Lecture 3:
n-gram Language Models II

Instructor: Swabha Swayamdipta
USC CSCI 499 LMs in NLP
Jan 22, 2024 Spring

Slides mostly adapted from Dan Jurafsky, some from Mohit Iyyer

Lecture Outline

- Announcements and Recap
  - Probabilistic language models
  - n-gram language models
    - Estimating n-gram probabilities
  - Evaluation and Perplexity
  - Generating from a language model
- Zeros!
- Smoothing
  - Add-1 / Laplace Smoothing
  - Interpolation
  - Back-off and Discounting

Announcements

+

Recap

Announcements

- HW1 Released last week and is due on 1/31
- Next Class — Wed (1/24) — Project Pitches! *Please do not miss class!*
    - 3 minute project pitch (5% of your grade)
        - What is the problem? Why should we care about it?
        - How is this connected to language models?
        - What would the inputs and outputs look like? Examples!
        - Come up with a good name for your project so it's memorable for voting
    - Slides encouraged
    - Everyone votes on teams. We will release all votes and it's up to you to form teams of 3
    - Piazza post soon: By 3pm on Wed, share title and tldr for your project pitch to be included in voting
    - It's natural for the ideas to morph between the project pitch and the project proposal
- Next Week: Extra Office Hours by TA : Tue (1/30) 10-11am
    - Correction: not this Thu

Probabilistic Language Modeling

Goal: compute the probability of a sentence or sequence of words:

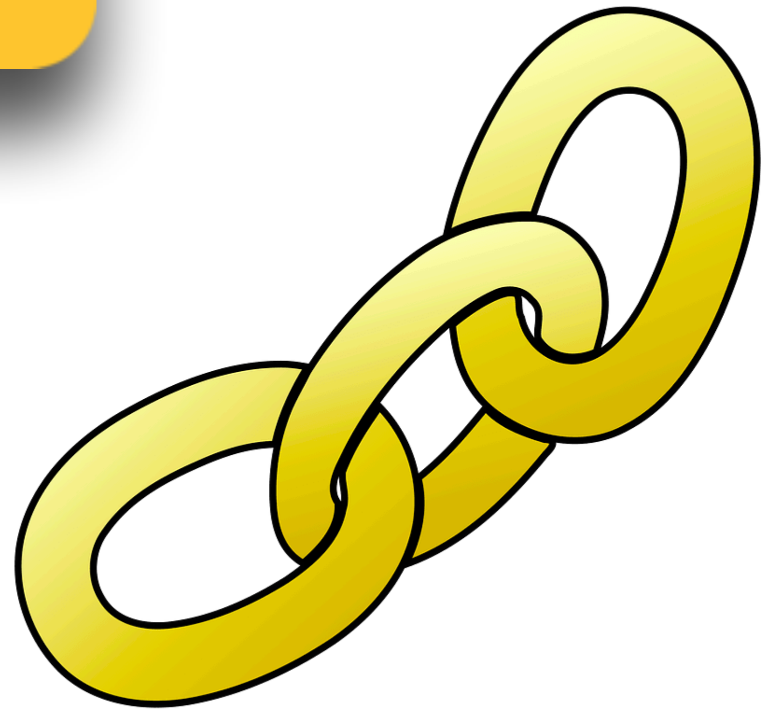
$$P(\mathbf{w}) = P(w_1, w_2, w_3, \ldots w_n)$$

Related task: probability of an upcoming word: $P(w_n \mid w_1, w_2, w_3, w_4, \ldots w_{n-1})$

A model that assigns probabilities to sequences of words is called a language model

Chain Rule

$$P(w_1, w_2, \ldots w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1} \ldots w_1)$$



How to estimate the probability of the next word?

$$P(\text{that} \mid \text{its water is so transparent}) = \frac{Count(\text{its water is so transparent that})}{Count(\text{its water is so transparent})}$$

Maximum Likelihood Estimate

Too many possibilities to count! Too few sentences that look like this…

Need to make some simplifying assumptions…

Markov Assumption

$$P(w_1, w_2, \ldots w_n) = \prod_i P(w_i \mid w_{i-k+1} \ldots w_{i-1})$$

In other words, we approximate each component in the product such that it is only conditioned on the previous k elements

k-gram Markov Assumption

$$P(w_i \mid w_1, w_2, \ldots w_{i-1}) \approx P(w_i \mid w_{i-k+1} \ldots w_{i-1})$$

n-gram models

Unigram Model

$$P(w_1, w2, \ldots w_n) \approx \prod_i P(w_i)$$

Bigram Model

$$P(w_1, w2, \ldots w_n) \approx \prod_i P(w_i \,|\, w_{i-1})$$

k-gram Model

$$P(w_1, w2, \ldots w_n) \approx \prod_i P(w_i \,|\, w_{i-k+1} \ldots w_{i-1})$$

n-gram Models: Limitations

In general this is an insufficient model of language

- "The computer which I had just put into the machine room on the fifth floor crashed."

At times the dependencies are not even clear!

- "The complex houses married and single soldiers and their families."
- "The horse raced past the barn fell."
- "The old man the boat."

Garden Path Sentences

But we can often get away with n-gram models

Language has long-distance dependencies

Estimating bigram probabilities

Maximum Likelihood Estimate

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

We do everything in log space to handle overflow issues

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

Special edge case tokens: <s> and </s> for beginning of sentence and end of sentence, respectively

Likelihood

Suppose we have a biased coin that's heads with probability p

- Let θ be the probability of heads - **parameter** of the model

Suppose we flip the coin four times and see (H, H, H, T)

- **Observed data**, D = (HHHT)

We don't know what θ is — this is the **estimation / learning problem**

- **Maximum Likelihood Estimate**: Choose parameter θ that **maximizes likelihood of observed data** $D, P(D\,|\,\theta)$
  - $\hat{\theta}_{MLE} = \arg\max_{\theta} P(D\,|\,\theta)$
  - Data likelihood, $P(D\,|\,\theta) = \theta\theta\theta(1-\theta)$ can be maximized by taking the derivative and set it equal to zero and find $\hat{\theta}_{MLE} = 0.75$.
- [Advanced] Maximum Aposteriori Probability **(MAP) estimate**
  - $\hat{\theta}_{MAP} = \arg\max_{\theta} P(\theta\,|\,D) = \arg\max_{\theta} P(D\,|\,\theta)P(\theta)$

Maximum Likelihood Estimates

The maximum likelihood estimate

- of some parameter of a model M from a training set T
- maximizes the likelihood of the training set T given the model M

Suppose the word “bagel” occurs 400 times in a corpus of a million words. What is the probability that a random word from some other text will be “bagel”?

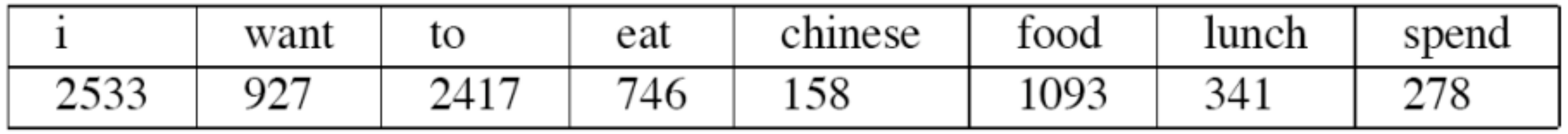
- MLE estimate is 400/1,000,000 = .0004

This may be a bad estimate for some other corpus

- But it is the estimate that makes it *most likely* that “bagel” will occur 400 times in a million word corpus

For the 9222 sentences in the Berkeley Restaurant Corpus:

Unigram Counts

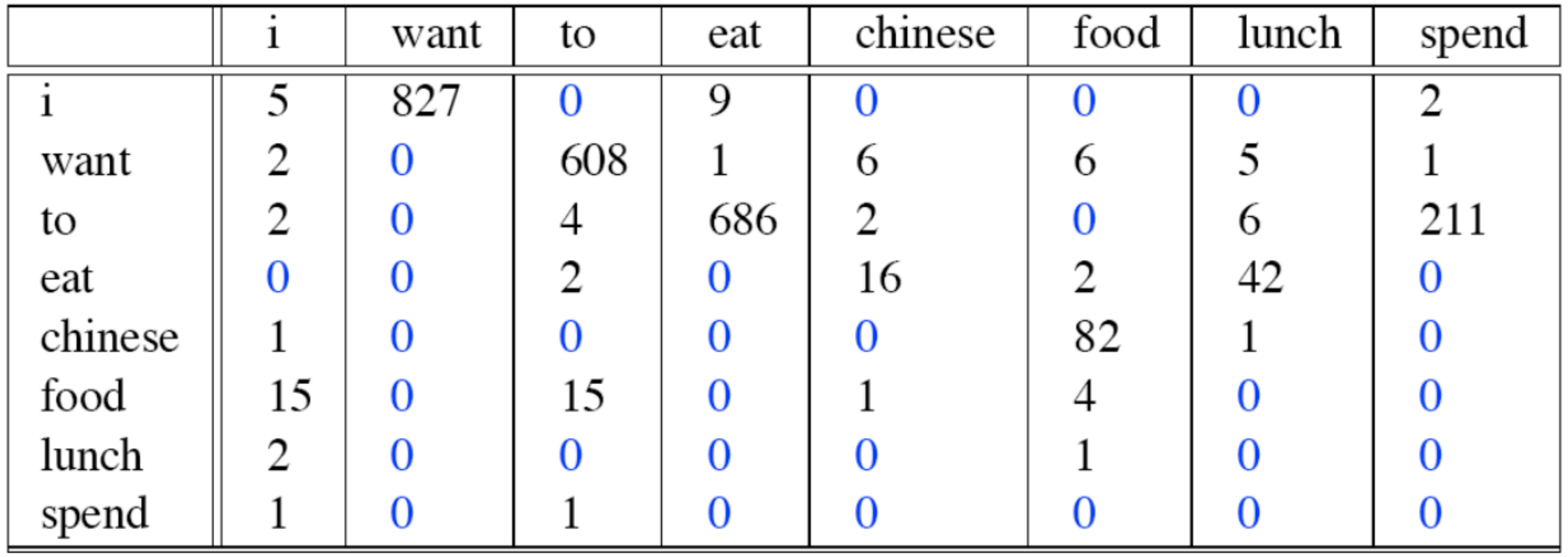


i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Next Word

Bigram Counts

History



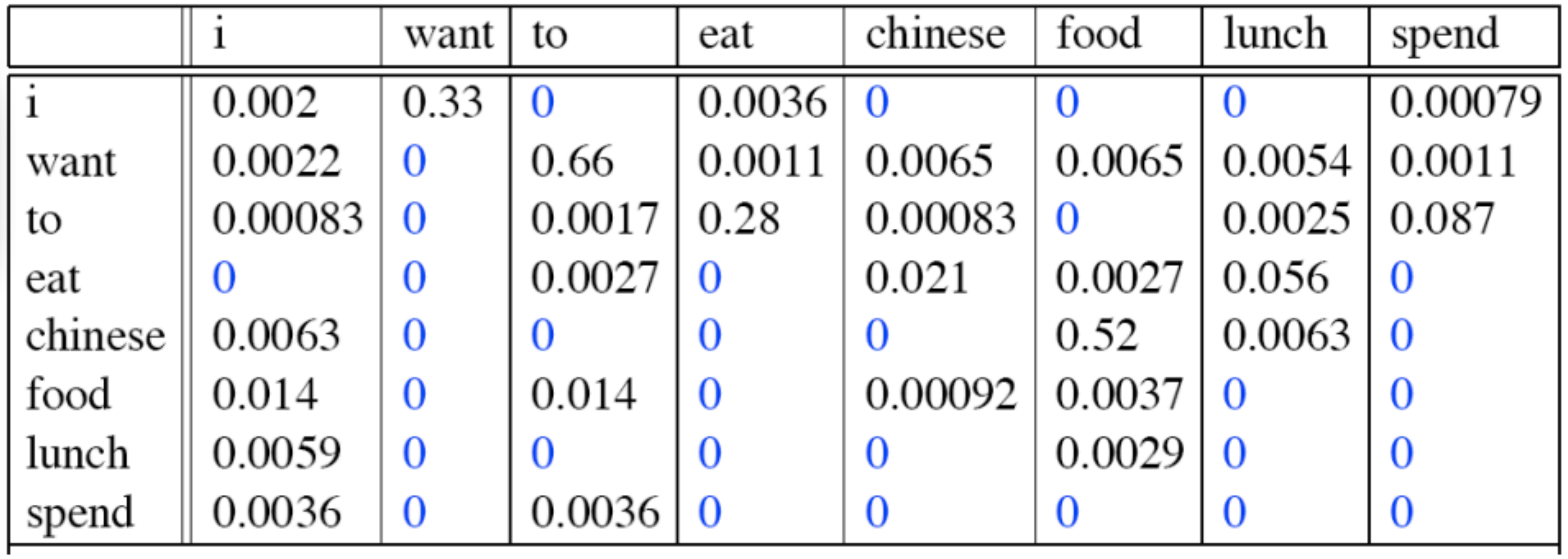
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Most n-grams are never seen!

w_i

Bigram Probabilities

w_{i-1}



	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

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

How good is a language model?

A better model of a text

- is one which assigns a higher probability to the word that actually occurs
- returns the highest probability when evaluated on an unseen test set
  - Mantra: I will never train my model on a test set

Perplexity

$$PPL(\mathbf{w}) = P(w_1 w_2 \ldots w_N)^{\boxed{-\frac{1}{N}}}$$

Normalization Factor

$$= \exp(\boxed{-\frac{1}{N} \log P(w_1 w_2 \ldots w_N)})$$

Negative log likelihood

Bigram Perplexity

$$PPL(\mathbf{w}) = \exp(-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i \,|\, w_{i-1}))$$

Lower the perplexity, better the language model

WSJ Perplexities

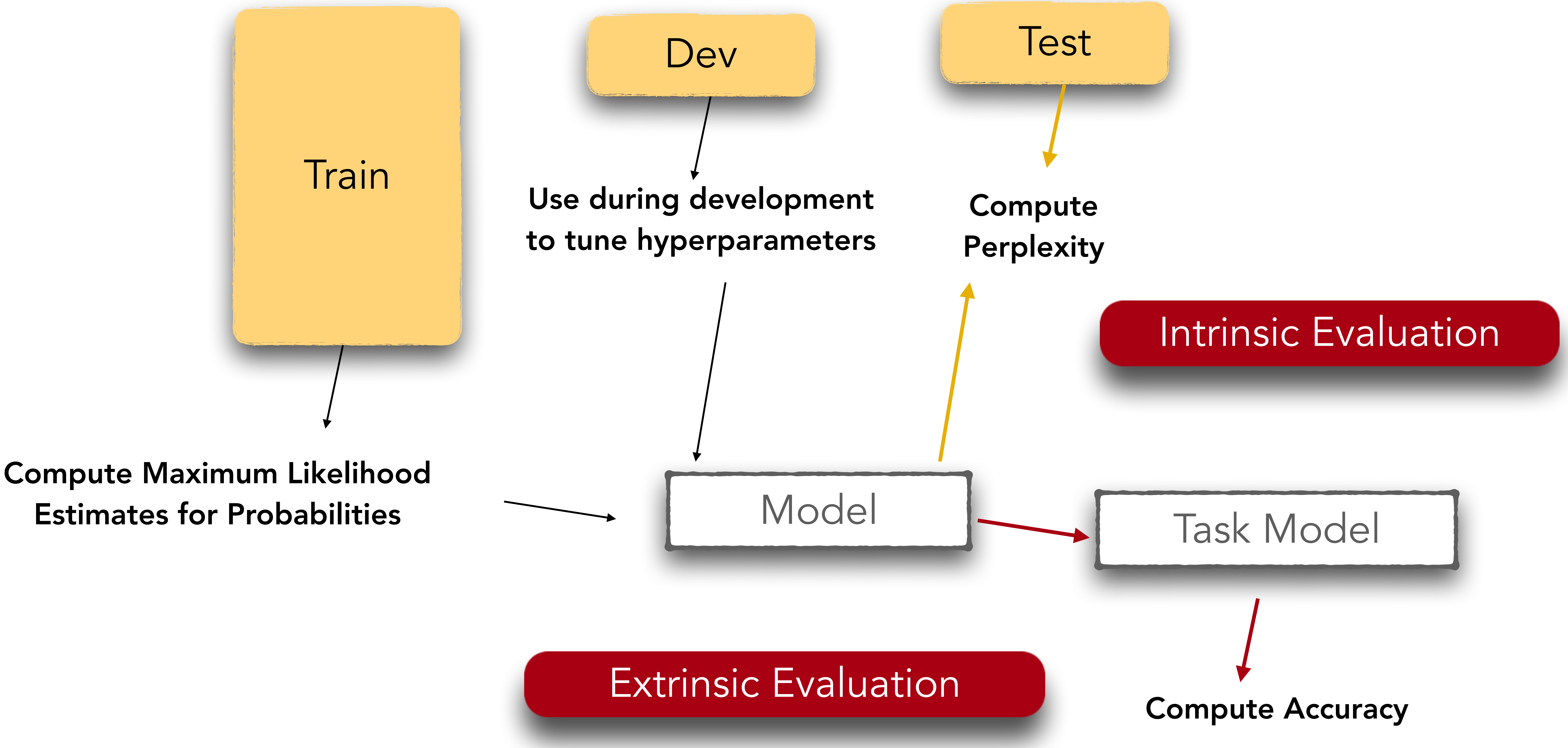
N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

n-grams do a better and better job of modeling the training corpus as we increase the value of *n*

How best to evaluate an LM?

- Extrinsic evaluation
  - On an external task (e.g. summarization) that uses an LM
  - More reliable
  - Can be time-consuming; hard to design
    - Which is the best task? How many tasks to try?
- Therefore, we often use intrinsic evaluation:
  - Bad approximation (less reliable)
    - Unless the test data looks just like the training data
  - Generally only useful in pilot experiments (faster to compute)

Language Model Development



Train

Dev

Test

Use during development to tune hyperparameters

Compute Perplexity

Intrinsic Evaluation

Compute Maximum Likelihood Estimates for Probabilities

Model

Task Model

Extrinsic Evaluation

Compute Accuracy

Generating from a bigram model

Done via Sampling

Sampling: choosing random points according to their likelihood

Method dates back to the early 1950s! Shannon era

1. Choose a random bigram (<s>, w) according to its probability
2. Now choose a random bigram (w, x) according to its probability
3. And so on until we choose </s>
4. Then string the words together

