Lecture 5: Language modeling
Announcements

- Recitation this week will be a P1 Q&A with Han
- P1 bakeoff: No pre-trained word embeddings or BERT. Out-of-the box linguistic annotations (e.g. pos tags), tf-idf, etc. are fine.
Probabilistic language models
Probabilistic language models

- Today’s goal: assign a probability to a sentence. Why?
Probabilistic language models

- Today’s goal: assign a probability to a sentence. Why?
  - Machine translation:
    \[ P(\text{high winds tonight}) > P(\text{large winds tonight}) \]
Today's goal: assign a probability to a sentence. Why?

- Machine translation:
  \[ P(\text{high winds tonight}) > P(\text{large winds tonight}) \]

- Spelling correction:
  \[ P(\text{I’ll be five minutes late}) > P(\text{I’ll be five minuets late}) \]
Probabilistic language models

Today’s goal: assign a probability to a sentence. Why?

- Machine translation:
  \[ P(\text{high winds tonight}) > P(\text{large winds tonight}) \]

- Spelling correction:
  \[ P(\text{I’ll be five minutes late}) > P(\text{I’ll be five minuets late}) \]

- Speech recognition:
  \[ P(\text{I saw a van}) > P(\text{eyes awe of an}) \]
Probabilistic language models

Today’s goal: assign a probability to a sentence. Why?

- Machine translation:
  \[ P(\text{high winds tonight}) > P(\text{large winds tonight}) \]

- Spelling correction:
  \[ P(\text{I’ll be five minutes late}) > P(\text{I’ll be five minuets late}) \]

- Speech recognition:
  \[ P(\text{I saw a van}) > P(\text{eyes awe of an}) \]

- Summarization, question answering, …
Probabilistic language models

- Today's goal: assign a probability to a sentence. Why?

Machine translation:
\[ P(\text{high winds tonight}) > P(\text{large winds tonight}) \]

Spelling correction:
\[ P(\text{I'll be five minutes late}) > P(\text{I'll be five minuets late}) \]

Speech recognition:
\[ P(\text{I saw a van}) > P(\text{eyes awe of an}) \]

Summarization, question answering, …
Probabilistic language models

Today’s goal: assign a probability to a sentence. Why?

- Machine translation: \( P(\text{high winds tonight}) > P(\text{large winds tonight}) \)
- Spelling correction: \( P(\text{I’ll be five minutes late}) > P(\text{I’ll be five minuets late}) \)
- Speech recognition: \( P(\text{I saw a van}) > P(\text{eyes awe of an}) \)
- Summarization, question answering, …
Probabilistic language models

Today's goal: assign a probability to a sentence. Why?

- Machine translation: $P(\text{high winds tonight}) > P(\text{large winds tonight})$
- Spelling correction: $P(\text{I'll be five minutes late}) > P(\text{I'll be five minuets late})$
- Speech recognition: $P(\text{I saw a van}) > P(\text{eyes awe of an})$
- Summarization, question answering, …
Probabilistic language models

- Today’s goal: assign a probability to a sentence. Why?

  - Machine translation: $P(\text{high winds tonight}) > P(\text{large winds tonight})$
  - Spelling correction: $P(\text{I’ll be five minutes late}) > P(\text{I’ll be five minuets late})$
  - Speech recognition: $P(\text{I saw a van}) > P(\text{eyes awe of an})$
  - Summarization, question answering, …

Kannan et al. 2016
Today's goal: assign a probability to a sentence. Why?

- Machine translation: $P(\text{high winds tonight}) > P(\text{large winds tonight})$
- Spelling correction: $P(\text{I'll be five minutes late}) > P(\text{I'll be five minuets late})$
- Speech recognition: $P(\text{I saw a van}) > P(\text{eyes awe of an})$
- Summarization, question answering, …
- and since ~2018, state-of-the-art vector representations of text.
Probabilistic language models
Probabilistic language models

Goal: compute the probability of a sentence (or sequence of words):

$$P(w) = P(w_1, w_2, w_3, ..., w_n)$$
Probabilistic language models

- Goal: compute the probability of a sentence (or sequence of words):
  \[ P(w) = P(w_1, w_2, w_3, \ldots, w_n) \]
- Related task: probability of the next word:
  \[ P(w_5 \mid w_4, w_3, w_2, w_1) \]
Probabilistic language models

- Goal: compute the probability of a sentence (or sequence of words):
  \[ P(w) = P(w_1, w_2, w_3, \ldots, w_n) \]

- Related task: probability of the next word:
  \[ P(w_5 \mid w_4, w_3, w_2, w_1) \]

- A model that computes either of these is called a language model (or LM).
How to compute $P(w)$?
How to compute $P(w)$?

- Want to compute the joint probability:

$$P(its, \text{ water, is, so, transparent, that})$$
How to compute $P(w)$?

- Want to compute the joint probability:

$$P(its,\ water,\ is,\ so,\ transparent,\ that)$$

- Intuition: use chain rule of probability

$$P(B \mid A) = \frac{P(A, B)}{P(A)} \quad \Rightarrow \quad P(A, B) = P(A)P(B \mid A)$$
How to compute $P(w)$?

- Want to compute the joint probability:

\[ P(\text{its}, \text{water}, \text{is}, \text{so}, \text{transparent}, \text{that}) \]

- Intuition: use chain rule of probability

\[
P(B \mid A) = \frac{P(A, B)}{P(A)} \quad \rightarrow \quad P(A, B) = P(A)P(B \mid A)
\]

- Chain rule in general:

\[
P(x_1, x_2, x_3, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)\ldots P(x_n \mid x_1, \ldots, x_{n-1})
\]
How to compute $P(w)$?
How to compute $P(w)$?

- Want to compute the joint probability:

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod_{i} P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$
How to compute $P(w)$?

- Want to compute the joint probability:

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod_{i} P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

- For example:

$$P(\text{its, water, is, so, transparent, that}) =$$

$$P(\text{its}) \cdot P(\text{water} \mid \text{its}) \cdot P(\text{is} \mid \text{its water})$$

$$\cdot P(\text{so} \mid \text{its water is}) \cdot P(\text{transparent} \mid \text{its water is so})$$
How to estimate $P(w_i)$?
How to estimate $P(w_i)$?

