CS5740: Natural Language Processing
Spring 2017

Language Models

Instructor: Yoav Artzi

Slides adapted from Dan Klein, Dan Jurafsky, Chris Manning, Michael Collins, Luke Zettlemoyer, and Yejin Choi

Overview

- The language modeling problem
- N-gram language models
- Evaluation: perplexity
- Smoothing
  - Add-N
  - Linear interpolation

The Language Modeling Problem

- Setup: Assume a (finite) vocabulary of words

$$\mathcal{V} = \{\text{the}, \text{a}, \text{man}, \text{telescope}, \text{Beckham}, \text{two}, \text{Madrid}, \ldots\}$$

- We can construct an (infinite) set of strings

$$\mathcal{V}^{\dagger} = \{\text{the}, \text{a}, \text{the a}, \text{the fan}, \text{the man}, \text{the man with the telescope}, \ldots\}$$

- Data: given a training set of example sentences
- Problem: estimate a probability distribution

$$\sum_{x \in \mathcal{V}^{\dagger}} p(x) = 1$$

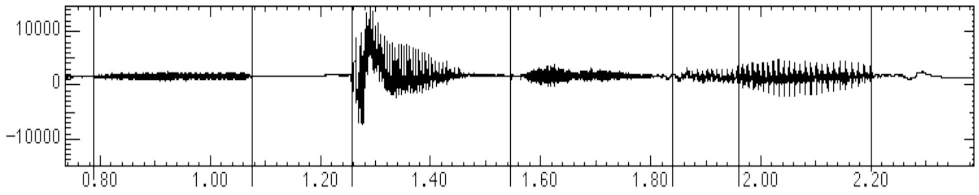
and $p(x) \geq 0$ for all $x \in \mathcal{V}^{\dagger}$

$$p(\text{the}) = 10^{-12}$$
$$p(\text{a}) = 10^{-13}$$
$$p(\text{the fan}) = 10^{-12}$$
$$p(\text{the fan saw Beckham}) = 2 \times 10^{-8}$$
$$p(\text{the fan saw saw}) = 10^{-15}$$
$$\ldots$$

- **Question: why would we ever want to do this?**

Speech Recognition

- Automatic Speech Recognition (ASR)
- Audio in, text out
- SOTA: 0.3% error for digit strings, 5% dictation, 50%+ TV



- “Wreck a nice beach?”
  - “Recognize speech”
- “Eye eight uh Jerry?”
  - “I ate a cherry”

SUPER ANTICS
BANG!
BANG!
BANG!
BANG!
BANG!
YOU-- YOU SAVED MY LIFE!
NONSENSE. I DID NOT SHAVE YOUR WIFE.
WHAT?
dig dig dig
BULLET.
HA HA HA HA HA HA HA HA
CALLEN

The Noisy Channel Model

- Goal: predict sentence given acoustics

$$w^* = \arg\max_X P(X \mid a)$$

- The noisy channel approach:

Language model: Distributions over sequences of words (sentences)

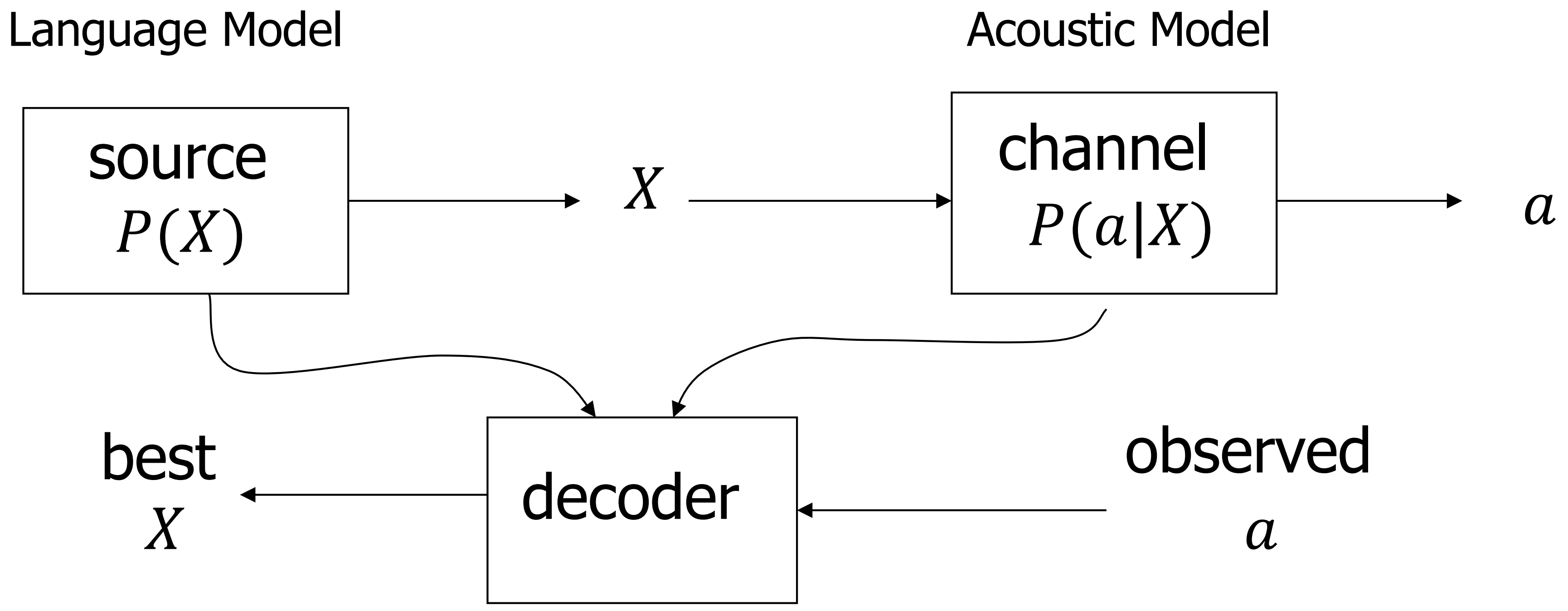
$$\begin{aligned} w^* &= \arg\max_X P(X \mid a) \\ &= \arg\max_X P(a \mid X) P(X) / P(a) \\ &= \arg\max_X P(a \mid X) P(X) \end{aligned}$$

Acoustic model: Distributions over acoustic waves given a sentence

Acoustically Scored Hypotheses

the station signs are in deep in english	-14732
the stations signs are in deep in english	-14735
the station signs are in deep into english	-14739
the station 's signs are in deep in english	-14740
the station signs are in deep in the english	-14741
the station signs are indeed in english	-14757
the station 's signs are indeed in english	-14760
the station signs are indians in english	-14790
the station signs are indian in english	-14799
the stations signs are indians in english	-14807
the stations signs are indians and english	-14815

