N-Grams and Corpus Linguistics

Lecture #4

September 6, 2012

1

Transition

- Up to this point we've mostly been discussing words in isolation
- Now we're switching to sequences of words
- And we're going to worry about assigning probabilities to sequences of words

2

Who Cares?

- Why would you want to assign a probability to a sentence or…
- Why would you want to predict the next word…
- Lots of applications

3

Real-Word Spelling Errors

- Mental confusions
  - Their/they're/there
  - To/too/two
  - Weather/whether
  - Peace/piece
  - You're/your
- Typos that result in real words
  - Lave for Have

4

Real Word Spelling Errors

- Collect a set of common pairs of confusions
- Whenever a member of this set is encountered compute the probability of the sentence in which it appears
- Substitute the other possibilities and compute the probability of the resulting sentence
- Choose the higher one

5

Next Word Prediction

- From a NY Times story...
  - Stocks ...
  - Stocks plunged this ….
  - Stocks plunged this morning, despite a cut in interest rates
  - Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall ...
  - Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began

6

– Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last …

– Stocks plunged this morning, despite a cut in interest rates by the Federal Reserve, as Wall Street began trading for the first time since last Tuesday's terrorist attacks.

Human Word Prediction

- Clearly, at least some of us have the ability to predict future words in an utterance.
- How?
  - Domain knowledge
  - Syntactic knowledge
  - Lexical knowledge

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

- Why do we want to predict a word, given some preceding words?
  - Rank the likelihood of sequences containing various alternative hypotheses, e.g. for ASR

  Theatre owners say popcorn/unicorn sales have doubled...
  - Assess the likelihood/goodness of a sentence, e.g. for text generation or machine translation

  The doctor recommended a cat scan.

  El doctor recommendó una exploración del gato.

N-Gram Models of Language

- Use the previous N-1 words in a sequence to predict the next word
- Language Model (LM)
  - unigrams, bigrams, trigrams,…
- How do we train these models?
  - Very large corpora

Counting Words in Corpora

- What is a word?
  - e.g., are cat and cats the same word?
  - September and Sept?
  - zero and oh?
  - Is _ a word? * ? '(' ?
  - How many words are there in don't ? Gonna ?
  - In Japanese and Chinese text -- how do we identify a word?

Terminology

- Sentence: unit of written language
- Utterance: unit of spoken language
- Word Form: the inflected form that appears in the corpus
- Lemma: an abstract form, shared by word forms having the same stem, part of speech, and word sense
- Types: number of distinct words in a corpus (vocabulary size)
- Tokens: total number of words

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

Corpora

- Corpora are online collections of text and speech
  - Brown Corpus
  - Wall Street Journal
  - AP news
  - Hansards
  - DARPA/NIST text/speech corpora (Call Home, ATIS, switchboard, Broadcast News, TDT, Communicator)
  - TRAINS, Radio News

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*

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(its water is so transparent that the)

P(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) =

Count(its water is so transparent that the)

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: $$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

$$P(A \wedge B) = P(A|B)P(B)$$

- 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)

Example

- The big red dog
- P(The)*P(big|the)*P(red|the big)*P(dog|the big red)
- Better P(The| <Beginning of sentence>) written as
  P(The | <S>)

25

General Case

- The word sequence from position 1 to n is w_1^n
- So the probability of a sequence is

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

26

Unfortunately

- That doesn’t help since its unlikely we’ll ever gather the right statistics for the prefixes.

27

Markov Assumption

- Assume that the entire prefix history isn’t necessary.
- In other words, an event doesn’t depend on all of its history, just a fixed length near history

28

Markov Assumption

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

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

29

N-Grams
The big red dog

- Unigrams: P(dog)
- Bigrams: P(dog|red)
- Trigrams: P(dog|big red)
- Four-grams: P(dog|the big red)

In general, we’ll be dealing with
P(Word| Some fixed prefix)

30

Caveat

- The formulation P(Word| Some fixed prefix) is not really appropriate in many applications.
- It is if we're dealing with real time speech where we only have access to prefixes.
- But if we're dealing with text we already have the right and left contexts. There's no a priori reason to stick to left contexts.

31

Training and Testing

- N-Gram probabilities come from a training corpus
  - overly narrow corpus: probabilities don't generalize
  - overly general corpus: probabilities don't reflect task or domain
- A separate test corpus is used to evaluate the model, typically using standard metrics
  - held out test set; development test set
  - cross validation
  - results tested for statistical significance

32

A Simple Example

- P(I want to eat Chinese food) = P(I | <start>) P(want | I) P(to | want) P(eat | to) P(Chinese | eat) P(food | Chinese)

33

A Bigram Grammar Fragment from BERP

eat on	.16	eat Thai	.03
eat some	.06	eat breakfast	.03
eat lunch	.06	eat in	.02
eat dinner	.05	eat Chinese	.02
eat at	.04	eat Mexican	.02
eat a	.04	eat tomorrow	.01
eat Indian	.04	eat dessert	.007
eat today	.03	eat British	.001