- Can we just count and divide (relative frequency estimate)?
How to estimate $P(w_i)$?

- Can we just count and divide (relative frequency estimate)?

$$P\left(\text{the} \mid \text{its water is so transparent that} \right) = \frac{\text{count(its water is so transparent that the)}}{\text{count(its water is so transparent that)}}$$
How to estimate $P(w_i)$?

- Can we just count and divide (relative frequency estimate)?

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

- No! Too many (infinitely) possible sentences.
How to estimate $P(w_i)$?

- Can we just count and divide (relative frequency estimate)?

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

- No! Too many (infinitely) possible sentences.

- Too sparse: we’ll never observe enough data to estimate.
Markov assumption
Markov assumption

- Make the simplifying assumption:

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

- Make the 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

- Make the 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**: conditional probability distribution of future states (words) depends only on the present state, not the sequence of events that preceded it.
Markov assumption

**Markov assumption**: conditional probability distribution of future states (words) depends only on the present state, not the sequence of events that preceded it.

\[
P(w_1, w_2, w_3, \ldots, w_n) \approx \prod_{i} P(w_i \mid w_{i-1}, \ldots, w_{i-k})
\]
Markov assumption

Markov assumption: conditional probability distribution of future states (words) depends only on the present state, not the sequence of events that preceded it.

\[ P(w_1, w_2, w_3, \ldots, w_n) \approx \prod_i P(w_i \mid w_{i-1}, \ldots, w_{i-k}) \]

In other words, approximate each component of the product:

\[ P(w_i \mid w_{i-1}, \ldots, w_2, w_1) \approx P(w_i \mid w_{i-1}, \ldots, w_{i-k}) \]
Simplest case: Unigram model
Simplest case: Unigram model

- Unigram model:

\[ P(w_1, w_2, \ldots, w_n) \approx \prod_i P(w_i) \]
Simplest case: Unigram model

- **Unigram model:**

\[ P(w_1, w_2, ..., w_n) \approx \prod_i P(w_i) \]

- Example sentences generated by a unigram model trained on financial news:

  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
Bigram model
Bigram model

- **Bigram model**: condition on the previous word

\[ P(w_i \mid w_{i-1}, \ldots, w_2, w_1) \approx P(w_i \mid w_{i-1}) \]
Bigram model

- **Bigram model**: condition on the previous word

\[ P(w_i \mid w_{i-1}, \ldots, w_2, w_1) \approx P(w_i \mid w_{i-1}) \]

- Example sentences generated by a bigram model trained on financial news:

  texaco rose one in this issue is pursuing growth in a boiler house said mr. gurria
  mexico ’s motion control proposal without permission from five hundred fifty five yen
  outside new car parking lot of the agreement reached
  this would be a record november
N-gram models
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, ...
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, ...

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

Figure 3.3 Eight sentences randomly generated from four n-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

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 ($N = 884,647$, $V = 29,066$), and our n-gram probability matrices are ridiculously sparse. There are $V^2 = 844,000,000$ possible bigrams alone, and the number of possible 4-grams is $V^4 = 7 \times 10^{17}$. 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. 3.4
N-gram models
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, ...
- In general, this is an insufficient model of language.
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …
- In general, this is an insufficient model of language.
  - Language has long-distance dependencies, e.g.:
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, ...
- In general, this is an insufficient model of language.
  - Language has **long-distance dependencies**, e.g.:

  The **computer(s)** that I just put into the machine room on the fifth floor **is (are)** crashing.
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, ...
- In general, this is an insufficient model of language.
  - Language has long-distance dependencies, e.g.:

  The **computer(s)** that I just put into the machine room on the fifth floor **is (are)** crashing.
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, …
- In general, this is an insufficient model of language.
  - Language has **long-distance dependencies**, e.g.:
    - But, n-grams require accounting for $V^n$ events. $V=10^4$, $n=7 \rightarrow 10^{28}$ events.

The computer(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

subject-verb agreement?
N-gram models

- Can extend to trigrams, 4-grams, 5-grams, ...
- In general, this is an insufficient model of language.
  - Language has **long-distance dependencies**, e.g.:

    subject-verb agreement?

    The **computer(s)** that I just put into the machine room on the fifth floor **is (are)** crashing.

    - But, n-grams require accounting for $V^n$ events. $V=10^4$, $n=7 \rightarrow 10^{28}$ events.

- Another example of **bias-variance tradeoff**
Estimating bigram probabilities
Estimating bigram probabilities

The Maximum Likelihood Estimate:

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

- The Maximum Likelihood Estimate:

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

- Example:

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

- The Maximum Likelihood Estimate:

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

- Example:

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

$$P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67$$
Estimating bigram probabilities

The Maximum Likelihood Estimate:

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

Example:

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

\[
P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67
\]

\[
P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5
\]
Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

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

\[ P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67 \]
\[ P(Sam \mid \langle s \rangle) = \frac{1}{3} = ??? \]
\[ P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5 \]
Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

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

\[
P(I \mid \langle s\rangle) = \frac{2}{3} = 0.67\quad P(Sam \mid \langle s\rangle) = \frac{1}{3} = ???
\]

\[
P(\langle /s\rangle \mid Sam) = \frac{1}{2} = 0.5\quad P(Sam \mid am) = \frac{1}{2} = ???
\]
Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

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

\[
P(I \mid <s>) = \frac{2}{3} = 0.67 \quad P(Sam \mid <s>) = \frac{1}{3} = 0.33
\]

\[
P(<s> \mid Sam) = \frac{1}{2} = 0.5 \quad P(Sam \mid am) = \frac{1}{2} = ???
\]
Estimating bigram probabilities

- The Maximum Likelihood Estimate:
  
  \[ P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})} = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]

- Example:

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

  \[
  P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67 \quad P(Sam \mid \langle s \rangle) = \frac{1}{3} = 0.33
  \]
  \[
  P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5 \quad P(Sam \mid am) = \frac{1}{2} = 0.5
  \]
Estimating bigram probabilities

- The Maximum Likelihood Estimate:

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

- Example:

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

\[
P(I \mid \langle s \rangle) = \frac{2}{3} = 0.67 \quad P(Sam \mid \langle s \rangle) = \frac{1}{3} = 0.33 \quad P(do \mid I) = \frac{1}{3} = 0.33
\]

\[
P(\langle /s \rangle \mid Sam) = \frac{1}{2} = 0.5 \quad P(Sam \mid am) = \frac{1}{2} = 0.5 \quad P(am \mid I) = \frac{2}{3} = 0.67
\]
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 venecia open during the day
## 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>
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

<table>
<thead>
<tr>
<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>2533</td>
<td>927</td>
<td>2417</td>
<td>746</td>
<td>158</td>
<td>1093</td>
<td>341</td>
<td>278</td>
</tr>
</tbody>
</table>
Raw bigram probabilities

- Normalize by unigrams:

<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></td>
<td>2533</td>
<td>927</td>
<td>2417</td>
<td>746</td>
<td>158</td>
<td>1093</td>
<td>341</td>
<td>278</td>
</tr>
</tbody>
</table>
Raw bigram probabilities

- Normalize by unigrams:

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

<table>
<thead>
<tr>
<th></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>2533</td>
<td>927</td>
<td>2417</td>
<td>746</td>
<td>158</td>
<td>1093</td>
<td>341</td>
</tr>
<tr>
<td></td>
<td>278</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Raw bigram probabilities

Normalize by unigrams:

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

<table>
<thead>
<tr>
<th></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>2533</td>
<td>927</td>
<td>2417</td>
<td>746</td>
<td>158</td>
<td>1093</td>
<td>341</td>
</tr>
</tbody>
</table>

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>i</td>
<td>0.002</td>
<td>0.33</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00079</td>
</tr>
<tr>
<td>want</td>
<td>0.0022</td>
<td>0</td>
<td>0.66</td>
<td>0.0011</td>
<td>0.0065</td>
<td>0.0065</td>
<td>0.0054</td>
<td>0.0011</td>
</tr>
<tr>
<td>to</td>
<td>0.00083</td>
<td>0</td>
<td>0.0017</td>
<td>0.28</td>
<td>0.00083</td>
<td>0</td>
<td>0.0025</td>
<td>0.087</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>0.0027</td>
<td>0</td>
<td>0.021</td>
<td>0.0027</td>
<td>0.056</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>0.52</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>0.00092</td>
<td>0.0037</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0059</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0029</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>0.0036</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Bigram estimates of sentence probabilities

\[
P(\langle s \rangle \text{ I want english food } \langle /s \rangle) = P(I \mid \langle s \rangle) \]
\[
\times P(\text{want} \mid I) \]
\[
\times P(\text{english} \mid \text{want}) \]
\[
\times P(\text{food} \mid \text{English}) \]
\[
\times P(\langle /s \rangle \mid \text{food}) \]
\[
= 0.000031
\]
Bigram estimates of sentence probabilities

\[ P(\langle s \rangle \ I \ \text{want english food} \ \langle /s \rangle) = P(I \ | \ \langle s \rangle) \]

\[ \times P(\text{want} \ | \ I) \]

\[ \times P(\text{english} \ | \ \text{want}) \]

\[ \times P(\text{food} \ | \ \text{English}) \]

\[ \times P(\langle /s \rangle \ | \ \text{food}) \]

\[ = 0.000031 \]

- In practice:

\[ \log P(I \ | \ \langle s \rangle) + \log P(\text{want} \ | \ I) + \log P(\text{english} \ | \ \text{want}) + ... \]
What kinds of knowledge?

- $P(\text{english} \mid \text{want}) = 0.0011$
- $P(\text{chinese} \mid \text{want}) = 0.0065$
- $P(\text{to} \mid \text{want}) = 0.66$
- $P(\text{eat} \mid \text{to}) = 0.28$
- $P(\text{food} \mid \text{to}) = 0$
- $P(\text{want} \mid \text{spend}) = 0$
- $P(i \mid <s>) = 0.25$
What kinds of knowledge?

- $P(\text{english} \mid \text{want}) = 0.0011$
- $P(\text{chinese} \mid \text{want}) = 0.0065$
- $P(\text{to} \mid \text{want}) = 0.66$
- $P(\text{eat} \mid \text{to}) = 0.28$
- $P(\text{food} \mid \text{to}) = 0$
- $P(\text{want} \mid \text{spend}) = 0$
- $P(i \mid <s>) = 0.25$
What kinds of knowledge?

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

world knowledge

grammar (infinitive verb)
What kinds of knowledge?

- $P(\text{english} \mid \text{want}) = 0.0011$
  - world knowledge

- $P(\text{chinese} \mid \text{want}) = 0.0065$

- $P(\text{to} \mid \text{want}) = 0.66$
  - grammar (infinitive verb)

- $P(\text{eat} \mid \text{to}) = 0.28$

- $P(\text{food} \mid \text{to}) = 0$
  - ungrammatical

- $P(\text{want} \mid \text{spend}) = 0$

- $P(i \mid <s>) = 0.25$
What kinds of knowledge?

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

*world knowledge*

*grammar (infinitive verb)*

*ungrammatical*

*people like to talk about themselves*
Evaluating language models
Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
  - Time consuming and hard
Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
  - Time consuming and hard

- **Intrinsic evaluations** test whether our model prefers “good” sentences to “bad” ones:
Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
  - Time consuming and hard

- **Intrinsic evaluations** test whether our model prefers “good” sentences to “bad” ones:
  - Does it assign higher probability to “real” or “frequently observed” sentences compared to “ungrammatical” or “rarely observed” sentences?
Evaluating language models

- **Extrinsic evaluations** test whether the model is useful for downstream tasks:
  - Spelling correction, speech recognition, machine translation, …
  - Time consuming and hard

- **Intrinsic evaluations** test whether our model prefers “good” sentences to “bad” ones:
  - Does it assign higher probability to “real” or “frequently observed” sentences compared to “ungrammatical” or “rarely observed” sentences?
  - **Perplexity**: A bad approximation for future performance.
Perplexity: Intuition
Perplexity: Intuition

- The Shannon Game: How well can we predict the next word?
Perplexity: Intuition

- The Shannon Game: How well can we predict the next word?