ASR System Components



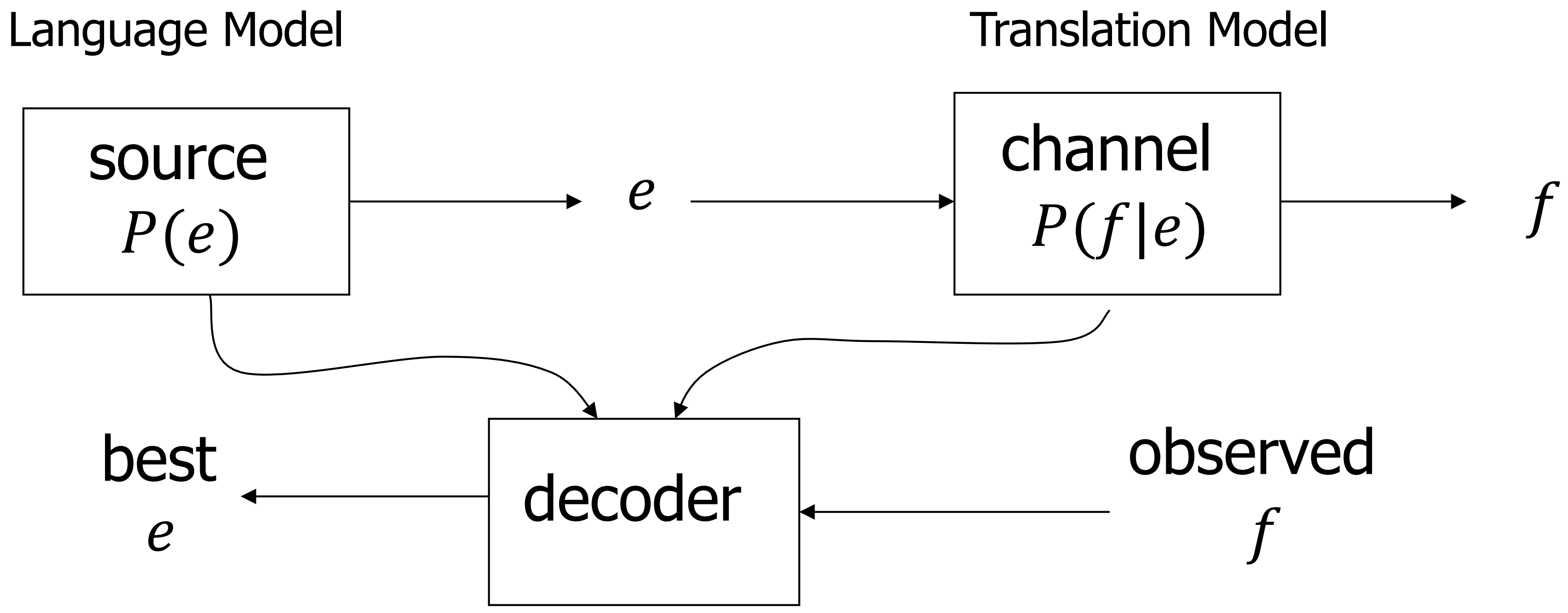
$$\arg\max_X P(X \mid a) = \arg\max_X P(a \mid X) P(X)$$

Translation as Codebreaking

"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.'** "

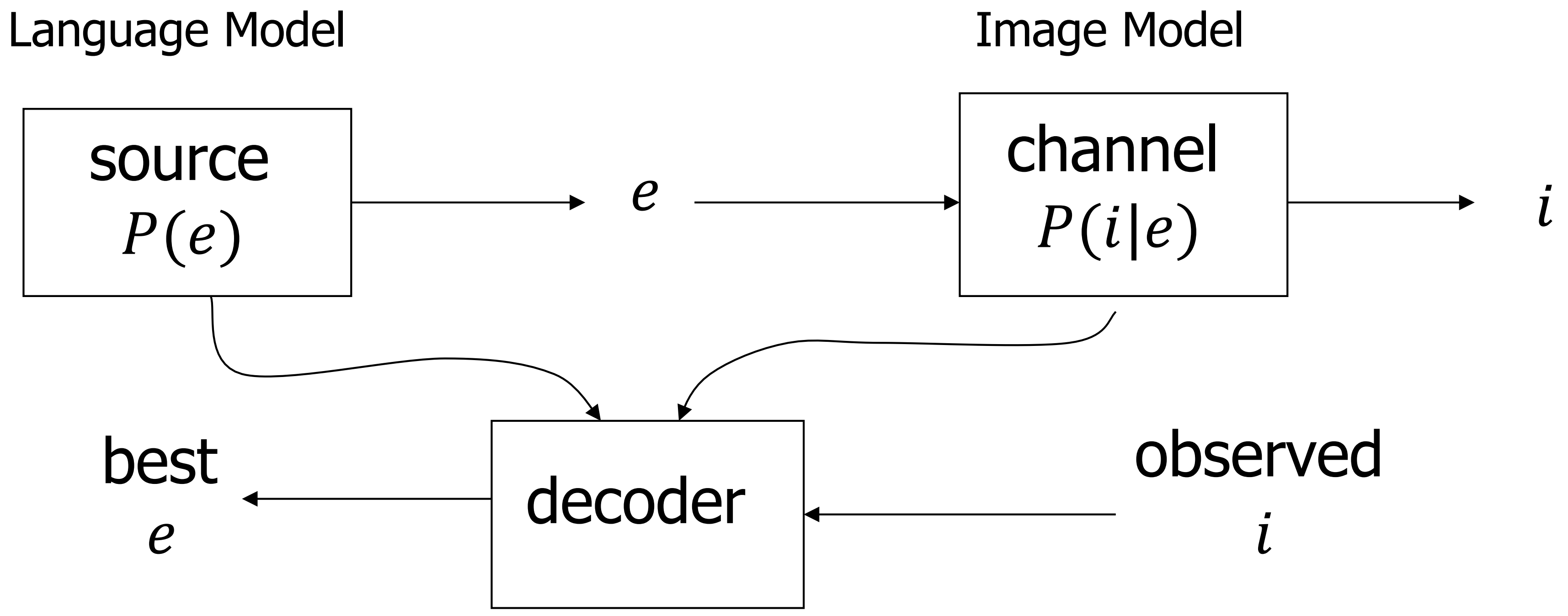
Warren Weaver
(1955:18, quoting a letter he wrote in 1947)

MT System Components



$$\arg\max_e P(e \mid f) = \arg\max_e P(f \mid e) P(e)$$

Caption Generation System Components



$$\arg\max_e P(e \mid i) = \arg\max_e P(i \mid e) P(e)$$

Learning Language Models

- Goal: Assign useful probabilities $P(X)$ to sentences X
  - Input: many observations of training sentences X
  - Output: system capable of computing $P(X)$

- Probabilities should broadly indicate plausibility of sentences
  - $P($I saw a van$)$ >> $P($eyes awe of an$)$
  - Not only grammaticality: $P($artichokes intimidate zippers$) \approx 0$
  - In principle, "plausible" depends on the domain, context, speaker…

- One option: empirical distribution over training sentences…

$$p(x_1 \ldots x_n) = \frac{c(x_1 \ldots x_n)}{N} \text{ for sentence } X = x_1 \ldots x_n$$

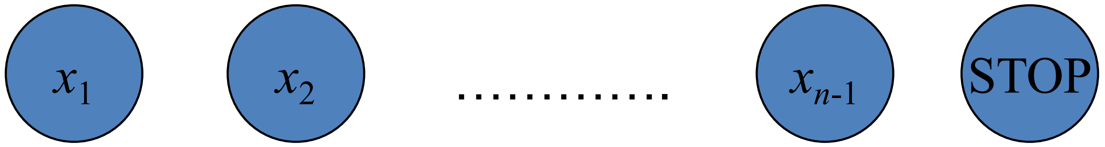
- Problem: does not generalize (at all)
- Need to assign non-zero probability to previously unseen sentences!

Unigram Models

- Simplest solution: unigrams

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} p(x_i)$$

- Generative process: pick a word, pick a word, … until you pick STOP
- As a graphical model:



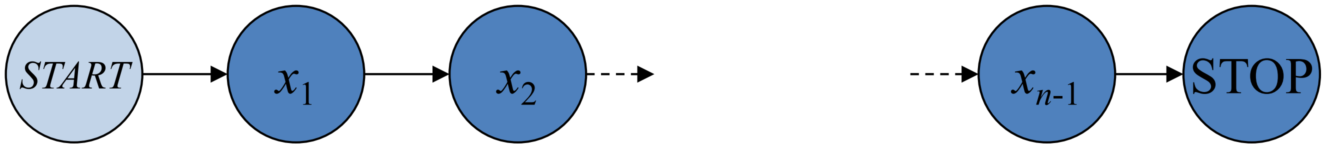
- Examples:
  - [fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass.]
  - [thrift, did, eighty, said, hard, 'm, july, bullish]
  - [that, or, limited, the]
  - []
  - [after, any, on, consistently, hospital, lake, of, of, other, and, factors, raised, 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]
- Big problem with unigrams: P(the the the the) >> P(I like ice cream)!

