CSE 447/547
Natural Language Processing
Winter 2020

Language Models

Yejin Choi

Slides adapted from Dan Klein, Michael Collins, Luke Zettlemoyer, Dan Jurafsky

Overview

- The language modeling problem
- N-gram language models
- Evaluation: perplexity
- Smoothing
  - Add-N
  - Linear Interpolation
  - Discounting Methods

The Language Modeling Problem

- Setup: Assume a (finite) vocabulary of words

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

- We can construct an (infinite) set of strings

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

- Data: given a *training set* of example sentences $x \in \mathcal{V}^\dagger$
- Problem: estimate a probability distribution

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

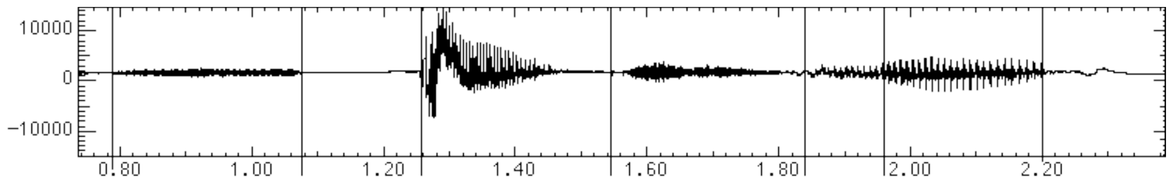
and $p(x) \geq 0$ for all $x \in \mathcal{V}^\dagger$

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

- 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"
- "I ate a cherry"

"Eye eight uh Jerry?"

The Noisy-Channel Model

- We want to predict a sentence given acoustics:

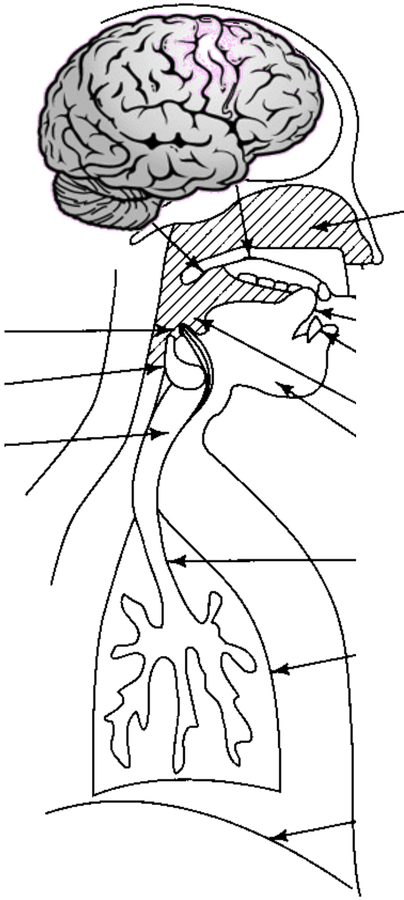
$$w^* = \arg\max_w P(w|a)$$

- The noisy channel approach:

$$\begin{aligned} w^* &= \arg\max_w P(w|a) \\ &= \arg\max_w P(a|w)P(w)/P(a) \\ &\propto \arg\max_w P(a|w)P(w) \end{aligned}$$

Acoustic model:
Distributions over acoustic waves given a sentence

Language model:
Distributions over sequences of words (sentences)

