Language Modeling

Dr. Demetrios Glinos
University of Central Florida

CAP6640 – Computer Understanding of Natural Language

Today

- Introduction to N-grams
- Estimating N-gram Probabilities
- Evaluation and Perplexity
- Generalization and Zeroes
- Laplace Smoothing
- Interpolation, Backoff, and Web-Scale LMs
- Advanced: Kneser-Ney Smoothing

Probabilistic Language Modeling

- A **language model (LM)** is a model that can compute
  - probability of a particular sentence or sequence of words

    $P(W) = P(w_1, w_2, w_3, \ldots, w_n)$

  - probability of the next word in a sequence

    $P(w_n \mid w_1, w_2, w_3, \ldots, w_{n-1})$

Language Modeling Uses

- Machine translation:
  - computing whether P(***high*** winds tonight) > P(***large*** winds tonight)
- Spelling correction
  - given the input: "My home is about fifteen ***minuets*** away"
  - comparing P(about fifteen ***minutes*** away) and P(about fifteen ***minuets*** away)
- Speech recognition
  - P("I saw a van") vs. ("Eyes awe of an")
- Plus many other applications in question-answering, summarization, etc.

Computing P(W)

- Consider the phrase, "in the still of the night"

- We wish to compute the joint probability

P(in, the, still, of, the, night)

Q1: How can we compute this?

Q2: What information do we need?

- Intuition: let's use the chair rule for probabilities

Chain Rule for Probabilities

- Recall the definition of conditional probability: $P(\, b \mid a \,) = \dfrac{P(\, a, b \,)}{P(\, a \,)}$

- Which can be rewritten as the product rule: $P(\, a,\, b \,) = P(\, a \,)\, P(\, b \mid a \,)$

- Adding another variable: $P(\, x_1,\, x_2,\, x_3 \,) = P(\, x_1 \,)\, P(\, x_2 | x_1 \,)\, P(\, x_3 | x_1,\, x_2 \,)$

- The Chain Rule in general: $\boxed{P(x_1, \ldots, x_n) = \prod_i P(x_i \mid x_1, \ldots, x_{i-1})}$

Example using the Chain Rule

$$P(x_1,\dots,x_n) = \prod_i P(x_i \mid x_1,\dots,x_{i-1})$$

- For the phrase, "in the still of the night"
- Using the Chain Rule

P(in, the, still, of, the, night)

= P(in)
P(the | in)
P(still | in, the)
P(of | in, the, still)
P(the | in, the, still, of)
P(night | in, the, still, of, the)

How Can We Estimate the Conditionals?

- Could we just count and divide?

$$P(\text{ night} \mid \text{in, the, still, of, the }) = \frac{\text{Count(in, the, still, of, the, night)}}{\text{Count(in, the, still, of, the)}}$$

Q: How many times are we likely to see this particular phrase?

A: We'll never see enough data to estimate these directly

Markov Assumption

- Basic idea: approximate the desired conditional probability using only a fixed limited number of predecessors

  - e.g., P(night | in the still of the) ≈ P(night | the) ← bigram

  - e.g., P(night | in the still of the) ≈ P(night | in the) ← trigram

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

- So that: $$\boxed{P(w_1 w_2 \ldots w_n) \approx \prod_i P(\underbrace{w_i \mid w_{i-k} \ldots w_{i-1}}_{\text{These are "k-grams"}})}$$

These are "k-grams"

N-gram Models

- Unigram model: $P(\mathrm{w}_1\, \mathrm{w}_2 \ldots \mathrm{w}_n) \approx \prod_i P(w_i)$
- Bigram model: $P(\mathrm{w}_1\, \mathrm{w}_2 \ldots \mathrm{w}_n) \approx \prod_i P(w_i \mid w_{i-1})$
- Trigram model: $P(\mathrm{w}_1\, \mathrm{w}_2 \ldots \mathrm{w}_n) \approx \prod_i P(w_i \mid w_{i-2}\, w_{i-1})$
- Can also extend to 4-grams, 5-grams, etc.
- But n-grams cannot capture long-distance dependencies
  - e.g. "The book that I was reading yesterday was bought online."
- Nevertheless, n-gram LMs often work well in practice

Today

- Introduction to N-grams
- Estimating N-gram Probabilities
- Evaluation and Perplexity
- Generalization and Zeroes
- Laplace Smoothing
- Interpolation, Backoff, and Web-Scale LMs
- Advanced: Kneser-Ney Smoothing

Estimating bigram probabilities

- Maximum Likelihood Estimate (MLE):

$$P(\, w_i \mid w_{i-1}\,) = \frac{count(w_{i-1}, w_i\,)}{count(w_{i-1}\,)}$$

which we often also write as

$$P(\, w_i \mid w_{i-1}\,) = \frac{c(w_{i-1}, w_i\,)}{c(w_{i-1}\,)}$$

- e.g., (# times "the" is followed by "night") / (# times "the" occurs in corpus)

Example: computing bigrams

- Consider this corpus: <s> I am Sam </s>
  <s> Sam I am </s>
  <s> I do not like green eggs and ham </s>

