Probabilistic Language Models

• Today’s goal: assign a probability to a sentence
  • Machine Translation:
    • $P(\text{high winds tonite}) > P(\text{large winds tonite})$
  • Spell Correction
    • The office is about fifteen minuets from my house
      • $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
  • Speech Recognition
    • $P(\text{I saw a van}) >> P(\text{eyes awe of an})$
  • + Summarization, question-answering, etc., etc.!!

Why?
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.

- Better: the grammar But language model or LM is standard
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

\[ p(B | A) = \frac{P(A,B)}{P(A)} \quad \text{Rewriting:} \quad P(A,B) = P(A)P(B | A) \]

• More variables:

\[ P(A,B,C,D) = P(A)P(B | A)P(C | A,B)P(D | A,B,C) \]

• The Chain Rule in General

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

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

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

- Could we just count and divide?

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

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

• Simplifying assumption:

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

• Or maybe

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

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

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

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

\[ P(w_1 w_2 \ldots w_n) \approx \prod_{i} P(w_i) \]

Some automatically generated sentences from a unigram model

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

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

that, or, limited, the
Bigram model

- Condition on the previous word:

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

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

- 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
Language Modeling

Introduction to N-grams
Language Modeling

Estimating N-gram Probabilities
Estimating bigram probabilities

- The Maximum Likelihood Estimate

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

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

\[
\begin{align*}
P(I \mid <s>) &= \frac{2}{3} = .67 \\
P(Sam \mid <s>) &= \frac{1}{3} = .33 \\
P(am \mid I) &= \frac{2}{3} = .67 \\
P(<s> \mid Sam) &= \frac{1}{2} = 0.5 \\
P(Sam \mid am) &= \frac{1}{2} = .5 \\
P(do \mid I) &= \frac{1}{3} = .33
\end{align*}
\]
More examples:
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

- 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>

- 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</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</td>
<td>0.0027</td>
<td>0.52</td>
</tr>
<tr>
<td>chinese</td>
<td>0.0063</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00092</td>
<td>0</td>
<td>0.0037</td>
<td>0.0029</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</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>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Bigram estimates of sentence probabilities

\[
P(\langle s \rangle \text{ I want english food } \langle /s \rangle ) = \\
P(\text{I} \mid \langle s \rangle) \\
\times P(\text{want} \mid \text{I}) \\
\times P(\text{english} \mid \text{want}) \\
\times P(\text{food} \mid \text{english}) \\
\times P(\langle /s \rangle \mid \text{food}) \\
= .000031
\]
What kinds of knowledge?

- $P(\text{english} \mid \text{want}) = .0011$
- $P(\text{chinese} \mid \text{want}) = .0065$
- $P(\text{to} \mid \text{want}) = .66$
- $P(\text{eat} \mid \text{to}) = .28$
- $P(\text{food} \mid \text{to}) = 0$
- $P(\text{want} \mid \text{spend}) = 0$
- $P(\text{i} \mid <s>) = .25$
Practical Issues

• We do everything in log space
  • Avoid underflow
  • (also adding is faster than multiplying)

\[
\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4
\]
Language Modeling Toolkits

- SRILM
- KenLM
  - [https://kheafield.com/code/kenlm/](https://kheafield.com/code/kenlm/)
All Our N-gram are Belong to You

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

Here at Google Research we have been using word n-gram models for a variety of R&D projects, ...

That’s why we decided to share this enormous dataset with everyone. 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.
Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html
Google Book N-grams

- http://ngrams.googlelabs.com/
Language Modeling

Estimating N-gram Probabilities
Evaluation and Perplexity

Language Modeling
How to evaluate N-gram models

"Extrinsic (in-vivo) Evaluation"

To compare models A and B

1. Put each model in a real task
   • Machine Translation, speech recognition, etc.
2. Run the task, get a score for A and for B
   • How many words translated correctly
   • How many words transcribed correctly
3. Compare accuracy for A and B
Intrinsic (in-vitro) evaluation

Extrinsic evaluation not always possible
• Expensive, time-consuming
• Doesn't always generalize to other applications

Intrinsic evaluation: *perplexity*
• Directly measures language model performance at predicting words.
• Doesn't necessarily correspond with real application performance
• But gives us a single general metric for language models
• Useful for large language models (LLMs) as well as n-grams
Training sets and test sets

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; different from training set.
- Intuition: we want to measure generalization to unseen data
- An **evaluation metric** (like **perplexity**) tells us how well our model does on the test set.
Choosing training and test sets

• If we're building an LM for a specific task
  • The test set should reflect the task language we want to use the model for

• If we're building a general-purpose model
  • We'll need lots of different kinds of training data
  • We don't want the training set or the test set to be just from one domain or author or language.
Training on the test set

We can’t allow test sentences into the training set
• Or else the LM will assign that sentence an artificially high probability when we see it in the test set
• And hence assign the whole test set a falsely high probability.
• Making the LM look better than it really is

This is called “Training on the test set”

Bad science!
Dev sets

• If we test on the test set many times we might implicitly tune to its characteristics
  • Noticing which changes make the model better.

