Speech Recognition: Acoustic Waves

- Human speech generates a wave
  - like a loudspeaker moving

- A wave for the words "speech lab" looks like:

Acoustic Sampling

- 10 ms frame (ms = millisecond = 1/1000 second)
- ~25 ms window around frame [wide band] to allow/smooth signal processing – it let’s you see formants

Spectral Analysis

- Frequency gives pitch; amplitude gives volume
  - sampling at ~8 kHz phone, ~16 kHz mic (kHz=1000 cycles/sec)
- Fourier transform of wave displayed as a spectrogram
  - darkness indicates energy at each frequency
  - hundreds to thousands of frequency samples

The Speech Recognition Problem

- The Recognition Problem: Noisy channel model
  - Build generative model of encoding: We started with English words, they were encoded as an audio signal, and we now wish to decode.
  - Find most likely sequence $w$ of “words” given the sequence of acoustic observation vectors $a$
    - Use Bayes’ rule to create a generative model and then decode
      $$ \text{ArgMax}_w P(w|a) = \text{ArgMax}_w P(a|w) P(w)/P(a) $$
      $$ = \text{ArgMax}_w P(a|w) P(w) $$

- Acoustic Model: $P(a|w)$
- Language Model: $P(w)$

- Why is this progress?

MT: Just a Code?

- “Also knowing nothing official about, but having guessed and inferred considerable about, the powerful new mechanized methods in cryptography—methods which I believe succeed even when one does not know what language has been coded—one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: ‘This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.’ ”
- Warren Weaver (1955:18, quoting a letter he wrote in 1947)
MT System Components

Language Model

Translation Model

source

P(e)

channel

P(fe)

decoder

observed

f

argmax P(e|f) = argmax P(f|e)P(e)

Other Noisy-Channel Processes

- Handwriting recognition
  \[ P(\text{text} \mid \text{strokes}) = P(\text{text})P(\text{strokes} \mid \text{text}) \]
- OCR
  \[ P(\text{text} \mid \text{pixels}) = P(\text{text})P(\text{pixels} \mid \text{text}) \]
- Spelling Correction
  \[ P(\text{text} \mid \text{typos}) = P(\text{text})P(\text{typos} \mid \text{text}) \]

Questions that linguistics should answer

- What kinds of things do people say?
- What do these things say/ask/request about the world?
  - Example: In addition to this, she insisted that women were regarded as a different existence from men unfairly.
- Text corpora give us data with which to answer these questions
- They are an externalization of linguistic knowledge
- What words, rules, statistical facts do we find?
- How can we build programs that learn effectively from this data, and can then do NLP tasks?

Probabilistic Language Models

- Want to build models which assign scores to sentences.
  - Example: \( P(\text{I saw a van}) >> P(\text{eyes awe of an}) \)
  - Not really grammaticality: \( P(\text{artichokes intimidate zippers}) \approx 0 \)
- One option: empirical distribution over sentences?
  - Problem: doesn’t generalize (at all)
- Two major components of generalization
  - Backoff: sentences generated in small steps which can be recombined in other ways
  - Discounting: allow for the possibility of unseen events

N-Gram Language Models

- No loss of generality to break sentence probability down with the chain rule
  \[ P(w_1w_2 \ldots w_n) = \prod P(w_i \mid w_{i-1} \ldots w_1) \]
- Too many histories!
  - \( P(??? \mid \text{no loss of generality to break sentence}) \)?
  - \( P(??? \mid \text{the water is so transparent that}) \)?
- N-gram solution: assume each word depends only on a short linear history (Markov assumption)
  \[ P(w_1w_2 \ldots w_n) = \prod P(w_i \mid w_{i-1} \ldots w_{i-1}) \]

Unigram Models

- Simplest case: unigrams
  \[ P(w_1w_2 \ldots w_n) = \prod P(w_i) \]
- Generative process: pick a word, pick a word, ...
- As a graphical model:

  \[ \begin{array}{c}
    w_1 \\
    w_2 \\
    \vdots \\
    w_n \\
    \text{STOP}\end{array} \]

  - To make this a proper distribution over sentences, we have to generate a special \( \text{STOP} \) symbol last. (Why?)