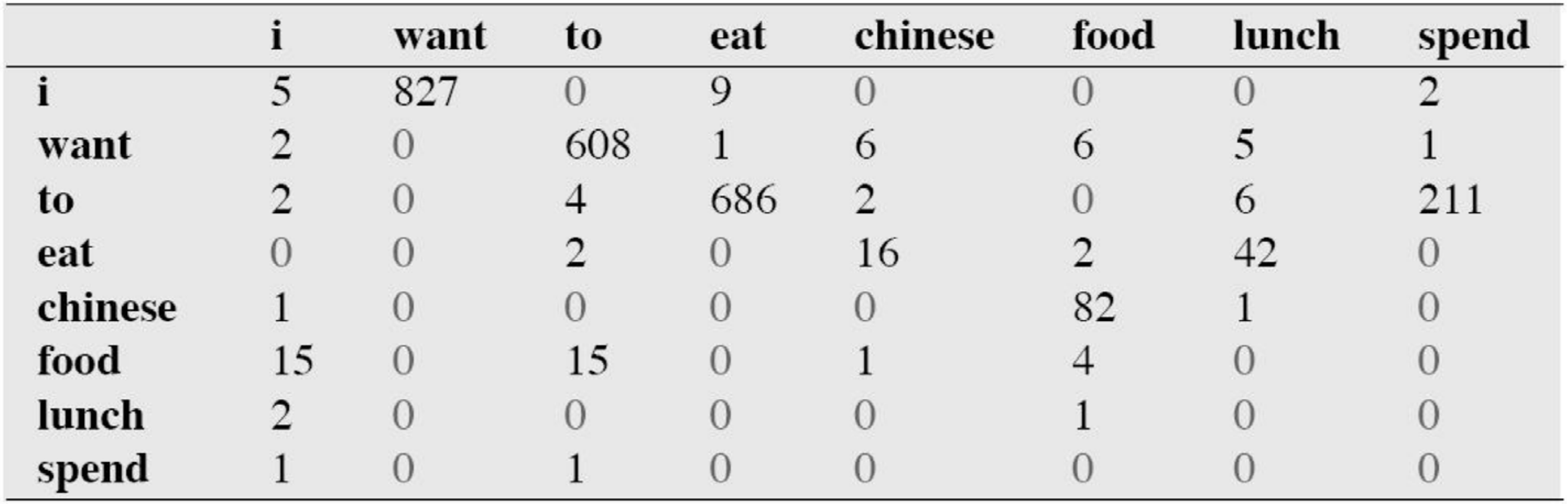
- Using $P(w_i \mid w_{i-1}) = \dfrac{c(w_{i-1}, w_i)}{c(w_{i-1})}$

- We compute: P(I | <s>) = 2/3 = .67
  P(am | I) = 2/3 = .67
  P(Sam | am) = 1/2 = .5
  P(</s> | Sam) = 1/2 = .5
  P(Sam | <s>) = 1/3 = .33
  P(do | I) = 1/3 = .33
  etc.

Another Example: Berkeley Restaurant Project

- The corpus consists of 9222 tokenized sentences like:
  - can you tell me about any good cantonese restaurants close by
  - mid priced tai 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

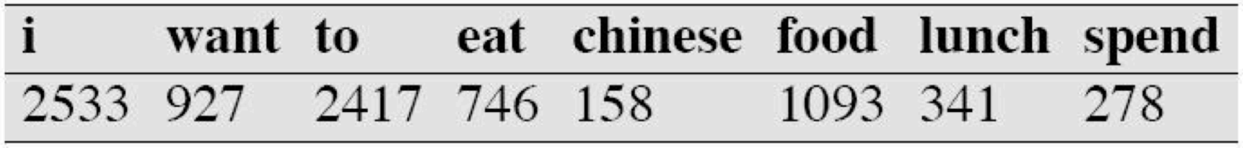


	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

source: Fig. 4.1

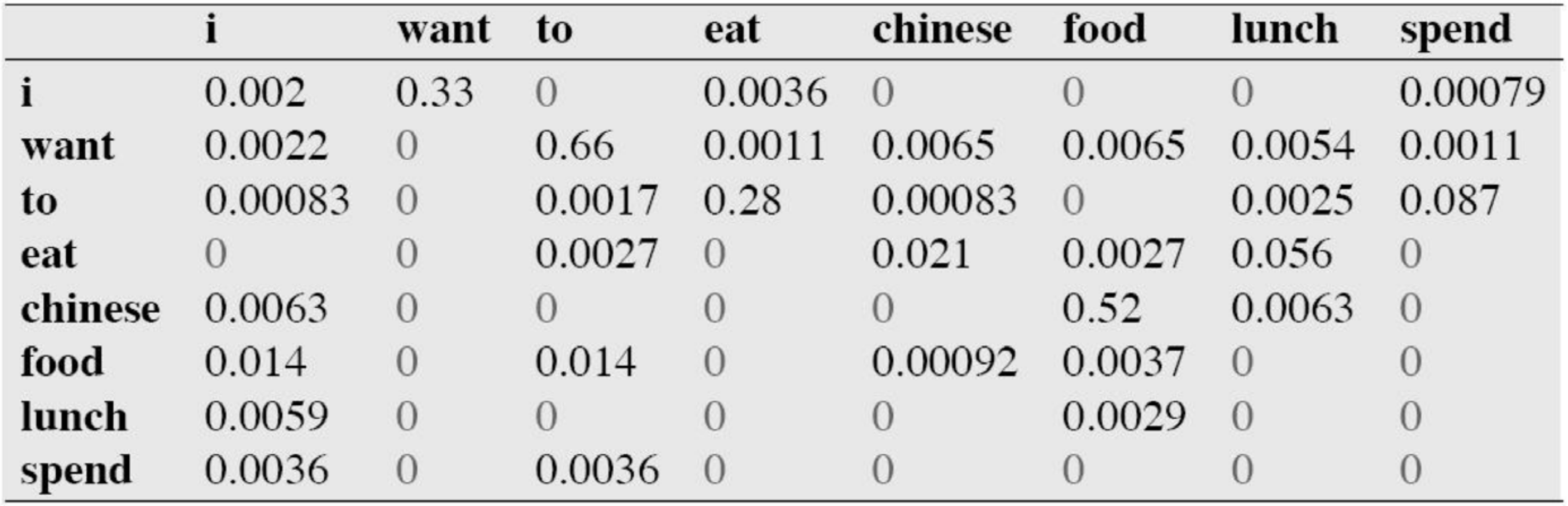
Normalized Bigram Counts

- Normalize by unigrams:



