consider that punctuation counts as a word). The trigram and 4-gram sentences are beginning to look a lot like Shakespeare. Indeed, a careful investigation of the 4-gram sentences shows that they look a little too much like Shakespeare. The words “It cannot be but so” are directly from “King John”. This is because, not to put the knock on Shakespeare, his oeuvre is not very large as corpora go (\(N \approx 884,647, V \approx 29,066\)), and our N-gram probability matrices are ridiculously sparse. There are \(V^2 \approx 844,000,000\) possible bigrams alone, and the number of possible 4-grams is \(V^4 \approx 7 \times 10^{17}\). Thus, once the generator has chosen the first 4-gram (“It cannot be but”), there are only five possible continuations (“that”, “I”, “he”, “thou”, and “so”); indeed, for many 4-grams, there is only one continuation.

To get an idea of the dependence of a grammar on its training set, let’s look at an N-gram grammar trained on a completely different corpus: the Wall Street Journal (WSJ) newspaper. Shakespeare and the Wall Street Journal are both English, so we might expect some overlap between our N-grams for the two genres. Fig. 4.4 shows sentences generated by unigram, bigram, and trigram grammars trained on 40 million words from WSJ.

Compare these examples to the pseudo-Shakespeare in Fig. 4.3. While superficially they both seem to model “English-like sentences”, there is obviously no over-
N-gram models

• We can extend to trigrams, 4-grams, 5-grams
• In general this is an insufficient model of language
  • because language has long-distance dependencies:
    “The computer which I had just put into the machine room on the fifth floor crashed.”
• But we can often get away with N-gram models

in the coming lectures, we will look at some models that can theoretically handle some of these longer-term dependencies
Estimating bigram probabilities

- The Maximum Likelihood Estimate (MLE)
  - relative frequency based on the empirical counts on a training set

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

\[ P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]
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>

\[ P(\text{I} \mid <s>) = \frac{2}{3} = 0.67 \]
\[ P(\text{Sam} \mid <s>) = ?? \]
\[ P(</s> \mid \text{Sam}) = \frac{1}{2} = 0.5 \]
\[ P(\text{Sam} \mid \text{am}) = ?? \]
An example

\[
P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}
\]

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

\[
P(I | <s>) = \frac{2}{3} = .67 \quad P(Sam | <s>) = \frac{1}{3} = .33 \quad P(am | I) = \frac{2}{3} = .67
\]

\[
P(<s> | Sam) = \frac{1}{2} = 0.5 \quad P(Sam | am) = \frac{1}{2} = .5 \quad P(do | I) = \frac{1}{3} = .33
\]
A bigger example:
Berkeley Restaurant Project sentences

• can you tell me about any good cantonese restaurants close by
• mid priced thai food is what i’m looking for
• tell me about chez panisse
• can you give me a listing of the kinds of food that are available
• i’m looking for a good place to eat breakfast
• when is caffe venezia open during the day
Raw bigram counts

- Out of 9222 sentences

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>5</td>
<td>827</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>want</td>
<td>2</td>
<td>0</td>
<td>608</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>686</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>211</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Raw bigram probabilities

\[ P(w_i | w_{i-1})^{\text{MLE}} = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]

- Normalize by unigrams:

  \[ P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]

- Result:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>want</td>
<td>0.002</td>
<td>0.33</td>
<td>0.66</td>
<td>0.0036</td>
<td>0.0011</td>
<td>0.28</td>
<td>0</td>
<td>0.00079</td>
</tr>
<tr>
<td>to</td>
<td>0.0022</td>
<td>0</td>
<td>0.0017</td>
<td>0.0065</td>
<td>0.0065</td>
<td>0.0054</td>
<td>0.0011</td>
<td>0.087</td>
</tr>
<tr>
<td>eat</td>
<td>0.00083</td>
<td>0</td>
<td>0.0027</td>
<td>0</td>
<td>0</td>
<td>0.056</td>
<td>0.0025</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0063</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0063</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>0.014</td>
<td>0</td>
<td>0.014</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0059</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Bigram estimates of sentence probabilities

\[
P(<s> \text{ I want english food } </s>) = \]
\[
P(I|<s>) \times P(\text{want}|I) \times P(\text{english}|\text{want}) \times P(\text{food}|\text{english}) \times P(</s>|\text{food})
\]
\[
= .000031
\]

these probabilities get super tiny when we have longer inputs w/ more infrequent words… how can we get around this?
What kinds of knowledge?

• $P(\text{english} \mid \text{want}) = 0.0011$
• $P(\text{chinese} \mid \text{want}) = 0.0065$
• $P(\text{to} \mid \text{want}) = 0.66$
• $P(\text{eat} \mid \text{to}) = 0.28$
• $P(\text{food} \mid \text{to}) = 0$
• $P(\text{want} \mid \text{spend}) = 0$
• $P(\text{i} \mid <s>) = 0.25$
Language Modeling Toolkits

• SRILM
  • http://www.speech.sri.com/projects/srilm/

• KenLM
  • https://kheafield.com/code/kenlm/
Evaluation: How good is our model?