Bigram Models

- Condition on previous single word:

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} p(x_i | x_{i-1})$$

- Generative process:
  - pick START, pick a word conditioned on previous one, repeat until to pick STOP
- Graphical Model:



- Any better?
  - [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]
  - [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, believe, chemical, prices, undoubtedly, will, be, as, much, is, scheduled, to, conscientious, teaching]
  - [this, would, be, a, record, november]
- But, what is the cost?

Higher Order N-grams?

Please close the door

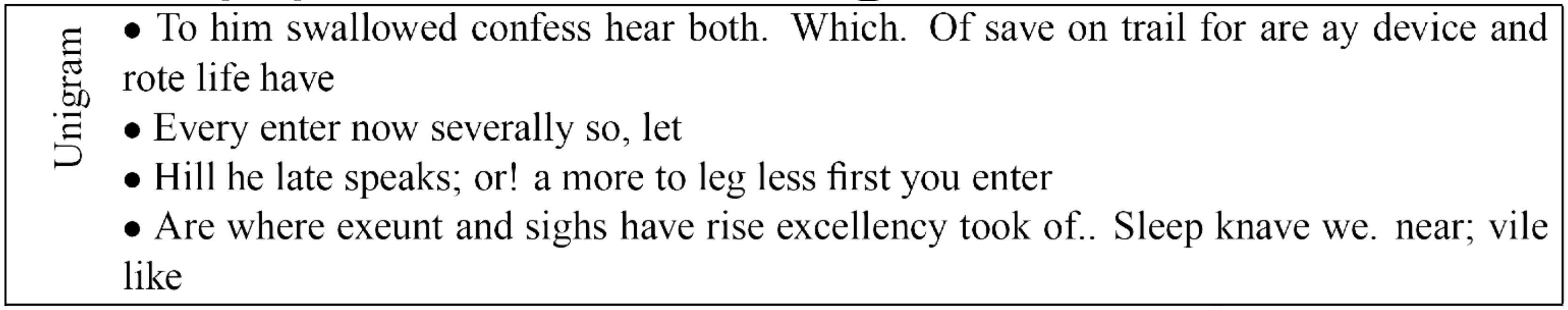
Please close the first window on the left

```
198015222 the first
194623024 the same
168504105 the following
158562063 the world
…
14112454 the door
-----------------
23135851162 the *
```

```
197302 close the window
191125 close the door
152500 close the gap
116451 close the thread
87298 close the deal
-----------------
3785230 close the *
```

```
3380 please close the door
1601 please close the window
1164 please close the new
1159 please close the gate
…
0 please close the first
-----------------
13951 please close the *
```

Approximating ________



Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Markov Assumption

- Simplifying assumption:

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

- Or if n=2:

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

N-gram Model Decomposition

- k-gram models ($k > 1$): condition on $k - 1$ previous words

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-(k-1)} \dots x_{i-1})$$

where $x_i \in \mathcal{V} \cup \{STOP\}$ and $x_{-k+2} \dots x_0 = *$

- Example: tri-gram (3-gram)

$$p(\text{the dog barks STOP}) =$$

$$q(\text{the}|*, *) \times q(\text{dog}|*, \text{the}) \times q(\text{barks}|\text{the, dog}) \times q(\text{STOP}|\text{dog, barks})$$

- Learning: estimate the distributions

Well Defined Distributions

Proof for Unigrams

- Simplest case: unigrams

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} p(x_i)$$

- Generative process: pick a word, pick a word, … until you pick STOP
- For all strings X (of any length): $p(X) \geq 0$
- Claim: the sum over string of all lengths is 1 : $\sum_X p(X) = 1$

(1) $$\sum_X p(X) = \sum_{n=1}^{\infty} \sum_{x_1 \dots x_n} p(x_1 \dots x_n)$$

(2) $$\sum_{x_1 \dots x_n} p(x_1 \dots x_n) = \sum_{x_1 \dots x_n} \prod_{i=1}^{n} p(x_i) = \sum_{x_1} \dots \sum_{x_n} p(x_1) \times \dots \times p(x_n)$$

$$= \sum_{x_1} p(x_1) \times \dots \times \sum_{x_n} p(x_n) = (1 - p_s)^{n-1} p_s \quad \text{where } p_s = p(\text{STOP})$$

(1)+(2) $$\sum_x p(x) = \sum_{n=1}^{\infty} (1 - p_s)^{n-1} p_s = p_s \sum_{n=1}^{\infty} (1 - p_s)^{n-1} = p_s \frac{1}{1 - (1 - p_s)} = 1$$

N-gram Model Parameters

- The parameters of an n-gram model:
  - Maximum likelihood estimate: relative frequency

$$q_{ML}(w) = \frac{c(w)}{c()}, \quad q_{ML}(w|v) = \frac{c(v,w)}{c(v)}, \quad q_{ML}(w|u,v) = \frac{c(u,v,w)}{c(u,v)}, \quad \ldots$$

where c is the empirical counts on a training set

- General approach
  - Take a training set D and a test set D'
  - Compute an estimate of the q's from D
  - Use it to assign probabilities to other sentences, such as those in D'

Training Counts

198015222 the first
194623024 the same
168504105 the following
158562063 the world
…
14112454 the door

23135851162 the *

$$q(\text{door}|\text{the}) = \frac{14112454}{2313581162}$$

$$= 0.0006$$

Regular Languages?

- N-gram models are (weighted) regular languages
  - Many linguists argue that language isn't regular.
    - Long-distance effects: "The computer which I had just put into the machine room on the fifth floor ___."
    - Recursive structure
  - Why **CAN** we often get away with n-gram models?

Measuring Model Quality

- The goal isn't to pound out fake sentences!
  - Obviously, generated sentences get "better" as we increase the model order
  - More precisely: using ML estimators, higher order always gives better likelihood on train, but not test

- What we really want to know is:
  - Will our model prefer good sentences to bad ones?
  - Bad $\neq$ ungrammatical!
  - Bad $\approx$ unlikely
  - Bad = sentences that our acoustic model really likes but aren't the correct answer

Measuring Model Quality

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

    When I eat pizza, I wipe off the ____

    Many children are allergic to ____

    I saw a ____

  - Unigrams are terrible at this game. (Why?)

grease	0.5
sauce	0.4
dust	0.05
….	
mice	0.0001
….	
the	1e-100

Claude Shannon

- A better model of a text…
  - is one which assigns a higher probability to the word that actually occurs

Measuring Model Quality

- For every sentences $X^{(i)}$ $(i = 1 \dots m)$ we can estimate its probability $p(X^{(i)})$
- A natural measure of model quality:

$$\prod_{i=1}^{m} p(X^{(i)})$$

- The higher this quantity is, the better we model unseen sentences

Perplexity

- Let M be the number of words in the corpus
- The average log probability is:

$$\frac{1}{M}\log_2\prod_{i=1}^{m}p(X^{(i)})=\frac{1}{M}\sum_{i=1}^{m}\log_2 p(X^{(i)})$$

- The perplexity is:

$$PP=2^{-l}$$

- Where:

$$l=\frac{1}{M}\sum_{i=1}^{m}\log_2 p(X^{(i)})$$

Perplexity

$$PP = 2^{-l} \qquad l = \frac{1}{M}\sum_{i=1}^{m} \log_2 p(X^{(i)})$$

- Lower is better!
- Perplexity is the inverse probability of the test set normalized by the number of words
- If we ever give a test n-gram zero probability → perplexity will be infinity
- How can we avoid this?