I always order pizza with cheese and ________

The 33rd president of the U.S. was ________

I saw a ________

\[
\begin{align*}
mushrooms & : 0.1 \\
pepperoni & : 0.1 \\
anchovies & : 0.01 \\
\ldots & \\
fried rice & : 0.0001 \\
\ldots & \\
\text{and} & : 1E-100
\end{align*}
\]
Perplexity: Intuition

- The Shannon Game: How well can we predict the next word?

  I always order pizza with cheese and ________

  The 33rd president of the U.S. was ________

  I saw a ________

- Unigrams are terrible at this.

  { mushrooms 0.1, pepperoni 0.1, anchovies 0.01, ...
    fried rice 0.0001, ...
    and 1E-100 }
Perplexity: Intuition

- The Shannon Game: How well can we predict the next word?

  I always order pizza with cheese and ________

  The 33rd president of the U.S. was ________

  I saw a ________

- Unigrams are terrible at this.

- Intuition: a better model of text is one that assigns a higher probability to the word that actually occurs.
Perplexity: Intuition

- The Shannon Game: How well can we predict the next word?

  I always order pizza with cheese and ________

  The 33rd president of the U.S. was ________

  I saw a ________

- Unigrams are terrible at this.

- Intuition: a better model of text is one that assigns a higher probability to the word that actually occurs.

- Compute per-word log likelihood on held out data: 
  \[
  \sum_{m=1}^{M} \log P(w_m | w_{m-1}, \ldots, w_1)
  \]
Perplexity: Intuition

- The Shannon Game: How well can we predict the next word?
  
  I always order pizza with cheese and ________

  The 33rd president of the U.S. was ________

  I saw a ________

- Unigrams are terrible at this.

- Intuition: a better model of text is one that assigns a higher probability to the word that actually occurs.

- Compute per-word log likelihood on held out data:

  \[
  \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)
  \]

  tokens in held out data
Perplexity
Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest $P(sentence)$. 
Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest $P(\text{sentence})$.

- Perplexity: deterministic transformation of the log likelihood into a pleasing information-theoretic value:
**Perplexity**

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest \( P(\text{sentence}) \).

- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

\[
\ell(\mathbf{w}) = \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)
\]
Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest $P(\text{sentence})$.

- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$
\ell(w) = \sum_{m=1}^{M} \log P(w_m \mid w_{m-1}, \ldots, w_1)
$$

$$
\text{ppl}(w) = 2^{-\frac{\ell(w)}{M}}
$$
Perplexity

■ The best language model is one that best predicts an unseen test set, i.e. one that gives the highest $P(\text{sentence})$.

■ Perplexity: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(w) = \sum_{m=1}^{M} \log P(w_m | w_{m-1}, ..., w_1) \quad \text{ppl}(w) = 2^{\frac{\ell(w)}{M}}$$

entropy: avg negative log likelihood per word
Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest $P(\text{sentence})$.

- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(w) = \sum_{m=1}^{M} \log P(w_m | w_{m-1}, \ldots, w_1) \quad \text{ppl}(w) = 2^{-\frac{\ell(w)}{M}}$$

**Entropy**: avg negative log likelihood per word

avg # bits required to encode each word under this model
The best language model is one that best predicts an unseen test set, i.e. one that gives the highest $P(\text{sentence})$.

**Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

$$\ell(w) = \sum_{m=1}^{M} \log P(w_m | w_{m-1}, \ldots, w_1)$$

$$\text{ppl}(w) = 2^{-\frac{\ell(w)}{M}}$$

How confused (*perplexed*) is the model on test data?
Perplexity

- The best language model is one that best predicts an unseen test set, i.e. one that gives the highest $P(\text{sentence})$.

- **Perplexity**: deterministic transformation of the log likelihood into a pleasing information-theoretic value:

  $$\ell(\textbf{w}) = \sum_{m=1}^{M} \log P(w_m | w_{m-1}, \ldots, w_1)$$

  $$\text{ppl}(\textbf{w}) = 2^{\frac{-\ell(\textbf{w})}{M}}$$

- How confused (*perplexed*) is the model on test data?

- Minimizing perplexity is the same as maximizing probability.

**Entropy**: avg negative log likelihood per word  
avg # bits required to encode each word under this model
Lower perplexity = better model
Lower perplexity = better model

- Training 38 million words, test 1.5 million words, financial news (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>

https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word
Lower perplexity = better model

- Training 38 million words, test 1.5 million words, financial news (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>

<table>
<thead>
<tr>
<th>RANK</th>
<th>MODEL</th>
<th>TEST PERPLEXITY</th>
<th>VALIDATION PERPLEXITY</th>
<th>PARAMS</th>
<th>EXTRA TRAINING DATA</th>
<th>PAPER</th>
<th>CODE</th>
<th>RESULT</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPT-3 (Zero-Shot)</td>
<td>20.5</td>
<td>175000M</td>
<td>✓</td>
<td>Language Models are Few-Shot Learners</td>
<td></td>
<td></td>
<td></td>
<td>2020</td>
</tr>
<tr>
<td>2</td>
<td>BERT-Large-CAS</td>
<td>31.3</td>
<td>36.1</td>
<td>X</td>
<td>Language Models with Transformers</td>
<td></td>
<td></td>
<td></td>
<td>2019</td>
</tr>
<tr>
<td>3</td>
<td>GPT-2</td>
<td>35.76</td>
<td>1542M</td>
<td>✓</td>
<td>Language Models are Unsupervised Multitask Learners</td>
<td></td>
<td></td>
<td></td>
<td>2019</td>
</tr>
<tr>
<td>4</td>
<td>Mogrifier LSTM + dynamic eval</td>
<td>44.9</td>
<td>44.8</td>
<td>24M</td>
<td>X</td>
<td>Mogrifier LSTM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word
Lower perplexity = better model?

| 1 gram | Months the my and issue of year foreign new exchange’s september were recession exchange new endorsed a acquire to six executives |
| 2 gram | Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her |
| 3 gram | They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions |
Zero probability bigrams
## Zero probability bigrams