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

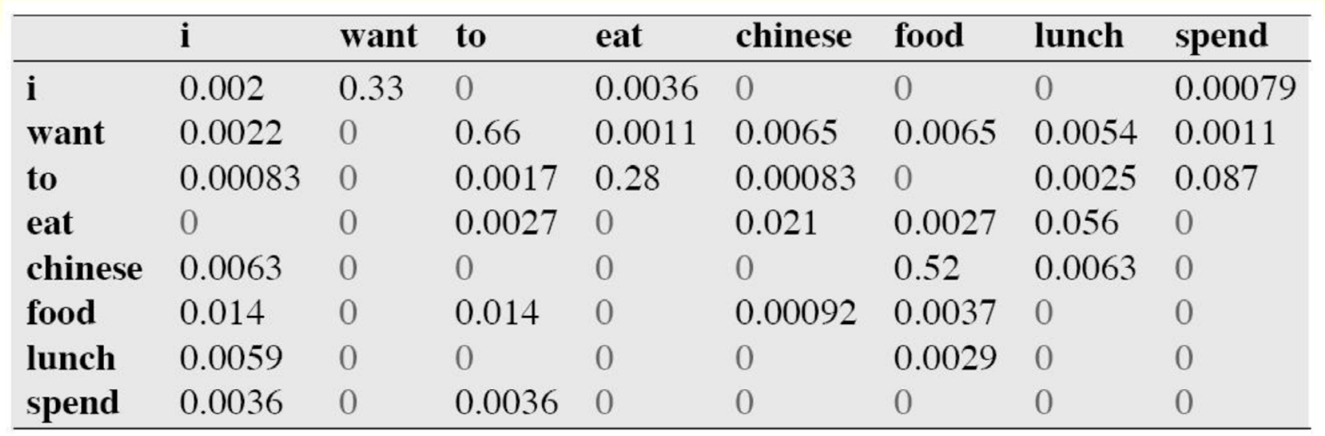
- Result:



	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

source: Fig. 4.2

Bigram estimate of sentence probability



	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

- Given also: P(i | <s>) = .25 and P(</s> | food) = .68

- We can compute: P(I want to eat chinese food)

=P(i|<s>) P(want |I) P(to|want) P(eat|to) P(Chinese|eat) P(food|Chinese) P(</s>|food)

= (.25) (.33) (.66) (.28) (.021) (.52) (.68)

= .0001132

source: Fig. 4.2

Practical Issue

- We compute these probabilities in log space
    - to avoid underflow
    - also: addition is faster than multiplication

$$\log(p_1 \times p_2 \times p_3 \times \ldots \times p_n) = \log(p_1) + \log(p_2) + \ldots + \log(p_n)$$

Q: Why are we able to use these values for our probabilities?

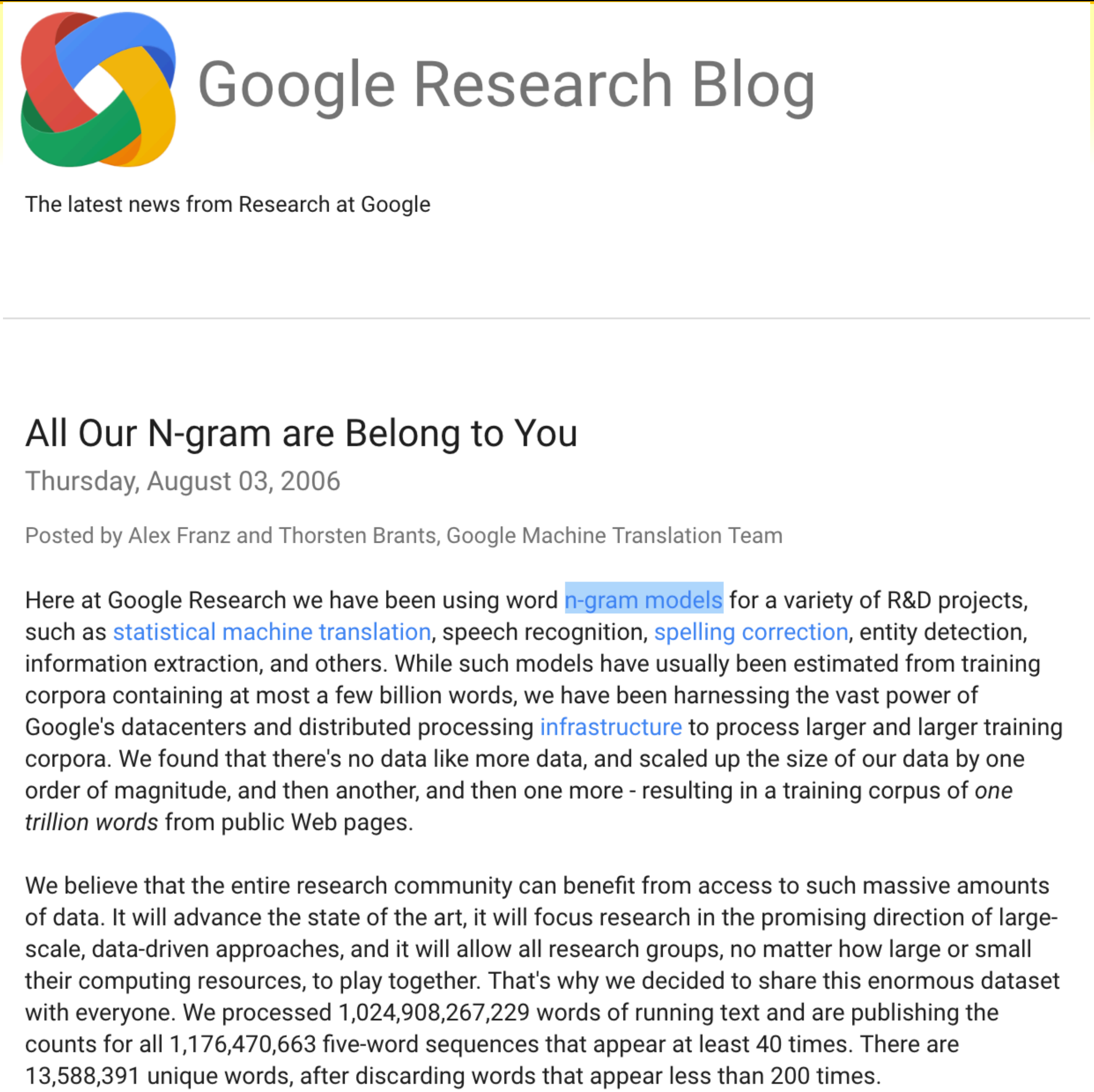
Language Modeling Toolkits

- SRILM

  http://www.speech.sri.com/projects/srilm/

- KenLM

  https://kheafield.com/code/kenlm/



Google Research Blog

The latest news from Research at Google

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, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of *one trillion words* from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small 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-Gram Release