Perplexity

$$PP = 2^{-l} \qquad l = \frac{1}{M}\sum_{i=1}^{m} \log_2 p(X^{(i)})$$

- Under a uniform distribution the perplexity will be the vocabulary size:
  - Let's suppose M sentences consisting of random digits
  - What is the perplexity of this data according to a model that assign P=1/10 to each digit?

$$\begin{aligned} PP &= 2^{-\frac{1}{M}\sum_{i=1}^{m}\log_2(\frac{1}{10})^{|X^{(i)}|}} \\ &= 2^{-\frac{1}{M}\sum_{i=1}^{m}|X^{(i)}|\log_2\frac{1}{10}} \\ &= 2^{-\log_2\frac{1}{10}} = 2^{-\log_2 10^{-1}} = 10 \end{aligned}$$

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, 20k word types WSJ

N-gram Order	Unigram	Bigram	Trigram*
Perplexity	962	170	109

- "An Estimate of an Upper Bound for the Entropy of English". Brown, Peter F.; et al. (March 1992). *Computational Linguistics* **18** (1)

- Important note:
  - It's easy to get bogus perplexities by having bogus probabilities that sum to more than one over their event spaces.

*With Good-Turing smoothing

Shannon Game

Shannon Game

https://goo.gl/yDUYHm

Measuring Model Quality: Speech

- Word Error Rate (WER)

$$\frac{\text{insertions} + \text{deletions} + \text{substitutions}}{\text{true sentence size}}$$

Correct answer: Andy saw a part of the movie

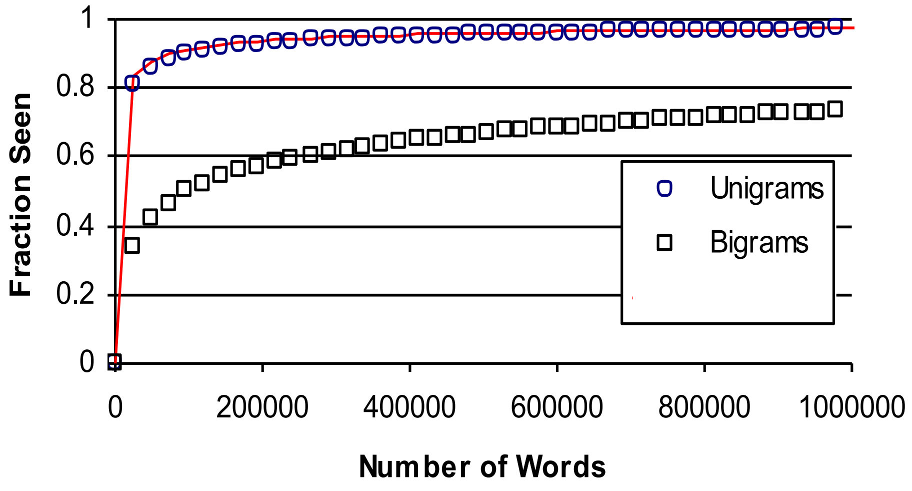
Recognizer output: And he saw apart of the movie

WER: 4/7
= 57%

- The “right” measure:
  - Task error driven
  - For speech recognition
  - For a specific recognizer!

- Common issue: intrinsic measures like perplexity are easier to use, but extrinsic ones are more credible

Sparsity in Language Models



- Problems with n-gram models:
  - New words appear all the time:
    - Synaptitute
    - 132,701.03
    - Multidisciplinarization
    - Post-truth
  - New n-grams: even more often
- 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
  - Not special to language: randomly generated character strings have this property (try it!)
- This is particularly problematic when…
  - Training set is small (does this happen for language modeling?)
  - Transferring domains: e.g., newswire, scientific literature, Twitter

Zeroes

- Training set:
  … denied the allegations
  … denied the reports
  … denied the claims
  … denied the request

- Test set:
  … denied the offer
  … denied the loan

P("offer" | denied the) = 0

- Bigrams with zero probability
  – Mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!

Parameter Estimation

- The parameters of an n-gram model:
  - Maximum likelihood estimate: relative frequency

$$q_{ML}(w) = \frac{c(w)}{c()}, \quad q_{ML}(w|v) = \frac{c(v,w)}{c(v)}, \quad q_{ML}(w|u,v) = \frac{c(u,v,w)}{c(u,v)}, \quad \ldots$$

where c is the empirical counts on a training set

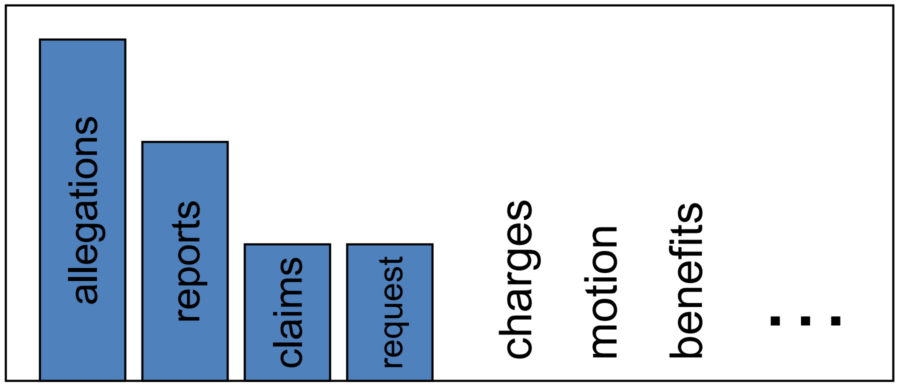
- Maximum likelihood estimates won’t get us very far
- Need to smooth these estimates
- General method (procedurally)
  - Take your empirical counts
  - Modify them in various ways to improve estimates
- General method (mathematically)
  - Often can give estimators a formal statistical interpretation … but not always
  - Approaches that are mathematically obvious don’t always what works

Smoothing

- We often want to make estimates from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request

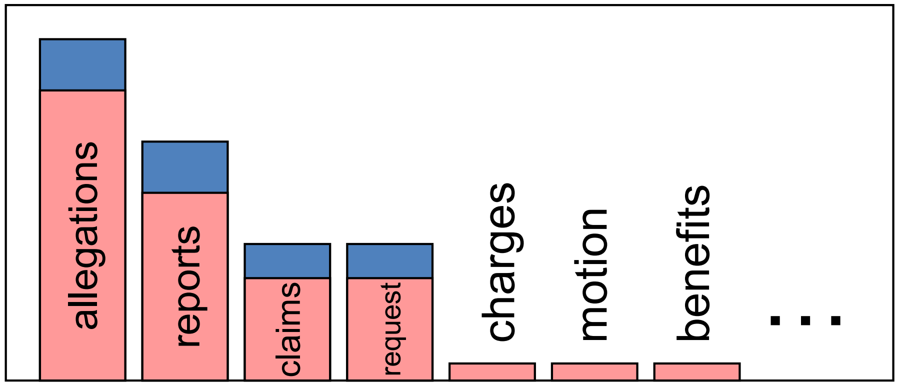
7 total



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other

7 total



- Very important all over NLP (and ML more generally), but easy to do badly!
- Question: what is the best way to do it?

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!