<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>0.002</td>
<td>0.33</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.002</td>
</tr>
<tr>
<td>want</td>
<td>0.0022</td>
<td>0</td>
<td>0.66</td>
<td>0.0011</td>
<td>0.0065</td>
<td>0.0065</td>
<td>0.0054</td>
<td>0.0011</td>
</tr>
<tr>
<td>to</td>
<td>0.00083</td>
<td>0</td>
<td>0.0017</td>
<td>0.28</td>
<td>0.00083</td>
<td>0</td>
<td>0.0025</td>
<td>0.087</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>0.0027</td>
<td>0</td>
<td>0.021</td>
<td>0.0027</td>
<td>0.056</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0063</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.52</td>
<td>0.0063</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>0.014</td>
<td>0</td>
<td>0.014</td>
<td>0</td>
<td>0.00092</td>
<td>0.0037</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0059</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0029</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>0.0036</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Zero probability bigrams
Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)

- Consider the corpus of Shakespeare’s works: $M = 884,647$ tokens; $V = 29,066$ types
Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
- Consider the corpus of Shakespeare's works: \( M = 884,647 \) tokens; \( V = 29,066 \) types
  - 300,000 bigram types out of \( V^2 = 884 \) million possible bigrams
Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
- Consider the corpus of Shakespeare’s works: $M = 884,647$ tokens; $V = 29,066$ types
  - 300,000 bigram types out of $V^2 = 884$ million possible bigrams
  - 99.96% of possible bigrams not observed in corpus (zero bigrams)
Zero probability bigrams

- Bigrams with zero probability mean that we will assign 0 probability to the test set!
  - Cannot compute perplexity (log(0) undefined)
- Consider the corpus of Shakespeare’s works: $M = 884,647$ tokens; $V = 29,066$ types
  - 300,000 bigram types out of $V^2 = 884$ million possible bigrams
  - 99.96% of possible bigrams not observed in corpus (zero bigrams)

**training set:**
...denied the allegations
...denied the reports
...denied the claims
...denied the request

**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!
  - Cannot compute perplexity (log(0) undefined)

- Consider the corpus of Shakespeare’s works: $M = 884,647$ tokens; $V = 29,066$ types
  - 300,000 bigram types out of $V^2 = 884$ million possible bigrams
  - 99.96% of possible bigrams not observed in corpus (zero bigrams)

**training set:**
- ...denied the allegations
- ...denied the reports
- ...denied the claims
- ...denied the request

**test set:**
- ...denied the offer
- ...denied the loan

\[ P(offer \mid denied \ the) = 0 \]
Smoothing!
Smoothing!

- As with counts for text classification, can use smoothing to improve generalization.
Smoothing!

- As with counts for text classification, can use **smoothing** to improve generalization.
- Intuition: steal some probability mass from observed n-grams, distribute the wealth.

Figure from Dan Klein
Smoothing!

- As with counts for text classification, can use **smoothing** to improve generalization.
- Intuition: steal some probability mass from observed n-grams, distribute the wealth.
- Can use add-one (Laplace) smoothing:

\[
P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}
\]
Laplace smoothing for bigrams?

\[ c^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V} \]
Laplace smoothing for bigrams?

- Compare “reconstituted” counts with original:

\[ c^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V} \]
Laplace smoothing for bigrams?

- Compare “reconstituted” counts with original:
  \[ c^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V} \]

<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>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>0</td>
<td>82</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>
Laplace smoothing for bigrams?

- Compare “reconstituted” counts with original: 
\[ c^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V} \]

<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>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>
Laplace smoothing for bigrams?

- Compare “reconstituted” counts with original: 
  \[ c^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V} \]

<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>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>0</td>
<td>82</td>
<td>1</td>
</tr>
<tr>
<td>food</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</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>

discount: \( d_c = \frac{c^*}{c} \)
Laplace smoothing for bigrams?

- Compare “reconstituted” counts with original: 
  \[ c^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V} \]

<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>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>82</td>
<td>1</td>
<td>0</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>1</td>
<td>0</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>

discount: 
\[ d_c = \frac{c^*}{c} \]

\[ d_c(\text{want to}) = 0.39 \]
Laplace smoothing for bigrams?

- Compare “reconstituted” counts with original: 
  \[ c^*(w_{i-1}, w_i) = \frac{(c(w_{i-1}, w_i) + 1) \cdot c(w_{i-1})}{c(w_{i-1}) + V} \]

<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>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>

**discount:** 
\[ d_c = \frac{c^*}{c} \]
\[ d_c(want to) = 0.39 \]
\[ d_c(chinese food) = 0.1 \]
Backoff and interpolation
Backoff and interpolation

- Sometimes it helps to use less context. Intuition: Condition on less context for contexts you haven’t learned much about.
Backoff and interpolation

- Sometimes it helps to use less context. Intuition: Condition on less context for contexts you haven’t learned much about.

- Two ways to use less context (in e.g. trigram model):
Backoff and interpolation

- Sometimes it helps to use less context. Intuition: Condition on less context for contexts you haven’t learned much about.

- Two ways to use less context (in e.g. trigram model):
  
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
Backoff and interpolation

- Sometimes it helps to use less context. Intuition: Condition on less context for contexts you haven’t learned much about.

- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
  - **Interpolation**: Always mix unigram, bigram, trigram statistics.
Backoff and interpolation

- Sometimes it helps to use less context. Intuition: Condition on less context for contexts you haven’t learned much about.

- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
  - **Interpolation**: Always mix unigram, bigram, trigram statistics.

- Interpolation works better
Backoff and interpolation

- Sometimes it helps to use less context. Intuition: Condition on less context for contexts you haven’t learned much about.

- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
  - **Interpolation**: Always mix unigram, bigram, trigram statistics.