<s> I

I want

want to

to eat

eat Chinese

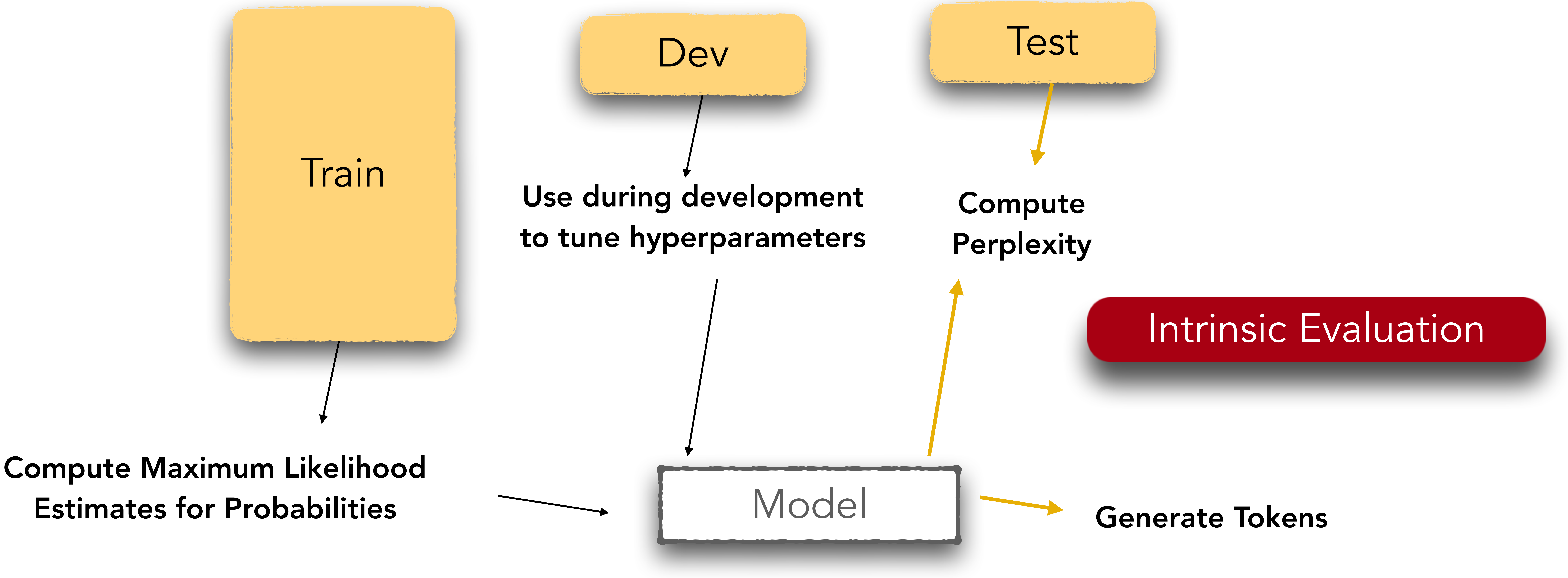
Chinese food

food </s>

I want to eat Chinese food

Helps visualize how good the model is!

Language Model Development



Train

Dev

Test

Use during development to tune hyperparameters

Compute Perplexity

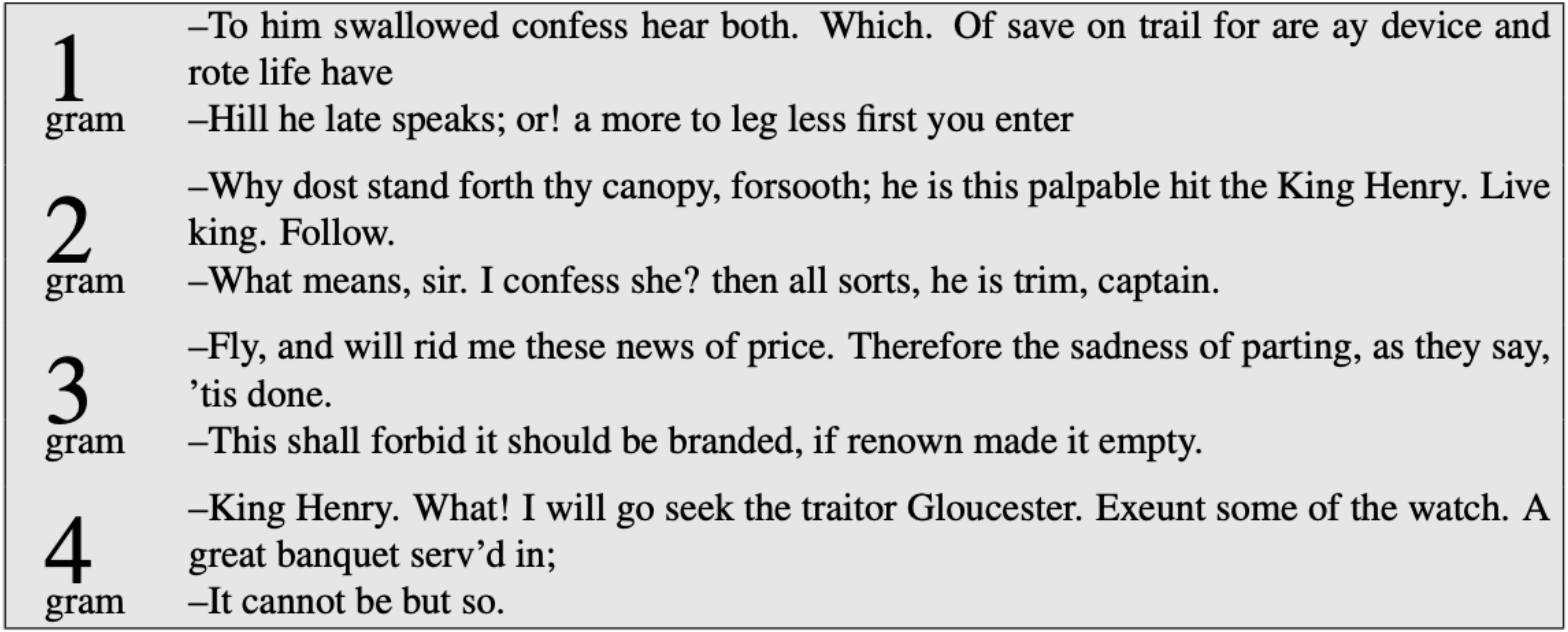
Intrinsic Evaluation

Compute Maximum Likelihood Estimates for Probabilities

Model

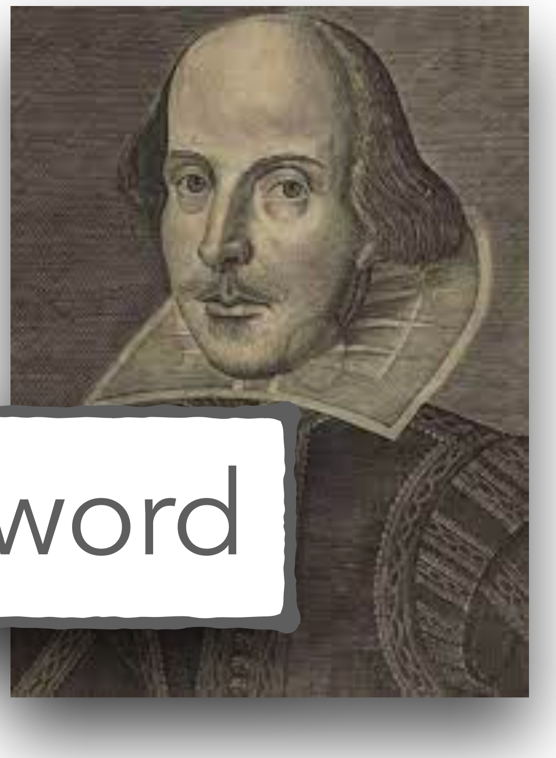
Generate Tokens

Shakespearean n-grams



1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

Shakespeare as a corpus



Token Type Vocabulary

A token is a technical term in NLP for what is commonly referred to as a word

$N = 884{,}647$ tokens, $V = 29{,}066$ vocabulary items (types)

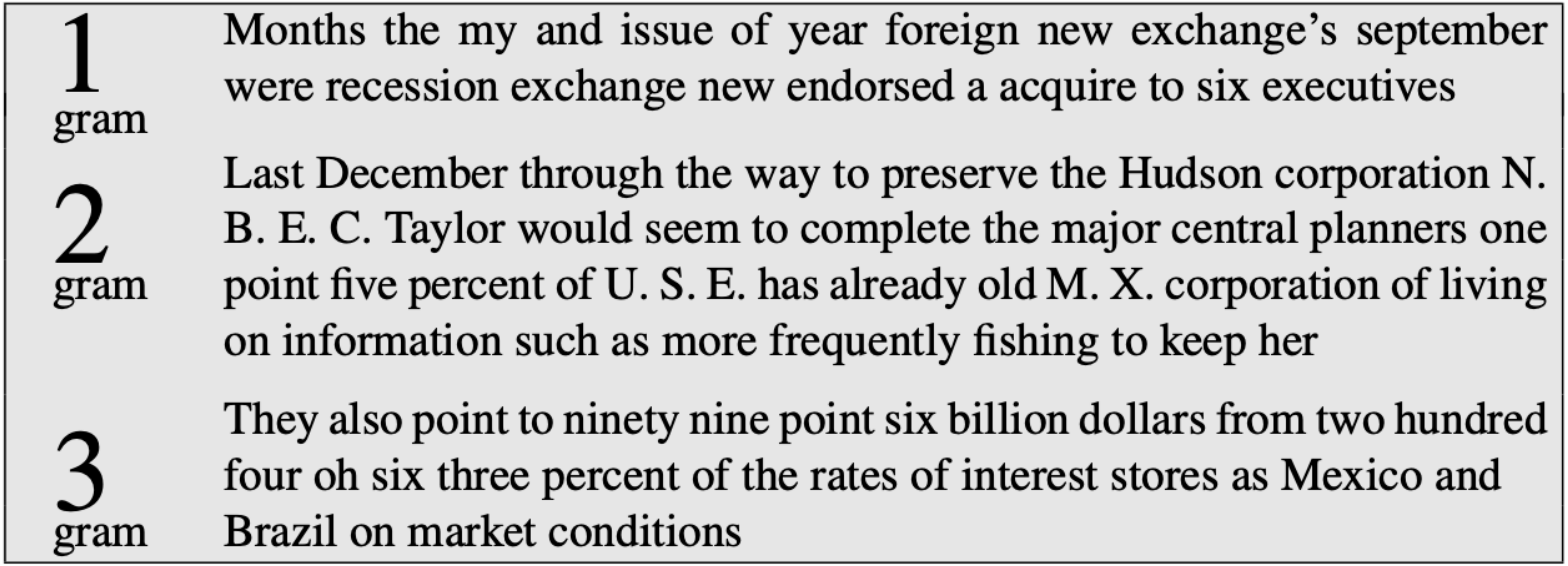
Shakespeare produced 300,000 bigram types out of $V^2 = 844$ million possible bigrams!