- MLE estimate:

$$P_{\text{MLE}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i)}{c(x_{i-1})}$$

- Add-1 estimate:

$$P_{\text{Add-1}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + 1}{c(x_{i-1}) + V}$$

More General Formulation

- Add-K:

$$P_{\text{Add-k}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + k}{c(x_{i-1}) + kV}$$

$$P_{\text{Add-k}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + m\frac{1}{V}}{c(x_{i-1}) + m}$$

- Unigram Prior Smoothing:

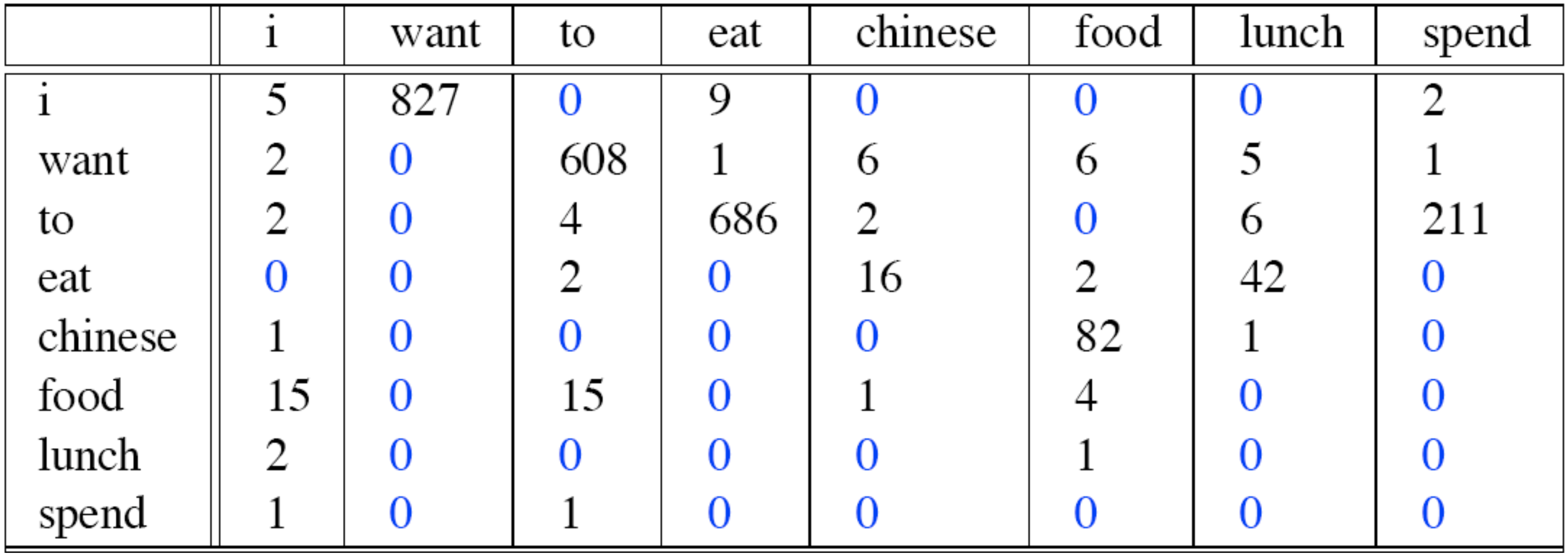
$$P_{\text{Add-k}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + mP(x_i)}{c(x_{i-1}) + m}$$

Berkeley Restaurant Corpus

- 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

- From 9222 sentences



	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Bigram Probabilities

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

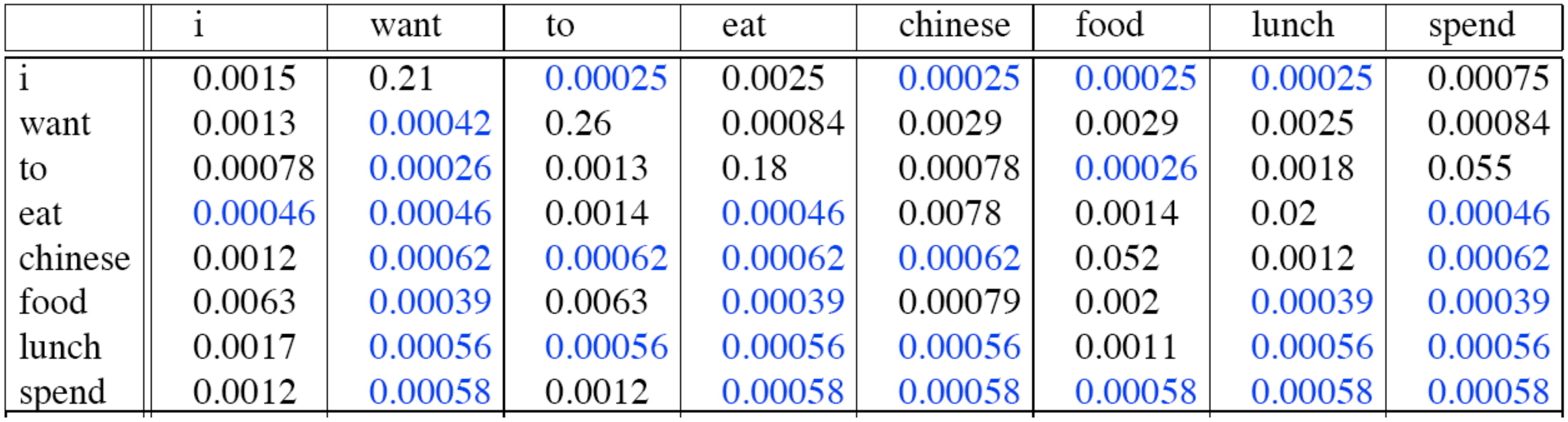
	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Add-1 on the Berkeley Restaurant Corpus

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Add-1 Smoothed Bigrams

$$P_{\text{Add-1}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + 1}{c(x_{i-1}) + V}$$



	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted Counts

$$P_{\text{Add-1}}(x_i \mid x_{i-1}) = \frac{c(x_{i-1}, x_i) + 1}{c(x_{i-1}) + V}$$

$$c^*(x_{i-1}, x_i) = \frac{(c(x_{i-1}, x_i) + 1)c(x_{i-1})}{c(x_{i-1}) + V}$$

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

Original vs. Add-1 (reconstituted) Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

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

Add-1 is a Blunt Instrument

- So Add-1 isn’t used for N-grams:
  - We’ll see better
- But Add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeroes isn’t so big

Add-1 Smoothing

- Classic solution: add counts (Laplace smoothing)

$$P_{\text{Add-}\delta}(w) = \frac{c(w) + \delta}{\sum_{w'} (c(w') + \delta)} = \frac{c(w) + \delta}{c() + \delta V}$$

  - Add-one smoothing especially often talked about
- For a bigram distribution, can add counts shaped like the unigram:

$$P_{\text{Uni-}\delta}(w|v) = \frac{c(v, w) + \delta q_{ML}(w)}{\sum_{w'} (c(v, w') + \delta q_{ML}(w'))} = \frac{c(v, w) + \delta P_{\text{MLE}}(w)}{c(v) + \delta}$$

- Can consider hierarchical formulations: trigram is recursively centered on smoothed bigram estimate, etc. [MacKay and Peto, 94]

- Bayesian: Can be derived from Dirichlet / multinomial conjugacy - prior shape shows up as pseudo-counts

- **Problem: works quite poorly!**

Linear Interpolation

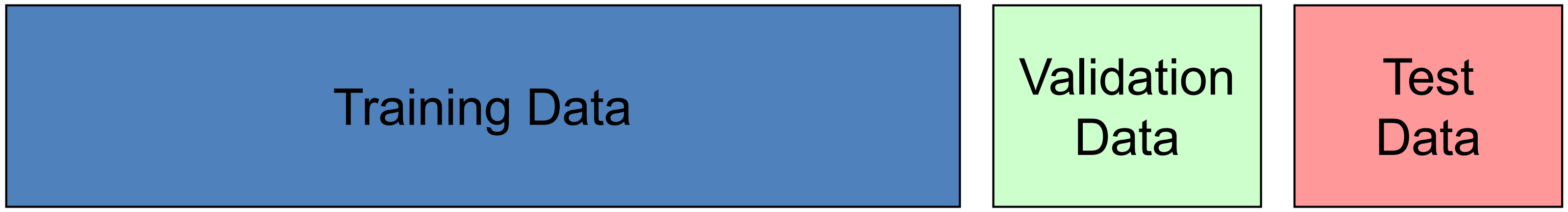
- Problem: MLE is supported by few counts
- Classic solution: mixtures of related, denser histories:

$$P_\lambda(w|u,v) = \lambda_3 P_{\text{MLE}}(w|u,v) + \lambda_2 P_{\text{MLE}}(w|v) + \lambda_1 P_{\text{MLE}}(w)$$

