Language Modeling with N-Grams
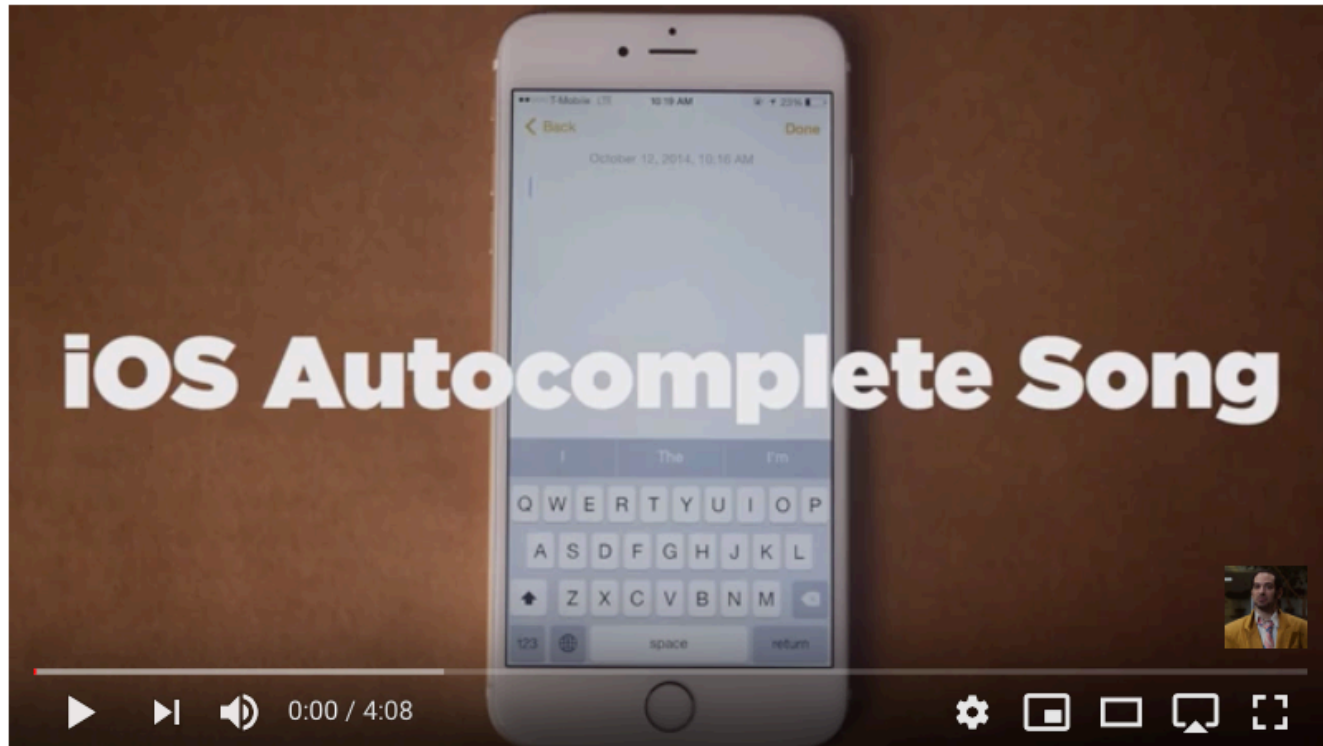
Speech and Language Processing (3rd Edition draft)

Chapter 3 "Language Modeling with N-Grams"

♫ iOS Autocomplete Song | Song A Day #2110

https://www.youtube.com/watch?v=M8MJFrdfGe0

Probabilistic Language Models

- Today's goal: assign a probability to a sentence

- Autocomplete for texting

- Machine Translation

- Spelling Correction

- Speech Recognition

- Other NLG tasks: summarization, question-answering, dialog systems

Probabilistic Language Modeling

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

- Related task: probability of an upcoming word

- A model that computes either of these

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

Probabilistic Language Modeling

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

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

- 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 $P(w_n | w_1, w_2 \ldots w_{n-1})$ 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(the, underdog, Philadelphia, Eagles, won)

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

The Chain Rule

The Chain Rule

- 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

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

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

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("the underdog Philadelphia Eagles won") =
  P(the) × P(underdog|the) × P(Philadelphia|the underdog)
    × P(Eagles|the underdog Philadelphia)
    × P(won|the underdog Philadelphia Eagles)

How to estimate these probabilities

- Could we just count and divide?

How to estimate these probabilities

- Could we just count and divide? Maximum likelihood estimation (MLE)

$$P(\text{won}|\text{the underdog team}) = \frac{\text{Count(the underdog team won)}}{\text{Count(the underdog team)}}$$

- Why doesn't this work?

Simplifying Assumption = Markov Assumption

Simplifying Assumption = Markov Assumption

- P(won|the underdog team) ≈ P(won|team)
- Only depends on the previous k words, not the whole context
- ≈ P(won|underdog team)
- ≈ $P(w_i | w_{i-2} w_{i-1})$
- $P(w_1 w_2 w_3 w_4 ... w_n) \approx \prod_i^n P(w_i | w_{i-k} ... w_{i-1})$
- K is the number of context words that we take into account

How much history should we use?

unigram	**no history**	$$\prod_i^n p(w_i)$$	$$p(w_i) = \frac{count(w_i)}{all\ words}$$
bigram	1 word as history	$$\prod_i^n p(w_i|w_{i-1})$$	$$p(w_i|w_{i-1}) = \frac{count(w_{i-1}w_i)}{count(w_{i-1})}$$
trigram	2 words as history	$$\prod_i^n p(w_i|w_{i-2}w_{i-1})$$	$$p(w_i|w_{i-2}w_{i-1}) = \frac{count(w_{i-2}w_{i-1}w_i)}{count(w_{i-2}w_{i-1})}$$
4-gram	3 words as history	$$\prod_i^n p(w_i|w_{i-3}w_{i-2}w_{i-1})$$	$$p(w_i|w_{i-3}w_{i-2}w_{i-1}) = \frac{count(w_{i-3}w_{i-2}w_{i-1}w_i)}{count(w_{i-3}w_{i-3}w_{i-1})}$$

