Speech and Language Processing

N-grams

Chapter 4 of SLP

Today

- Word prediction task
- Language modeling (N-grams)
  - N-gram intro
  - The chain rule
  - Model evaluation
  - Smoothing

Word Prediction

- Guess the next word...
  - *... I notice three guys standing on the ???*
- There are many sources of knowledge that can be used to inform this task, including arbitrary world knowledge.
- But it turns out that you can do pretty well by simply looking at the preceding words and keeping track of some fairly simple counts.

Word Prediction

- We can formalize this task using what are called *N*-gram models.
- *N*-grams are token sequences of length *N*.
- Our earlier example contains the following 2-grams (aka bigrams)
  - (I notice), (notice three), (three guys), (guys standing), (standing on), (on the)
- Given knowledge of counts of N-grams such as these, we can guess likely next words in a sequence.

N-Gram Models

- More formally, we can use knowledge of the counts of *N*-grams to assess the conditional probability of candidate words as the next word in a sequence.
- Or, we can use them to assess the probability of an entire sequence of words.
  - Pretty much the same thing as we'll see...

Applications

- It turns out that being able to predict the next word (or any linguistic unit) in a sequence is an extremely useful thing to be able to do.
- As we'll see, it lies at the core of the following applications
  - Automatic speech recognition
  - Handwriting and character recognition
  - Spelling correction
  - Machine translation
  - And many more.

Counting

- Simple counting lies at the core of any probabilistic approach. So let's first take a look at what we're counting.
  - *He stepped out into the hall, was delighted to encounter a water brother.*
    - 13 tokens, 15 if we include "," and "." as separate tokens.
    - Assuming we include the comma and period, how many bigrams are there?

Counting: Types and Tokens

- How about
  - *They picnicked by the pool, then lay back on the grass and looked at the stars.*
    - 18 tokens (again counting punctuation)
- But we might also note that "*the*" is used 3 times, so there are only 16 unique types (as opposed to tokens).
- In going forward, we'll have occasion to focus on counting both types and tokens of both words and *N*-grams.

Counting: Wordforms

- Should "cats" and "cat" count as the same when we're counting?
- How about "geese" and "goose"?
- Some terminology:
  - **Lemma**: a set of lexical forms having the same **stem**, major part of speech, and rough word sense
  - Wordform: fully inflected surface form
- Again, we'll have occasion to count both lemmas and wordforms

Counting: Corpora

- So what happens when we look at large bodies of text instead of single utterances?
- Brown et al (1992) large corpus of English text
  - 583 million wordform tokens
  - 293,181 wordform types
- Google
  - Crawl of 1,024,908,267,229 English tokens
  - 13,588,391 wordform types
    - That seems like a lot of types... After all, even large dictionaries of English have only around 500k types. Why so many here?

- Numbers
- Misspellings
- Names
- Acronyms
- etc

Language Modeling

- Back to word prediction
- We can model the word prediction task as the ability to assess the conditional probability of a word given the previous words in the sequence
  - $P(w_n|w_1,w_2...w_{n-1})$
- We'll call a statistical model that can assess this a *Language Model*
  - A probabilistic estimation for the frequency of words and word sequences, usually derived from a corpus.

Language Modeling

- How might we go about calculating such a conditional probability?
  - One way is to use the definition of conditional probabilities and look for counts. So to get
  - P(*the* | *its water is so transparent that*)
- By definition that's

  P(A|B) = P(A ∩ B) / P(B)

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

  We can get each of those from counts in a large corpus.

Very Easy Estimate

- How to estimate?
  - P(the | its water is so transparent that)

P(the | its water is so transparent that) =

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

Very Easy Estimate

- According to Google those counts are 5/9.
  - Unfortunately... 2 of those were to these slides... So maybe it's really
  - 3/7
  - In any case, that's not terribly convincing due to the small numbers involved.

Language Modeling

- Unfortunately, for most sequences and for most text collections we won't get good estimates from this method.
  - What we're likely to get is 0. Or worse 0/0.
- Clearly, we'll have to be a little more clever.
  - Let's use the chain rule of probability
  - And a particularly useful independence assumption.