- Is this a well defined distribution?
  - Yes, if all $\lambda_i \geq \mathbf{0}$ and they sum to 1
- The mixture approach tends to work better than Add-δ approach
  - Can flexibly include multiple back-off contexts
  - Good ways of learning the mixture weights with EM (later)
  - But: not entirely clear why it works so much better
- All the details you could ever want: [Chen and Goodman, 98]

Estimating Lambdas

- Use a **validation** corpus



- Choose λs to maximize the probability of validation data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to validation set:

$$\log P(x_1, \ldots, x_n \mid \lambda_1, \ldots, \lambda_k) = \sum_i \log P_\lambda(x_i \mid x_{i-1})$$

Advanced Smoothing Algorithms

- Intuition: Use the count of things we've **seen once**
  - To help estimate the count of things we've **never seen**
- Used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Also: Witten-Bell

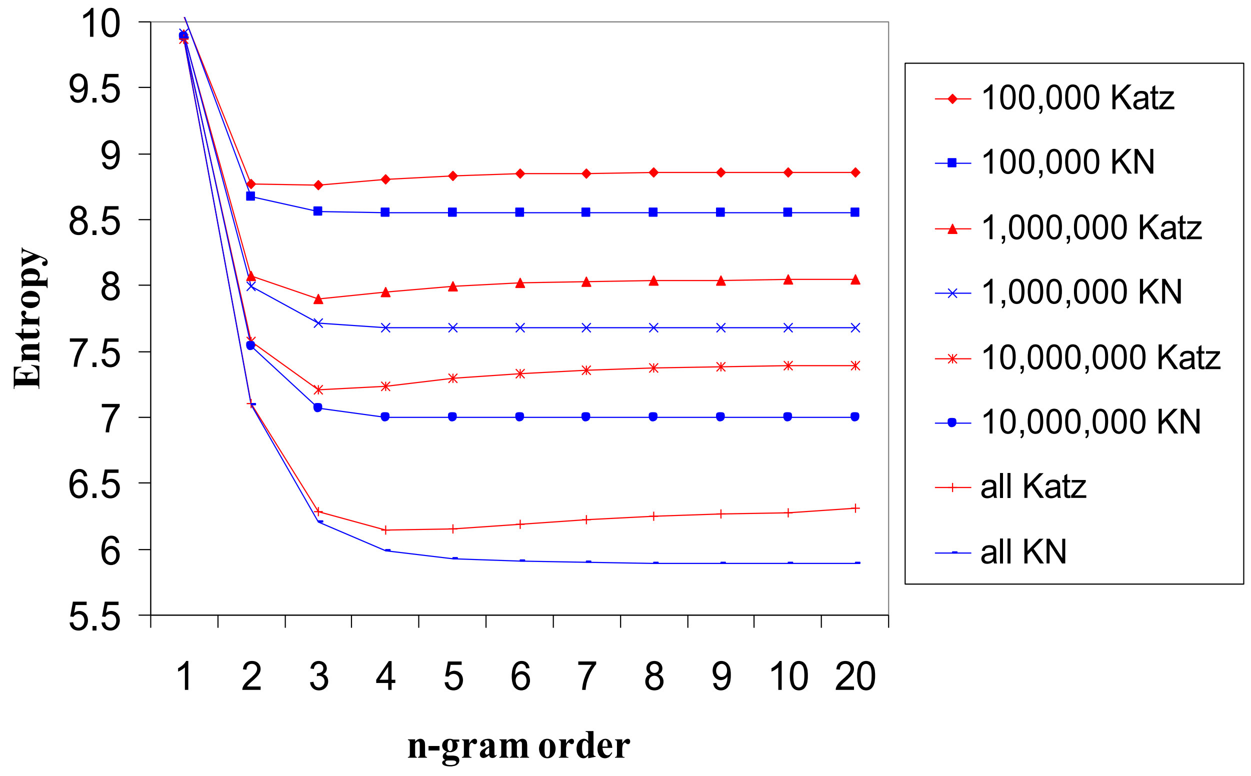
Invented during WWII by Alan Turing and later published by Good. Frequency estimates were needed for Enigma code-breaking effort

What Actually Works?

- Trigrams and beyond:
  - Unigrams, bigrams generally useless
  - Trigrams much better (when there's enough data)
  - 4-, 5-grams really useful in MT, but not so much for speech
- Discounting
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell, etc…
- See [Chen+Goodman] reading for tons of graphs…

Data vs. Method?

- Having more data is better…



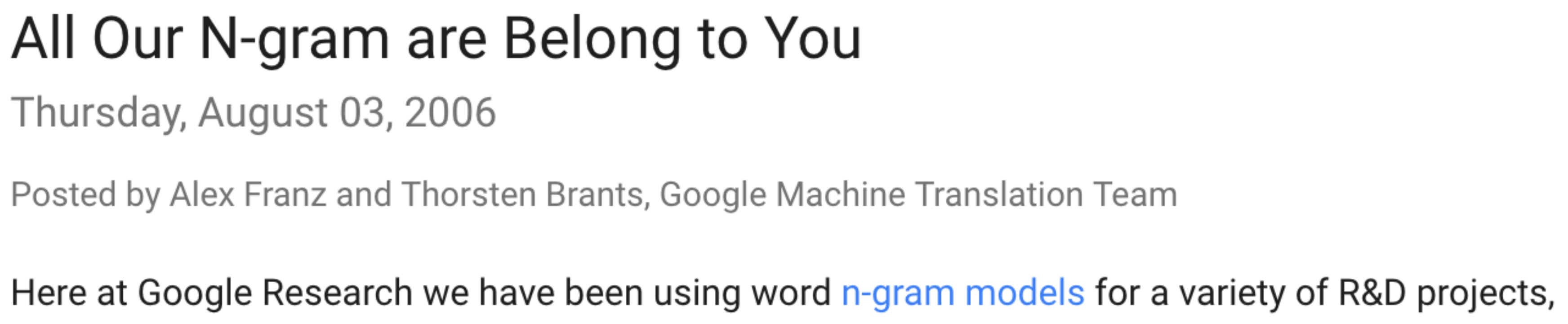
- … but so is using a better estimator
- Another issue: $N > 3$ has huge costs in speech recognizers

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)
  - (though log can be slower than multiplication)

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

Web-scale N-grams



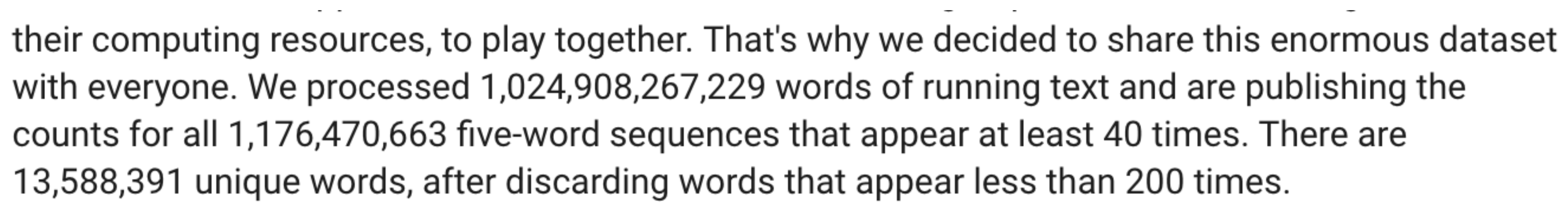
All Our N-gram are Belong to You

Thursday, August 03, 2006

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,

…



their computing resources, to play together. 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-grams