- Interpolation works better

- State-of-the-art: Extended interpolated Kneser-Ney smoothing
Linear interpolation
Linear interpolation

Simple interpolation:

\[
\hat{P}(w_i \mid w_{i-1}, w_{i-2}) = \lambda_1 P(w_i \mid w_{i-1}, w_{i-2}) \\
+ \lambda_2 P(w_i \mid w_{i-1}) \\
+ \lambda_3 P(w_i)
\]
Linear interpolation

Simple interpolation:

\[ \hat{P}(w_i \mid w_{i-1}, w_{i-2}) = \lambda_1 P(w_i \mid w_{i-1}, w_{i-2}) \]
\[ + \lambda_2 P(w_i \mid w_{i-1}) \]
\[ + \lambda_3 P(w_i) \]
\[ \sum_i \lambda_i = 1 \]
Linear interpolation

■ Simple interpolation:
\[
\hat{P}(w_i | w_{i-1}, w_{i-2}) = \lambda_1 P(w_i | w_{i-1}, w_{i-2}) + \lambda_2 P(w_i | w_{i-1}) + \lambda_3 P(w_i)
\]
\[\sum_i \lambda_i = 1\]

■ Better: \(\lambda_i\) depends on specific context
\[
\hat{P}(w_i | w_{i-1}, w_{i-2}) = \lambda_1 (w_{i-2}) P(w_i | w_{i-1}, w_{i-2}) + \lambda_2 (w_{i-2}) P(w_i | w_{i-1}) + \lambda_3 (w_{i-2}) P(w_i)
\]
Dealing with unseen words
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.
- **Open vocabulary** task: may encounter out-of-vocabulary (OOV) words.
- Common solution: create an unknown word token: <UNK>
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.

- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.

- Common solution: create an unknown word token: `<UNK>`

  - Choose a fixed vocabulary of size $V$ in advance
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: `<UNK>`
  
  - Choose a fixed vocabulary of size $V$ in advance
  - Convert any token in training set not in $V$ to `<UNK>"
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: `<UNK>`

  - Choose a fixed vocabulary of size $V$ in advance
  - Convert any token in training set not in $V$ to `<UNK>`
  - Estimate probabilities of `<UNK>` just like any other word in the training set.
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.
- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.
- Common solution: create an unknown word token: `<UNK>`

- Choose a fixed vocabulary of size $V$ in advance
- Convert any token in training set not in $V$ to `<UNK>`
- Estimate probabilities of `<UNK>` just like any other word in the training set.

Threshold words by frequency
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.

- **Open vocabulary** task: may encounter **out-of-vocabulary (OOV)** words.

- Common solution: create an unknown word token: <UNK>

  - Choose a fixed vocabulary of size $V$ in advance
  - Convert any token in training set not in $V$ to <UNK>
  - Estimate probabilities of <UNK> just like any other word in the training set.

- Another solution: sub-word representations (e.g. FastText word embeddings)

Threshold words by frequency
Dealing with unseen words

- **Closed vocabulary** task: we know all possible words in advance; $V$ is fixed.
- **Open vocabulary** task: may encounter out-of-vocabulary (OOV) words.

Common solution: create an unknown word token: <UNK>

- Choose a fixed vocabulary of size $V$ in advance
- Convert any token in training set not in $V$ to <UNK>
- Estimate probabilities of <UNK> just like any other word in the training set.

Another solution: sub-word representations (e.g. FastText word embeddings)

$$skiing = \{^{skiing}$, $^{ski}$, $^skii$, $^kiin$, $^iing$, $^ing$\}$$
Huge, web-scale n-grams

Google n-gram release:

All Our N-gram are Belong to You
Thursday, August 3, 2006

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html
Huge, web-scale n-grams

- Google n-gram release:

All Our N-gram are Belong to You
Thursday, August 3, 2006

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

File sizes: approx. 24 GB compressed (gzip'ed) text files

- Number of tokens: 1,024,908,267,229
- Number of sentences: 95,119,665,584
- Number of unigrams: 13,588,391
- Number of bigrams: 314,843,401
- Number of trigrams: 977,069,902
- Number of fourgrams: 1,313,818,354
- Number of fivegrams: 1,176,470,663

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html
Huge, web-scale n-grams

Google n-gram release:

All Our N-gram are Belong to You

Thursday, August 3, 2006

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

File sizes: approx. 24 GB compressed (gzip'ed) text files

| Number of tokens: | 1,024,908,267,229 |
| Number of sentences: | 95,119,665,584 |
| Number of unigrams: | 13,588,391 |
| Number of bigrams: | 314,843,401 |
| Number of trigrams: | 977,069,902 |
| Number of fourgrams: | 1,313,818,354 |
| Number of fivegrams: | 1,176,470,663 |

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html
Scaling to huge, web-scale n-grams?
Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren’t informative
Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren’t informative
- Computational efficiency:
Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren’t informative
- Computational efficiency:
  - Efficient data structures like tries.
Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count $>\$ threshold
  - Entropy-based pruning: prune n-grams that aren’t informative

- Computational efficiency:
  - Efficient data structures like tries.
  - Bloom filters: approximate language models.
Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren’t informative

- Computational efficiency:
  - Efficient data structures like tries.
  - Bloom filters: approximate language models.
  - Don’t use strings: Huffman coding to stuff many words into two bytes.
Scaling to huge, web-scale n-grams?

- Pruning: only store n-grams with count > threshold
  - Entropy-based pruning: prune n-grams that aren’t informative

- Computational efficiency:
  - Efficient data structures like tries.
  - Bloom filters: approximate language models.
  - Don’t use strings: Huffman coding to stuff many words into two bytes.
  - Quantization: store probabilities in 4-8 bits rather than 32 bit float.
Language modeling toolkits

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

- KenLM
  
  https://kheafield.com/code/kenlm/
Language modeling toolkits

- **SRILM**
  

- **KenLM**
  
  [https://kheafield.com/code/kenlm/](https://kheafield.com/code/kenlm/)
Backoff and interpolation

- Sometimes it helps to use less context. Intuition: Condition on less context for contexts you haven’t learned much about.

- Two ways to use less context (in e.g. trigram model):
  - **Backoff**: Use trigram counts if you have good evidence; otherwise, bigram; otherwise, unigram.
  - **Interpolation**: Always mix unigram, bigram, trigram statistics.

- Interpolation works better

- State-of-the-art: **Extended interpolated Kneser-Ney smoothing**
Absolute discounting
Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare
Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare
- How much to subtract?