The Chain Rule

- Recall the definition of conditional probabilities
- Rewriting:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A \cap B) = P(A \mid B)P(B)$$

- For sequences…
  - P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)
- 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

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

P(its water was so transparent)=

P(its)*

P(water|its)*

P(was|its water)*

P(so|its water was)*

P(transparent|its water was so)

Unfortunately

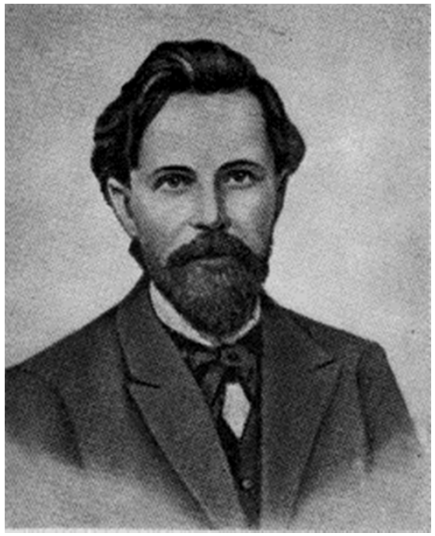
- There are still a lot of possible sentences
- In general, we'll never be able to get enough data to compute the statistics for those longer prefixes
  - Same problem we had for the strings themselves

Independence Assumption

- Make the simplifying assumption
  - P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|a)
- Or maybe
  - P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|saw,a)
- That is, the probability in question is independent of its earlier history.

Independence Assumption

- This particular kind of independence assumption is called a *Markov assumption* after the Russian mathematician Andrei Markov.



Markov Assumption

So for each component in the product replace with the approximation (assuming a prefix of N)

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

Bigram version

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)

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

An Example

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

$$P(\text{I}\,|\,\text{<s>}) = \tfrac{2}{3} = .67 \qquad P(\text{Sam}\,|\,\text{<s>}) = \tfrac{1}{3} = .33 \qquad P(\text{am}\,|\,\text{I}) = \tfrac{2}{3} = .67$$
$$P(\text{</s>}\,|\,\text{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\text{Sam}\,|\,\text{am}) = \tfrac{1}{2} = .5 \qquad P(\text{do}\,|\,\text{I}) = \tfrac{1}{3} = .33$$

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

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*

Bigram Counts

- Out of 9222 sentences
  - Eg. "I want" occurred 827 times

	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

Bigram Probabilities

- Divide bigram counts by prefix unigram counts to get probabilities.

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

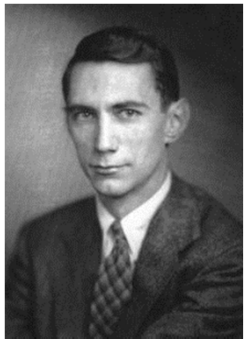
	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram Estimates of Sentence Probabilities

- P(<s> I want english food </s>) =
  P(i|<s>)*
  P(want|I)*
  P(english|want)*
  P(food|english)*
  P(</s>|food)*
  =.000031

Shannon's Method

- Assigning probabilities to sentences is all well and good, but it's not terribly illuminating . A more interesting task is to turn the model around and use it to generate random sentences that are *like* the sentences from which the model was derived.
- Generally attributed to Claude Shannon.



Shannon's Method

- Sample a random bigram (<s>, w) according to its probability
- Now sample a random bigram (w, x) according to its probability
  - Where the prefix w matches the suffix of the first.
- And so on until we randomly choose a (y, </s>)
- Then string the words together
- <s> I
  I want
  want to
  to eat
  eat Chinese
  Chinese food
  food </s>

Shakespeare

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

Shakespeare as a 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)
  - This is the biggest problem in language modeling; we'll come back to it.
- Quadrigrams are worse: What's coming out looks like Shakespeare because it ***is*** Shakespeare

Evaluation

- How do we know if our models are any good?
  - And in particular, how do we know if one model is better than another.
- Well Shannon's game gives us an intuition.
  - The generated texts from the higher order models sure look better. That is, they sound more like the text the model was obtained from.
  - But what does that mean? Can we make that notion operational?

