CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang

Outline

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
- Interpolation, backoff, and web-scale LMs
- Smoothing: Kneser-Ney Smoothing

[Modified from Dan Jurafsky's slides]

Probabilistic Language Models

- Assign a probability to a sentence
  - Machine Translation:
    - P(**high** winds tonight) > P(**large** winds tonight)
  - Spell Correction
    - The office is about fifteen **minuets** from my house
      - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
  - Speech Recognition
    - P(I saw a van) >> P(eyes awe of an)
  - Text Generation in general:
    - Summarization, question-answering …

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:
  $P(W) = P(w_1,w_2,w_3,w_4,w_5…w_n)$
- 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 $P(w_n|w_1,w_2…w_{n-1})$ 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(its, water, is, so, transparent, that)
- Intuition: let's rely on the Chain Rule of Probability

Quick Review: Probability

- Recall the definition of conditional probabilities
  $p(B|A) = P(A,B)/P(A)$ Rewriting: **$P(A,B) = P(A)P(B|A)$**
- More variables:
  $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$
- The Chain Rule in General
  $P(x_1,x_2,x_3,…,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)…P(x_n|x_1,…,x_{n-1})$

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

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

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

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

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

How to estimate these probabilities

- Could we just count and divide?

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

How to estimate these probabilities

- Could we just count and divide?

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

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Markov Assumption

- Simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption

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

- In other words, we approximate each component in the product

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

Simplest case: Unigram model

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

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:

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

```
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

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(s) which I had just put into the machine room on the fifth floor is (are) crashing."
- But we can often get away with N-gram models

Today's Outline

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
- Interpolation, backoff, and web-scale LMs
- Smoothing: Kneser-Ney Smoothing

Estimating bigram probabilities

- The Maximum Likelihood Estimate for bigram probability

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

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

An example

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

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

An example

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

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

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

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:

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

Bigram estimates of sentence probabilities

P(<s> I want english food </s>) =
P(I|<s>)
× P(want|I)
× P(english|want)
× P(food|english)
× P(</s>|food)
= .000031

Knowledge

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

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

Language Modeling Toolkits

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

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 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

- 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

Today's Outline

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
- Interpolation, backoff, and web-scale LMs
- Smoothing: Kneser-Ney Smoothing

Evaluation: How good is our model?

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.

Training on the test set

- We can’t allow test sentences into the training set
- We will assign it an artificially high probability when we set it in the test set
- “Training on the test set”
- Bad science!
- And violates the honor code

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 evaluation of N-gram models

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - Sometimes use **intrinsic** evaluation: **perplexity**

Difficulty of extrinsic 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 33rd President of the US was ____

    I saw a ____
  - Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?

    I always order pizza with cheese and ____

    The 33rd President of the US was ____

    I saw a ____
  - Unigrams are terrible at this game. (Why?)
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

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

Perplexity

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

- Gives the highest P(sentence)

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

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

Perplexity

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

- Gives the highest P(sentence)

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

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

Chain rule:

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

For bigrams:

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

Perplexity

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

- Gives the highest P(sentence)

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

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

Chain rule:

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

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

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?

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?

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

Lower perplexity = better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Today's Outline

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
- Interpolation, backoff, and web-scale LMs
- Smoothing: Kneser-Ney Smoothing

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!

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

In training set, we see
... denied the allegations
... denied the reports
... denied the claims
... denied the request

But in test set,
... denied the offer
... denied the loan

P("offer" | denied the) = 0

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)!

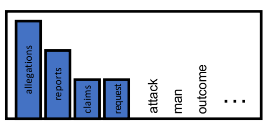
Today's Outline

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
- Interpolation, backoff, and web-scale LMs
- Smoothing: Kneser-Ney Smoothing

The intuition of smoothing (from Dan Klein)

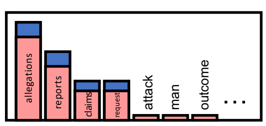
- When we have sparse statistics:

  P(w | denied the)
  3 allegations
  2 reports
  1 claims
  1 request
  7 total



- Steal probability mass to generalize better

  P(w | 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! (*Instead of taking away counts*)
- MLE estimate: $$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$
- Add-1 estimate: $$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

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

Why add V?

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

Add-1 estimation is a blunt instrument

- So add-1 isn't used for N-grams:
  - We'll see better methods
  - (nowadays, neural LM becomes popular, will discuss later)
- But add-1 is used to smooth other NLP models
  - For text classification (coming soon!)
  - In domains where the number of zeros isn't so huge.

Today's Outline

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
- Interpolation, backoff, and web-scale LMs
- Smoothing: Kneser-Ney Smoothing

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
- In general, interpolation works better

Linear Interpolation

- Simple interpolation

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

How to set the lambdas?

- Use a **held-out** corpus

Training Data | Held-Out Data | Test Data

- 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:

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

A Common Method – Grid Search

- Take a list of possible values, e.g. [0.1, 0.2, … ,0.9]
- Try all combinations

Linear Interpolation

- Simple interpolation

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

- Lambdas conditional on context:

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

Linear Interpolation

- Simple interpolation

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

- Lambdas conditional on context:

How to estimate?

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

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

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 (e.g. selecting high frequency words)
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At test time
    - If text input: Use UNK probabilities for any word not in training

Smoothing for Web-scale N-grams

- "Stupid backoff" (Brants *et al*. 2007)
- No discounting, just use relative frequencies

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

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

Until unigram probability

Today's Outline

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
- Interpolation, backoff, and web-scale LMs
- Smoothing: 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)!

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 I

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading*___________? *glasses* / *Francisco*
  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"
- 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 unique bigrams it completes
  - Every unique bigram was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

← Unique bigrams w is in

Kneser-Ney Smoothing II

- How many times does w appear as a novel continuation (unique bigrams):

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

- Normalized by the total number of word bigram types

$$\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|$$

← All unique bigrams in the corpus

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

Kneser-Ney Smoothing III

- Alternative metaphor: The number of # of unique words 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 •

Language Modeling

- Probabilistic language model and n-grams
- Estimating n-gram probabilities
- Language model evaluation and perplexity
- Generalization and zeros
- Smoothing: add-one
- Interpolation, backoff, and web-scale LMs
- Smoothing: Kneser-Ney Smoothing

Homework

- Reading J&M ch1 and ch4.1-4.9
- Start thinking about course project and find a team
- Project proposal due **Oct 1st**
- The format of the proposal will be posted on Piazza