34

<start> I	.25	want some	.04
<start> I'd	.06	want Thai	.01
<start> Tell	.04	to eat	.26
<start> I'm	.02	to have	.14
I want	.32	to spend	.09
I would	.29	to be	.02
I don't	.08	British food	.60
I have	.04	British restaurant	.15
want to	.65	British cuisine	.01
want a	.05	British lunch	.01

35

- P(I want to eat British food) = P(I|<start>) P(want|I) P(to|want) P(eat|to) P(British|eat) P(food|British) = .25*.32*.65*.26*.001*.60 = .000080
- vs. I want to eat Chinese food = .00015
- Probabilities seem to capture ``syntactic'' facts, ``world knowledge''
  - eat is often followed by an NP
  - British food is not too popular
- N-gram models can be trained by counting and normalization

36

An Aside on Logs

- You don't really do all those multiplies. The numbers are too small and lead to underflows
- Convert the probabilities to logs and then do additions.
- To get the real probability (if you need it) go back to the antilog.

How do we get the N-gram probabilities?

- N-gram models can be trained by counting and normalization

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>

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

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

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

Maximum Likelihood Estimates

- The maximum likelihood estimate of some parameter of a model M from a training set T
  - Is the estimate that maximizes the likelihood of the training set T given the model M
- Suppose the word Chinese occurs 400 times in a corpus of a million words (Brown corpus)
- What is the probability that a random word from some other text from the same distribution will be "Chinese"
- MLE estimate is 400/1000000 = .004
  - This may be a bad estimate for some other corpus
- But it is the **estimate** that makes it **most likely** that "Chinese" will occur 400 times in a million word corpus.

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*

BERP Bigram Counts

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

43

BERP Bigram Probabilities

- Normalization: divide each row's counts by appropriate unigram counts for w_{n-1}

I	want	to	eat	Chinese	food	lunch
3437	1215	3256	938	213	1506	459

- Computing the bigram probability of I I
  - C(I,I)/C(all I)
  - p (I|I) = 8 / 3437 = .0023
- Maximum Likelihood Estimation (MLE): relative frequency of e.g.

$$\frac{freq(w_1, w_2)}{freq(w_1)}$$

44

BERP Table: Bigram Probabilities

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

45

What do we learn about the language?

- What's being captured with ...
  - P(want | I) = .32
  - P(to | want) = .65
  - P(eat | to) = .26
  - P(food | Chinese) = .56
  - P(lunch | eat) = .055
- What about...
  - P(I | I) = .0023
  - P(I | want) = .0025
  - P(I | food) = .013

46

- P(I | I) = .0023 I I I I want
- P(I | want) = .0025 I want I want
- P(I | food) = .013 the kind of food I want is ...

47

Kinds of Knowledge

- As crude as they are, *N*-gram probabilities capture a range of interesting facts about language.

- P(english|want) = .0011
- P(chinese|want) = .0065

World knowledge

- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0

Syntax

- P (i | <s>) = .25

Discourse

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

The Wall Street Journal is Not Shakespeare

unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

bigram: 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

trigram: 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

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)

9/5/2012 Speech and Language Processing - Jurafsky and Martin 55

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

9/5/2012 Speech and Language Processing - Jurafsky and Martin 56

Perplexity

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

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

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

9/5/2012 Speech and Language Processing - Jurafsky and Martin 57

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

9/5/2012 Speech and Language Processing - Jurafsky and Martin 58

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***

9/5/2012 Speech and Language Processing - Jurafsky and Martin 59

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.

9/5/2012 Speech and Language Processing - Jurafsky and Martin 60

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!

Smoothing Techniques

- Every n-gram training matrix is sparse, even for very large corpora (Zipf's law)
- Solution: estimate the likelihood of unseen n-grams
- Problems: how do you adjust the rest of the corpus to accommodate these 'phantom' n-grams?

Problem

- Let's assume we're using N-grams
- How can we assign a probability to a sequence where one of the component n-grams has a value of zero
- Assume all the words are known and have been seen
  - Go to a lower order n-gram
  - Back off from bigrams to unigrams
  - Replace the zero with something else

Add-One (Laplace)

- Make the zero counts 1.
- Rationale: They're just events you haven't seen yet. If you had seen them, chances are you would only have seen them once… so make the count equal to 1.