- `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`

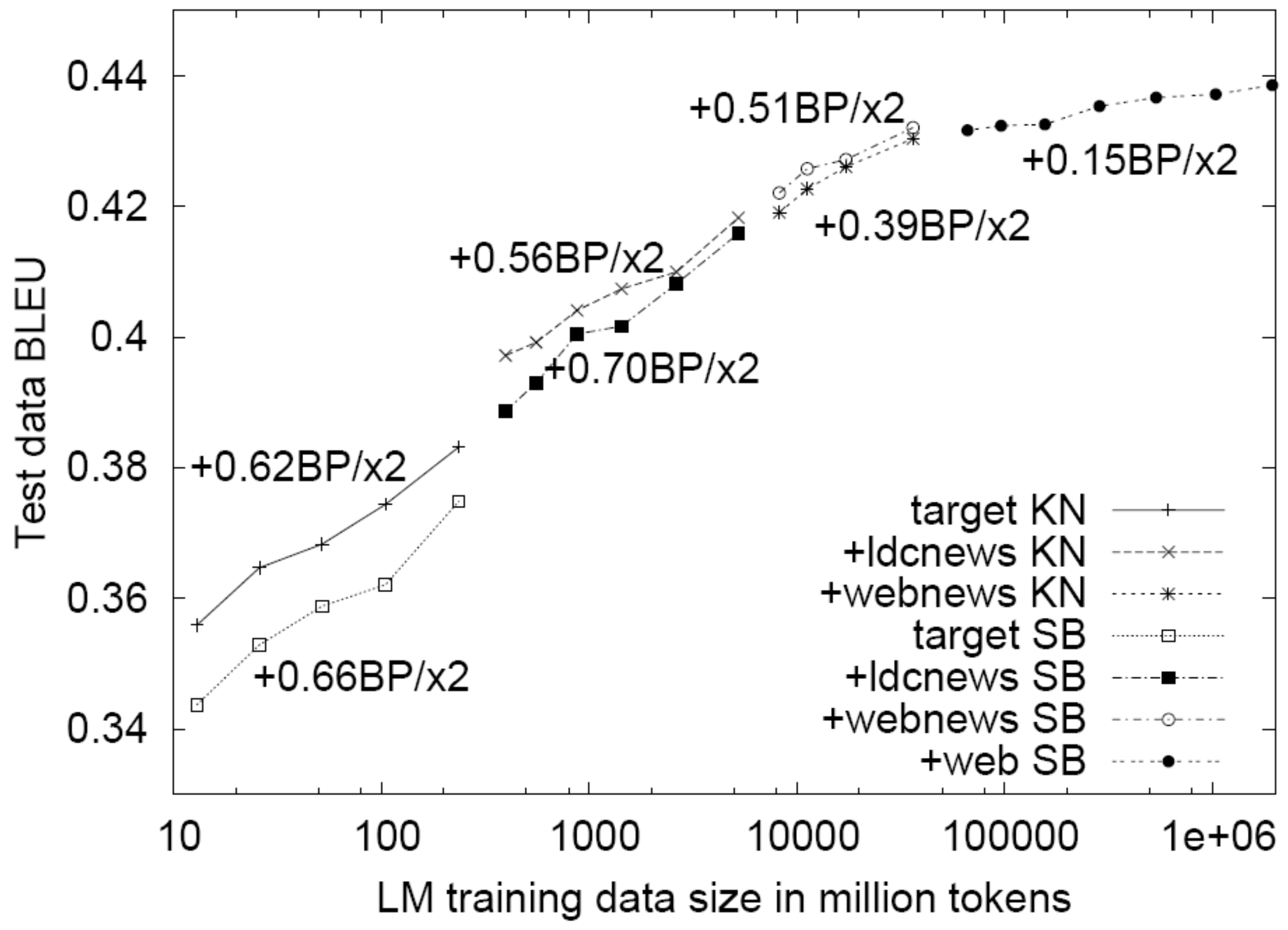
http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

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 (more advanced)
- 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)

Even More Data!

Tons of data closes gap, for extrinsic MT evaluation



http://www.aclweb.org/anthology/D07-1090.pdf

Handling Unknown Words

- 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

Beyond N-gram LMs

- Lots of ideas we won't have time to discuss:
  - Caching models: recent words more likely to appear again
  - Trigger models: recent words trigger other words
  - Topic models
- More advanced ideas
  - Syntactic models: use tree models to capture long-distance syntactic effects [Chelba and Jelinek, 98]
  - Discriminative models: set n-gram weights to improve final task accuracy rather than fit training set density [Roark, 05, for ASR; Liang et. al., 06, for MT]
  - Structural zeroes: some n-grams are syntactically forbidden, keep estimates at zero [Mohri and Roark, 06]
  - Bayesian document and IR models [Daume 06]

Case Study: Language Identification

- How can we tell what language a document is in?

The 38th Parliament will meet on Monday, October 4, 2004, at 11:00 a.m. The first item of business will be the election of the Speaker of the House of Commons. Her Excellency the Governor General will open the First Session of the 38th Parliament on October 5, 2004, with a Speech from the Throne.

La 38e législature se réunira à 11 heures le lundi 4 octobre 2004, et la première affaire à l'ordre du jour sera l’élection du président de la Chambre des communes. Son Excellence la Gouverneure générale ouvrira la première session de la 38e législature avec un discours du Trône le mardi 5 octobre 2004.

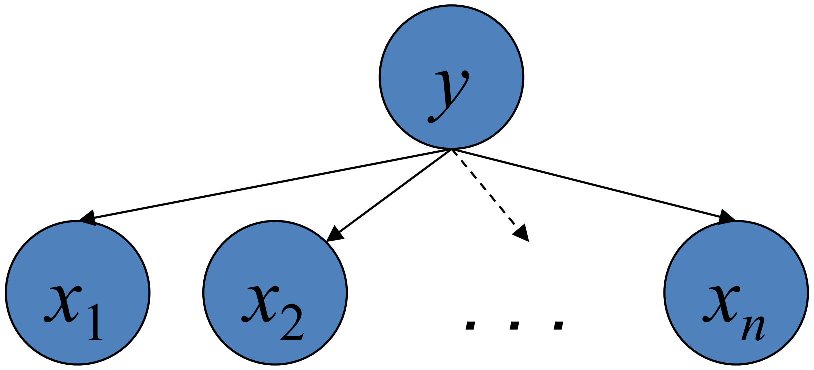
- How to tell the French from the English?
  - Treat it as word-level text categorization?
  - Overkill, and requires a lot of training data
    - You don’t actually need to know about words!
- Option: build a character-level language model

Σύμφωνο σταθερότητας και ανάπτυξης
Patto di stabilità e di crescita

Naïve-Bayes Models

- Generative model: pick a topic, then generate a document using a language model for that topic
- Naïve-Bayes assumption:
  - All words are independent given the topic.

$$p(y, X) = q(y) \prod_{i=1}^{|X|} q(x_i \mid y)$$



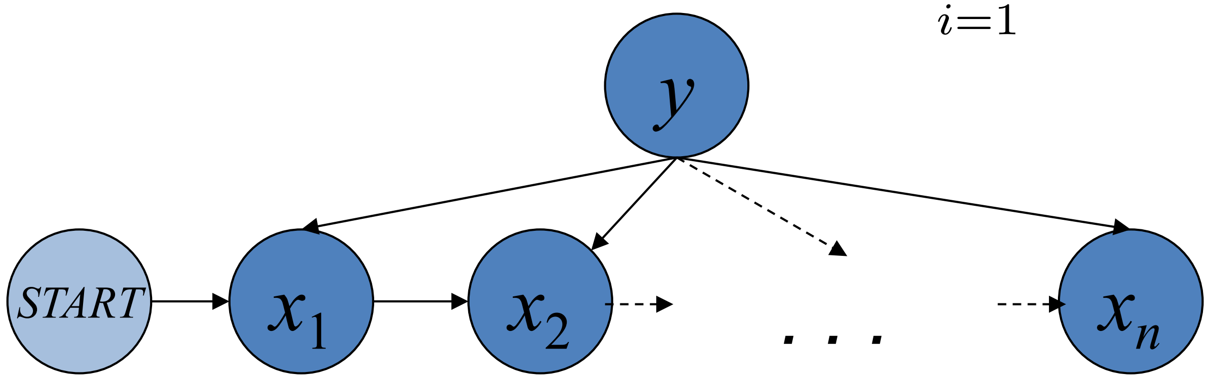
- Compare to a unigram language model:

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} p(x_i)$$

One Option: Class-Conditional LMs

- Can add a topic variable to richer language models

$$p(y, X) = q(y) \prod_{i=1}^{|X|} q(x_i \mid y, x_{i-1})$$



- Could be characters instead of words, used for language ID
- Could sum out the topic variable and use as a language model
- How might a class-conditional n-gram language model behave differently from a standard n-gram model?

