Logistics

- **Office hours**
  - Prof. Lu Wang, Mondays 1:30pm - 2:30pm, or by appointment, Rm 911, 177 Huntington Ave
  - To attend OH at 177 Huntington Ave., you need to put down your name on Piazza beforehand (by 2pm each Monday), and then bring a photo ID (e.g. husky card) with you and check in at the front desk.
  - Nikhil Badugu (email: badugu.n@husky.neu.edu), Tuesdays 4-5pm, Ryder Hall 147
  - Eva Sharma (email: sharma.ev@husky.neu.edu), Thursdays 2-3pm, Ryder Hall 220
- **Sign up on piazza!**
  - piazza.com/northeastern/fall2019/cs4120cs6120
- **Course website**

Project proposal (due Sep 30)

- In general, we want to see that you have a clear goal in the project. The technical details can be described in a rough manner, but in principle, you need to show what problem you want to study, and what is novel of your project.
- **Introduction**: the problem has to be well-defined. What are the input and output. Why this is an important problem to study.
- **Related work**: put your work in context. Describe what has been done in previous work on the same or related subject. And why what you propose to do here is novel and different.
- **Datasets**: what data do you want to use? What is the size of it? What information is contained? Why is it suitable for your task?
- **Methodology**: what models do you want to use? You may change the model as the project goes, but you may want to indicate some type of models that might be suitable for your problem. Is it a supervised learning problem or unsupervised? What classifiers can you start with? Are you making improvements? You don't have to be crystal clear on this section, but it can be used to indicate the direction that your project goes to.
- **Evaluation**: what metrics do you want to use for evaluating your models?

*Length: 1 page (or more if necessary). Single space if MS word is used. Or you can choose latex templates, e.g. [https://www.acm.org/publications/proceedings-template](https://www.acm.org/publications/proceedings-template).*

Grading: based on each section described above, roughly 20 points per section. But as you can tell, they're related to each other.

Each group just need to submit one copy on blackboard with group member names indicated.

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

Probabilistic Language Models

- Assign a probability to a sentence
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 minutes from my house
  • P(about fifteen minutes from) > P(about fifteen minutes 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:
  \[ P(W) = P(w_1, w_2, w_3, w_4, w_5, \ldots) \]
• 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:
  \[ P(W) \text{ or } P(w_n | w_1, w_2, \ldots w_{n-1}) \]
  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:
  \[ P(\text{its, water, is, so, transparent, that}) \]

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

Quick Review: Probability

• Recall the definition of conditional probabilities
  \[ p(B|A) = \frac{P(A,B)}{P(A)} \]
  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

\[ P(w_1, w_2, \ldots w_n) = \prod_i P(w_i | w_1, w_2, \ldots w_{i-1}) \]
The Chain Rule applied to compute joint probability of words in sentence

\[ P(w_1 w_2 \ldots w_n) = \prod_i P(w_i | w_1 w_2 \ldots w_{i-1}) \]

\[ P("its water is so transparent") = P(its) \times P(water|its) \times P(is|its water) \times P(so|its water is) \times P(transparent|its water is so) \]

How to estimate these probabilities

• Could we just count and divide?

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

• No! Too many possible sentences!
• We’ll never see enough data for estimating these

Markov Assumption

• Simplifying assumption:

\[ P(\text{the its water is so transparent that}) \approx P(\text{the that}) \]

• Or maybe

\[ P(\text{the its water is so transparent that}) \approx P(\text{the transparent that}) \]

Markov Assumption

\[ P(w_1 w_2 \ldots w_n) = \prod_i P(w_i | w_{i-k} \ldots w_{i-1}) \]

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

\[ P(w_i | w_1 w_2 \ldots w_{i-1}) \approx P(w_i | w_{i-k} \ldots w_{i-1}) \]

Simplest case: Unigram model

\[ P(w_1 w_2 \ldots w_n) = \prod_i P(w_i) \]

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:

\[ P(w_i | w_1 w_2 \ldots w_{i-1}) \approx P(w_i | w_{i-1}) \]

- 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

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

\[ P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})} \]