So 99.96% of the possible bigrams were never seen (have zero entries in the table)

4-grams (quadrigrams) are worse: What's coming out looks like Shakespeare because it is Shakespeare!

Most n-grams are never seen!

The WSJ is no Shakespeare!

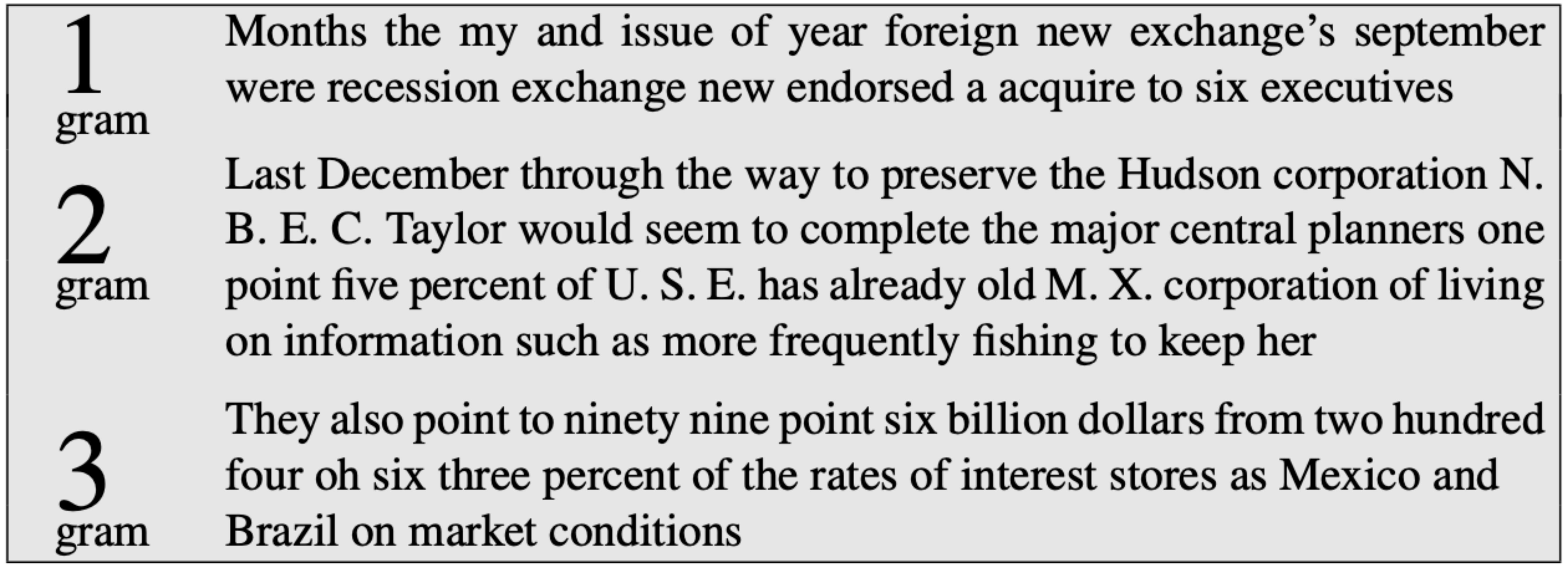


1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

So why not just sample from very high order n-gram models? Do we even need GPT?

Can only produce n-grams from training data!

Shakespearean corpus cannot produce WSJ vocabulary and vice versa



1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

The successes we are seeing here are due to a phenomena commonly known as overfitting

Overfitting bad!

Robustness

n-grams only work well for word prediction if the test corpus looks like the training corpus

- In real life, it often doesn’t
- We need to train **robust** models that **generalize**!
  - Technical terms for “doing well on the test data” or “doing well on any test data”

Generalization

- One kind of generalization pitfall: Zeros!
  - Things that don’t ever occur in the training set
    - But occur in the test set

Zeros

Training set:

allegations	report	claims	request
4	1	2	2

Test set

… offer c (**offer**) $= 0$

… loan c (**loan**) $= 0$

Bigram Probabilities:

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

Undefined probabilities!

N-gram models: Zero Counts

- Problem: Word “offer” didn’t appear in the train set…

At test time, we may encounter tokens never seen (unigram with 0 frequency)

  - Very severe problem resulting in undefined probabilities
  - Problem: Many words like “Swayamdipta” won’t appear in most training sets!
  - Other problems: new terms, different dialects, evolving language
  - These are known as **OOV** for “out of vocabulary”, or **unknown tokens**
- Design: Open Vocabulary vs. Closed Vocabulary

Open vs. Closed Vocabularies

Solution for missing tokens: the <UNK> token

These are known as **OOV** for “out of vocabulary”, or **unknown tokens**

One way to handle OOV tokens is by adding a pseudo-word called <UNK>

We can replace all words that occur fewer than n times in the training set, where n is some small number by <UNK> and re-estimate the counts and probabilities

When not done carefully, may artificially lower perplexity

Does this solve all our problems?

Training set:

… denied the allegations

… denied the reports

… denied the claims

… denied the request

Test set

… denied the offer

… denied the loan

We might still encounter tokens never seen *in context* (i.e. n-gram with 0 frequency)

$$P\left(\textbf{offer} \mid \textbf{denied the}\right) = 0$$

will assign 0 probability to the test set!

What happens to perplexity??

Zero probability bigrams

Bigrams with zero probability

- mean that we will assign 0 probability to the test set!

And hence we cannot compute perplexity

- No one can divide by 0!

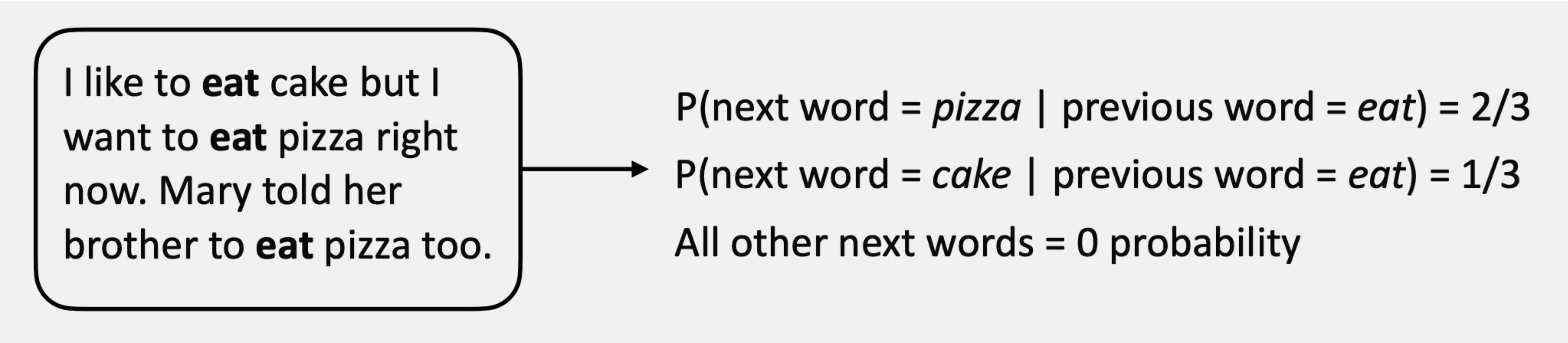
$$PPL(\mathbf{w}) = \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Lecture Outline

- Announcements and Recap
  - Probabilistic language models
  - n-gram language models
    - Estimating n-gram probabilities
  - Evaluation and Perplexity
  - Generating from a language model
- Zeros!
- Smoothing
  - Add-1 / Laplace Smoothing
  - Interpolation
  - Back-off and Discounting

Smoothing

Intuition for Smoothing



I like to **eat** cake but I want to **eat** pizza right now. Mary told her brother to **eat** pizza too.