• So we run on the test set only once, or a few times

• That means we need a third dataset:
  • A **development test set** or, **devset**.
  • We test our LM on the devset until the very end
  • And then test our LM on the **test set** once
Intuition of perplexity as evaluation metric: How good is our language model?

Intuition: A good LM prefers "real" sentences

• Assign higher probability to “real” or “frequently observed” sentences
• Assigns lower probability to “word salad” or “rarely observed” sentences?
Intuition of perplexity 2: Predicting upcoming words

The Shannon Game: **How well can we predict the next word?**
- Once upon a ____
- That is a picture of a ____
- For breakfast I ate my usual ____

Unigrams are terrible at this game (Why?)

A good LM is one that assigns a higher probability to the next word that actually occurs

Claude Shannon

![Image of Claude Shannon](https://creativecommons.org/licenses/by/2.0/)

**Time** 0.9
**Dream** 0.03
**Midnight** 0.02

... and 1e-100
Intuition of perplexity 3: The best language model is one that best predicts the entire unseen test set

• We said: a good LM is one that assigns a higher probability to the next word that actually occurs.

• Let's generalize to all the words!
  • The best LM assigns high probability to the entire test set.

• When comparing two LMs, A and B
  • We compute $P_A$(test set) and $P_B$(test set)
  • The better LM will give a higher probability to (=be less surprised by) the test set than the other LM.
Intuition of perplexity 4: Use perplexity instead of raw probability

- Probability depends on size of test set
  - Probability gets smaller the longer the text
  - Better: a metric that is per-word, normalized by length

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

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

\[
= \frac{1}{\sqrt[N]{P(w_1w_2...w_N)}}
\]
Intuition of perplexity 5: the inverse

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

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

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

(The inverse comes from the original definition of perplexity from cross-entropy rate in information theory)

Probability range is [0,1], perplexity range is [1,∞]

**Minimizing perplexity is the same as maximizing probability**
Intuition of perplexity 6: N-grams

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

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

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

Bigrams:
\[ PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \]
Intuition of perplexity 7: Weighted average branching factor

Perplexity is also the **weighted average branching factor** of a language.

**Branching factor**: number of possible next words that can follow any word

Example: Deterministic language \( L = \{\text{red, blue, green}\} \)

Branching factor = 3 (any word can be followed by red, blue, green)

Now assume LM A where each word follows any other word with equal probability \( \frac{1}{3} \)

Given a test set \( T = \text{"red red red red blue"} \)

\[
\text{Perplexity}_A(T) = P_A(\text{red red red red blue})^{-1/5} = ((\frac{1}{3})^5)^{-1/5} = (\frac{1}{3})^{-1} = 3
\]

But now suppose red was very likely in training set, such that for LM B:

- \( P(\text{red}) = .8 \)  \( p(\text{green}) = .1 \)  \( p(\text{blue}) = .1 \)

We would expect the probability to be higher, and hence the perplexity to be smaller:

\[
\text{Perplexity}_B(T) = P_B(\text{red red red red blue})^{-1/5} = (0.8 * 0.8 * 0.8 * 0.1)^{-1/5} = 0.04096^{-1/5} = 0.527^{-1} = 1.89
\]
Holding test set constant:
Lower perplexity = better language model

Training 38 million words, test 1.5 million words, WSJ

<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>
Language Modeling

Evaluation and Perplexity
The Shannon (1948) Visualization Method
Sample words from an LM

Unigram:

REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

Bigram:

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.
How Shannon sampled those words in 1948

"Open a book at random and select a letter at random on the page. This letter is recorded. The book is then opened to another page and one reads until this letter is encountered. The succeeding letter is then recorded. Turning to another page this second letter is searched for and the succeeding letter recorded, etc."
Sampling a word from a distribution

the 0.06
of 0.03
a 0.02
of 0.02
to 0.02
in 0.02

however (p=.0003) polyphonic p=.0000018
Choose a random bigram \((<s>, w)\) according to its probability \(p(w|<s>)\)

Now choose a random bigram \((w, x)\) according to its probability \(p(x|w)\)

And so on until we choose \(<s>\)

Then string the words together

\[
\text{I want to eat Chinese food}
\]
Note: there are other sampling methods

Used for neural language models

Many of them avoid generating words from the very unlikely tail of the distribution

We'll discuss when we get to neural LM decoding:

- Temperature sampling
- Top-k sampling
- Top-p sampling
### 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. |
| 3 gram | –Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. |
| 4 gram | –This shall forbid it should be branded, if renown made it empty. |
| 4 gram | –King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; |
| 4 gram | –It cannot be but so. |
Shakespeare as corpus

N=884,647 tokens, V=29,066

Shakespeare produced 300,000 bigram types out of $V^2=844$ million possible bigrams.