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

An example

<i> I am Sam </i>
<i> Sam I am </i>
<i> I do not like green eggs and ham </i>
An example

\[
P(w_1 | w_{-1}) = \frac{c(w_1, w_2)}{c(w_1)} \quad \text{<s> I am Sam </s>}
\]
\[
\text{<s> Sam I am </s>}
\]
\[
\text{<s> I do not like green eggs and ham </s>}
\]

\[
P(I | <s>) = \frac{3}{5} = .67 \quad P(San | <s>) = \frac{3}{5} = .33 \quad P(San | I) = \frac{3}{5} = .67
\]

\[
P(\text{Sam} | <s>) = \frac{3}{5} = .6 \quad P(\text{Sam} | \text{am}) = \frac{3}{5} = .5 \quad P(\text{do} | I) = \frac{3}{5} = .33
\]

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 venizia open during the day

Raw bigram counts

- Out of 9222 sentences

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>5</td>
<td>827</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>want</td>
<td>2</td>
<td>608</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>2</td>
<td>4</td>
<td>686</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>42</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>chinese</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>food</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>lunch</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>spend</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Raw bigram probabilities

- Normalize by unigrams:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>253</td>
<td>927</td>
<td>247</td>
<td>746</td>
<td>158</td>
<td>1093</td>
<td>341</td>
<td>276</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0.026</td>
<td>0.33</td>
<td>0.66</td>
<td>0.011</td>
<td>0.0065</td>
<td>0</td>
<td>0.007</td>
<td>0.0079</td>
</tr>
<tr>
<td>want</td>
<td>0.0022</td>
<td>0</td>
<td>0.0017</td>
<td>0.28</td>
<td>0.00038</td>
<td>0</td>
<td>0.002</td>
<td>0.0077</td>
</tr>
<tr>
<td>to</td>
<td>0</td>
<td>0</td>
<td>0.0027</td>
<td>0.021</td>
<td>0.0027</td>
<td>0.52</td>
<td>0.006</td>
<td>0</td>
</tr>
<tr>
<td>eat</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>chinese</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>food</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lunch</td>
<td>0.0059</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spend</td>
<td>0.0036</td>
<td>0</td>
<td>0.0036</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bigram estimates of sentence probabilities

\[
P(\text{<s> I want english food </s>}) = P(I | <s>) \times P(\text{want} | I) \times P(\text{english} | \text{want}) \times P(\text{food} | \text{english}) \times P(\text{food} | <s>) = .000031
\]

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

Google N-Gram Release, August 2006

All Our N-gram are Belong to You

Posted by Alex Frantz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of NLP projects. ...

That’s why we decided to share this enormous dataset with everyone. We processed 1,024,000,267,229 words of running text and are publishing the counts for all 1,176,470,883 five-word sequences that appear at least 40 times. There are 12,585,591 unique words, after discarding words that appear less than 200 times.

[http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html](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?

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

I always order pizza with cheese and ____
The 33rd President of the US was ____
I saw a ____
mushrooms 0.1
pepperoni 0.1
anchovies 0.01
...
...
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:

\[ P(W) = \sum_{i=1}^{N} \log P(w_i) \]

Chain rule:
For bigrams:

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

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 assigns $P = 1/10$ to each digit?

$$\text{PP(W)} = \frac{1}{P(W)} = \frac{1}{\frac{1}{10}} = 10$$

Lower perplexity = better model

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

<table>
<thead>
<tr>
<th>N-gram Order</th>
<th>Unigram</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perplexity</td>
<td>962</td>
<td>170</td>
<td>109</td>
</tr>
</tbody>
</table>

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!

- 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

$$P(“offer” | \text{denied the}) = 0$$
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:
  - Steal probability mass to generalize better
  - P(w | denied the)
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total
- Steal probability mass to generalize better

- 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:
    \[ P_{\text{MLE}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \]
  - Add-1 estimate:
    \[ P_{\text{Add-1}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V} \]
    \( V \) is the size of vocabulary

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>want</th>
<th>to</th>
<th>eat</th>
<th>chinese</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6</td>
<td>828</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>want</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>eat</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>chinese</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>83</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>food</td>
<td>16</td>
<td>1</td>
<td>16</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>lunch</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>spend</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Simple interpolation makes the counts of the trigrams based on this bigram will be more trustworthy, so we can make the counts of the trigrams higher and thus give that trigram more weight in computing the trigram probability.