Examples:

- Two in 1984, was incorporated, is, the, inflation, most, dollars, quarter, in, is, mass.
- That, or, limited, the
- After, any, on, consistently, hospital, of, other, factors, raised, analyst, too, allowed, mexico, never, consider, fall, bungled, davison, that, obtain, price, lines, the, to, sass, the, the, further, board, a
- Details, machinists, the, companies, which, rivals, an, because, longer, oakes, percent, a, they, three, edward, it, currier, an, within, in, three, wrote, is, you, s., longer, institute, dentistry, pay, however, said, possible, to, rooms, hiding, eggs, approximate, financial, canada, the, so, workers, advancers, half, between, nasdaq
Bigram Models

- Big problem with unigrams: \( P(\text{the the the the}) \gg P(\text{I like ice cream})! \)
- Condition on previous word:
  \[
P(w_i | w_{i-1}) = \prod_{i} P(w_i | w_{i-1})
\]
- Any better?
  - (textaco, 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)
  - (although, common, shares, rose, forty, six, point, four, hundred, dollars, from, thirty, seconds, at, the, greatest, play, disingenuous, to, be, reset, annually, the, buy, out, of, american, brands, vying, for, mr., womack, currently, sharedata, incorporated, believes, chemical, prices, undoubtedly, will, be, as, much, is, scheduled, to, conscientious, teaching)
  - (this, would, be, a, record, november)

Regular Languages?

- N-gram models are (weighted) regular languages
- You can extend to trigrams, fourgrams, ...
- Why can’t we model language like this?
  - Linguists have many arguments why language can’t be regular.
  - Long-distance effects:
    - “The computer which I had just put into the machine room on the fifth floor crashed.”
  - Why CAN we often get away with n-gram models?
  - PCFG language models do model tree structure (later):
    - “This, quarter, ’s, surprisingly, independent, attack, paid, off, the, risk, involving, IRS, leaders, and, transportation, prices.”
    - “It, could, be, announced, sometime.”
    - “Mr., Toseland, believes, the, average, defense, economy, is, drafted, from, slightly, more, than, 12, stocks.”

Estimating bigram probabilities: The maximum likelihood estimate

- \( P(\text{I am Sam}) = \frac{5}{827} \)
- \( P(\text{Sam I am}) = \frac{9}{827} \)
- \( P(\text{I do not like green eggs and ham}) = \frac{67}{5} \)

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

- This is the Maximum Likelihood Estimate, because it is the one which maximizes \( P(\text{Training set} | \text{Model}) \)

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

<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>2</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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></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>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>0.033</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>want</td>
<td>0.0022</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>0.0017</td>
<td>0.28</td>
<td>0.00038</td>
<td>0.0025</td>
<td>0.0072</td>
<td>0.0077</td>
</tr>
<tr>
<td>Chinese</td>
<td>0.0036</td>
<td>0.0027</td>
<td>0.021</td>
<td>0.0075</td>
<td>0.0056</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>food</td>
<td>0.0014</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>0.0009</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>0.0036</td>
<td>0.0036</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Evaluation**

- What we want to know is:
  - Will our model prefer good sentences to bad ones?
    - That is, does it assign higher probability to "real" or "frequently observed" sentences than "ungrammatical" or "rarely observed" sentences?
  - As a component of Bayesian inference, will it help us discriminate correct utterances from noisy inputs?
- We train parameters of our model on a training set.
- To evaluate how well our model works, we look at the model’s performance on some new data.
- This is what happens in the real world; we want to know how our model performs on data we haven’t seen.
- So a test set. A dataset which is different from our training set. Preferably totally unseen/unused.

**Measuring Model Quality**

- For Speech: Word Error Rate (WER) – *insertions + deletions + substitutions* / *true sentence size*.
- Correct answer: Andy saw a part of the movie.
- Recognizer output: And he saw a part of the movie.
- The "right" measure:
  - Task error driven
  - For speech recognition
  - For a specific recognizer
- Extrinsic, task-based evaluation is in principle best, but …
- For general evaluation, we want a measure which references only good text, not mistake text.

**Measuring Model Quality**

- Problem with entropy:
  - 0.1 bits of improvement doesn’t sound so good
- Solution: perplexity
  - Intrinsic measure: may not reflect task performance (but is helpful as a first thing to measure and optimize on)
- Note: Even though our models require a stop step, people typically don’t count it as a symbol when taking these averages.
- E.g.,
  - N-gram Order
  - Perplexity
  - Unigram
  - Bigram
  - Trigram
  - Perplexity
  - 982
  - 170
  - 109

**The Shannon Visualization Method**

- Generate random sentences:
- Choose a random bigram <s>, w according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- Then string the words together
- etc.