P(next word = *pizza* | previous word = *eat*) = 2/3

P(next word = *cake* | previous word = *eat*) = 1/3

All other next words = 0 probability

- Types: I, like, to, eat, cake, but, want, pizza, right, now, ., Mary, told, her, brother, too
  - $|V| = ?$ $|V_{\text{bigrams}}| = ?$
- All other vocabulary tokens getting 0 probability just doesn't seem right. We want to assign some probability to other words
- We want to **smooth the distribution from our counts**

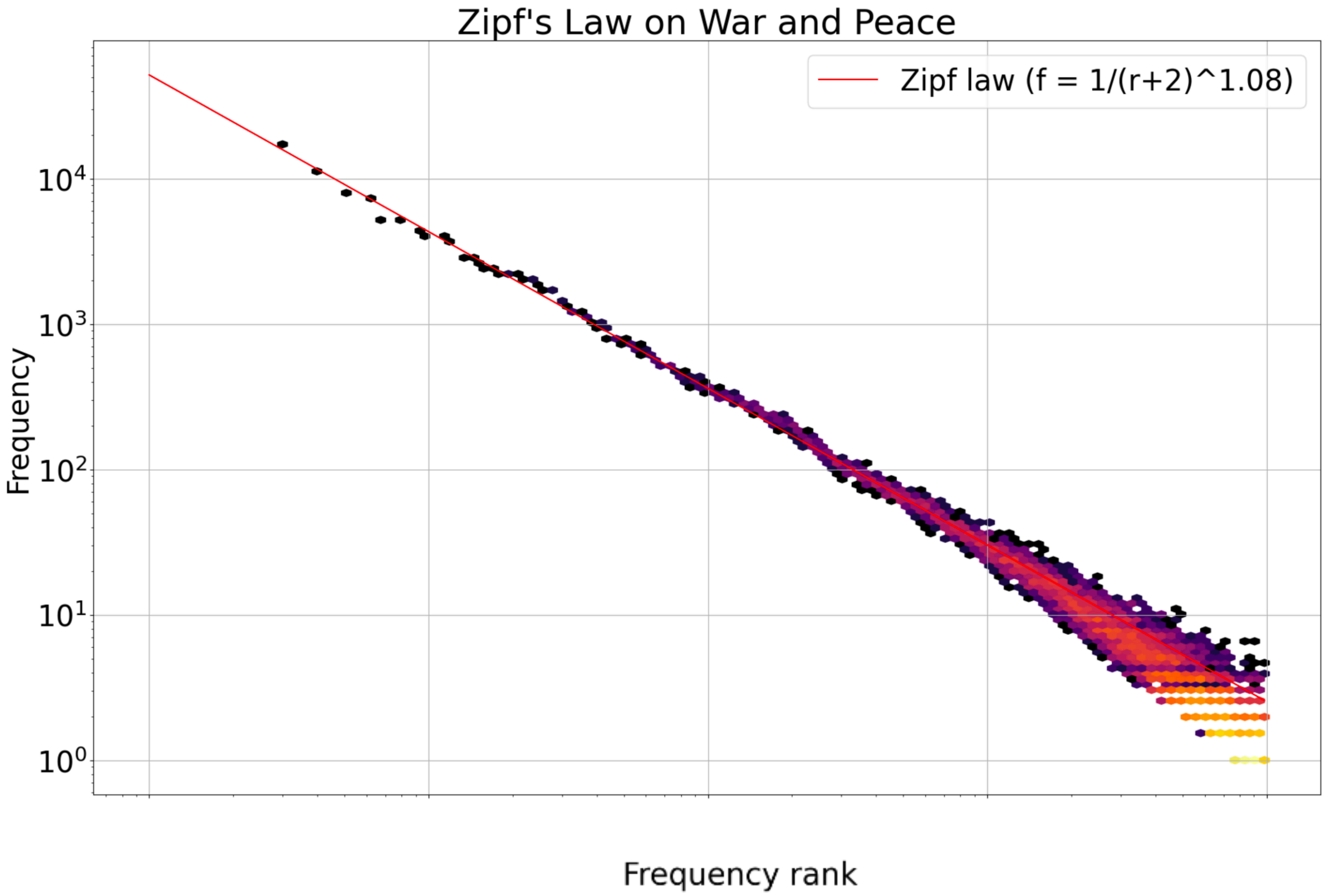
What does a count distribution look like?

Zipf’s Law

The distribution over words resembles that of a power law:

- there will be a few words that are very frequent, and a long tail of words that are rare
- $freq_w(r) \approx r^{-s}$

NLP algorithms must be especially robust to observations that do not occur or rarely occur in the training data

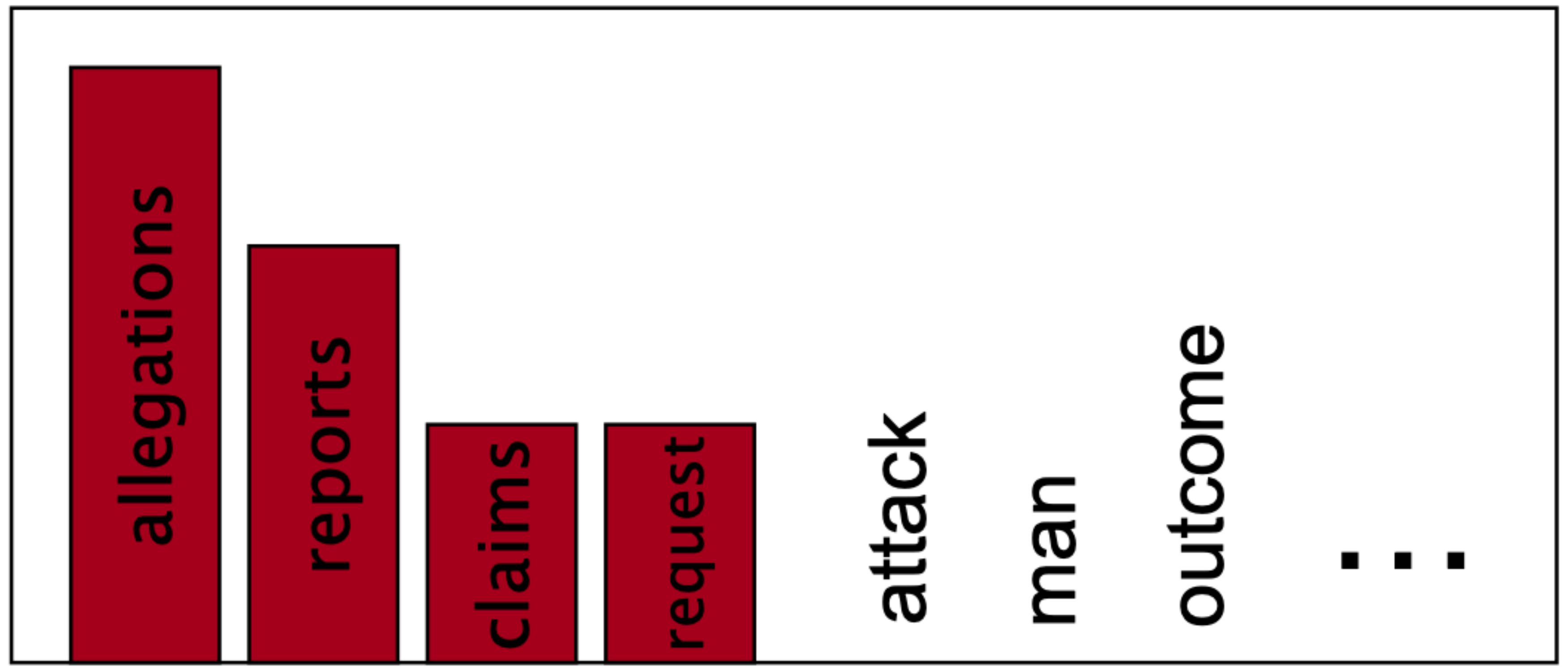


Zipf, G. K. (1949). Human behavior and the principle of least effort.

Smoothing ~ Massaging Probability Masses

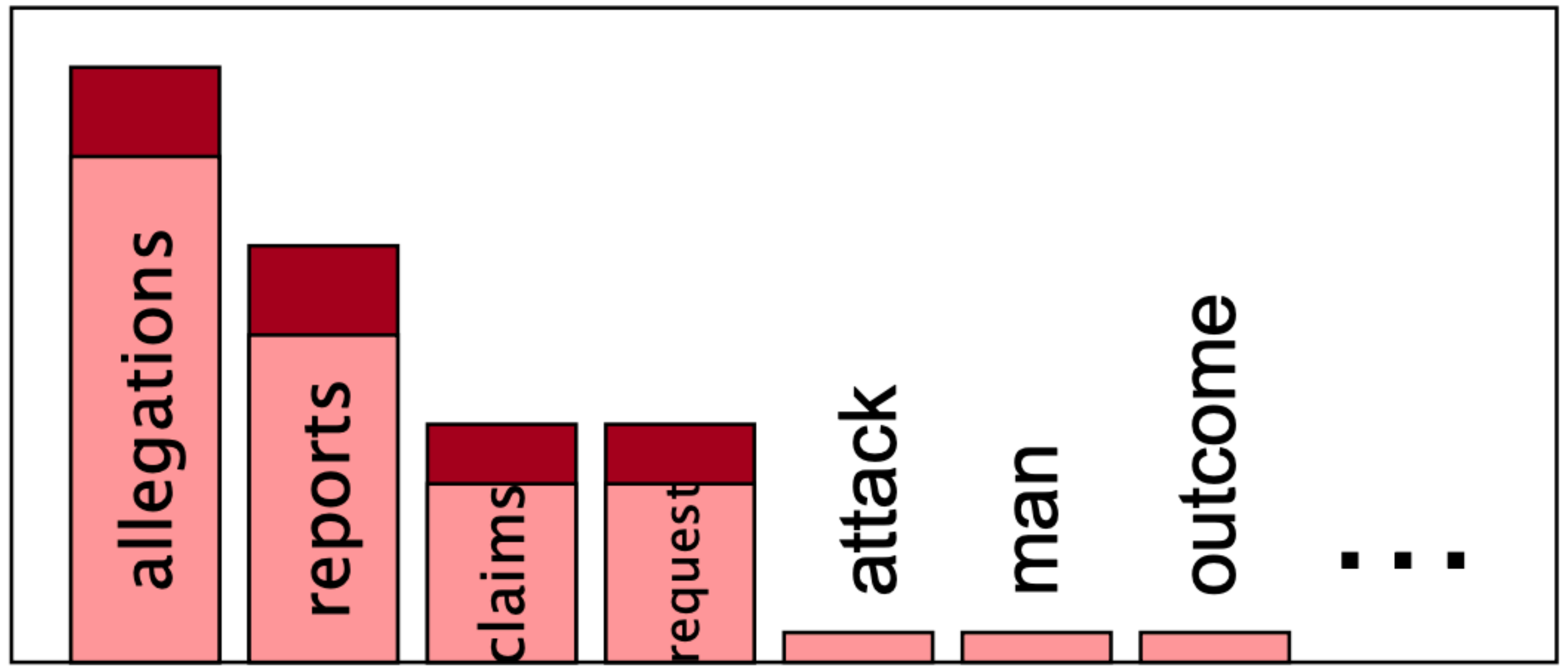
When we have sparse statistics: $Count(w \mid \text{denied the})$

3 allegations
2 reports
1 claims
1 request
7 total



Steal probability mass to generalize better: $Count(w \mid \text{denied the})$

2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



Add-One Estimation

MLE estimate

$$P_{MLE}(w_i) = \frac{c(w_i)}{\sum_w c(w)}$$

Laplace smoothing

75 year old method!