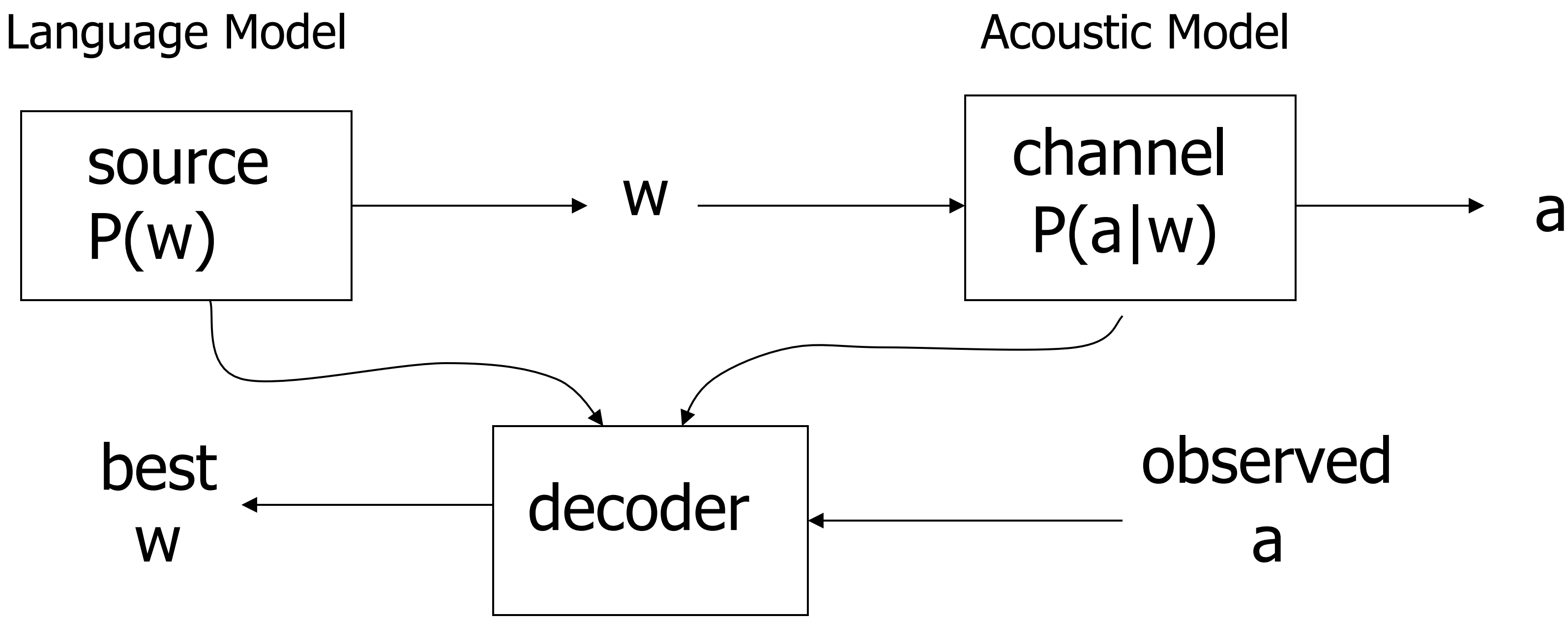


Acoustically Scored Hypotheses

$$\propto \arg\max_{w} \boxed{P(a|w)} P(w)$$

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



$$\underset{w}{\operatorname{argmax}} P(w|a) = \underset{w}{\operatorname{argmax}} P(a|w)P(w)$$

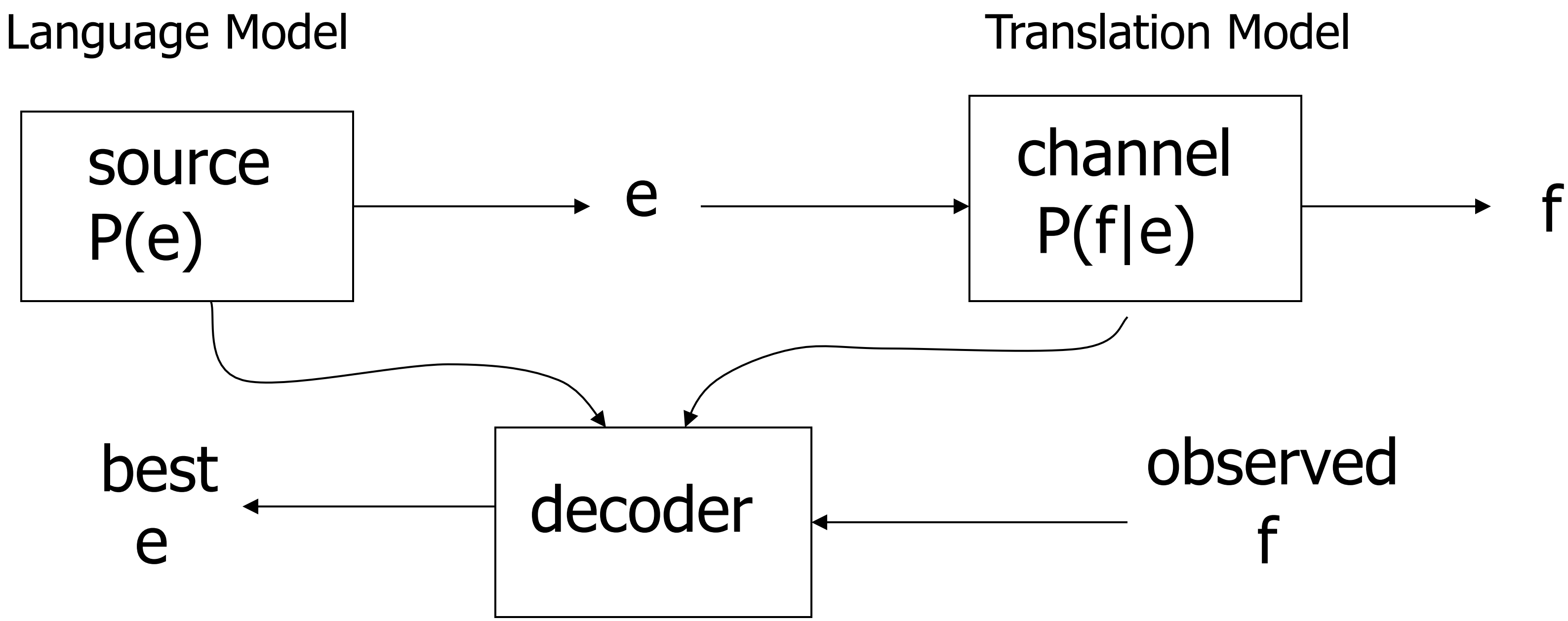
Translation: Codebreaking?

"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



$$\operatorname*{argmax}_{e} P(e|f) = \operatorname*{argmax}_{e} P(f|e)P(e)$$

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 grammaticality*: P(artichokes intimidate zippers) ≈ 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

- Assumption: each word x_i is generated i.i.d.

$$p(x_1 ... x_n) = \prod_{i=1}^{n} q(x_i) \quad \text{where} \sum_{x_i \in \mathcal{V}*} q(x_i) = 1 \quad \text{and} \quad \mathcal{V}^* := \mathcal{V} \cup \{\text{STOP}\}$$

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

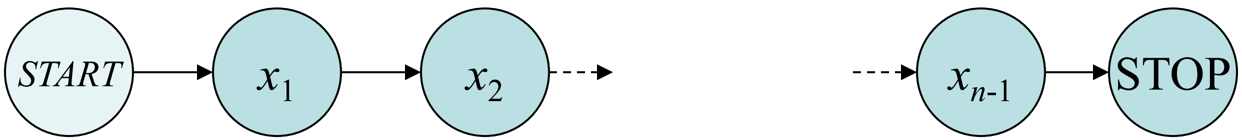
- Examples:
  - [fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass.]
  - [thrift, did, eighty, said, hard, 'm, july, bullish]
  - [that, or, limited, the]
  - []
  - [after, any, on, consistently, hospital, lake, of, of, other, and, factors, raised,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