**What’s in our text corpora**

- Common words in *Tom Sawyer* (71,370 words)
  - the: 3332, and: 1775, to: 1725, of: 1440, was: 1161, it: 1027, that: 896, he: 686, Tom: 679
  - Word Frequency of Frequency
    - 1 3993
    - 2 1292
    - 3 1164
    - 5 410
    - 6 243
    - 8 119
    - 9 172
    - 10 131
    - 11–50 1540
    - >50 99
    - >100 102
Sparsity

- Problems with n-gram models:
  - New words appear regularly:
    - Synaptitude
    - 132,701.03
  - New bigrams: even more often
    - Trigrams or more – still worse!

- Zipf’s Law
  - Types (words) vs. tokens (word occurrences)
  - Broadly: most word types are rare ones
  - Specifically:
    - Rank word types by token frequency
    - Frequency inversely proportional to rank: \( f \propto \frac{1}{r} \)
    - Statistically: word distributions are heavy-tailed
  - Not special to language: randomly generated character strings have this property (try it?)

Zipf’s Law (on the Brown corpus)

Smoothing

- We often want to make estimates from sparse statistics:
  - \( P(w | \text{denied the}) \)
    - 3 allegations
    - 2 reports
    - 1 claim
    - 1 request
    - 7 total

Smoothing: Add-One, Add-\( \delta \) (for bigram models)

- Estimating multinomials
  - We want to know what words follow some history \( h \)
  - We saw some small sample of \( N \) words from \( P(w | h) \)
  - Counts of events we didn’t see are always too low (0 < \( N P(w | h) \))
  - Counts of events we did see are in aggregate too high

- Example:
  - \( P(w | \text{denied the}) \)
    - 3 allegations
    - 2 reports
    - 1 claim
    - 1 request
    - 7 total

  - \( P(w | \text{affirmed the}) \)
    - 1 award

- Two issues:
  - Discounting: how to reserve mass what we haven’t seen
  - Interpolation: how to allocate that mass amongst unseen events

Five types of smoothing

- Today we’ll cover
  - Add-\( \delta \) smoothing (Laplace)
  - Simple interpolation
  - Good-Turing smoothing
  - Katz smoothing
  - Kneser-Ney smoothing

- Or less if we run out of time … and then you’ll just have to read the textbook!
Add-One Estimation

- Idea: pretend we saw every word once more than we actually did [Laplace]
  \[ P(w|h) = \frac{c(w,h) + 1}{c(h) + V} \]
  - Think of it as taking items with observed count \( r > 1 \) and treating them as having count \( r^* = r - 1 \)
  - Holds out \( V/(N+V) \) for "fake" events
  - \( N \) of which is distributed back to seen words
  - Actually tells us not only how much to hold out, but where to put it
  - Works astonishingly poorly in practice
- Quick fix: add some small \( \delta \) instead of 1 [Lidstone, Jefferys]
  - Slightly better, holds out less mass, still a bad idea

Laplace-smoothed bigrams

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

Quiz Question!

- Suppose I'm making a language model with a vocabulary size of 20,000 words
- In my training data, I saw the bigram "comes across" 10 times
  - 5 times it was followed by as
  - 5 times it was followed by other words (like, less, again, most, in)

  - What is the MLE of \( P(as|comes across) \)?
  - What is the add-1 estimate of \( P(as|comes across) \)?

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

<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>1</td>
<td>3</td>
<td>1</td>
<td>609</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>687</td>
<td>3</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>1</td>
<td>3</td>
</tr>
<tr>
<td>chinese</td>
<td>2</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>
</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>
</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>
</tr>
</tbody>
</table>

How Much Mass to Withhold?