- So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- That sparsity is even worse for 4-grams, explaining why our sampling generated actual Shakespeare.
The Wall Street Journal is not Shakespeare

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

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
Can you guess the author? These 3-gram sentences are sampled from an LM trained on who?

1) 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 gram Brazil on market conditions.

2) This shall forbid it should be branded, if renown made it empty.

3) “You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.
Choosing training data

If task-specific, use a training corpus that has a similar genre to your task.

- If legal or medical, need lots of special-purpose documents

Make sure to cover different kinds of dialects and speaker/authors.

- Example: *African-American Vernacular English (AAVE)*
  - One of many varieties that can be used by African Americans and others
  - Can include the auxiliary verb *finna* that marks immediate future tense:
    - "My phone finna die"
The perils of overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus
  • But even when we try to pick a good training corpus, the test set will surprise us!
  • We need to train robust models that generalize!

One kind of generalization: **Zeros**
  • Things that don’t ever occur in the training set
  • But occur in the test set
Zeros

Training set:
... ate lunch
... ate dinner
... ate a
... ate the

P(“breakfast” | ate) = 0

• Test set
  ... ate lunch
  ... ate breakfast
Zero probability bigrams

Bigrams with zero probability
- Will hurt our performance for texts where those words appear!
- And mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity (can’t divide by 0)!
Sampling and Generalization

Language Modeling
Language Modeling

Smoothing: Add-one (Laplace) smoothing
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

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

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

- MLE estimate:

\[ P_{Add-1}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V} \]

- Add-1 estimate:
Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is $400/1,000,000 = .0004$
- This may be a bad estimate for some other corpus
  - But it is the estimate that makes it most likely that “bagel” will occur 400 times in a million word corpus.
### 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>0.00025</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>
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.
Language Modeling

Smoothing: Add-one (Laplace) smoothing
Language Modeling

Interpolation, Backoff, and Web-Scale LMs
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:

\[ \hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 (w_{n-1}^{n-2}) P(w_n|w_{n-2}w_{n-1}) + \lambda_2 (w_{n-1}^{n-2}) P(w_n|w_{n-1}) + \lambda_3 (w_{n-1}^{n-2}) P(w_n) \]
How to set the lambdas?

- Use a **held-out** corpus

Training Data  | Held-Out Data  | Test Data

- Choose $\lambda$s to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for $\lambda$s that give largest probability to held-out set:

$$
\log P(w_1...w_n \mid M(\lambda_1...\lambda_k)) = \sum_i \log P_{M(\lambda_1...\lambda_k)}(w_i \mid w_{i-1})
$$
Unknown words: Open versus closed vocabulary tasks

• If we know all the words in advanced
  • Vocabulary V is fixed
  • Closed vocabulary task
• Often we don’t know this
  • **Out Of Vocabulary** = OOV words
  • Open vocabulary task
• Instead: create an unknown word token <UNK>
  • Training of <UNK> probabilities
    • Create a fixed lexicon L of size V
    • At text normalization phase, any training word not in L changed to <UNK>
    • Now we train its probabilities like a normal word
  • At decoding time
    • If text input: Use UNK probabilities for any word not in training
Huge web-scale n-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count > threshold.
  - Remove singletons of higher-order n-grams
- Entropy-based pruning
- Efficiency
  - Efficient data structures like tries
  - Bloom filters: approximate language models
  - Store words as indexes, not strings
    - Use Huffman coding to fit large numbers of words into two bytes
  - Quantize probabilities (4-8 bits instead of 8-byte float)
Smoothing for Web-scale N-grams

- “Stupid backoff” (Brants et al. 2007)
- No discounting, just use relative frequencies

\[
S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} 
\frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\
0.4S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise}
\end{cases}
\]

\[
S(w_i) = \frac{\text{count}(w_i)}{N}
\]
N-gram Smoothing Summary

- Add-1 smoothing:
  - OK for text categorization, not for language modeling
- The most commonly used method:
  - Extended Interpolated Kneser-Ney
- For very large N-grams like the Web:
  - Stupid backoff
Advanced Language Modeling

• Discriminative models:
  • choose n-gram weights to improve a task, not to fit the training set

• Parsing-based models

• Caching Models
  • Recently used words are more likely to appear

\[ P_{\text{CACHE}}(w | \text{history}) = \lambda P(w_i | w_{i-2}w_{i-1}) + (1 - \lambda) \frac{c(w \in \text{history})}{|\text{history}|} \]

• These perform very poorly for speech recognition (why?)
Language Modeling

Interpolation, Backoff, and Web-Scale LMs