EXTRAS

Notation: N_c = Frequency of frequency c

- N_c = the count of things we've seen c times
- Sam I am I am Sam I do not eat

I	3
sam	2
am	2
do	1
not	1
eat	1

$N_1 = 3$

$N_2 = 2$

$N_3 = 1$

Good-Turing Smoothing Intuition

- You are fishing (a scenario from Josh Goodman), and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next species is trout?
  - 1/18
- How likely is it that next species is new (i.e. catfish or bass)
  - Let's use our estimate of things-we-saw-once to estimate the new things.
  - 3/18 (because N_1=3)
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18
  - How to estimate?

Good-Turing Calculations

$$P^{*}_{GT}(\text{things with zero frequency}) = \frac{N_1}{N} \qquad c^{*} = \frac{(c+1)N_{c+1}}{N_c}$$

Unseen (bass or catfish)

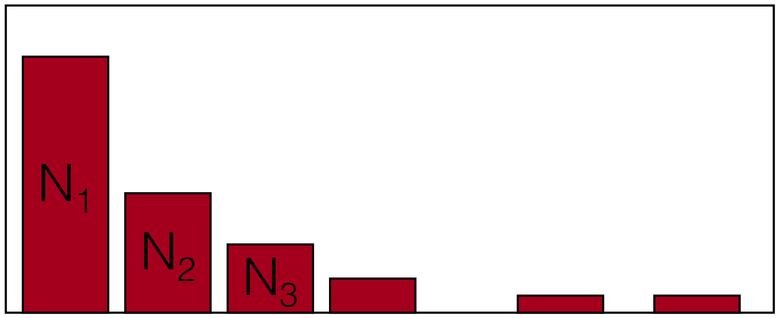
- $c = 0$:
- MLE $p = 0/18 = 0$
- P^{*}_{GT} (unseen) $= N_1/N = 3/18$

Seen once (trout)

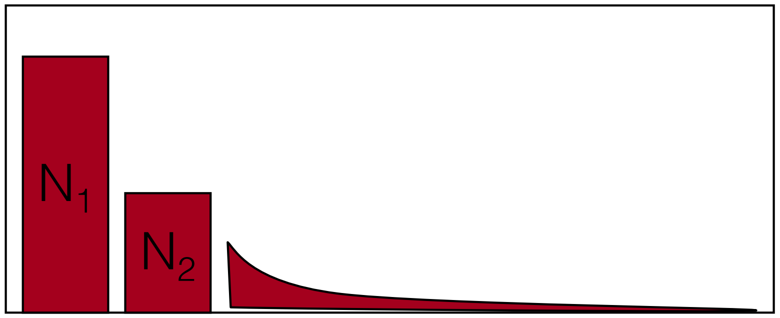
- $c = 1$
- MLE $p = 1/18$
- $C^{*}(\text{trout}) = 2 * N_2/N_1 = 2 * 1/3 = 2/3$
- $P^{*}_{GT}(\text{trout}) = 2/3 / 18 = 1/27$

Good-Turing Complications

- Problem: what about "the"? (say c=4417)
  - For small k, $N_k > N_k+1$
  - For large k, too jumpy, zeroes wreck estimates



  - Simple Good-Turing [Gale and Sampson]: replace empirical N_k with a best-fit power law once counts get unreliable



Good-Turing Numbers

- Numbers from Church and Gale (1991)
- 22 million words of AP Newswire

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

- It sure looks like
  c* = (c - .75)

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Absolute Discounting

- Idea: observed n-grams occur more in training than they will later:

Count in 22M Words	Future c* (Next 22M)
1	0.448
2	1.25
3	2.24
4	3.23

- Absolute Discounting (Bigram case)
  - No need to actually have held-out data; just subtract 0.75 (or some d)

$$c^*(v,w) = c(v,w) - 0.75 \text{ and } q(w|v) = \frac{c^*(v,w)}{c(v)}$$

  - But, then we have "extra" probability mass

$$\alpha(v) = 1 - \sum_w \frac{c^*(v,w)}{c(v)}$$

  - Question: How to distribute α between the unseen words?

Katz Backoff

- Absolute discounting, with backoff to unigram estimates

$$c^*(v,w) = c(v,w) - \beta \qquad \alpha(v) = 1 - \sum_w \frac{c^*(v,w)}{c(v)}$$

- Define seen and unseen bigrams:

$$\mathcal{A}(v) = \{w : c(v,w) > 0\} \quad \mathcal{B}(v) = \{w : c(v,w) = 0\}$$

- Now, backoff to maximum likelihood unigram estimates for unseen bigrams

$$q_{BO}(w|v) = \begin{cases} \frac{c^*(v,w)}{c(v)} & \text{If } w \in \mathcal{A}(v) \\ \alpha(v) \times \frac{q_{ML}(w)}{\sum_{w' \in \mathcal{B}(v)} q_{ML}(w')} & \text{If } w \in \mathcal{B}(v) \end{cases}$$

- Can consider hierarchical formulations: trigram is recursively backed off to Katz bigram estimate, etc
- Can also have multiple count thresholds (instead of just 0 and >0)
- Problem?
  - Unigram estimates are bad predictors

Kneser-Ney Smoothing

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: I can't see without my reading ___*Francisco* *glasses*___?
  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"
- Instead of P(w): "How likely is w"
- $P_{\text{continuation}}(w)$: "How likely is w to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

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

Kneser-Ney Smoothing

- How many times does w appear as a novel continuation:

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

- Normalized by the total number of word bigram types

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

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

Kneser-Ney Smoothing

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability
- Replace unigram in discounting:

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

λ is a normalizing constant; the probability mass we've discounted

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

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing: Recursive Formulation

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

$$c_{KN}(\bullet) = \begin{cases} count(\bullet) & \text{for the highest order} \\ continuationcount(\bullet) & \text{for lower order} \end{cases}$$

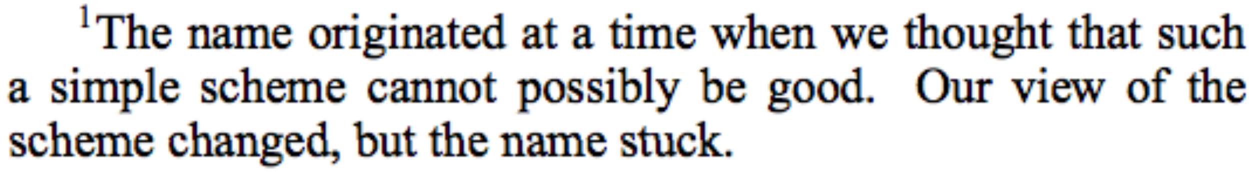
Continuation count = Number of unique single word contexts for •

Smoothing at Web-scale

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

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$



[1]The name originated at a time when we thought that such a simple scheme cannot possibly be good. Our view of the scheme changed, but the name stuck.