Smoothing Continued…

Manish Shrivastava

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

Resulting Good-Turing numbers

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

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

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

Good-Turing smoothing

- The distribution of N_C has “gaps” for frequent objects
- The GT count for the most frequent word (e.g., “the”) is undefined
- Still probabilities of object with same count are assumed equal
- Need another “smoothing”!
  - Many possible GT smoothers
  - Difficult in general case, hindered use of GT in practice

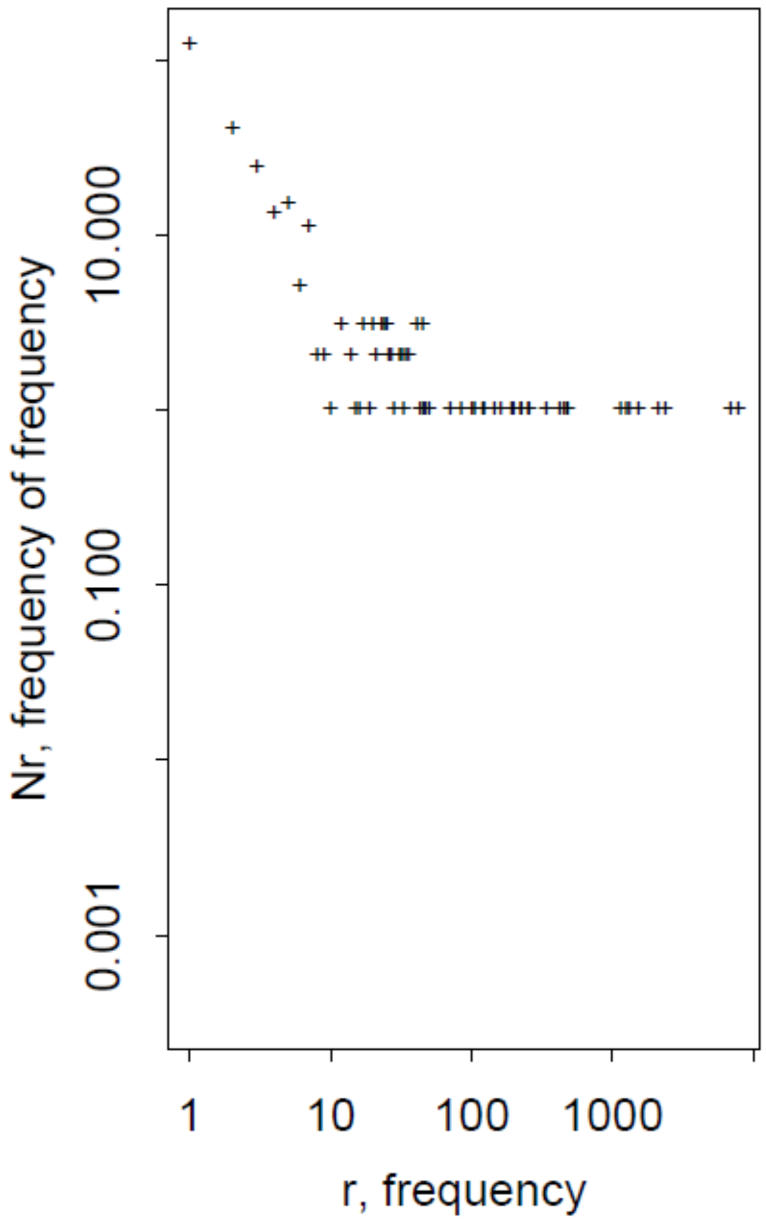
Good-Turing smoothing

- Another example (“Prosody”, Gale 1995)
  - Typical for linguistic data
  - Uncertainties in N_C vary greatly with c

frequency	frequency of frequency
1	120
2	40
3	24
4	13
5	15
6	5
7	11
8	2
9	2
10	1
11	0
12	3

Good-Turing smoothing

- Another representation of the same data
  - Too “granular” due to integer counts



Church and Gate (1991)