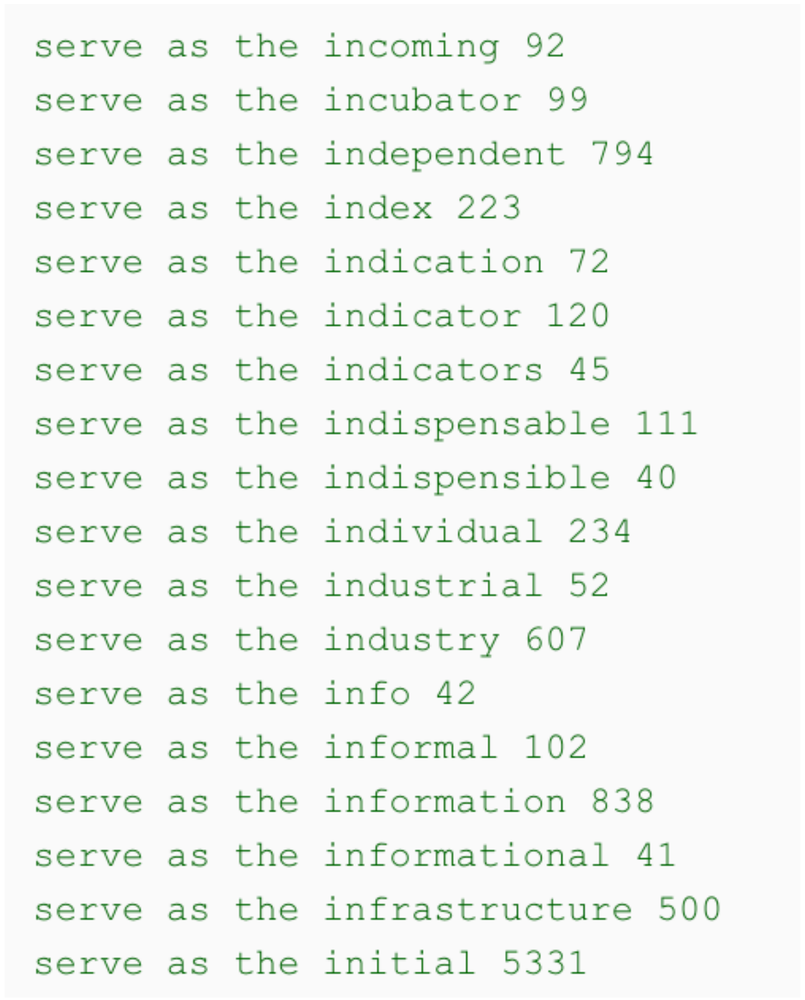
Corpus statistics:

```
File sizes: approx. 24 GB compressed (gzip'ed) text files

Number of tokens:    1,024,908,267,229
Number of sentences:    95,119,665,584
Number of unigrams:         13,588,391
Number of bigrams:         314,843,401
Number of trigrams:        977,069,902
Number of fourgrams:     1,313,818,354
Number of fivegrams:     1,176,470,663
```