1. Pretend we saw each word one more time than we did
2. Just add one to all the counts!
3. All the counts that used to be zero will now have a count of 1…

Add-1 estimate

$$P_{Add-1}(w_i) = \frac{c(w_i)+1}{\sum_w (c(w)+1)} = \frac{c(w_i)+1}{V+\sum_w c(w)}$$

What happens to our *P* if we don't increase the denominator?

Add-1 Estimation Bigrams

MLE estimate

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

Pretend we saw each bigram one more time than we did

Add-1 estimate

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}w_i) + 1}{c(w_{i-1}) + V}$$

What does this do to the unigram counts?

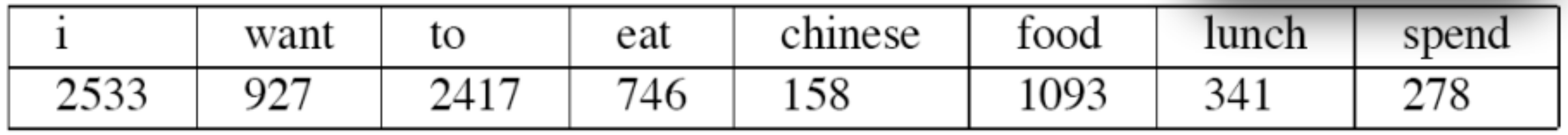
Keep the same denominator as before and reconstruct bigram counts

$$= \frac{c^*(w_{i-1}w_i)}{c(w_{i-1})}$$

Recall: BRP 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

Unigrams

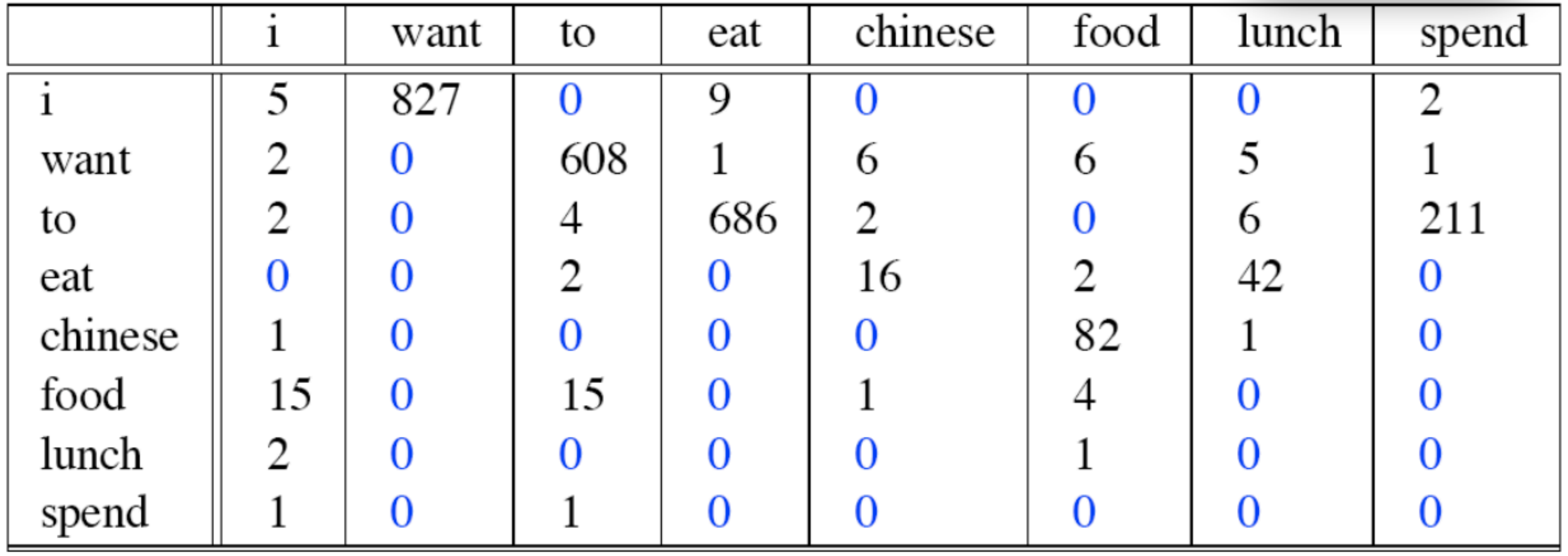


i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Bigrams

w_i

w_{i-1}



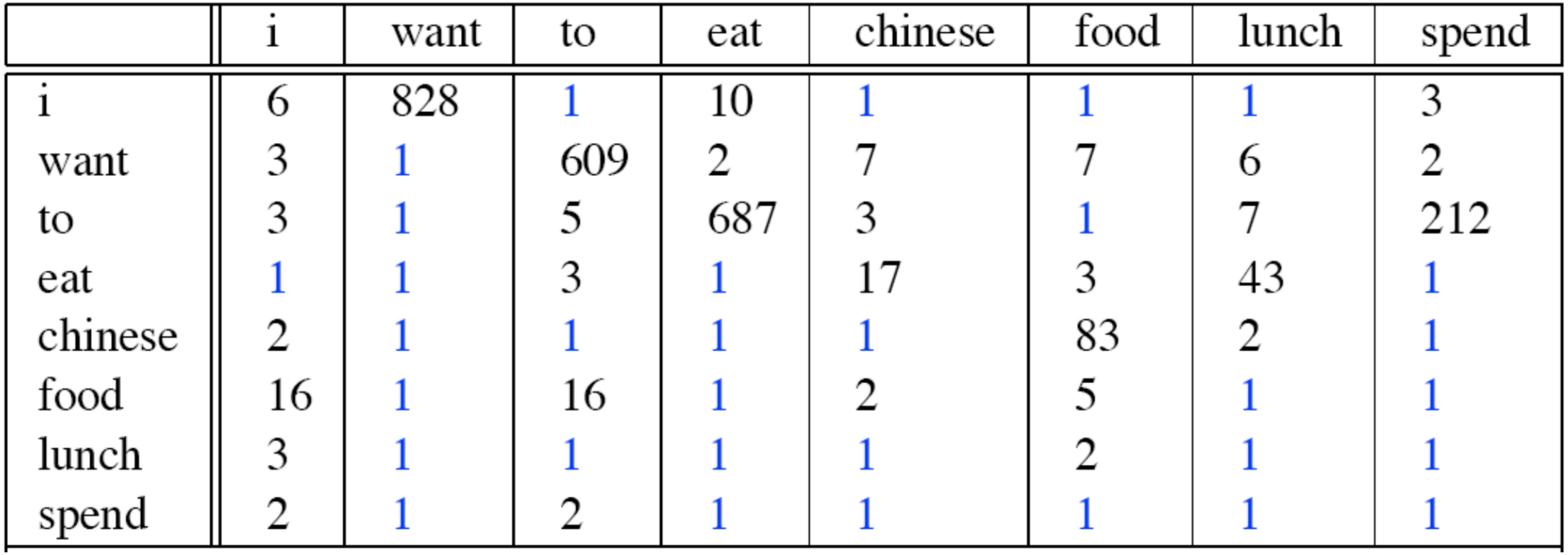
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Laplace-smoothed bigram counts

Just add one to all the counts!