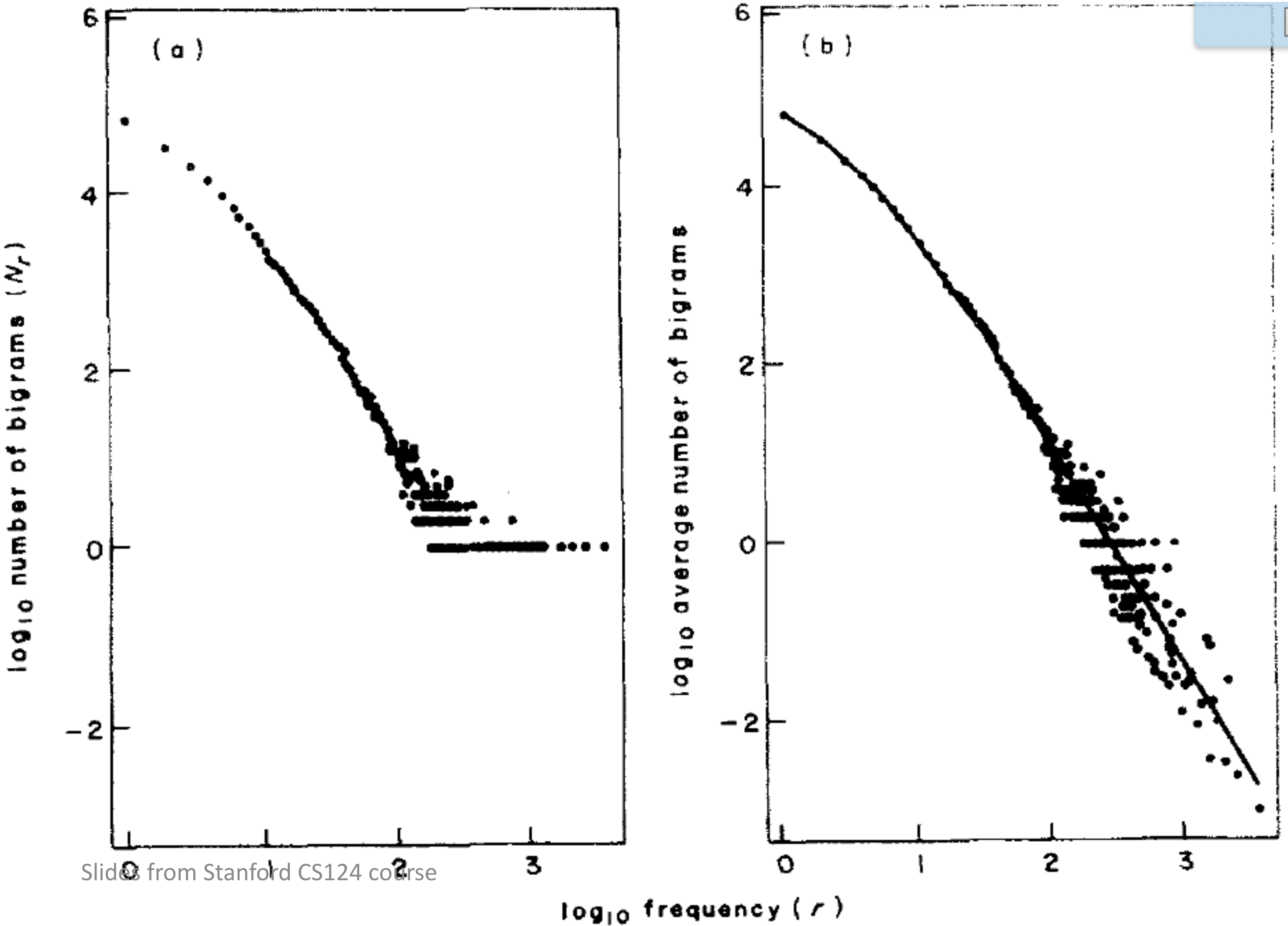
- Bi-gram smoothing based on unigram frequencies
  - A type of back-off
- Bi-grams are bucketed into N bins based on jii
  - “joint if independent”
  - Logarithmic bins, typically 3 per decade
- For a bi-gram *xy*:

$$jii = Ne(p(x))e(p(y))$$

- Basic estimator is used within each bin
  - Good-Turing, Held-off, Deleted Estimate, or other (?)