Google N-Gram Release

Sample 4-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
serve as the industrial 52
serve as the industry 607
serve as the info 42
serve as the informal 102
serve as the information 838
serve as the informational 41
serve as the infrastructure 500
serve as the initial 5331
```

Today

- Introduction to N-grams
- Estimating N-gram Probabilities
- Evaluation and Perplexity
- Generalization and Zeroes
- Laplace Smoothing
- Interpolation, Backoff, and Web-Scale LMs
- Advanced: Kneser-Ney Smoothing

Evaluating a Language Model

- OK, so we have a corpus and we pulled some n-grams out of it.
- Questions
  - How good is our model?
  - Does it prefer "real" or "frequently observed" sentences to obscure or ungrammatical ones?
- We use standard supervised learning evaluation method
  - train on a **training set**
  - test against a **test set**
  - use an **evaluation metric** to quantify how well our model performs
  - **Note:** training on the test set is bad science and unethical

Extrinsic evaluation of N-gram models

- Best method of evaluation for comparing models is to have them perform some useful task
  - e.g., spelling correction, speech recognition, machine translation
- Run the task, get accuracy results for each model, and compare the results
  - e.g., how many misspelled words were properly corrected
  - e.g., how many words were translated correctly
- There are many such ongoing tasks in NLP
  - e.g., annual conferences such as SemEval, CLEF, PAN, etc.

Intrinsic Evaluation

- Extrinsic evaluation is time-consuming
  - can take days or weeks to do the work, but often much longer for the comparitive evaluation of systems

- Intrinsic evaluation
  - measures **perplexity**
  - how good the approximation is depends on how closely the test data matches the training data

  - useful for pilot experiments, and when time/resources insufficient to perform extrinsic evaluation

Intuition for Perplexity

- Consider the Shannon game
  - How well can we predict the next word?

    "I always order pizza with onions and _____

pepperoni	0.1
sausage	0.1
ham	0.05
mushrooms	0.01
…	
egg rolls	0.0001
…	

  - Unigrams don't work very well here (Q: Why?)
  - A ***good*** LM is one that computes a higher probability to a word that actually occurs in the real world.
  - The ***best*** LM is one that best predicts an unseen test sentence
    - i.e., gives the highest P(sentence)

Perplexity defined

Definition:

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

$$PP(W) = P(w_1 w_2 \dots w_n)^{1/n} = \sqrt[n]{\frac{1}{P(w_1 w_2 \dots w_n)}}$$

Chain rule:

$$PP(W) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 w_2 \dots w_{i-1})}}$$

For bigrams:

$$PP(W) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_{i-1})}}$$

➔ We wish to minimize perplexity (this maximizes the probability)

Perplexity as a branching factor

- Suppose a sentence consists of N random digits

Q: What is the perplexity of this sentence according to a model that assigns a probability of 1/10 to each digit?