- Remember the key discounting problem:
  - What count should \( r^* \) should we use for an event that occurred \( r \) times in \( N \) samples?
    - \( r \) is too big
  - Idea: held-out data [Jelinek and Mercer]
    - Get another \( N \) samples
    - See what the average count of items occurring \( r \) times is (e.g., doubletons on average might occur 1.78 times)
    - Use those averages as \( r^* \)

- Works better than fixing counts to add in advance
**Backoff and Interpolation**

- **Discounting** says, “I saw event X n times, but I will really treat it as if I saw it fewer than n times.
- **Backoff (and interpolation)** says, “In certain cases, I will condition on less of my context than in other cases.”
  - The sensible thing is to condition on less in contexts that you haven’t learned much about.
- **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram
- **Interpolation**: mix all three

**Linear Interpolation**

- One way to ease the sparsity problem for n-grams is to use less-sparse n-1-gram estimates
- General linear interpolation:
  \[ P(w|w_{-j}) = [1 - \lambda(w_j)]P(w|w_{-j}) + \lambda j P(w) \]
  - Having a single global mixing constant is generally not ideal:
    \[ P(w|w_{-j}) = [1 - \lambda]P(w|w_{-j}) + \lambda P(w) \]
  - But it actually works surprisingly well – simplest competent approach
  - A better yet still simple alternative is to vary the mixing constant as a function of the conditioning context
    \[ P(w|w_{-j}) = [1 - \lambda(w_j)]P(w|w_{-j}) + \lambda(w_j)P(w) \]

**Held-Out Data**

- Important tool for getting models to generalize:
- When we have a small number of parameters that control the degree of smoothing, we set them to maximize the log-likelihood of held-out data
  \[ LL = \sum \log P_{\text{held-out}}(w_{-j} | w_{-i}) \]
  - Can use any optimization technique (line search or EM usually easiest)
  - Example:
    \[ P(w|w_{-j}) = [1 - \lambda]P(w|w_{-j}) + \lambda P(w) \]

**Good-Turing Reweighting I**

- We’d like to not need held-out data (why?)
- Idea: leave-one-out validation
  - Take each of the c training words out in turn
  - c training sets of size c-1, hold-out of size 1
- What fraction of held-out words are unseen k times in training?
  - \( N_k \)
  - What fraction of held-out words are seen k times?
    - \( \hat{N}_k \)
- So in the future we expect \( (k+1)\hat{N}_k / c \)
- There are \( N_k \) words with training count k
- Each should occur with probability:
  - \( (k+1)\hat{N}_k / c \)
  - \( \hat{N}_k \)

**Good-Turing Reweighting II**

- Problem: what about “the”?
  - (say c=4417)
- For small \( k, N_k > \hat{N}_k \)
- For large \( k, \) too jumpy, zeros wreck estimates
- Simple Good-Turing [Gale and Sampson] replace empirical \( \hat{N}_k \) with a best-fit power law once count counts get unavailable

**Good-Turing smoothing intuition**

- Imagine you are fishing
  - You have caught
    - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
  - How likely is it that next species is new (i.e. catfish or bass)
    - 3/18
  - Assuming so, how likely is it that next species is trout?
    - Must be less than 1/18

[Slide adapted from Josh Goodman]
Good Turing calculations

<table>
<thead>
<tr>
<th>unseen (bass or catfish)</th>
<th>trout</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
</tr>
<tr>
<td>MLE p</td>
<td>( \frac{\text{Count}}{N} = 0 )</td>
</tr>
<tr>
<td>c*</td>
<td>( \frac{\text{Count}}{N} = \frac{1}{2} )</td>
</tr>
</tbody>
</table>

\[ \text{GT: } p_{\text{GT}}(\text{unseen}) = \frac{\text{Count}}{N} = \frac{1}{2} = 0.17 \]

\[ \text{Katz: } p_{\text{Katz}}(\text{trout}) = \frac{\text{Count} + 1}{N + 1} = \frac{1}{3} = 0.33 \]

Good-Turing Reweighting III

- Hypothesis: counts of \( k \) should be \( k^* = \frac{(k+1)N}{k} \)

Katz Smoothing

- Extends G-T smoothing into a backoff model from higher to lower order contexts
- Use G-T discounted bigram counts (roughly – Katz left large counts alone)
- Whatever mass is left goes to empirical unigram

\[ P_{\text{Katz}}(w | w_{-1}) = \frac{\sum \alpha(c(w, w_{-1})) + \alpha(w_{-1}) \hat{P}(w)}{\sum c(w, w_{-1})} \]

Intuition of Katz backoff + discounting

- How much probability to assign to all the zero trigrams?
- Use GT or some other discounting algorithm to tell us
- How do we divide that probability mass among different words in the vocabulary?
  - Use the \((n-1)\)-gram estimates to tell us
- What do we do for the unigram words not seen in training (i.e., not in our vocabulary)
  - The problem of Out Of Vocabulary = OOV words
  - Important, but messy … mentioned at end of class