Bigram Models

$$p(x_1...x_n) = \prod_{i=1}^{n} q(x_i|x_{i-1}) \quad \text{where} \sum_{x_i \in \mathcal{V}^*} q(x_i|x_{i-1}) = 1$$

$$x_0 = \text{START} \quad \& \quad \mathcal{V}^* := \mathcal{V} \cup \{\text{STOP}\}$$

- Generative process: (1) generate the very first word conditioning on the special symbol START, then, (2) pick the next word conditioning on the previous word, then repeat (2) until the special word STOP gets picked.
- Graphical Model:



- Subtleties:
  - If we are introducing the special START symbol to the model, then we are making the assumption that the sentence always starts with the special start word START, thus when we talk about $p(x_1...x_n)$ it is in fact $p(x_1...x_n|x_0 = \text{START})$
  - While we add the special STOP symbol to the vocabulary $\mathcal{V}^*$, we do not add the special START symbol to the vocabulary. Why?

Bigram Models

- Alternative option:

$$p(x_1 ... x_n) = q(x_1) \prod_{i=2}^{n} q(x_i | x_{i-1}) \quad \text{where} \sum_{x_i \in \mathcal{V}^*} q(x_i | x_{i-1}) = 1$$

- Generative process: (1) generate the very first word based on the unigram model, then, (2) pick the next word conditioning on the previous word, then repeat (2) until the special word STOP gets picked.
- 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]

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

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

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

- Learning: estimate the distributions $q(x_i | x_{i-(k-1)} \dots x_{i-1})$

Generating Sentences by Sampling from N-Gram Models

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

Unigram LMs are Well Defined Dist'ns*

- Simplest case: unigrams

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

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

(1) $$\sum_x p(x) = \sum_{n=1}^{\infty} \sum_{x_1...x_n} p(x_1...x_n)$$

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

$$= \sum_{x_1} q(x_1) \times ... \times \sum_{x_n} q(x_n) = (1 - q_s)^{n-1} q_s \quad \text{where } q_s = q(\text{STOP})$$

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

N-Gram Model Parameters

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

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

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*(.) 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 *

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

$$= 0.0006$$

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 is always 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 ____

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

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

Claude Shannon

- How good are we doing?

Compute per word log likelihood (M words, m test sentences s_i):

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

Perplexity

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

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Perplexity is the inverse probability of the test set, normalized by the number of words (why?)

$$equivalently:$$

$$PP(W) = 2^{-l}$$

$$\text{where } \; l = \frac{1}{N} \log P(w_1 w_2 \ldots w_N)$$

$$2^{-l} \text{ where } l = \frac{1}{M} \sum_{i=1}^{m} \log p(s_i)$$

The Shannon Game intuition for perplexity

- How hard is the task of recognizing digits '0,1,2,3,4,5,6,7,8,9' at random
  - Perplexity 10
- How hard is recognizing (30,000) names at random
  - Perplexity = 30,000
- If a system has to recognize
  - Operator (1 in 4)
  - Sales (1 in 4)
  - Technical Support (1 in 4)
  - 30,000 names (1 in 120,000 each)
  - Perplexity is 53
- Perplexity is weighted equivalent branching factor

$$
\begin{aligned}
PP(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

Perplexity as branching factor

- Language with higher perplexity means the number of words branching from a previous word is larger on average.
- The difference between the perplexity of a language model and the true perplexity of the language is an indication of the quality of the model.

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity			

- "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. Be careful in homeworks!

Extrinsic Evaluation

- Intrinsic evaluation: e.g., perplexity
  - Easier to use, but does not necessarily correlate the model performance when situated in a downstream application.
- Extrinsic evaluation: e.g., speech recognition, machine translation
  - Harder to use, but shows the true quality of the model in the context of a specific downstream application.
  - Better perplexity might not necessarily lead to better Word Error Rate (WER) for speech recognition.

- Word Error Rate (WER) := $\dfrac{\text{insertions} + \text{deletions} + \text{substitutions}}{\text{true sentence size}}$

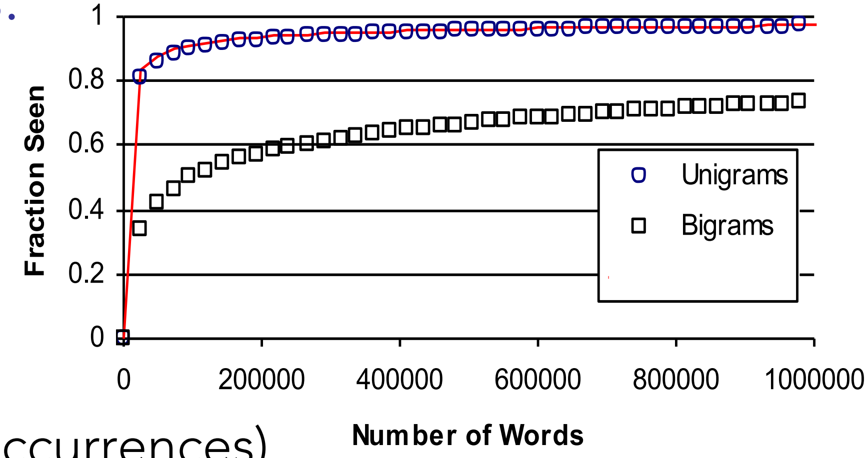
Correct answer: Andy saw a part of the movie

