

# Language Modeling

## Introduction to N-grams

# Probabilistic Language Models

Today's goal: assign a probability to a sentence

- Machine Translation:
  - $P(\text{high winds tonite}) > P(\text{large winds tonite})$

- Spell Correction

Why?

- The office is about fifteen **minuets** from my house
  - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
- Speech Recognition
  - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
- + Summarization, question-answering, etc., etc.!!

# Probabilistic Language Modeling

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

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

A model that computes either of these:

$P(W)$  or  $P(w_n | w_1, w_2 \dots w_{n-1})$  is called a **language model**.

Better: **the grammar** But **language model** or **LM** is standard

# How to compute $P(W)$

How to compute this joint probability:

- $P(\text{its, water, is, so, transparent, that})$

Intuition: let's rely on the Chain Rule of Probability

# Reminder: The Chain Rule

Recall the definition of conditional probabilities

$$p(\mathbf{B} | \mathbf{A}) = \mathbf{P}(\mathbf{A}, \mathbf{B}) / \mathbf{P}(\mathbf{A}) \quad \text{Rewriting:} \quad \mathbf{P}(\mathbf{A}, \mathbf{B}) = \mathbf{P}(\mathbf{A})\mathbf{P}(\mathbf{B} | \mathbf{A})$$

More variables:

$$P(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}) = P(\mathbf{A})P(\mathbf{B} | \mathbf{A})P(\mathbf{C} | \mathbf{A}, \mathbf{B})P(\mathbf{D} | \mathbf{A}, \mathbf{B}, \mathbf{C})$$

The Chain Rule in General

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2 | x_1)P(x_3 | x_1, x_2) \dots P(x_n | x_1, \dots, x_{n-1})$$

# The Chain Rule applied to compute joint probability of words in sentence



$$\begin{aligned} P(\text{"its water is so transparent"}) = \\ P(\text{its}) \times P(\text{water} | \text{its}) \times P(\text{is} | \text{its water}) \\ \times P(\text{so} | \text{its water is}) \times P(\text{transparent} | \text{its water is so}) \end{aligned}$$

# How to estimate these probabilities

Could we just count and divide?



No! too many possible sentences:

We'll never see enough data for estimating these

# Markov Assumption



Andrei Markov

Simplifying assumption:



Or maybe





# Markov Assumption



In other words, we approximate each component in the product



# Simplest case: Unigram model



Some automatically generated sentences from a unigram model

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

# Bigram model

- Condition on the previous word:



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

this, would, be, a, record, november

# N-gram models

We can extend to trigrams, 4-grams, 5-grams

In general this is an insufficient model of language

- because language has **long-distance dependencies**:

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

But we can often get away with N-gram models

# Language Modeling

## Introduction to N-grams

# Language Modeling

## Estimating N-gram Probabilities

# Estimating bigram probabilities

## The Maximum Likelihood Estimate



# An example



<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\text{I} \mid \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{Sam} \mid \text{<s>}) = \frac{1}{3} = .33$$

$$P(\text{am} \mid \text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>} \mid \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} \mid \text{am}) = \frac{1}{2} = .5$$

$$P(\text{do} \mid \text{I}) = \frac{1}{3} = .33$$



## More examples: Berkeley Restaurant Project sentences

can you tell me about any good cantonese restaurants close by  
mid priced thai food is what i'm looking for  
tell me about chez panisse  
can you give me a listing of the kinds of food that are available  
i'm looking for a good place to eat breakfast  
when is caffe venezia open during the day

# Raw bigram counts

Out of 9222 sentences

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

# Raw bigram probabilities

Normalize by unigrams:

Result:

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

	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

# Bigram estimates of sentence probabilities

$$\begin{aligned} P(<s> \text{ I want english food } </s>) = \\ & P(\text{I} | <s>) \\ & \times P(\text{want} | \text{I}) \\ & \quad \times P(\text{english} | \text{want}) \\ & \quad \times P(\text{food} | \text{english}) \\ & \quad \times P(</s> | \text{food}) \\ & = .000031 \end{aligned}$$