w_i

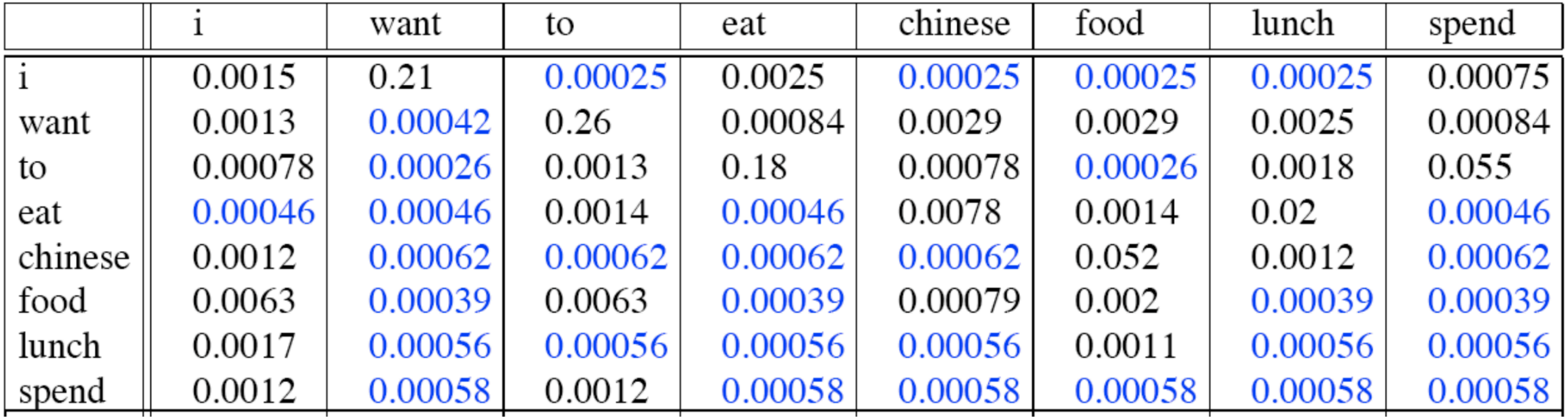
w_{i-1}



	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

Laplace-smoothed bigram probabilities

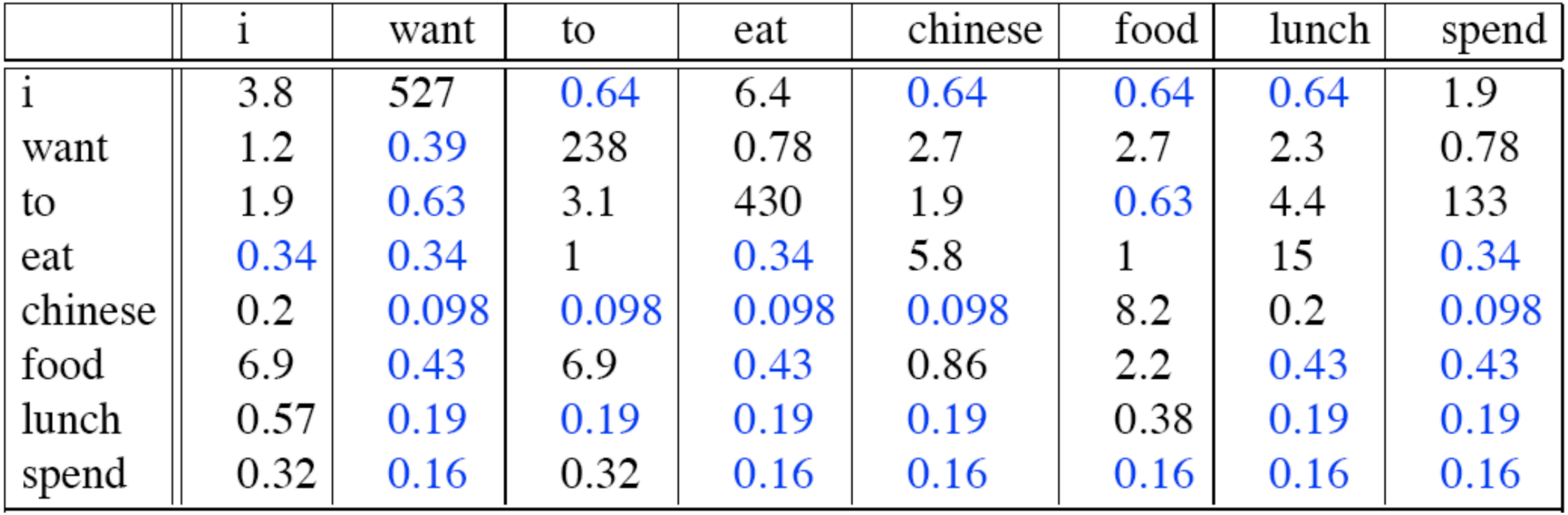
$$P_{Add-1}\left(w_i \mid w_{i-1}\right) = \frac{c(w_{i-1}w_i) + 1}{c(w_{i-1}) + V}$$



	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

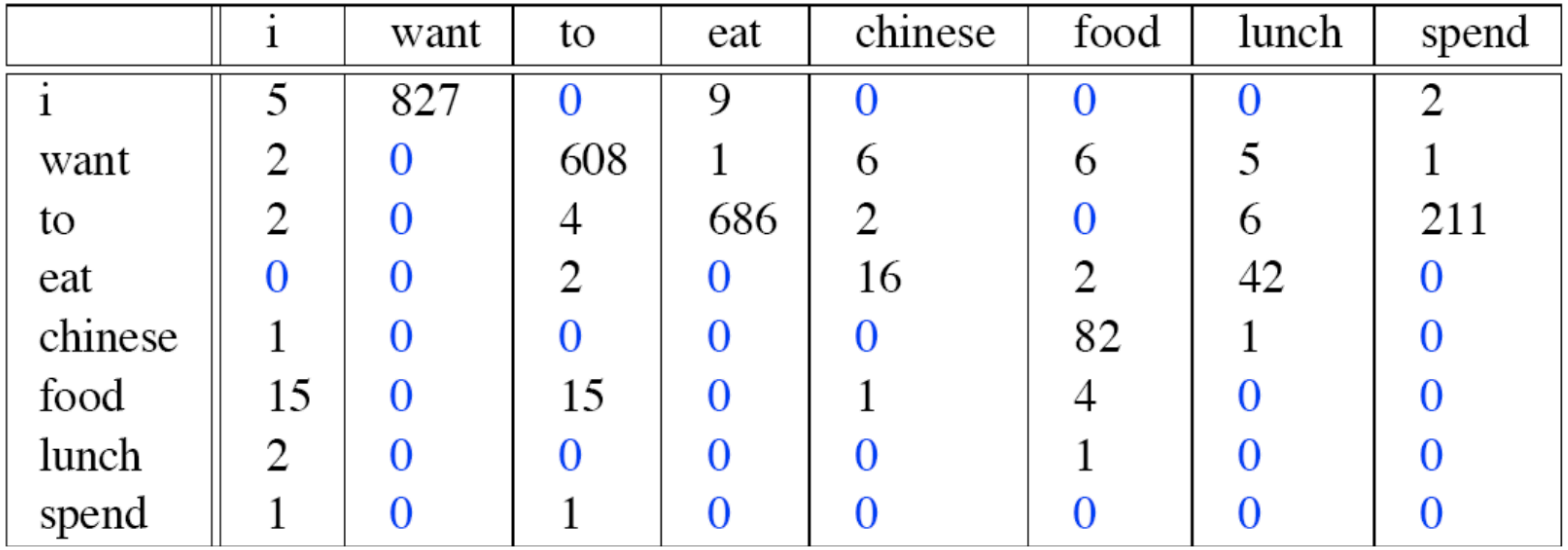
$$c*(w_{i-1}w_i) = \frac{[c(w_{i-1}w_i) + 1]c(w_{i-1})}{c(w_{i-1}) + V}$$