Add-one Smoothing

- For unigrams:
  - Add 1 to every word (type) count
  - Normalize by N (tokens) /(N (tokens) +V (types))
  - Smoothed count (adjusted for additions to N) is $\left(c_i+1\right)\frac{N}{N+V}$
  - Normalize by N to get the new unigram probability:

$$p_i^* = \frac{c_i+1}{N+V}$$

- For bigrams:
  - Add 1 to every bigram $c(w_{n-1}\, w_n) + 1$
  - Incr unigram count by vocabulary size $c(w_{n-1}) + V$

Original BERP Counts

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

67

BERP Table: Bigram Probabilities

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

68

BERP After Add-One

Was .65

	I	want	to	eat	Chinese	food	lunch
I	.0018	.22	.00020	.0028	.00020	.00020	.00020
want	.0014	.00035	.28	.00035	.0025	.0032	.0025
to	.00082	.00021	.0023	.18	.00082	.00021	.0027
eat	.00039	.00039	.0012	.00039	.0078	.0012	.021
Chinese	.0016	.00055	.00055	.00055	.00055	.066	.0011
food	.0064	.00032	.0058	.00032	.00032	.00032	.00032
lunch	.0024	.00048	.00048	.00048	.00048	.00096	.00048

69

Add-One Smoothed BERP Reconstituted

	I	want	to	eat	Chinese	food	lunch
I	6	740	.68	10	.68	.68	.68
want	2	.42	331	.42	3	4	3
to	3	.69	8	594	3	.69	9
eat	.37	.37	1	.37	7.4	1	20
Chinese	.36	.12	.12	.12	.12	15	.24
food	10	.48	9	.48	.48	.48	.48
lunch	1.1	.22	.22	.22	.22	.44	.22

70

- Discount: ratio of new counts to old (e.g. add-one smoothing changes the BERP count (to|want) from 786 to 331 (d_c=.42) and p(to|want) from .65 to .28)
- Problem: add one smoothing changes counts drastically:
  - too much weight given to unseen ngrams
  - in practice, unsmoothed bigrams often work better!

71

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

9/5/2012 Speech and Language Processing - Jurafsky and Martin 72

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
  - $c_0^* = c_1$ $p_0 = N_1/N$ $p_0=N_1/N=3/18$

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

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

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

P(eel) = c*(1) = (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

Complications

- In practice, assume large counts (c>k for some k) are reliable:

$$c^* = c \text{ for } c > k$$

- That complicates c*, making it:

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \text{ for } 1 \le c \le k.$$

- Also: we assume singleton counts c=1 are unreliable, so treat N-grams with count of 1 as if they were count=0
- Also, need the Nk to be non-zero, so we need to smooth (interpolate) the Nk counts before computing c* from them

Backoff and Interpolation

- Another really useful source of knowledge
- If we are estimating:
  - trigram p(z|x,y)
  - but count(xyz) is zero
- Use info from:
  - Bigram p(z|y)
- Or even:
  - Unigram p(z)
- How to combine this trigram, bigram, unigram info in a valid fashion?

Backoff Vs. Interpolation

- **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram
- **Interpolation**: mix all three

Interpolation

- Simple interpolation

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

- 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, or development,** corpus
- Choose lambdas which maximize the probability of some held-out data
  - I.e. fix the *N*-gram probabilities
  - Then search for lambda values
  - That when plugged into previous equation
  - Give largest probability for held-out set
  - Can use EM to do this search

Katz Backoff

$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.}\end{cases}$$

$$P_{\text{katz}}(z|x,y) = \begin{cases} P^*(z|x,y), & \text{if } C(x,y,z) > 0 \\ \alpha(x,y)P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\ P^*(z), & \text{otherwise.}\end{cases}$$

$$P_{\text{katz}}(z|y) = \begin{cases} P^*(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y)P^*(z), & \text{otherwise.}\end{cases}$$

Why discounts P* and alpha?

- MLE probabilities sum to 1

$$\sum_i P(w_i|w_j w_k) = 1$$

- So if we used MLE probabilities but backed off to lower order model when MLE prob is zero
- We would be adding extra probability mass
- And total probability would be greater than 1

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

GT Smoothed Bigram Probabilities

	i	want	to	eat	chinese	food	lunch	spend
i	0.0014	0.326	0.00248	0.00355	0.000205	0.0017	0.00073	0.000489
want	0.00134	0.00152	0.656	0.000483	0.00455	0.00455	0.00384	0.000483
to	0.000512	0.00152	0.00165	0.284	0.000512	0.0017	0.00175	0.0873
eat	0.00101	0.00152	0.00166	0.00189	0.0214	0.00166	0.0563	0.000585
chinese	0.00283	0.00152	0.00248	0.00189	0.000205	0.519	0.00283	0.000585
food	0.0137	0.00152	0.0137	0.00189	0.000409	0.00366	0.00073	0.000585
lunch	0.00363	0.00152	0.00248	0.00189	0.000205	0.00131	0.00073	0.000585
spend	0.00161	0.00152	0.00161	0.00189	0.000205	0.0017	0.00073	0.000585

Intuition of Backoff+Discounting

- How much probability to assign to all the zero trigrams?
  - Use GT or other discounting algorithm to tell us
- How to divide that probability mass among different contexts?
  - Use the N-1 gram estimates to tell us
- What do we do for the unigram words not seen in training?
  - **Out Of Vocabulary** = OOV words

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- We don't use GT smoothing for these
  - Because GT assumes we know the number of unseen events
- 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

Practical Issues

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

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

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.

Summary

- N-gram probabilities can be used to *estimate* the likelihood
  - Of a word occurring in a context (N-1)
  - Of a sentence occurring at all
- Smoothing techniques deal with problems of unseen words in a corpus