# What kinds of knowledge?

$$P(\text{english} | \text{want}) = .0011$$

$$P(\text{chinese} | \text{want}) = .0065$$

$$P(\text{to} | \text{want}) = .66$$

$$P(\text{eat} | \text{to}) = .28$$

$$P(\text{food} | \text{to}) = 0$$

$$P(\text{want} | \text{spend}) = 0$$

$$P(i | \langle s \rangle) = .25$$

# Practical Issues

We do everything in log space

- Avoid underflow
- (also adding is faster than multiplying)



# Language Modeling Toolkits

## SRILM

- <http://www.speech.sri.com/projects/srilm/>

## KenLM

- <https://kheafield.com/code/kenlm/>

# Google N-Gram Release, August 2006



...





# Google N-Gram Release

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 indispensable 40  
serve as the individual 234

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

# Google Book N-grams

<http://ngrams.googlelabs.com/>

# Language Modeling

## Estimating N-gram Probabilities

# Language Modeling

## Evaluation and Perplexity

# Evaluation: How good is our model?

Does our language model prefer good sentences to bad ones?

- Assign higher probability to “real” or “frequently observed” sentences
- Than “ungrammatical” or “rarely observed” sentences?

We train parameters of our model on a **training set**.

We test the model’s performance on data we haven’t seen.

- A **test set** is an unseen dataset that is different from our training set, totally unused.
- An **evaluation metric** tells us how well our model does on the test set.

# Extrinsic evaluation of N-gram models

Best evaluation for comparing models A and B

- Put each model in a task
  - spelling corrector, speech recognizer, MT system
- Run the task, get an accuracy for A and for B
  - How many misspelled words corrected properly
  - How many words translated correctly
- Compare accuracy for A and B

# Difficulty of extrinsic (in-vivo) evaluation of N-gram models

## Extrinsic evaluation

- Time-consuming; can take days or weeks

So

- Sometimes use **intrinsic** evaluation: **perplexity**
- Bad approximation
  - unless the test data looks **just** like the training data
  - So **generally only useful in pilot experiments**
- But is helpful to think about.

# Intuition of Perplexity

## The Shannon Game:

- How well can we predict the next word?  
I always order pizza with cheese and \_\_\_\_  
The 33<sup>rd</sup> President of the US was \_\_\_\_  
I saw a \_\_\_\_
- Unigrams are terrible at this game. (Why?)

mushrooms 0.1  
pepperoni 0.1  
anchovies 0.01  
....  
fried rice 0.0001  
....  
and 1e-100

## A better model of a text

- is one which assigns a higher probability to the word that actually occurs



# Perplexity

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

- Gives the highest  $P(\text{sentence})$

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



Chain rule:

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

For bigrams:

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

**Minimizing perplexity is the same as maximizing probability**

# The Shannon Game intuition for perplexity

From Josh Goodman

Perplexity is weighted equivalent branching factor

How hard is the task of recognizing digits '0,1,2,3,4,5,6,7,8,9'

- Perplexity 10

How hard is recognizing (30,000) names at Microsoft.

- Perplexity = 30,000

Let's imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1 in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)
- What is the perplexity? Next slide

# The Shannon Game intuition for perplexity

Josh Goodman: imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1 in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)

We get the perplexity of this sequence of length 120K by first multiplying 120K probabilities (90K of which are  $1/4$  and 30K of which are  $1/120K$ ), and then taking the inverse 120,000th root:

$$\text{Perp} = (\frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \dots * \frac{1}{120K} * \frac{1}{120K} * \dots)^{(-1/120K)}$$

But this can be arithmetically simplified to just  $N = 4$ : the operator ( $1/4$ ), the sales ( $1/4$ ), the tech support ( $1/4$ ), and the 30,000 names ( $1/120,000$ ):

$$\text{Perplexity} = ((\frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{120K})^{(-1/4)}) = 52.6$$

# Perplexity as branching factor

Let's suppose a sentence consisting of random digits

What is the perplexity of this sentence according to a model that assign  $P=1/10$  to each digit?