Answer:

$$PP(W) = P(w_1 w_2 \ldots w_n)^{-1/N} = \left[\left(\frac{1}{10}\right)^N\right]^{-1/N} = 10$$

Lower perplexity = better model

- Consider perplexity values for different models trained on 38 million words, tested on 1.5 million words, from the Wall Street Journal corpus (vocabulary of 19,979 words)

Model	Perplexity
Unigram	962
Bigram	170
Trigram	109

Today

- Introduction to N-grams
- Estimating N-gram Probabilities
- Evaluation and Perplexity
- Generalization and Zeroes
- Laplace Smoothing
- Interpolation, Backoff, and Web-Scale LMs
- Advanced: Kneser-Ney Smoothing

The problem of zeroes

- Suppose these are all the instances of trigrams beginning with "denied the" in the training corpus

  denied the allegations

  denied the reports

  denied the claims

  denied the request

- And suppose the test set contains these word combinations

  denied the offer

  denied the loan

- Result: P(offer | denied the) = 0 and can't compute perplexity, either

Shakespeare corpus

- Corpus
  - number of tokens: 884,647
  - size of vocabulary: 29,066
- The corpus contains about 300K bigram types out of V^2 = 844 million possible bigrams
  - Thus, 99.96% of all possible bigrams were never seen in the corpus
  - i.e., they have zero entries
- We need to make our n-gram models that generalize better
  - i.e., we want models that work for word combinations that do not occur in training data, but which do occur in test set

Today

- Introduction to N-grams
- Estimating N-gram Probabilities
- Evaluation and Perplexity
- Generalization and Zeroes
- Laplace Smoothing
- Interpolation, Backoff, and Web-Scale LMs
- Advanced: Kneser-Ney Smoothing

Intuition for Smoothing

- Intuition: steal some of the probability mass from the words that did occur in the corpus, to cover the words that did not occur in the corpus

P(w | denied the)	# occurrences
allegations	3
reports	2
claims	1
request	1
TOTAL	7

$\Rightarrow$

P(w | denied the)	# occurrences
allegations	2.5
reports	1.5
claims	0.5
request	0.5
other	2
TOTAL	7

Laplace Smoothing

- **Laplace (also called "Add-one") smoothing or estimation**"
- Basic idea: pretend we saw each word one more time than we actually did see it
  i.e., just add 1 to each count
- MLE estimate: $$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$
- Laplace estimate: $$P_{Laplace}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Maximum Likelihood Estimates

- Maximum likelihood estimation maximizes the likelihood of training set T given model M

- Suppose the word "bagel" occurs 400 times in a corpus of a million words

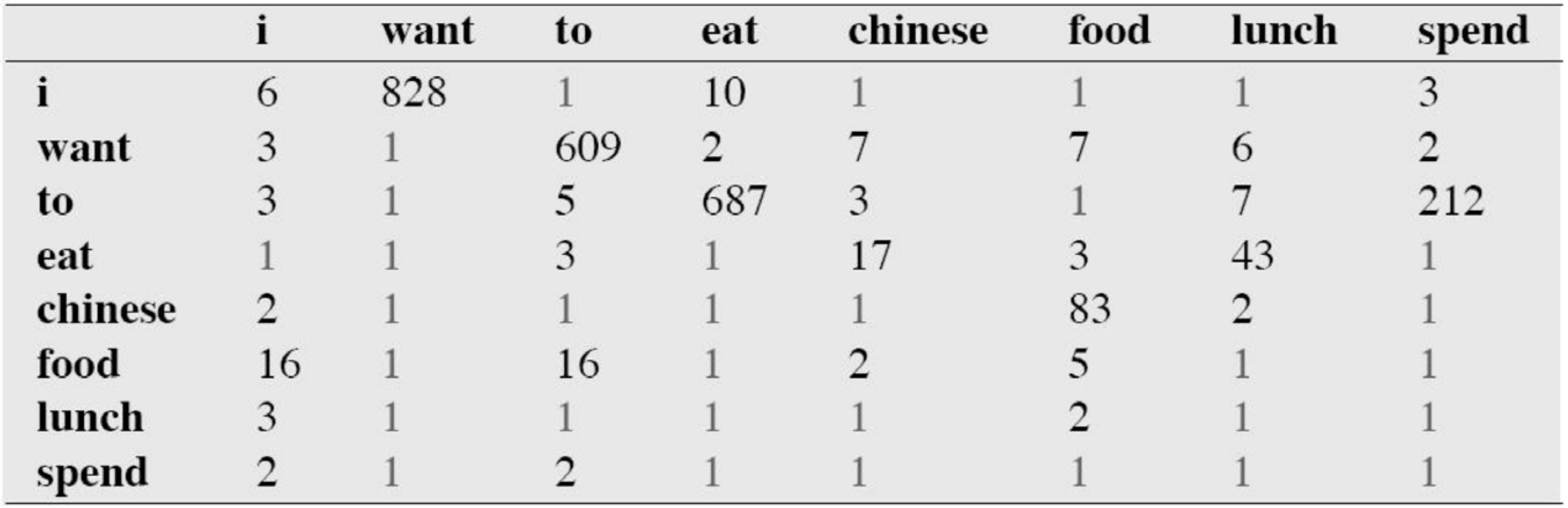
Q: What is the probability that a random word from some other text will be the word "bagel"?

Answer: MLE estimate is 400 / 1 million = .0004

Note: This may not be a good estimate for a particular other corpus, but it is the estimate that maximizes the likelihood that "bagel" will occur 400 times in a million word corpus