• Does our language model prefer good sentences to bad ones?
  • Assign higher probability to “real” or “frequently observed” sentences
    • Than “ungrammatical” or “rarely observed” sentences?
• We train parameters of our model on a training set.
• We test the model’s performance on data we haven’t seen.
  • A test set is an unseen dataset that is different from our training set, totally unused.
  • An evaluation metric tells us how well our model does on the test set.
Evaluation: How good is our model?

• The goal isn’t to pound out fake sentences!
  • Obviously, generated sentences get “better” as we increase the model order
  • More precisely: using maximum likelihood estimators, higher order is always better likelihood on training set, but not test set
Training on the test set

• We can’t allow test sentences into the training set
• We will assign it an artificially high probability when we set it in the test set
• “Training on the test set”
• Bad science!
• And violates the honor code
Intuition of Perplexity

• The Shannon Game:
  • How well can we predict the next word?
    I always order pizza with cheese and ____
  • The 33rd President of the US was ____
  • I saw a ____
  • Unigrams are terrible at this game. (Why?)

• A better model of a text
  • is one which assigns a higher probability to the word that actually occurs
  • compute per word log likelihood
    \( M \) words, \( m \) test sentence \( s_i \)

\[
l = \frac{1}{M} \sum_{i=1}^{m} \log p(s_i)
\]
Perplexity

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

- Gives the highest P(sentence)

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

$$PP(W) = P(w_1w_2...w_N)^{-\frac{1}{N}}$$

Chain rule:

For bigrams:

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

Minimizing perplexity is the same as maximizing probability
Perplexity as branching factor

Let’s suppose a sentence consisting of random digits
What is the perplexity of this sentence according to a model that assign \( P = \frac{1}{10} \) to each digit?

\[
\begin{align*}
PP(W) &= P(w_1w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \left( \frac{1}{10} \right)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= \frac{1}{10} \\
&= 10
\end{align*}
\]
Lower perplexity = better model

- Training 38 million words, test 1.5 million words, Wall Street Journal

<table>
<thead>
<tr>
<th>N-gram Order</th>
<th>Unigram</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perplexity</td>
<td>962</td>
<td>170</td>
<td>109</td>
</tr>
</tbody>
</table>
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)!

\[
PP(W) = P(w_1w_2...w_N)^\frac{1}{N}
\]

\[
= \sqrt[N]{\frac{1}{P(w_1w_2...w_N)}}
\]

Q: How do we deal with ngrams of zero probabilities?
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
Zeros

Training set:
... denied the allegations
... denied the reports
... denied the claims
... denied the request

Test set
... denied the offer
... denied the loan

\[ P(\text{“offer”} \mid \text{denied the}) = 0 \]
The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:
  \[ P(w \mid \text{denied the}) \]
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total

- Steal probability mass to generalize better
  \[ P(w \mid \text{denied the}) \]
  - 2.5 allegations
  - 1.5 reports
  - 0.5 claims
  - 0.5 request
  - 2 other
  - 7 total
Add-one estimation (again!)

- Also called Laplace smoothing
- 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:
\[
P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}
\]
### Berkeley Restaurant Corpus: Laplace smoothed bigram counts

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
<td>828</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>want</td>
<td>3</td>
<td>1</td>
<td>609</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>687</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>212</td>
</tr>
<tr>
<td>eat</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>17</td>
<td>3</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>chinese</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>83</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>food</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>lunch</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>spend</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Laplace-smoothed bigrams

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

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0.0015</td>
<td>0.21</td>
<td>0.00025</td>
<td>0.0025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00075</td>
</tr>
<tr>
<td>want</td>
<td>0.0013</td>
<td>0.00042</td>
<td>0.26</td>
<td>0.00084</td>
<td>0.0029</td>
<td>0.0029</td>
<td>0.0025</td>
<td>0.00084</td>
</tr>
<tr>
<td>to</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0013</td>
<td>0.18</td>
<td>0.00078</td>
<td>0.00026</td>
<td>0.0018</td>
<td>0.055</td>
</tr>
<tr>
<td>eat</td>
<td>0.00046</td>
<td>0.00046</td>
<td>0.0014</td>
<td>0.00046</td>
<td>0.0078</td>
<td>0.0014</td>
<td>0.02</td>
<td>0.00046</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0012</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.052</td>
<td>0.0012</td>
<td>0.00062</td>
</tr>
<tr>
<td>food</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.0063</td>
<td>0.00039</td>
<td>0.00079</td>
<td>0.002</td>
<td>0.00039</td>
<td>0.00039</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0017</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.0011</td>
<td>0.00056</td>
<td>0.00056</td>
</tr>
<tr>
<td>spend</td>
<td>0.0012</td>
<td>0.00058</td>
<td>0.0012</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
<td>0.00058</td>
</tr>
</tbody>
</table>
Reconstituted counts

\[ c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V} \]