	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

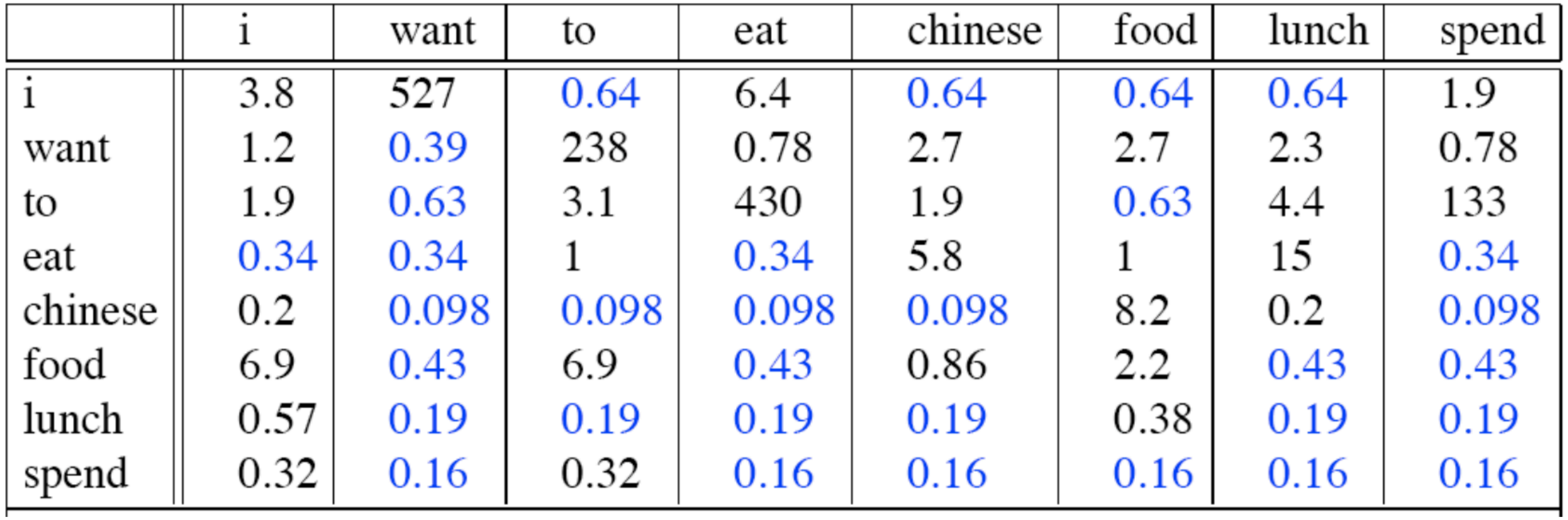
Compare with raw bigram counts

Original, Raw



	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

Reconstructed



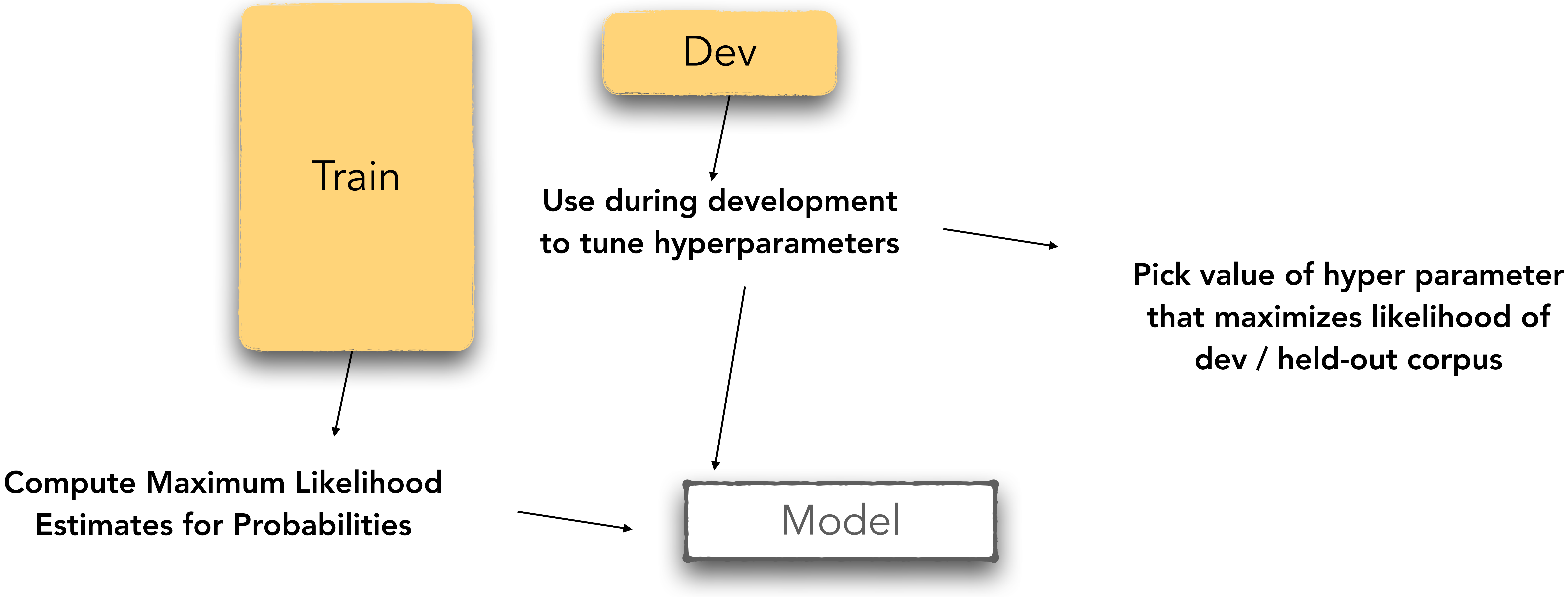
	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19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

Big change to the counts!

Perhaps 1 is too much, add a fraction?

Add-k smoothing

Language Model Development



Train

Dev

Use during development to tune hyperparameters

Pick value of hyper parameter that maximizes likelihood of dev / held-out corpus

Compute Maximum Likelihood Estimates for Probabilities

Model

Add-1 Estimation: Last thoughts

So add-1 isn't used for n-grams, being something of a blunt instrument

- One-size-fits-all

Add-1 is used to smooth other NLP models though…

- For text classification
- In domains where the number of zeros isn't so huge

Interpolation

Perhaps use some pre-existing evidence

- Condition on less context for contexts you haven’t learned much about

Interpolation

- mix unigram, bigram, trigram probabilities for a trigram LM
- mix n-gram, (n-1)-gram, … unigram probabilities for an n-gram LM

Interpolation works better than Add-1 / Laplace

Linear Interpolation

Simple Interpolation

$$\hat{P}(w_i | w_{i-2}w_{i-1}) = \lambda_1 P(w_i) + \lambda_2 P(w_i | w_{i-1}) + \lambda_3 P(w_i | w_{i-2}w_{i-1})$$

$$\sum_k \boxed{\lambda_k} = 1$$

Hyperparameters!

Context-Conditional Interpolation

$$\hat{P}(w_i | w_{i-2}w_{i-1}) = \lambda_3(w_{i-2}^{i-1}) P(w_i | w_{i-2}w_{i-1}) + \lambda_2(w_{i-2}^{i-1}) P(w_i | w_{i-1}) + \lambda_1(w_{i-2}^{i-1}) P(w_i)$$

Reconstituted Counts

Different for every unique context

How to set the λs?

Choose λs to maximize the probability of held-out data:

- Fix the n-gram probabilities (on the training data)
- Then search for λs that give largest probability to held-out set:

$$logP(w_1 \ldots w_n | M(\lambda_1 \ldots \lambda_k)) = \sum_i logP_{M(\lambda_1 \ldots \lambda_k)}(w_i | w_{i-1})$$

Backoff and Discounting

Backoff

- use trigram if you have good evidence,
- otherwise bigram, otherwise unigram

Still need a correct probability distribution!

- discount the higher-order n-grams by d to save some probability mass for the lower order n-grams
- need a function α to distribute this probability mass to the lower order n-grams

Stupid Backoff

No discounting, just use relative frequencies

Don't care about a valid language model

Not a probability distribution (usually denoted as P)

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

Hyperparameter!

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

Brants *et al.* 2007

Katz Backoff

Rely on a discounted probability $P*$

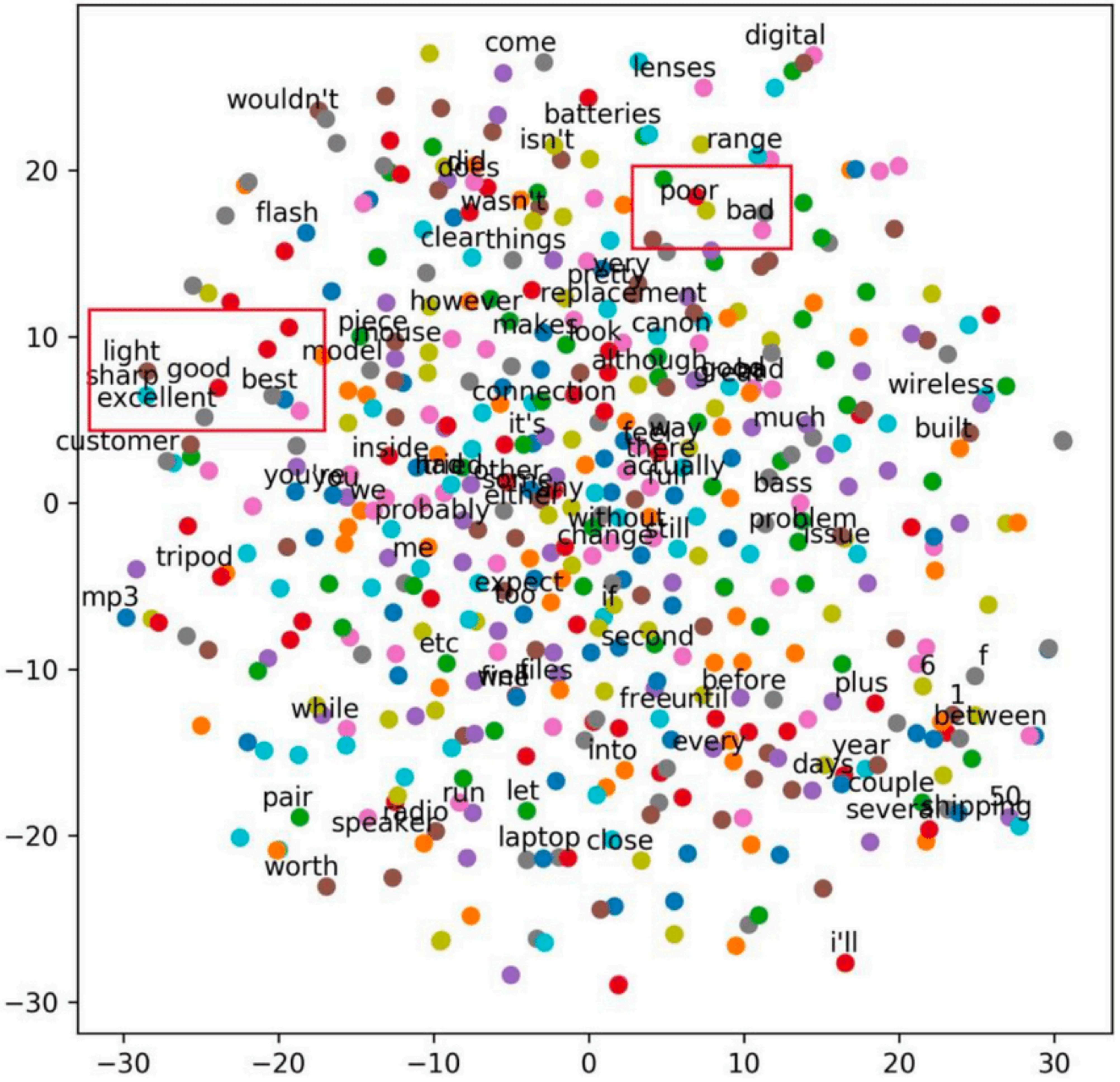
- if we've seen this n-gram before (i.e., if we have non-zero counts)
- otherwise, we recursively back off to the Katz probability for the shorter-history (n-1)-gram

$$P_{BO}(w_i \mid w_{i-k+1:i-1}) = \begin{cases} P*(w_i \mid w_{i-k+1:i-1}), & \textbf{if } c(w_{i-k+1:i}) > 0 \\ \alpha(i-k+1:i-1)P_{BO}(w_i \mid w_{i-k+2:i-1}), & \textbf{otherwise} \end{cases}$$

Recursive Formulation!

Next Topics

- Words are more than discrete symbols
- Better feature representations than n-grams
- Parameters!!!
- But first, Logistic Regression and Classification in Machine Learning


come
digital
lenses
wouldn't
batteries
isn't
range
did
does
wasn't
poor
bad
flash
clearthings
very
pretty
however
replacement
piece
mouse
makes
look
canon
model
light
sharp
good
best
excellent
although
great
connection
wireless
it's
much
built
customer
inside
way
there
actually
youre
we
other
either
bass
probably
without
change
still
problem
issue
tripod
me
expect
too
if
mp3
etc
second
f
files
while
free
until
before
plus
between
into
every
year
days
couple
pair
run
let
radio
speaker
several
shipping
laptop
close
worth
i'll