In a slightly more sophisticated version of linear interpolation, each N-gram is weighted by mixing together the unigram, bigram, and trigram probabilities, each weighted by a different lambda. Thus, we estimate the trigram probability by:

\[ P(w_n|w_{n-1}) = \lambda_1 P(w_n|w_{n-1}) + \lambda_2 P(w_n|w_{n-2}) + \lambda_3 P(w_n) \]

where \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \).

Add-1 estimation is a blunt instrument

- So add-1 isn’t used for N-grams:
  - [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.
- Add-1 can be extended to add-k (k can be any positive real number, sometimes also called add-alpha)

### Laplace-smoothed bigrams

The discounting we have been discussing so far can help solve the problem of zero language modeling, generating counts with poor variances and often inappropriate.

One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events. Instead of adding 1 to each count, we add a fraction:

\[ P'(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \]

### Table: Laplace-smoothed bigrams

<table>
<thead>
<tr>
<th></th>
<th>eat</th>
<th>sat</th>
<th>see</th>
<th>she</th>
<th>food</th>
<th>lunch</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>eat</td>
<td>0.0013</td>
<td>0.23</td>
<td>0.00025</td>
<td>0.0025</td>
<td>0.00025</td>
<td>0.00025</td>
<td>0.00025</td>
</tr>
<tr>
<td>sat</td>
<td>0.0003</td>
<td>0.00042</td>
<td>0.36</td>
<td>0.000841</td>
<td>0.00029</td>
<td>0.00029</td>
<td>0.00029</td>
</tr>
<tr>
<td>see</td>
<td>0.00078</td>
<td>0.000015</td>
<td>0.0013</td>
<td>0.0018</td>
<td>0.00078</td>
<td>0.00018</td>
<td>0.00055</td>
</tr>
<tr>
<td>she</td>
<td>0.00046</td>
<td>0.00046</td>
<td>0.0014</td>
<td>0.00046</td>
<td>0.00078</td>
<td>0.0014</td>
<td>0.02</td>
</tr>
<tr>
<td>food</td>
<td>0.00012</td>
<td>0.00012</td>
<td>0.00062</td>
<td>0.00062</td>
<td>0.0002</td>
<td>0.00012</td>
<td>0.00062</td>
</tr>
<tr>
<td>lunch</td>
<td>0.00017</td>
<td>0.00017</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
<td>0.00056</td>
</tr>
<tr>
<td>spend</td>
<td>0.0012</td>
<td>0.0012</td>
<td>0.00054</td>
<td>0.00054</td>
<td>0.00054</td>
<td>0.00054</td>
<td>0.00054</td>
</tr>
</tbody>
</table>

### 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

\[ P(w_n|w_{n-1}) = \lambda_1 P(w_n|w_{n-1}) + \lambda_2 P(w_n|w_{n-2}) + \lambda_3 P(w_n) \]

where \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \).

### How to set the lambdas?

- Use a held-out corpus
- Choose \( \lambda \) to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for \( \lambda \) that give largest probability to held-out set:
    \[ \log P(w_1, w_2, \ldots, w_n | M(\lambda_1, \ldots, \lambda_n)) = \sum \log P(w_1, w_2 | M(\lambda_1, \ldots, \lambda_n)) \]
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
  \[ P(w_n|w_{n-2}w_{n-1}) = \lambda_0 P(w_n|w_{n-2}w_{n-1}) + \lambda_1 P(w_n|w_{n-1}) + \lambda_2 P(w_n) \sum \lambda_i = 1 \]
- Lambda conditional on context:
  \[ P(w_n|w_{n-2}w_{n-1}) = \lambda_0 (w_1|w_0)^{n-2} P(w_n|w_{n-2}w_{n-1}) + \lambda_1 (w_2|w_1)^{n-1} P(w_n|w_{n-1}) + \lambda_2 (w_3|w_2)^{n} P(w_n) \]