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>3.8</td>
<td>527</td>
<td>0.64</td>
<td>6.4</td>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
<td>1.9</td>
</tr>
<tr>
<td>want</td>
<td>1.2</td>
<td>0.39</td>
<td>238</td>
<td>0.78</td>
<td>2.7</td>
<td>2.7</td>
<td>2.3</td>
<td>0.78</td>
</tr>
<tr>
<td>to</td>
<td>1.9</td>
<td>0.63</td>
<td>3.1</td>
<td>430</td>
<td>1.9</td>
<td>0.63</td>
<td>4.4</td>
<td>133</td>
</tr>
<tr>
<td>eat</td>
<td>0.34</td>
<td>0.34</td>
<td>1</td>
<td>0.34</td>
<td>5.8</td>
<td>1</td>
<td>15</td>
<td>0.34</td>
</tr>
<tr>
<td>chinese</td>
<td>0.2</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>8.2</td>
<td>0.2</td>
<td>0.098</td>
</tr>
<tr>
<td>food</td>
<td>6.9</td>
<td>0.43</td>
<td>6.9</td>
<td>0.43</td>
<td>0.86</td>
<td>2.2</td>
<td>0.43</td>
<td>0.43</td>
</tr>
<tr>
<td>lunch</td>
<td>0.57</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.38</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>spend</td>
<td>0.32</td>
<td>0.16</td>
<td>0.32</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
</tr>
</tbody>
</table>
## Compare with raw bigram counts

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>5</td>
<td>827</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>want</td>
<td>2</td>
<td>0</td>
<td>608</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>686</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>211</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>3.8</td>
<td>527</td>
<td>0.64</td>
<td>6.4</td>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
<td>1.9</td>
</tr>
<tr>
<td>want</td>
<td>1.2</td>
<td>0.39</td>
<td>238</td>
<td>0.78</td>
<td>2.7</td>
<td>2.7</td>
<td>2.3</td>
<td>0.78</td>
</tr>
<tr>
<td>to</td>
<td>1.9</td>
<td>0.63</td>
<td>3.1</td>
<td>430</td>
<td>1.9</td>
<td>0.63</td>
<td>4.4</td>
<td>133</td>
</tr>
<tr>
<td>eat</td>
<td>0.34</td>
<td>0.34</td>
<td>1</td>
<td>0.34</td>
<td>5.8</td>
<td>1</td>
<td>15</td>
<td>0.34</td>
</tr>
<tr>
<td>chinese</td>
<td>0.2</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>0.098</td>
<td>8.2</td>
<td>0.2</td>
<td>0.098</td>
</tr>
<tr>
<td>food</td>
<td>6.9</td>
<td>0.43</td>
<td>6.9</td>
<td>0.43</td>
<td>0.86</td>
<td>2.2</td>
<td>0.43</td>
<td>0.43</td>
</tr>
<tr>
<td>lunch</td>
<td>0.57</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.38</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>spend</td>
<td>0.32</td>
<td>0.16</td>
<td>0.32</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
</tr>
</tbody>
</table>

48
Add-1 estimation is a blunt instrument

- So add-1 isn’t used for N-grams:
  - We’ll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn’t so huge.
Backoff and Interpolation

- 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
Linear Interpolation

• Simple interpolation

\[
\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)
\]

\[\sum_i \lambda_i = 1\]

• Lambdas conditional on context:

\[
\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 (w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) + \lambda_2 (w_{n-2}^{n-1}) P(w_n|w_{n-1}) + \lambda_3 (w_{n-2}^{n-1}) P(w_n)
\]
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
- Divide up 22 million words of AP Newswire
  - Training and held-out set
  - for each bigram in the training set
  - see the actual count in the held-out set!

<table>
<thead>
<tr>
<th>Bigram count in training</th>
<th>Bigram count in heldout set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.0000270</td>
</tr>
<tr>
<td>1</td>
<td>0.448</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
</tr>
<tr>
<td>6</td>
<td>5.23</td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>7.21</td>
</tr>
<tr>
<td>9</td>
<td>8.26</td>
</tr>
</tbody>
</table>
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
- Divide up 22 million words of AP Newswire
  - Training and held-out set
  - for each bigram in the training set
  - see the actual count in the held-out set!

<table>
<thead>
<tr>
<th>Bigram count in training</th>
<th>Bigram count in heldout set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.0000270</td>
</tr>
<tr>
<td>1</td>
<td>0.448</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
</tr>
<tr>
<td>6</td>
<td>5.23</td>
</tr>
<tr>
<td>7</td>
<td>6.21</td>
</tr>
<tr>
<td>8</td>
<td>7.21</td>
</tr>
<tr>
<td>9</td>
<td>8.26</td>
</tr>
</tbody>
</table>

Why do you think the training and heldout counts differ?
Absolute Discounting Interpolation

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

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

- (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 Intuition

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can’t see without my reading*?
  - “Francisco” is more common than “glasses”
  - ... but “Francisco” always follows “San”
- The unigram is useful exactly when we haven’t seen this bigram!
- Instead of $P(w)$: “How likely is $w$”
- $P_{\text{continuation}}(w)$: “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

\[
P_{\text{CONTINUATION}}(w) \propto \left| \{w_{i-1} : c(w_{i-1}, w) > 0\} \right|
\]
exercise!