Recognizer output: And he saw apart of the movie

WER: 4/7
= 57%

Sparsity

- Problems with n-gram models:
  - New words appear all the time:
    - Synaptitute
    - 132,701.03
    - multidisciplinarization
  - 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

Zeros

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

  P("offer" | denied the) = 0

- Test set
  … denied the offer
  … denied the loan

Zero probability bigrams

- Bigrams with zero probability
  - mean that we will assign 0 probability to the test set!
- It also means that we cannot compute perplexity (can't divide by 0)!

$$
\begin{aligned}
PP(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}
\end{aligned}
$$

$equivalently:$

$$PP(W) = 2^{-l}$$

$$\text{where } \; l = \frac{1}{N} \log P(w_1 w_2 \ldots w_N)$$

Parameter Estimation

- Maximum likelihood estimates won't get us very far

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

- 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 aren't always what works

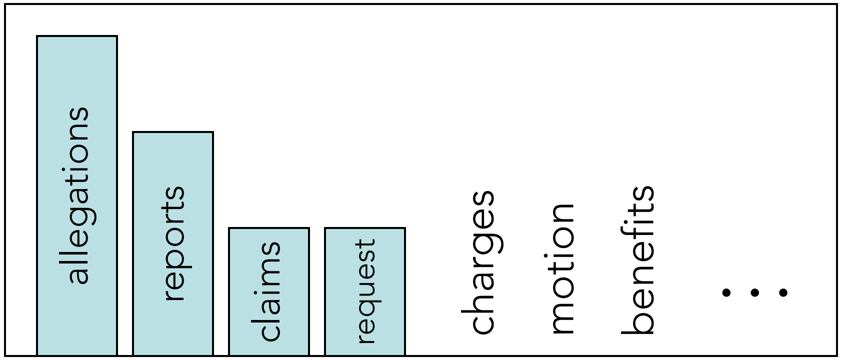
3516 wipe off the excess
1034 wipe off the dust
547 wipe off the sweat
518 wipe off the mouthpiece
…
120 wipe off the grease
0 wipe off the sauce
0 wipe off the mice

28048 wipe off the *

Smoothing

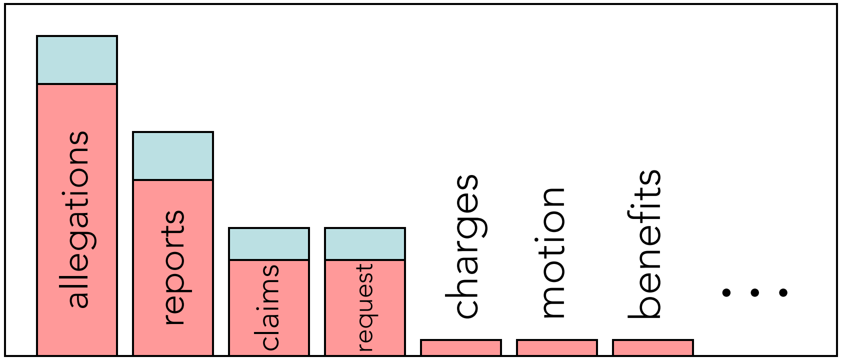
- We often want to make estimates from sparse statistics:

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:

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

- Add-1 estimate:

$$q_{\text{ADD-1}}(x_i|x_{i-1}) = \frac{c(x_{i-1}, x_i) + 1}{c(x_{i-1}) + }$$

More general formulations

Add-K:

$$q_{\text{ADD-K}}(x_i|x_{i-1}) = \frac{c(x_{i-1}, x_i) + k}{c(x_{i-1}) + \ }$$

$$q_{\text{ADD-K}}(x_i|x_{i-1}) = \frac{c(x_{i-1}, x_i) + \ }{c(x_{i-1}) + m}$$

Unigram Prior Smoothing:

$$q_{\text{UNIFORM-PRIOR}}(x_i|x_{i-1}) = \frac{c(x_{i-1}, x_i) + \ }{c(x_{i-1}) + m}$$

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.

Linear Interpolation

- Problem: $q_{ML}(w|u,v)$ is supported by few counts
- Classic solution: mixtures of related, denser histories:

$$q(w|u,v) = \lambda_3 q_{ML}(w|u,v) + \lambda_2 q_{ML}(w|v) + \lambda_1 q_{ML}(w)$$

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

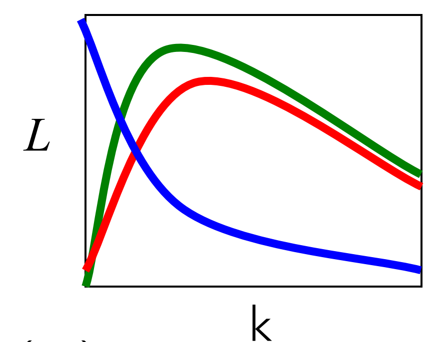
Experimental Design

- Important tool for optimizing how models generalize:

Training Data | Validation Data | Test Data

  - **Training data:** use to estimate the base n-gram models without smoothing
  - **Validation data (or "development" data):** use to pick the values of "hyper-parameters" that control the degree of smoothing by maximizing the (log-)likelihood of the validation data
  - Can use any optimization technique (line search or EM usually easiest)