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

# Lower perplexity = better model

Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

# Language Modeling

## Evaluation and Perplexity

# Language Modeling

Generalization and zeros

# The Shannon Visualization Method

Choose a random bigram

( $\langle s \rangle$ ,  $w$ ) according to its probability

Now choose a random bigram ( $w$ ,  $x$ ) according to its probability

And so on until we choose  $\langle /s \rangle$

Then string the words together

$\langle s \rangle$  I  
I want  
want to  
to eat  
eat Chinese  
Chinese food  
food  $\langle /s \rangle$   
I want to eat Chinese food



# Approximating Shakespeare



# Shakespeare as corpus

N=884,647 tokens, V=29,066

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)

Quadrigrams worse: What's coming out looks like Shakespeare because it *is* Shakespeare

The Wall Street Journal is not  
Shakespeare (no offense)



Can you guess the training set author of the LM that generated these random 3-gram sentences?

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 gram Brazil on market conditions

This shall forbid it should be branded, if renown made it empty.

“You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

# The perils of overfitting

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

# Zeros

Training set:

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

$$P(\text{"offer"} \mid \text{denied the}) = 0$$

- 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!

And hence we cannot compute perplexity (can't divide by 0)!

# Language Modeling

Generalization and zeros



# Language Modeling

Smoothing: Add-one  
(Laplace) smoothing

# The intuition of smoothing (from Dan Klein)

When we have sparse statistics:

$P(w \mid \text{denied the})$

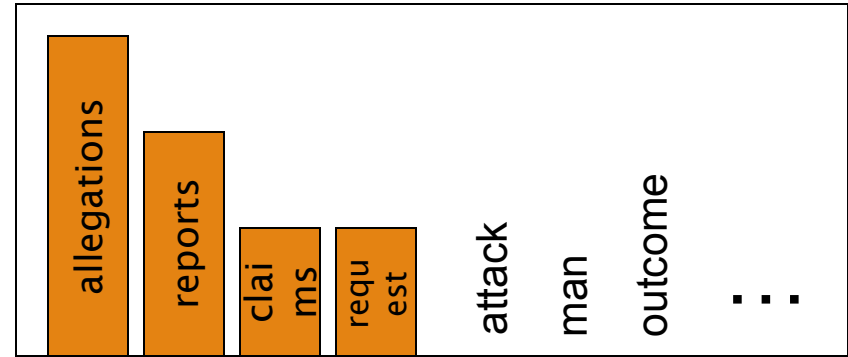
3 allegations

2 reports

1 claims

1 request

7 total



Steal probability mass to generalize better

$P(w \mid \text{denied the})$

2.5 allegations

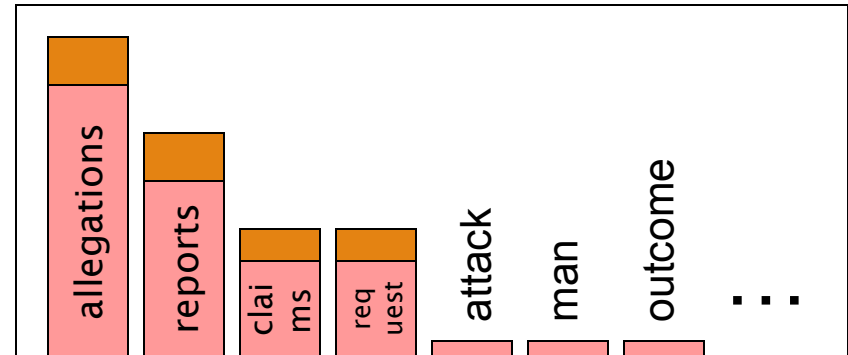
1.5 reports

0.5 claims

0.5 request

2 other

7 total



# 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:



Add-1 estimate:



# 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.

# 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

# Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

	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

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

	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

# 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

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



# Add-1 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.

# Language Modeling

Smoothing: Add-one  
(Laplace) smoothing

# Language Modeling

Interpolation, Backoff, and  
Web-Scale LMs

# Backoff and Interpolation

Sometimes it helps to use **less** context

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

## **Backoff:**

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

## **Interpolation:**

- mix unigram, bigram, trigram

Interpolation works better

# Linear Interpolation

Simple interpolation



Lambdas conditional on context:

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2(w_{n-2}^{n-1}) P(w_n | w_{n-1}) \\ &\quad + \lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

# How to set the lambdas?

Use a **held-out** corpus

Training Data

Held-  
Out  
Data

Test  
Data

Choose  $\lambda$ s to maximize the probability of held-out data:

- Fix the N-gram probabilities (on the training data)
- Then search for  $\lambda$ s that give largest probability to held-out set:



# Unknown words: Open versus closed vocabulary tasks

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

# 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



# N-gram Smoothing Summary

Add-1 smoothing:

- OK for text categorization, not for language modeling

The most commonly used method:

- Extended Interpolated Kneser-Ney

For very large N-grams like the Web:

- Stupid backoff

# Advanced Language Modeling

## Discriminative models:

- choose n-gram weights to improve a task, not to fit the training set

## Parsing-based models

## Caching Models

- Recently used words are more likely to appear



- These turned out to perform very poorly for speech recognition (why?)

# Language Modeling

Interpolation, Backoff, and  
Web-Scale LMs

# Language Modeling

Advanced:  
Kneser-Ney Smoothing

# Absolute discounting: just subtract a little from each count

Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros

How much to subtract ?

Church and Gale (1991)'s clever idea

Divide up 22 million words of AP Newswire

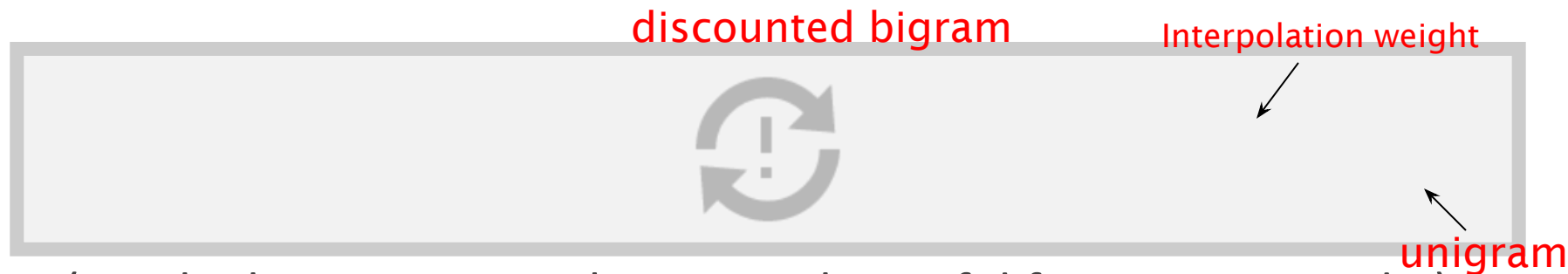
- Training and held-out set
- for each bigram in the training set
- see the actual count in the held-out set!

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

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

# Absolute Discounting Interpolation

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



- (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 Kongses?*
- “Kong” turns out to be more common than “glasses”
- ... but “Kong” always follows “Hong”

The unigram is useful exactly when we haven't seen this bigram!

Instead of  $P(w)$ : “How likely is  $w$ ”

$P_{\text{continuation}}(w)$ : “How likely is  $w$  to appear as a novel continuation?”

- For each word, count the number of bigram types it completes
- Every bigram type was a novel continuation the first time it was seen





# Kneser-Ney Smoothing II

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



Normalized by the total number of word bigram types



# Kneser-Ney Smoothing III

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



normalized by the # of words preceding all words:



A frequent word (Kong) occurring in only one context (Hong) will have a low continuation probability

# Kneser-Ney Smoothing IV



$\lambda$  is a normalizing constant; the probability mass we've discounted



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



Continuation count = Number of unique single word contexts for  
.

# Language Modeling

Advanced:  
Kneser-Ney Smoothing