Dan Jurafsky

When we have sparse statistics:

- Steal probability mass to generalize better

\[
\begin{align*}
P(w|\text{denied the}) &= 2.5 \text{ allegations} + 1.5 \text{ reports} + 0.5 \text{ claims} + 0.5 \text{ request} + 2 \text{ other} \\
&= 7 \text{ total}
\end{align*}
\]

- Other...

allegations  reports  claims  request  attack  man  outcome  ...

\begin{align*}
P(w|\text{denied the}) &= 2.5 \text{ allegations} + 1.5 \text{ reports} + 0.5 \text{ claims} + 0.5 \text{ request} + 2 \text{ other} \\
&= 7 \text{ total}
\end{align*}
Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare

- How much to subtract?

  - Church and Gale (1991): How do the counts actually differ for each bigram in train vs. held-out set?
Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare

- How much to subtract?

  - Church and Gale (1991): How do the counts actually differ for each bigram in train vs. held-out set?

<table>
<thead>
<tr>
<th>Bigram count in training</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bigram count in held-out</td>
<td>0.0000270</td>
<td>0.448</td>
<td>1.25</td>
<td>2.24</td>
<td>3.23</td>
<td>4.21</td>
<td>5.23</td>
<td>6.21</td>
<td>7.21</td>
<td>8.26</td>
</tr>
<tr>
<td>Difference</td>
<td>-0.000027</td>
<td>0.552</td>
<td>0.75</td>
<td>0.76</td>
<td>0.77</td>
<td>0.79</td>
<td>0.77</td>
<td>0.79</td>
<td>0.79</td>
<td>0.74</td>
</tr>
</tbody>
</table>
Absolute discounting

- Removing counts from frequent n-grams, rather than adding to rare

- How much to subtract?

  - Church and Gale (1991): How do the counts actually differ for each bigram in train vs. held-out set?

<table>
<thead>
<tr>
<th>Bigram count in training</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bigram count in held-out</td>
<td>0.0000270</td>
<td>0.448</td>
<td>1.25</td>
<td>2.24</td>
<td>3.23</td>
<td>4.21</td>
<td>5.23</td>
<td>6.21</td>
<td>7.21</td>
<td>8.26</td>
</tr>
<tr>
<td>Difference</td>
<td>-0.000027</td>
<td>0.552</td>
<td>0.75</td>
<td>0.76</td>
<td>0.77</td>
<td>0.79</td>
<td>0.77</td>
<td>0.79</td>
<td>0.79</td>
<td>0.74</td>
</tr>
</tbody>
</table>
Absolute discounting interpolation
Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount $d$ from each count
Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount $d$ from each count
  
  - (Maybe keep a couple extra values of $d$ for counts 1 and 2.)
Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount $d$ from each count
  - (Maybe keep a couple extra values of $d$ for counts 1 and 2.)

- **Absolute discounting interpolation**: 
Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount $d$ from each count
  - (Maybe keep a couple extra values of $d$ for counts 1 and 2.)

- **Absolute discounting interpolation**:

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w_i)$$
Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount $d$ from each count
  - (Maybe keep a couple extra values of $d$ for counts 1 and 2.)

- **Absolute discounting interpolation**:

\[
P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w_i)
\]
Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount $d$ from each count
  - (Maybe keep a couple extra values of $d$ for counts 1 and 2.)

- **Absolute discounting interpolation**: 

\[
P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w_i)
\]
Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount $d$ from each count
  - (Maybe keep a couple extra values of $d$ for counts 1 and 2.)
- **Absolute discounting interpolation**:

  
  \[
  P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \lambda(w_{i-1})P(w_i) + \lambda(w_{i-1})P(w_i)
  \]
Absolute discounting interpolation

- **Absolute discounting**: just subtract a fixed discount $d$ from each count
  - (Maybe keep a couple extra values of $d$ for counts 1 and 2.)

- **Absolute discounting interpolation**:

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

- But, should we really use the unmodified unigram $P(w_i)$?
Kneser-Ney Smoothing
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading __________ ?
Kneser-Ney Smoothing

■ Better estimate for probabilities of unigrams.

■ Shannon game w/ unigrams: I can’t see without my reading _________ ? glasses
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading _______ ? King
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading ________ ?
  - *Kong* is more common than *glasses*
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading _________ ?
  - *Kong* is more common than *glasses*
  - …but *Kong* very frequently follows *Hong*
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading _________ ? Kong
  - Kong is more common than glasses
  - …but Kong very frequently follows Hong
- Instead of modeling $P(w_i)$: “How likely is $w_i$?”
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading _________ ?
  - *Kong* is more common than *glasses*
  - ...but *Kong* very frequently follows *Hong*
- Instead of modeling $P(w_i)$: “How likely is $w_i$?”
- $P_{continuation}(w_i)$: “How likely is $w_i$ to appear as a novel *continuation*?”
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading ________ Kong?
  - Kong is more common than glasses
  - ...but Kong very frequently follows Hong
- Instead of modeling $P(w_i)$: “How likely is $w_i$?”
- $P_{\text{continuation}}(w_i)$: “How likely is $w_i$ to appear as a novel continuation?”
  - For each word, count the number of bigram types it completes:
Kneser-Ney Smoothing

- Better estimate for probabilities of unigrams.
  - Shannon game w/ unigrams: I can’t see without my reading Kong ?
  - Kong is more common than glasses
  - …but Kong very frequently follows Hong
- Instead of modeling $P(w_i)$: “How likely is $w_i$?”
- $P_{\text{continuation}}(w_i)$: “How likely is $w_i$ to appear as a novel continuation?”
  - For each word, count the number of bigram types it completes:
    \[
    P_{\text{continuation}}(w_i) \propto \left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|
    \]
Kneser-Ney Smoothing
Kneser-Ney Smoothing

How many times does $w_i$ appear as a novel continuation:

$$P_{\text{continuation}}(w_i) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$
Kneser-Ney Smoothing

- How many times does $w_i$ appear as a novel continuation:

$$P_{continuation}(w_i) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- Normalize by the total number of word bigram types:

$$|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|$$
Kneser-Ney Smoothing

How many times does \( w_i \) appear as a novel continuation:

\[
P_{\text{continuation}}(w_i) \propto |\{ w_{i-1} : c(w_{i-1}, w) > 0 \}|
\]