- Examples:



$$q_{\text{ADD-K}}(x_i|x_{i-1}) = \frac{c(x_{i-1}, x_i) + k}{c(x_{i-1}) + k|\mathcal{V}^*|}$$

$$q(w|u,v) = \lambda_3 q_{ML}(w|u,v) + \lambda_2 q_{ML}(w|v) + \lambda_1 q_{ML}(w)$$

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

Practical Issues

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

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

* Advanced Topics for Smoothing

Held-Out Reweighting

- What' s wrong with add-d smoothing?
- Let' s look at some real bigram counts [Church and Gale 91]:

Count in 22M Words	Actual c* (Next 22M)	Add-one' s c*	Add-0.0000027' s c*
1	0.448	2/7e-10	~1
2	1.25	3/7e-10	~2
3	2.24	4/7e-10	~3
4	3.23	5/7e-10	~4
5	4.21	6/7e-10	~5

Mass on New	9.2%	~100%	9.2%
Ratio of 2/1	2.8	1.5	~2

- Big things to notice:
  - Add-one vastly overestimates the fraction of new bigrams
  - Add-0.0000027 vastly underestimates the ratio 2*/1*
- One solution: use held-out data to predict the map of c to c*

Absolute Discounting

- Idea 1: 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 the words into seen and unseen

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

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

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

- 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)

Good-Turing Discounting*

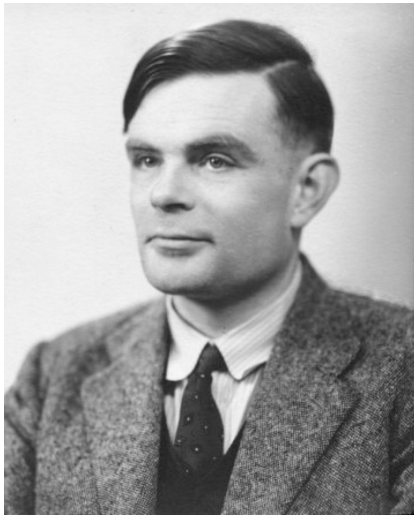
- Question: why the same d for all n-grams?
- Good-Turing Discounting: invented during WWII by Alan Turing and later published by Good. Frequency estimates were needed for Enigma code-breaking effort.
- Let n_r be the number of n-grams x for which c(x) = r
- Now, use the modified counts

$$c^*(x) = (r+1)\frac{n_{r+1}}{n_r} \text{ iff } c(x) = r, \;\; r > 0$$

- Then, our estimate of the missing mass is:

$$\alpha(v) = \frac{n_1}{N}$$

- Where N is the number of tokens in the training set



Good-Turing Smoothing Without Tears

William A. Gale
AT&T Bell Laboratories
P. O. Box 636
Murray Hill, NJ, 07974-0636

gale@research.att.com

ABSTRACT

rmance of statistically based techniques for many tasks such as spelling correct
ation, and translation is improved if one can estimate a probability for an object of inte
een seen before. Good-Turing methods are one means of estimating these probab
unseen objects. However, the use of Good-Turing methods requires a smoothing s
th in regions of vastly different accuracy. Such smoothers are difficult to use, and
e use of Good-Turing methods in computational linguistics.

Kneser-Ney Backoff*

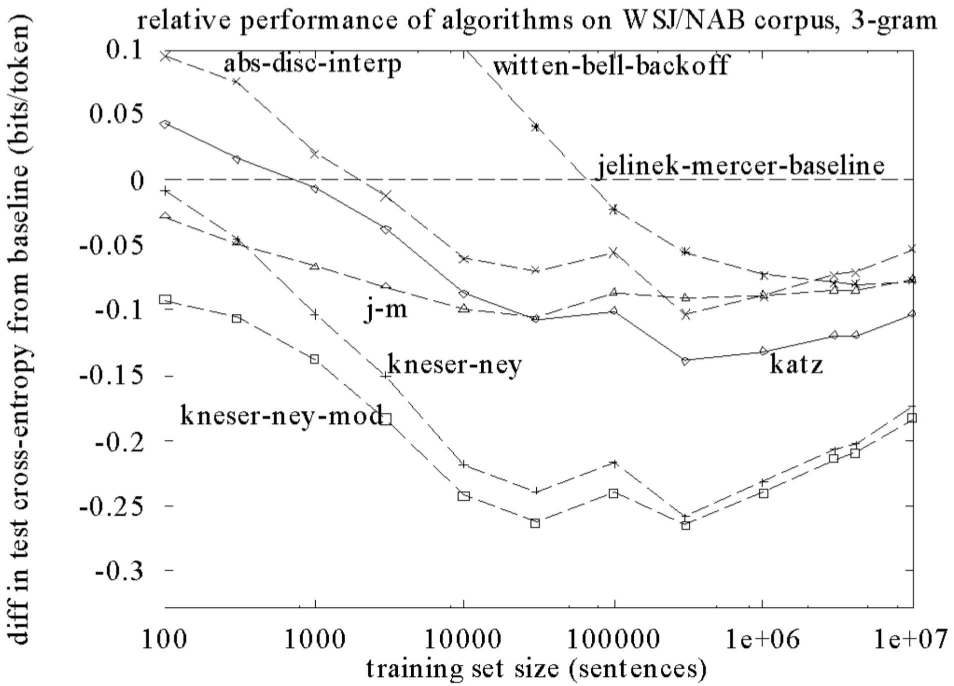
- Idea: Type-based fertility
  - Shannon game: There was an unexpected ____?
    - delay?
    - Francisco?
  - "Francisco" is more common than "delay"
  - … but "Francisco" (almost) always follows "San"
  - … so it's less "fertile"
- Solution: type-continuation probabilities
  - In the back-off model, we don't want the unigram estimate p_{ML}
  - Instead, want the probability that w is *allowed in a novel context*
  - For each word, count the number of bigram types it completes

$$P_C(w) \propto |w' : c(w', w) > 0|$$

  - KN smoothing repeatedly proven effective
  - [Teh, 2006] shows it is a kind of approximate inference in a hierarchical Pitman-Yor process (and other, better approximations are possible)

What Actually Works?