Linear Interpolation

- Simple interpolation
  \[ P(w_n|w_{n-2}w_{n-1}) = \lambda_0 P(w_n|w_{n-2}w_{n-1}) + \lambda_1 P(w_n|w_{n-1}) + \lambda_2 P(w_n) \]
- Lambda conditional on context:
  \[ P(w_n|w_{n-2}w_{n-1}) = \lambda_0 (w_1|w_0)^{n-2} P(w_n|w_{n-2}w_{n-1}) + \lambda_1 (w_2|w_1)^{n-1} P(w_n|w_{n-1}) + \lambda_2 (w_3|w_2)^{n} P(w_n) \]

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
- 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 test normalization phase, any training word not in L changes to `<UNK>`
  * Use word probabilities like a normal word
  * At test time
  * If test input: Use OOV probabilities for any word not in training

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 test normalization phase, any training word not in L changes to `<UNK>`
  * Use word probabilities like a normal word
  * At test time
  * If test input: Use OOV 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

\[
S(w_j | w_{i-1} \ldots w_{i-k}) = \begin{cases} 
\frac{\text{count}(w_j^+)}{\text{count}(w_{i-1}^+)} & \text{if count}(w_{i-1}^+) \neq 0 \\
0.4 \times S(w_j | w_{i-1}^+) & \text{otherwise}
\end{cases}
\]

\[
S(w_0) = \frac{\text{count}(w_0)}{N}
\]

Unit 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 Interpolation

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

\[
P_{\text{Absolute Discounting}}(w_i | w_{j-1}) = \frac{c(w_{j-1}, w_i) - d}{c(w_{j-1})} + \lambda \frac{P(w_i)}{\text{unigram}}
\]

• 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_______?
  - “Francisco” is more common than “glasses”
  - “Francisco” always follows “San”

Kneser-Ney Smoothing II

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

\[
P_{\text{CONTINUATION}}(w) = \left| \left\{ w_{j-1}, w : c(w_{j-1}, w) > 0 \right\} \right|
\]

• Normalized by the total number of word bigram types

\[
P_{\text{CONTINUATION}}(w) = \frac{\left| \left\{ w_{j-1}, w : c(w_{j-1}, w) > 0 \right\} \right|}{\left( \left| \left\{ w_{j-1}, w : c(w_{j-1}, w) > 0 \right\} \right| \}}
\]

Kneser-Ney Smoothing III

• Alternative metaphor: The number of # of unique words seen to precede \( w \)

\[
P_{\text{CONTINUATION}}(w) = \frac{\left| \left\{ w_{j-1}, w : c(w_{j-1}, w) > 0 \right\} \right| \text{ normalized by the # of words preceding all words:}}{\sum_{w'} \left| \left\{ w_{j-1}, w' : c(w_{j-1}, w') > 0 \right\} \right|}
\]

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

<table>
<thead>
<tr>
<th>Bigram count in training</th>
<th>Bigram count in heldout set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.648</td>
</tr>
<tr>
<td>1</td>
<td>1.25</td>
</tr>
<tr>
<td>2</td>
<td>2.28</td>
</tr>
<tr>
<td>3</td>
<td>3.23</td>
</tr>
<tr>
<td>4</td>
<td>4.24</td>
</tr>
<tr>
<td>5</td>
<td>5.23</td>
</tr>
<tr>
<td>6</td>
<td>6.21</td>
</tr>
<tr>
<td>7</td>
<td>7.21</td>
</tr>
<tr>
<td>8</td>
<td>8.26</td>
</tr>
</tbody>
</table>

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

Church and Gale (1991)’s clever idea

Divide up 22 million words of AP Newswire

• Training and held-out set

How much to subtract?

• For each bigram in the training set

- see the actual count in the held-out set!
Kneser-Ney Smoothing IV

\[ P_{KN}(w_i | w_{i-1}) = \max \left( \frac{c(w_{i-1}, w_i) - d(w_{i-1})}{c(w_{i-1})}, 0 \right) + \lambda(w_{i-1}) P_{CONTINUOUS}(w_i) \]

\( \lambda \) is a normalizing constant; the probability mass we’ve discounted

\[ \lambda(w_{i-1}) = \frac{d(w_{i-1})}{c(w_{i-1})} \left[ w : c(w_{i-1}, w) > 0 \right] \]

Homework

• Reading J&M ch1 and ch4.1-4.9
• Start thinking about course project and find a team
• Project proposal due Sep 30th
• The format of the proposal will be posted on Piazza