Church and Gale (1991)

- Single bin
  - j = 33, jii=1.4
  - Averaged around zeros (right)
  - Hastie and Shirey. A variable bandwidth kernel smoother. AT&T Technical report, 1988.
  - AP Wire, 4.4x10^7 words



Church and Gale (1991)

- Overall, the smoothing algorithm is complicated
- Shown to work well on large corpora [Chen, Goodman 1996]
- Defined for bigrams, generalization is ambiguous
- Unigram probabilities are estimated using MLE, other methods are possible

Simple Good-Turing (SGT)

- By Gale (1995)
- Simple linear interpolation between log(r) and log(Nr)

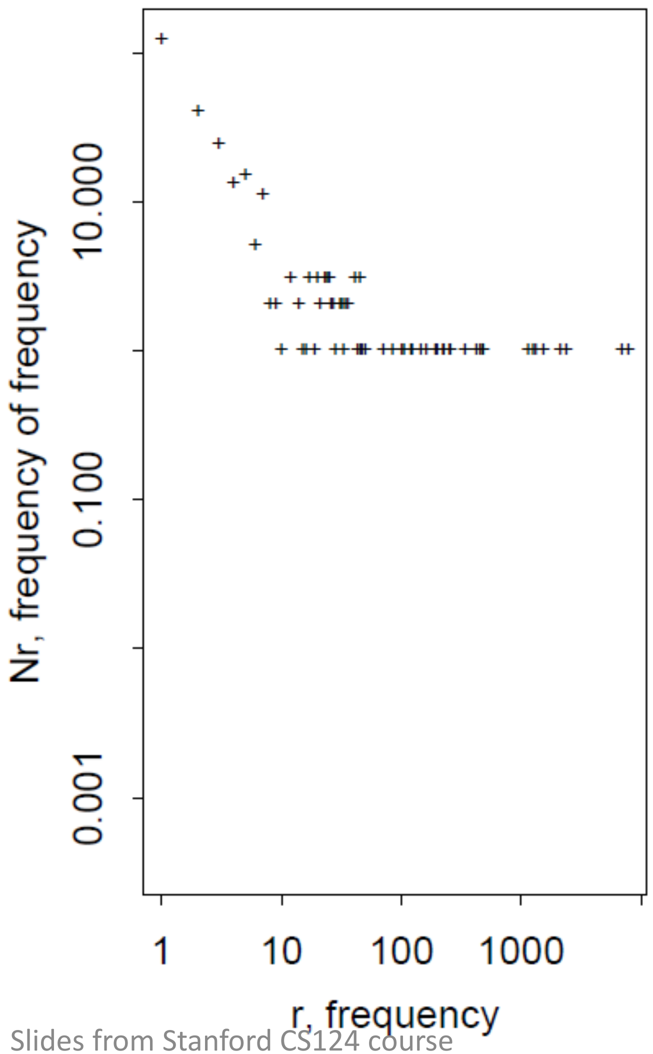
$$\log(N_r) = a + b\log(r)$$

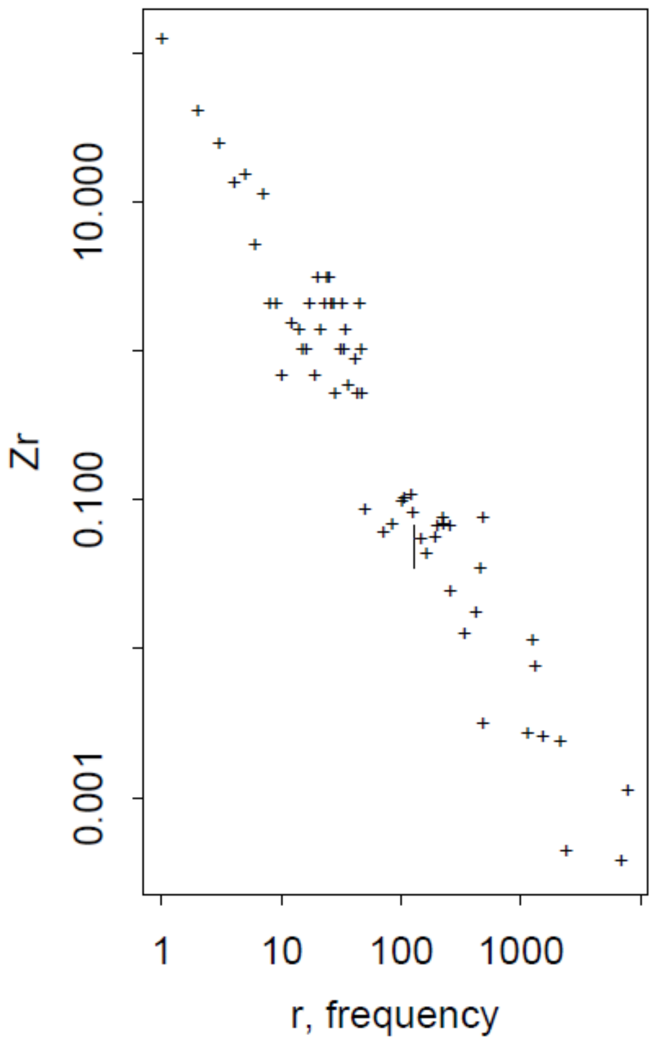
  - a and b are fit by linear regression
- Turing estimates are used for smaller r, linear estimates “kick in” once they become “significantly different”

Simple Good-Turing (SGT)

- Again, “Prosody” data from Gale (1995)

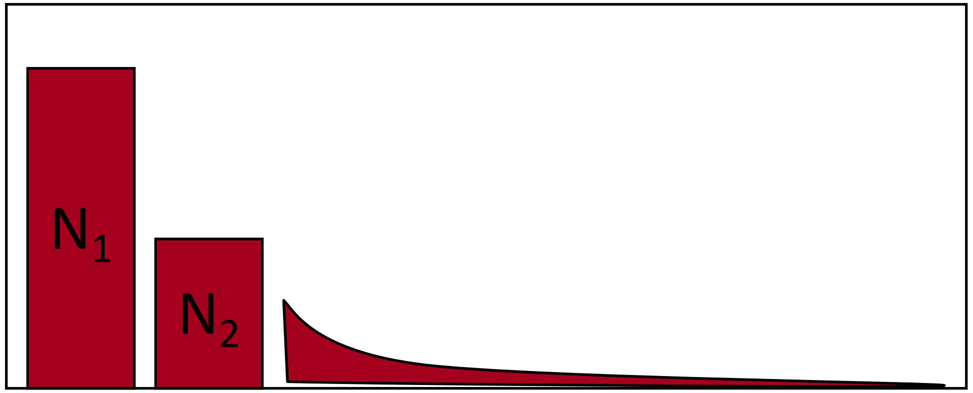
- Good-Turing (left)
- Good-Turing with zero averaging (right)





Simple Good-Turing (SGT)

- The author claimed that SGT is not only very simple but very accurate too
- The claim is based on Monte Carlo simulation, using Zipfian distribution



N_r vs r

Illustration from Dan Jurafsky lectures

Kneser-Ney smoothing I

- Constant discounting (Ney, Essen, Kneser, 1994)
- From AP Wire:
  - Looks like the difference is ~0.75 between MLE and GT for most objects

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

- 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

- But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing

- 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

Kneser-Ney Smoothing

- 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

- 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

- 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

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

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 •

Witten-Bell Smoothing

- Intuition:
  - An unseen n-gram is one that just did not occur yet
  - When it does happen, it will be its first occurrence
  - So give to unseen n-grams the probability of seeing a new n-gram

Witten-Bell – Unigram Case

- N: number of tokens
- T: number of types (diff. observed words) - can be different than V (number of words in dictionary)