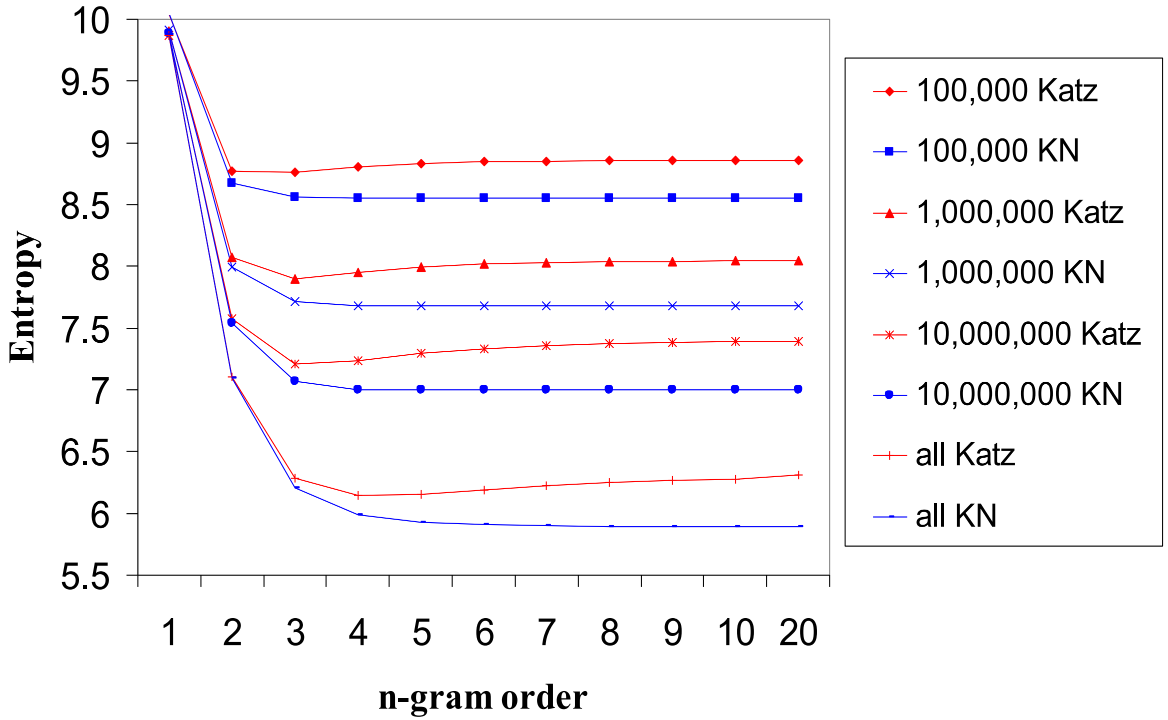
- Trigrams and beyond:
  - Unigrams, bigrams generally useless
  - Trigrams much better (when there's enough data)
  - 4-, 5-grams really useful in MT, but not so much for speech
- Discounting
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell, etc…
- See [Chen+Goodman] reading for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



[Graphs from Joshua Goodman]