Berkeley Restaurant Corpus

- Laplace smoothed bigram **counts:**

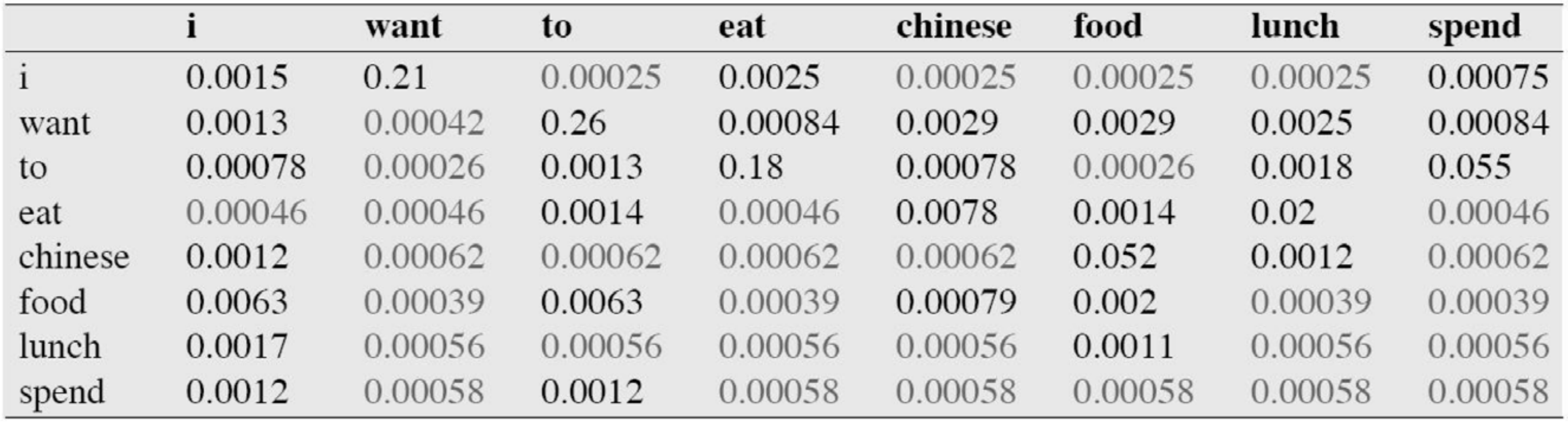


	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

source: Fig. 4.5

Berkeley Restaurant Corpus

- Laplace smoothed bigram **probabilities (using V = 1446):**



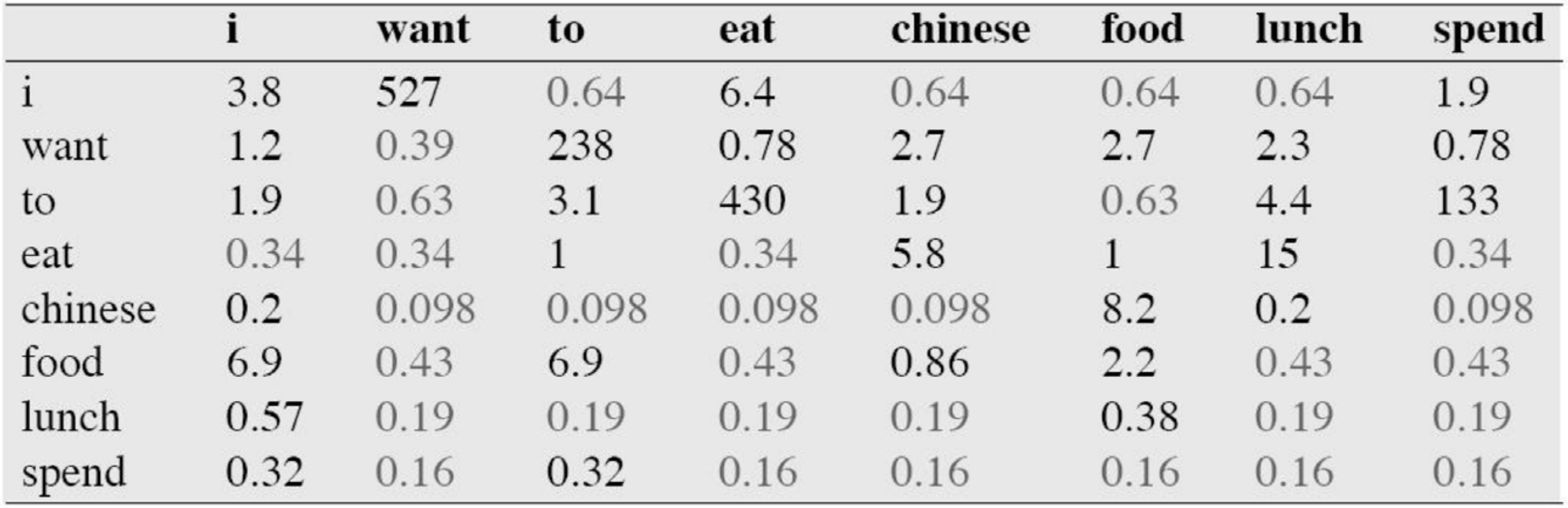
	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	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

source: Fig. 4.6

Berkeley Restaurant Corpus

- Reconsitituted counts:

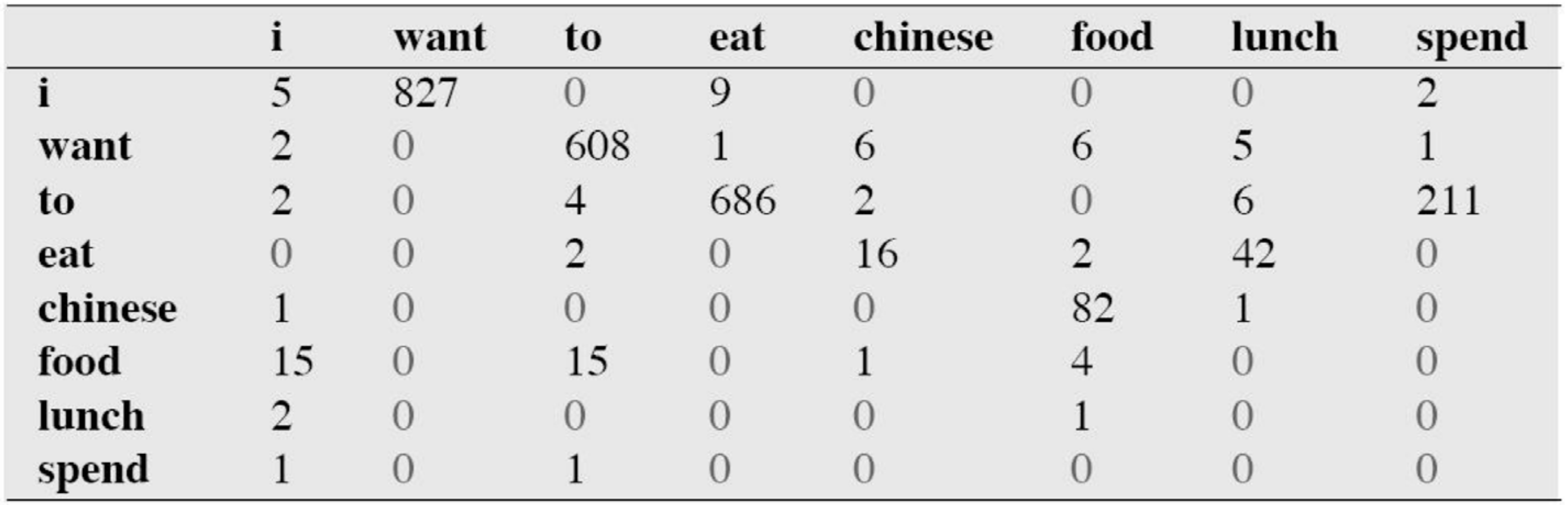
$$c^*(w_{n-1}\, w_n) = \frac{[c(w_{n-1}\, w_n) + 1] \cdot c(w_{n-1})}{c(w_{n-1}\,) + V}$$



	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

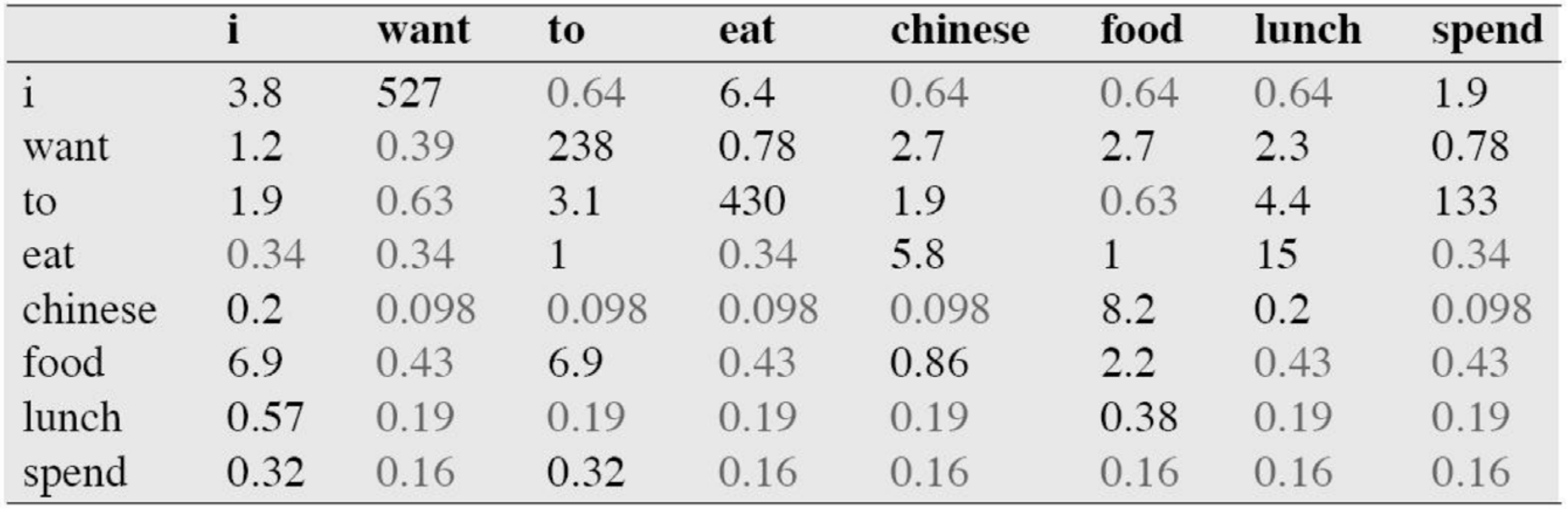
source: Fig. 4.7

Compare with Raw 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

source: Fig. 4.1



	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

source: Fig. 4.7

Today

- Introduction to N-grams
- Estimating N-gram Probabilities
- Evaluation and Perplexity
- Generalization and Zeroes
- Laplace Smoothing
- Interpolation, Backoff, and Web-Scale LMs
- Advanced: Kneser-Ney Smoothing

Backoff and Interpolation

- The reality: Sometimes, less context is better
    - Particularly if you don't know much about the domain
- Backoff
    - use trigrams if have good evidence
    - otherwise back off and use bigrams, otherwise unigram
- Interpolation
    - typically mixes unigrams, bigrams, and trigrams
    - generally works better than backoff

Linear Interpolation

- Basic idea: combine different order N-grams by linearly interpolating all models

- Simple interpolation:

$$\hat{P}(w_n | w_{n-2}\, w_{n-1}) = \lambda_1 P(w_n | w_{n-2}\, w_{n-1}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n)$$

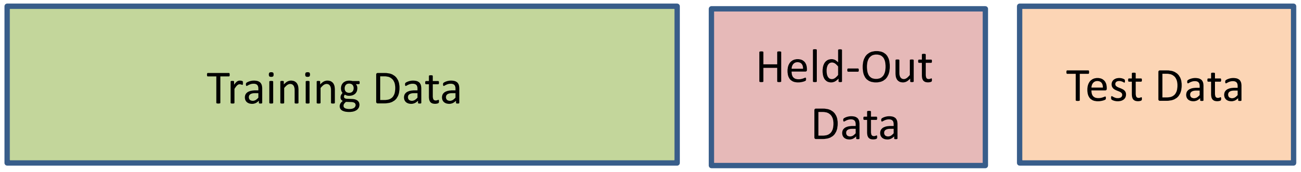
where $\sum_i \lambda_i = 1$

- Can also have lambdas depend on the word context:

$$\hat{P}(w_n | w_{n-2}\, w_{n-1}) = \lambda_1(w_{n-2}^{n-1}) P(w_n | w_{n-2}\, w_{n-1}) + \lambda_2(w_{n-2}^{n-1}) P(w_n | w_{n-1}) + \lambda_3(w_{n-2}^{n-1}) P(w_n)$$

How to set the lambdas?

- Use a **held-out** corpus



- Use training corpus to develop basic N-grams

- Use held-out corpus to choose lambdas that maximize the probability of the held-out data

$$\log P(w_1 \dots w_n \mid M(\lambda_1 \dots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \dots \lambda_k)}(w_i \mid w_{i-1})$$

- Evaluate model on the test data set

Handling Unknown Words

- Closed vocabulary task
  - we know all the words in advance
  - vocabulary V is fixed
  - nice, if you can get it
- Open vocabulary task
  - we don't know all the words
  - often the case
  - there are **out-of-vocabulary (OOV)** words
- Solution: create an "unknown word" token <UNK>
  - create a fixed lexicon L of V words
  - at text normalization, any word not in L is changed to <UNK>; train normally
  - at test time, use probabilities for <UNK> for any word not in training

Huge web-scale N-grams

- What to do with, e.g., the Google N-gram corpus
- Pruning