Historical Notes

1913	Andrei Markov counts 20k letters in *Eugene Onegin*
1948	Claude Shannon uses n-grams to approximate English
1950s/60s	Noam Chomsky decries finite-state Markov Models
1980s	Fred Jelinek at IBM TJ Watson uses n-grams for ASR, think about 2 other ideas for models: (1) MT, (2) stock market prediction
1993	Jelinek at team develops statistical machine translation $$argmax_e p(e|f) = p(e)\, p(f|e)$$
	Jelinek left IBM to found CLSP at JHU Peter Brown and Robert Mercer move to Renaissance Technology

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
- 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

Language Modeling

Estimating N-gram Probabilities

Estimating bigram probabilities

- The Maximum Likelihood Estimate

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

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

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>

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

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

Problems for MLE

- Zeros

Train	Test
denied the allegations	denied the memo
denied the reports	
denied the claims	
denied the requests	

- P(memo|denied the) = 0
- And we also assign 0 probability to all sentences containing it!

Problems for MLE

- Out of vocabulary items (OOV)
- <unk> to deal with OOVs
- Fixed lexicon L of size V
- Normalize training data by replacing any word not in L with <unk>

- Avoid zeros with smoothing

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

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

Google Book N-grams

- http://ngrams.googlelabs.com/

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.

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 (task-based) 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 _____

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

Minimizing perplexity is the same as maximizing probability

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:

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

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?

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

Lower perplexity = better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Language Modeling

Generalization and zeros

The Shannon Visualization Method

- Choose a random bigram
  (<s>, w) according to its probability
- Now choose a random bigram   (w, x) according to its probability
- And so on until we choose </s>
- 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

Approximating Shakespeare

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 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)
- 4-grams worse: What's coming out looks like Shakespeare because it *is* Shakespeare

The wall street journal is not shakespeare (no offense)

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

Can you guess the author of 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

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 | denied the)
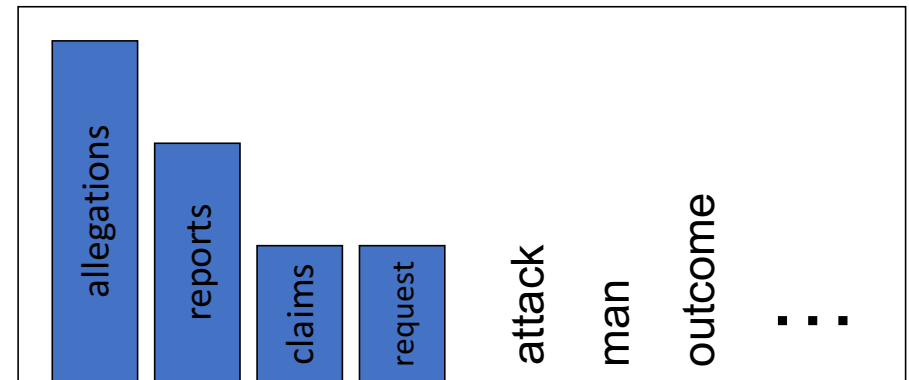 - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request

  7 total

- Steal probability mass to generalize better

  P(w | denied the)
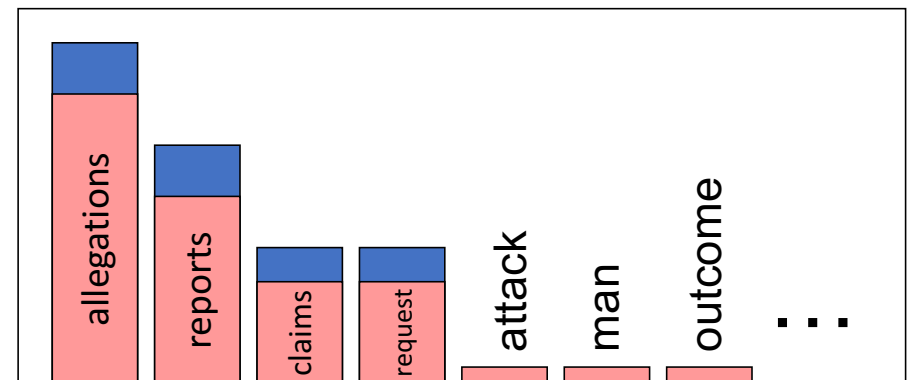 - 2.5 allegations
  - 1.5 reports
  - 0.5 claims
  - 0.5 request
  - 2 other

  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!

- MLE estimate:

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

- Add-1 estimate:

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

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.

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

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

How to set the lambdas?

- Use a **held-out** corpus

Training Data	Held-Out Data	Test Data

- Choose 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:

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

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

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

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

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

$$P_{CACHE}(w \mid history) = \lambda P(w_i \mid w_{i-2}w_{i-1}) + (1 - \lambda)\frac{c(w \in history)}{\mid history \mid}$$

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)!

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

- (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_____?* ~~Francisco~~ ~~glasses~~
  - "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 bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

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

Kneser-Ney Smoothing II

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

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

- Normalized by the total number of word bigram types

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

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

Kneser-Ney Smoothing III

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

$$| \{ w_{i-1} : c(w_{i-1}, w) > 0 \} |$$

- normalized by the # of words preceding all words:

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

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability

Kneser-Ney Smoothing 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

73

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

Advanced: Kneser-Ney Smoothing