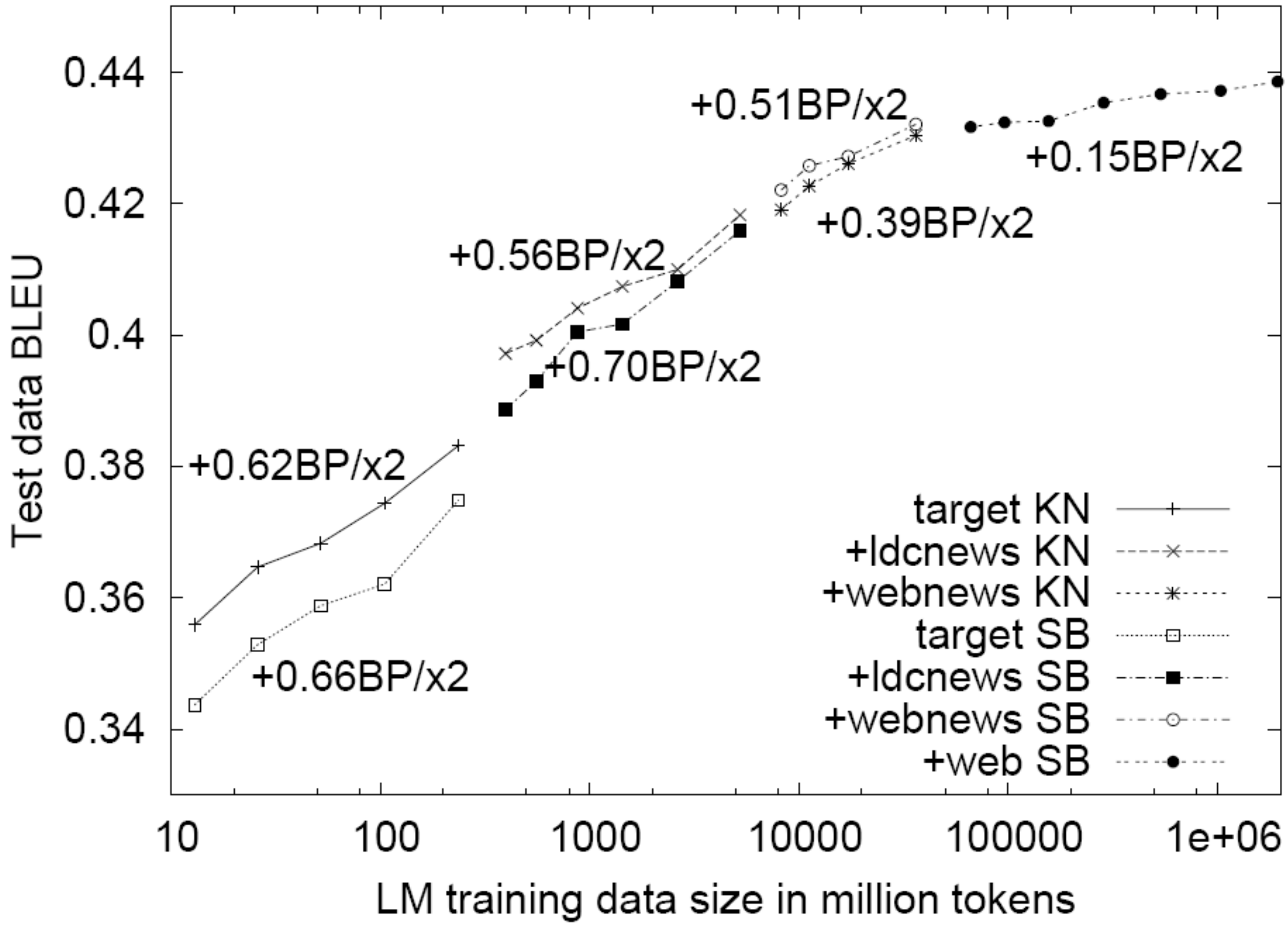
Data vs. Method?

- Having more data is better…



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

Tons of Data?



- Tons of data closes gap, for extrinsic MT evaluation

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
- A few recent 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 zeros: some n-grams are syntactically forbidden, keep estimates at zero [Mohri and Roark, 06]
  - Bayesian document and IR models [Daume 06]

Google N-Gram Release, August 2006

AUG

3

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 five-word sequences that appear at least 40 times. There are unique words, after discarding words that appear less than 200 times.

Google N-Gram

- 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

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

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

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

* Additional details on

1. Good Turing
2. Kneser-Ney

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)

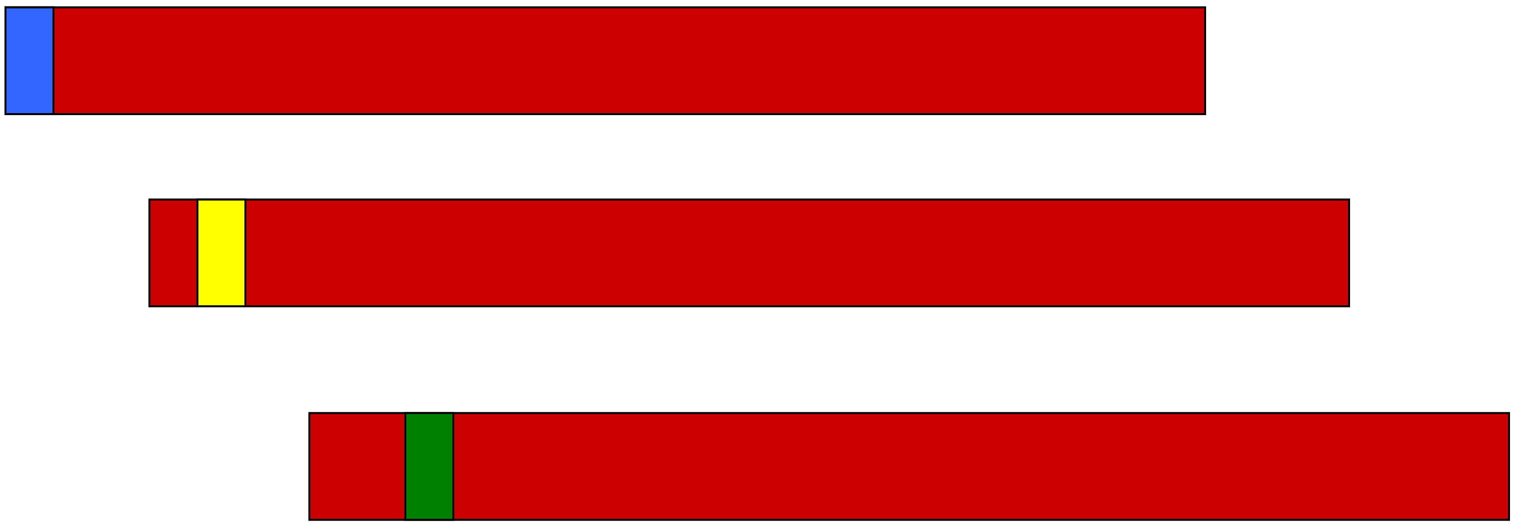
- c = 0:
- MLE p = 0/18 = 0
- P^*_{GT} (unseen) = N_1/N = 3/18

Seen once (trout)

- c = 1
- MLE p = 1/18
- C*(trout) = 2 * N2/N1 = 2 * 1/3 = 2/3
- P^*_{GT}(trout) = 2/3 / 18 = 1/27