Evaluation

- Standard method
  - Train parameters of our model on a **training set**.
  - Look at the models performance on some new data
    - This is exactly what happens in the real world; we want to know how our model performs on data we haven't seen
  - So use a **test set**. A dataset which is different than our training set, but is drawn from the same source
  - Then we need an **evaluation metric** to tell us how well our model is doing on the test set.
    - One such metric is **perplexity** (to be introduced below)

Unknown Words

- But once we start looking at test data, we'll run into words that we haven't seen before (pretty much regardless of how much training data you have).
- With an *Open Vocabulary* task
  - Create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L, of size V
      - From a dictionary or
      - A subset of terms from the training set
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we count that like a normal word
  - At test time
    - Use UNK counts for any word not in training

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

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

- Chain rule:

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

- For bigrams:

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

- Minimizing perplexity is the same as maximizing probability
  - **The best language model is one that best predicts an unseen test set**

Lower perplexity means a better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Evaluating *N*-Gram Models

- Best evaluation for a language model
  - Put model A into an application
    - For example, a speech recognizer
  - Evaluate the performance of the application with model A
  - Put model B into the application and evaluate
  - Compare performance of the application with the two models
  - ***Extrinsic evaluation***

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

- Extrinsic evaluation
  - This is really time-consuming
  - Can take days to run an experiment
- So
  - As a temporary solution, in order to run experiments
  - To evaluate N-grams we often use an **intrinsic** evaluation, an approximation called **perplexity**
  - But perplexity is a poor approximation unless the test data looks **just** like the training data
  - So is **generally only useful in pilot experiments (generally is not sufficient to publish)**
  - **But is helpful to think about.**

Zero Counts

- Back to Shakespeare
  - Recall that 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)
  - Does that mean that any sentence that contains one of those bigrams should have a probability of 0?

Zero Counts

- Some of those zeros are really zeros...
  - Things that really can't or shouldn't happen.
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count (probably a count of 1!).
- Zipf's Law (long tail phenomenon):
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Result:
  - Our estimates are **sparse**! We have no counts at all for the vast bulk of things we want to estimate!
- Answer:
  - Estimate the likelihood of unseen (zero count) N-grams!

Laplace Smoothing

- Also called add-one smoothing
- Just add one to all the counts!
- Very simple

- MLE estimate: $P(w_i) = \frac{c_i}{N}$
- Laplace estimate: $P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$
- Reconstructed counts: $c_i^* = (c_i + 1)\frac{N}{N + V}$

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 Bigram Probabilities

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

	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

Big Change to the Counts!

- C(count to) went from 608 to 238!
- P(to|want) from .66 to .26!
- Discount d= c*/c
  - d for "chinese food" =.10!!! A 10x reduction
  - So in general, Laplace is a blunt instrument
  - Could use more fine-grained method (add-k)
- But Laplace smoothing not used for N-grams, as we have much better methods
- Despite its flaws Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - in domains where the number of zeros isn't so huge.

Better Smoothing

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell

- Is to use the count of things we've seen once to help estimate the count of things we've never seen

Good-Turing
Josh Goodman Intuition

- Imagine you are fishing
  - There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?
  - 3/18
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18

Slide adapted from Josh Goodman

Good-Turing

- Notation: N_x is the frequency-of-frequency-x
  - So $N_{10}=1$
    - Number of fish species seen 10 times is 1 (carp)
  - $N_1=3$
    - Number of fish species seen 1 is 3 (trout, salmon, eel)
- To estimate total number of unseen species
  - Use number of species (words) we've seen once
  - $c_0^* = c_1$ $p_0 = N_1/N$

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

- All other estimates are adjusted (down) to give probabilities for unseen

Slide from Josh Goodman

Good-Turing Intuition

- Notation: N_x is the frequency-of-frequency-x
  - So $N_{10}=1$, $N_1=3$, etc
- To estimate total number of unseen species
  - Use number of species (words) we've seen once
  - $c_0^* = c_1$ $\quad p_0 = N_1/N = 3/18$

$$P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N}$$

- All other estimates are adjusted (down) to give probabilities for unseen

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

c*(trout) = (1+1) 1/ 3 = 2/3

Slide from Josh Goodman

GT Fish Example

	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT p^*_{GT}	$p^*_{\text{GT}}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{\text{GT}}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

Bigram Frequencies of Frequencies and GT Re-estimates

AP Newswire			Berkeley Restaurant—		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

Google N-Gram Release

All Our N-gram are Belong to You

By Peter Norvig - 8/03/2006 11:26:00 AM

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, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training

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

Google Caveat

- Remember the lesson about test sets and training sets... Test sets should be similar to the training set (drawn from the same distribution) for the probabilities to be meaningful.
- So... The Google corpus is fine if your application deals with arbitrary English text on the Web.
- If not then a smaller domain specific corpus is likely to yield better results.