Kneser-Ney Smoothing I

- Something’s been very broken all this time
  - Shannon game: There was an unexpected ___?
  - Francisco?
  - “Francisco” is more common than “delay”
- Solution: Kneser-Ney smoothing
  - In the back-off model, we don’t want the unigram probability of \( w \)
- Instead, probability given that we are observing a novel continuation
- Every bigram type was a novel continuation the first time it was seen

\[ P(w | w_{-1}) = \frac{\sum c(w, w_{-1}) \alpha(w_{-1}) \hat{P}(w)}{c(w, w_{-1})} \]

Kneser-Ney Smoothing II

- One more aspect to Kneser-Ney:
  - Look at the GT counts:

<table>
<thead>
<tr>
<th>Count in 22M Words</th>
<th>Actual c* (Next 22M)</th>
<th>GT’s c*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.448</td>
<td>0.446</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
<td>1.24</td>
</tr>
<tr>
<td>3</td>
<td>2.23</td>
<td>2.24</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
<td>3.24</td>
</tr>
</tbody>
</table>

- Absolute Discounting
  - Save ourselves some time and just subtract 0.75 (or some \( d \))
  - Maybe have a separate value of \( d \) for very low counts

\[ P_{\text{KNS}}(w | w_{-1}) = \frac{c(w, w_{-1}) - D}{\sum c(w', w_{-1})} + \alpha(w_{-1}) \hat{P}(w) \]

What Actually Works?

- Trigrams:
  - Unigrams, bigrams too little context
  - Trigrams much better (when there’s enough data)
  - 4-, 5-grams usually not worth the cost (which is more than it seems, due to how speech recognizers are constructed)
- Good-Turing-like methods for count adjustment
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell
- Kneser-Ney equalization for lower-order models
  - See [Chen+Goodman] reading for tons of graphs!
Data >> Method?

- Having more data is always good...
- ... but so is picking a better smoothing mechanism!
- $N > 3$ often not worth the cost (greater than you’d think)

Google N-Gram Release

Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 233
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensable 40
- serve as the individual 234

Beyond N-Gram LMs

- Caching Models
  - Recent words more likely to appear again
  - $P_{\text{cache}}(w|\text{history}) = \lambda P(w|\text{w}_{-1:2}) \times (1-\lambda)^{-1}(\text{w} \in \text{history})$
  - Can be disastrous in practice for speech (why?)

- Skipping Models
  - $P_{\text{skip}}(w|\text{w}_{-1:2}) = \lambda P(w|\text{w}_{-1:2}) + \lambda P(w|\text{w}_{-1:2})$

- Clustering Models: condition on word classes when words are too sparse
  - Trigger Models: condition on bag of history words (e.g., sentence)

- Structured Models: use parse structure (we’ll see these later)

- Language Modeling toolkits
  - SRILM
  - CMU-Cambridge LM Toolkit
  - IRST LM Toolkit

Unknown words: Open versus closed vocabulary tasks

- If we know all the words in advance
  - Vocabulary $V$ is fixed
  - Closed vocabulary task: Easy
  - Continue in speech recognition
- Often we don’t know the set of all words
  - 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 $V$ [Can we work out right size for it?]
    - At test normalization phase, any training word not in L changed to <UNK>
- There may be no such instance if L covers the training data
- Now we train its probabilities
  - If few counts are mapped to <UNK>, we may treat it like a normal word
  - Otherwise, techniques like Good-Turing estimation are applicable
- At decoding time
  - If test input: Use UNK probabilities for any word not in training

Practical Considerations

- The unknown word symbol <UNK>
  - In many cases, open vocabularies use multiple types of OOVs (e.g., numbers & proper names)
  - For the programming assignment:
    - OK to assume there is only one unknown word type, UNK
    - UNK be quite common in new text!
    - OK to assume there is only one unknown word type, UNK
  - UNK stands for all unknown word types (define probability event model thus – it is a union of basic outcomes)
  - To model the probability of individual new words occurring, you can use spelling models for them, but people usually don’t

- Numerical computations
  - We usually do everything in log space (log probabilities)
  - Avoid underflow
  - (also adding is faster than multiplying)
  - $p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$