Ney et al.'s Good Turing Intuition

H. Ney, U. Essen, and R. Kneser, 1995. On the estimation of 'small' probabilities by leaving-one-out. IEEE Trans. PAMI. 17:12,1202-1212

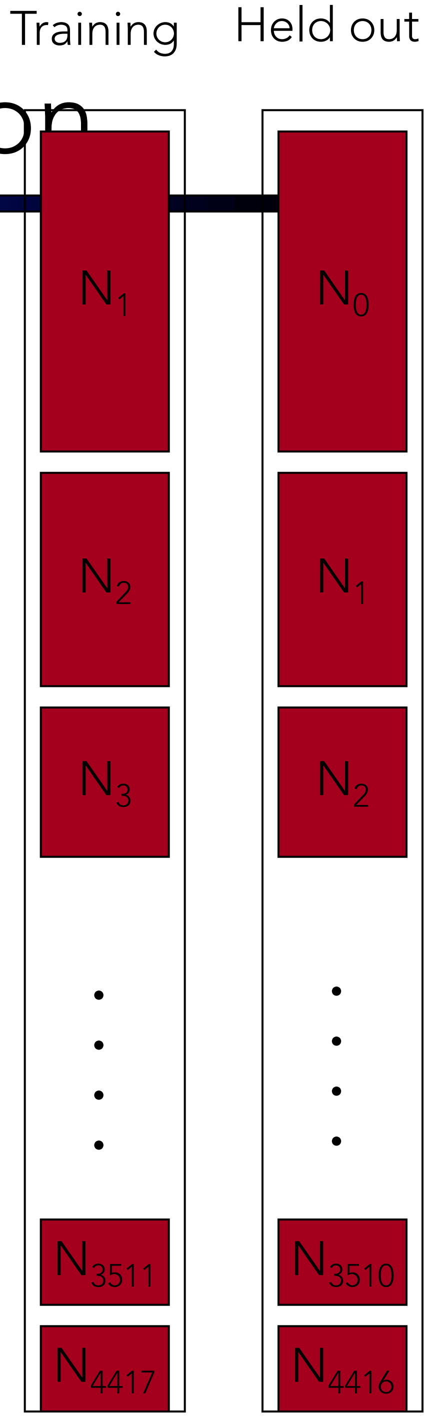


Held-out words:

Ney *et al.* Good Turing Intuition

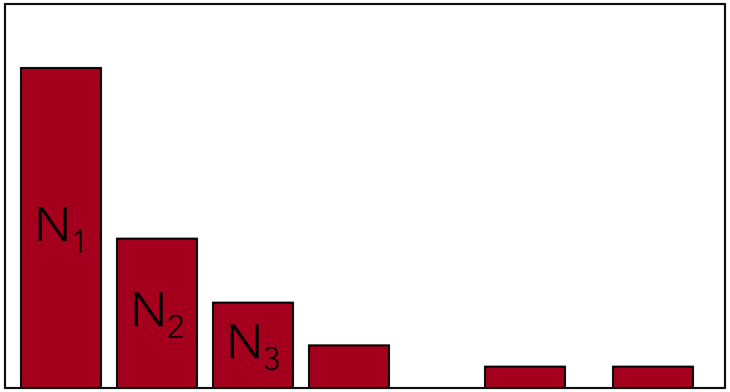
- Intuition from leave-one-out validation
  - Take each of the *c* training words out in turn
  - *c* training sets of size *c*–1, held-out of size 1
  - What fraction of held-out words are unseen in training?
    - N_1/c
  - What fraction of held-out words are seen *k* times in training?
    - $(k+1)N_{k+1}/c$
  - So in the future we expect $(k+1)N_{k+1}/c$ of the words to be those with training count *k*
  - There are N_k words with training count *k*
  - Each should occur with probability:
    - $(k+1)N_{k+1}/c/N_k$
  - …or expected count:

$$k^* = \frac{(k+1)N_{k+1}}{N_k}$$

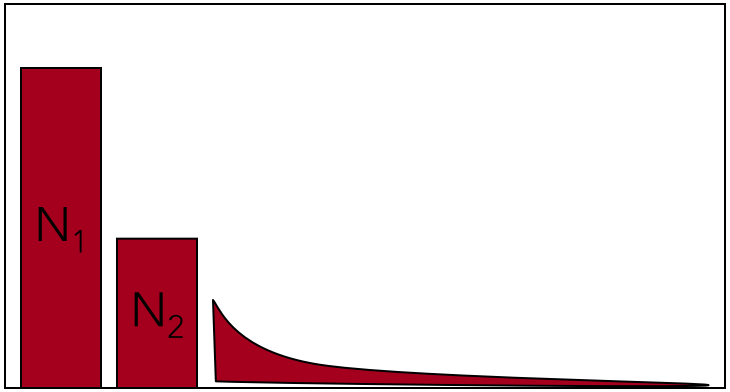


Good-Turing complications

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



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



Resulting 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 Interpolation

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

discounted bigram

Interpolation weight

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

unigram

- (Maybe keeping a couple extra values of d for counts 1 and 2)
- But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing I

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading___________?*
  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"

Francisco
glasses

- The unigram is useful exactly when we haven't seen this bigram!
- Instead of P(w): "How likely is w"
- $P_{continuation}(w)$: "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 II

- 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 III

- Alternative metaphor: The number of # of word types seen to precede w

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

- normalized by the # of words preceding all words:

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

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

Kneser-Ney Smoothing IV

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

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 •