Normalize by the total number of word bigram types:

\[
|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0 \}|
\]

To get an estimate for the probability that \( w_i \) will appear:

\[
P_{\text{continuation}}(w_i) = \frac{|\{ w_{i-1} : c(w_{i-1}, w) > 0 \}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0 \}|}
\]
Kneser-Ney Smoothing

- How many times does $w_i$ appear as a novel continuation:

$$P_{\text{continuation}}(w_i) \propto \left| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} \right|$$

- Normalize by the total number of word bigram types:

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

- To get an estimate for the probability that $w_i$ will appear:

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

- A frequent word (Kong) occurring mostly in one context (Hong) will have a low continuation probability.
Kneser-Ney Smoothing

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

- Interpolated Kneser-Ney smoothing bigram probability:

\[
P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{\text{continuation}}(w_i)
\]

\[
\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{w : c(w_{i-1}, w) > 0 \} \right|
\]
Kneser-Ney Smoothing

- Interpolated Kneser-Ney smoothing bigram probability:

\[ P_{KN}(w_i | w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{continuation}(w_i) \]

- where \( \lambda \) is a normalizing constant to distribute the discounted probability mass:

\[ \lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}\left|\{w : c(w_{i-1}, w) > 0\}\right| \]
Kneser-Ney Smoothing

- Interpolated Kneser-Ney smoothing bigram probability:

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

- where \( \lambda \) is a normalizing constant to distribute the discounted probability mass:

\[ \lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right| \]
Kneser-Ney Smoothing

- Interpolated Kneser-Ney smoothing bigram probability:

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

- where \( \lambda \) is a normalizing constant to distribute the discounted probability mass:

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

(normalized discount)
**Kneser-Ney Smoothing**

- Interpolated Kneser-Ney smoothing bigram probability:

\[
P_{KN}(w_i | w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{continuation}(w_i)
\]

- where \( \lambda \) is a normalizing constant to distribute the discounted probability mass:

\[
\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}| \quad \text{normalized discount}
\]

\( \lambda(w_{i-1}) \) = # word types that can follow \( w_{i-1} \)

= # word types that we discounted

= # of times we applied normalized discount
Kneser-Ney Smoothing
Kneser-Ney Smoothing

- Recursive formulation for n-grams (typically ~5):

\[
P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i-1}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i \mid w_{i-n+2}^{i-1})
\]

where:

\[
c_{KN}(\cdot) = \begin{cases} 
\text{count(\cdot)} & \text{for the highest order} \\
\text{continuation_count(\cdot)} & \text{for lower orders}
\end{cases}
\]
Kneser-Ney Smoothing

- Recursive formulation for n-grams (typically ~5):

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

where:

\[
c_{KN}(\cdot) = \begin{cases} 
\text{count(\cdot)} & \text{for the highest order} \\
\text{continuation\_count(\cdot)} & \text{for lower orders}
\end{cases}
\]

- Continuation count: number of unique single word contexts for \( \cdot \)
Kneser-Ney Smoothing

- Recursive formulation for n-grams (typically ~5):

\[
P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i-1}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i \mid w_{i-n+2}^{i-1})
\]

where:

\[
c_{KN}(\cdot) = \begin{cases} 
\text{count}(\cdot) & \text{for the highest order} \\
\text{continuation} \_ \text{count}(\cdot) & \text{for lower orders}
\end{cases}
\]

- Continuation count: number of unique single word contexts for \( \cdot \)

- **Modified Kneser-Ney smoothing**: Uses three different discounts for n-grams with counts 1, 2, and 3+ [Chen and Goodman 1998].
n-gram language model challenges
n-gram language model challenges

- Bias/variance trade-off
n-gram language model challenges

- Bias/variance trade-off
- Generalization
n-gram language model challenges

- Bias/variance trade-off

- Generalization
  
  I’ll have a ________  
  Give me the ________

  I’d like to order the ________  
  I’ll take one ________
n-gram language model challenges

- Bias/variance trade-off
- Generalization
  - I’ll have a ________
  - Give me the ________
  - I’d like to order the ________
  - I’ll take one ________
- Context size
n-gram language model challenges

- Bias/variance trade-off

- Generalization

  I’ll have a ________
  
  Give me the ________
  
  I’d like to order the ________
  
  I’ll take one ________

- Context size

The **computer(s)** that I just put into the machine room on the fifth floor **is (are)** crashing.
n-gram language model challenges

- Bias/variance trade-off
- Generalization
  - I’ll have a ________
  - I’d like to order the ________
  - Give me the ________
  - I’ll take one ________
- Context size
  The computer(s) that I just put into the machine room on the fifth floor is (are) crashing.
- Solution: neural language models. Next class.
n-gram language model challenges

- Bias/variance trade-off

- Generalization
  - I’ll have a ________  Give me the ________
  - I’d like to order the ________  I’ll take one ________

- Context size
  The computer(s) that I just put into the machine room on the fifth floor is (are) crashing.

- Solution: neural language models. Next class.
  - Better empirical performance.
n-gram language model challenges

- Bias/variance trade-off

- Generalization
  - I’ll have a _______  
  - I’d like to order the _______
  - Give me the _______

- Context size
  
The **computer(s)** that I just put into the machine room on the fifth floor is (are) crashing.

- Solution: **neural language models.** Next class.
  - Better empirical performance.
  - Provide state-of-the-art dense word representations.
n-gram language model challenges

- Bias/variance trade-off

- Generalization
  
  I’ll have a ________
  I’d like to order the ________
  Give me the ________
  I’ll take one ________

- Context size

  The **computer**(s) that I just put into the machine room on the fifth floor **is** (are) crashing.

- Solution: **neural language models**. Next class.

  - Better empirical performance.
  
  - Provide state-of-the-art dense word representations.
  
  - But: a lot more computation. Slower to train and infer.
Announcements

- Recitation this week will be a P1 Q&A with Han
- P1 bakeoff: No pre-trained word embeddings or BERT. Out-of-the box linguistic annotations (e.g. pos tags), tf-idf, etc. are fine.