  - only store N-grams with count > threshold
  - remove sinletons of higher-order n-grams
  - use entropy-based pruning
- Use efficient data structures
  - tries
  - Bloom filters
  - Store words as indexes, not strings
    - use Huffman encoding 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

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

$$S(w_i) = \frac{c(w_i)}{N}$$

Today

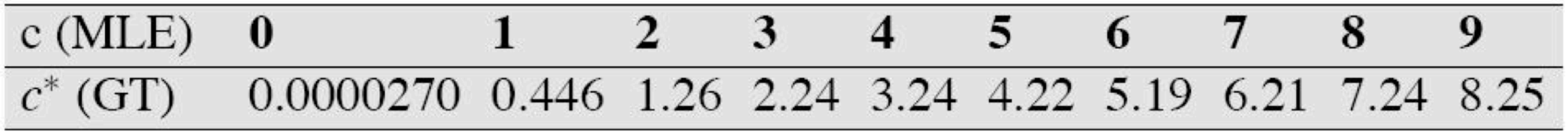
- Introduction to N-grams
- Estimating N-gram Probabilities
- Evaluation and Perplexity
- Generalization and Zeroes
- Laplace Smoothing
- Interpolation, Backoff, and Web-Scale LMs
- Advanced: Kneser-Ney Smoothing

Kneser-Ney Smoothing

- Most commonly used method for N-gram smoothing
  - Laplace smoothing is fine for text classification, but not for all NLP tasks
- Recall: smoothing is all about reducing probability values from what is observed, for handling the zeroes
- Question is: how to do the reduction
- Kneser-Ney is based on a method called absolute discounting

Absolute Discounting

- Church and Gale (1991)
  - Look at the frequency of each bigram in the training set
  - Compare it to the frequency of the same bigram in the held-out set



c (MLE)	0	1	2	3	4	5	6	7	8	9
c^* (GT)	0.0000270	0.446	1.26	2.24	3.24	4.22	5.19	6.21	7.24	8.25

source: Figure, p. 110

  - Data is from AP newswire data set of 22 million words
  - Anecdotal data suggest that we can subtract a constant value from counts to handle the zeroes
    - here: subtract the constant value D = 0.75 (≈ 2 – 1.26, etc.)

$$P_{absolute}(\, w_i \mid w_{i-1}\,) = \frac{c(w_{i-1}, w_i\,) - D}{c(w_{i-1}\,)}$$

- in practice, also common to keep separate discount factors for the 0 and 1 counts

Kneser-Ney Smoothing

- Gives better estimates for probabilities of lower-order unigrams
- Consider: I can't see without my reading ______ ?
  - correct answer: glasses
  - but in almost every corpus, other words are more frequent, e.g., York
  - BUT, "York" almost always follows "New"
- Kneser-Ney (1995)
  - change the question: "How likely is w", i.e., what is P(w) ?
  - into the question: "How likely is w to be a novel continuation?"
- For each word, count the number of bigram types it completes
- Note: every bigram type was novel the first time it was encountered

$$P_{CONTINUATION}(w_i) = \frac{|\{ w_{i-1} : c(w_{i-1}w_i) > 0 \}|}{\sum_{w_i} |\{ w_{i-1} : c(w_{i-1}w_i) > 0 \}|}$$

- A frequent word (York) occurring in only one context (New) will have low continuation probability

Interpolated Kneser-Ney

An interpolated form of Kneser-Ney is most commonly used

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

where λ is a normalizing constant for the probability mass we have discounted

$$\lambda(w_{i-1}\,) = \underbrace{\frac{D}{c(w_{i-1}\,)}}_{\text{the normalized discount}} \underbrace{|\{\, w : c(\, w_{i-1}, w\,) > 0\,\}|}_{\text{The number of word types that can follow } w_{i-1}}$$

the normalized discount

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