Prob. of **unseen** unigrams

$$p_i^* = \frac{T}{Z(T+N)}$$

$$Z = \sum_{i:c_i=0} 1$$

Prob. of **seen** unigrams

$$p_i^* = \frac{c_i}{N+T}$$

Witten-Bell – bigram case

Prob. of **unseen** unigrams

$$p^*(w_i|w_x) = \frac{T(w_x)}{Z(w_x)(N(w_x) + T(w_x))}$$

Prob. of **seen** unigrams

$$p^*(w_i|w_x) = \frac{c(w_x, w_i)}{N(w_x) + T(w_x)}$$

Witten-Bell Example

- The original counts were –

	I	*want*	*to*	*eat*	*Chinese*	*food*	*lunch*	…	***N(w)* seen bigram tokens**	***T(w)* seen bigram types**	***Z(w)* unseen bigram types**
I	8	1087	0	13	0	0	0		3437	95	1521
want	3	0	786	0	6	8	6		1215	76	1540
to	3	0	10	860	3	0	12		3256	130	1486
eat	0	0	2	0	19	2	52		938	124	1492
Chinese	2	0	0	0	0	120	1		213	20	1592
food	19	0	17	0	0	0	0		1506	82	534
lunch	4	0	0	0	0	1	0		459	45	1571

- T(w)= number of different <u>seen</u> bigrams types starting with w
- We have a vocabulary of 1616 words, so we can compute

  Z(w)= number of <u>unseen</u> bigrams types starting with w

  Z(w) = 1616 - T(w)

- N(w) = number of bigrams <u>tokens</u> starting with w

Witten-Bell Example

- WB smoothed probabilities:

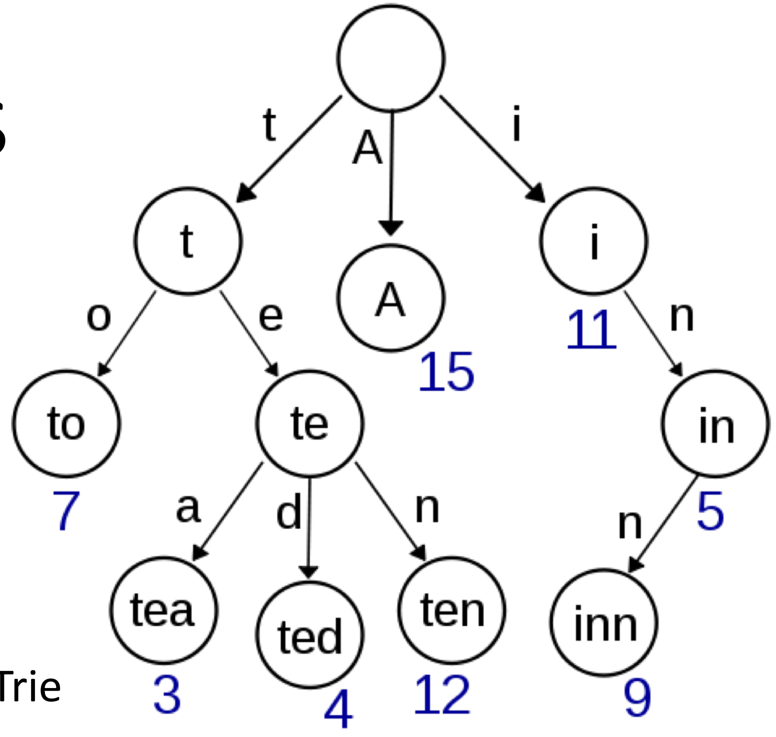
	I	*want*	*to*	*eat*	*Chinese*	*food*	*lunch*	…	**Total**
I	.0022 (7.78/3437)	.3078	.000002	.0037	.000002	.000002	.000002		1
want	.00230	.00004	.6088	.00004	.0047	.0062	.0047		1
to	.00009	.00003	.0030	.2540	.00009	.00003	.0038		1
eat	.00008	.00008	.0021	.00008	.0179	.0019	.0490		1
Chinese	.00812	.00005	.00005	.00005	.00005	.5150	.0042		1
food	.0120	.00004	.0107	.00004	.00004	.00004	.00004		1
lunch	.0079	.00006	.00006	.00006	.00006	.0020	.00006		1

Practical issue: 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

Huge web-scale n-grams

- Efficiency
  - Efficient data structures
    - e.g. tries



https://en.wikipedia.org/wiki/Trie

  - Store words as indexes, not strings
  - Quantize probabilities (4-8 bits instead of 8-byte float)