Foundations of Natural Language Processing
Lecture 7b: LMs and Smoothing: Further Challenges

Alex Lascarides

School of informatics

Recap: Ngram LMs and Smoothing

- Different smoothing methods account for different aspects of sparse data and word behaviour.
  - Laplace and Good Turing: Recognise non-zero prob. mass needed on unseen ngrams.
  - Interpolation/backoff: leverage advantages of both higher and lower order N-grams.

Now: More challenges for making smoothing work!

Do our smoothing methods work here?

Example from MacKay and Peto (1995):

> Imagine, you see, that the language, you see, has, you see, a frequently occurring couplet, 'you see', you see, in which the second word of the couplet, see, follows the first word, you, with very high probability, you see. Then the marginal statistics, you see, are going to become hugely dominated, you see, by the words you and see, with equal frequency, you see.

- $P(\text{see})$ and $P(\text{you})$ both high, but see nearly always follows you.
- So $P(\text{see}|novel)$ should be much lower than $P(\text{you}|novel)$.

Diversity of histories matters!

- A real example: the word York
  - fairly frequent word in Europarl corpus, occurs 477 times
  - as frequent as foods, indicates and providers

  → in unigram language model: a respectable probability

- However, it almost always directly follows New (473 times)

- So, in unseen bigram contexts, York should have low probability
  - lower than predicted by unigram model as used in interpolation/backoff.

Kneser-Ney Smoothing

- Kneser-Ney smoothing takes diversity of histories into account
- Count of distinct histories for a word:

$$N_{1+}(\bullet w_i) = |\{w_{i-1} : c(w_{i-1}, w_i) > 0\}|$$

- Recall: maximum likelihood est. of unigram language model:

$$P_{ML}(w_i) = \frac{C(w_i)}{\sum_w C(w)}$$

- In KN smoothing, replace raw counts with count of histories:

$$P_{KN}(w_i) = \frac{N_{1+}(\bullet w_i)}{\sum_w N_{1+}(\bullet w)}$$

Kneser-Ney in practice

- Original version used backoff, later "modified Kneser-Ney" introduced using interpolation.
- Fairly complex equations, but until recently the best smoothing method for word n-grams.
- See Chen and Goodman (1999) for extensive comparisons of KN and other smoothing methods.
- KN (and other methods) implemented in language modelling toolkits like SRILM (classic), KenLM (good for really big models), OpenGrm Ngram library (uses finite state transducers), etc.

Are we done with smoothing yet?

We've considered methods that predict rare/unseen words using

- Uniform probabilities (add-α, Good-Turing)
- Probabilities from lower-order n-grams (interpolation, backoff)
- Probability of appearing in new contexts (Kneser-Ney)

What's left?

Word similarity

- Suppose we have two words with $C(w_1) \gg C(w_2)$
  - salmon
  - swordfish
- Can $P(\text{salmon}|\text{caught two})$ tell us anything about $P(\text{swordfish}|\text{caught two})$?
- N-gram models: no.

Word similarity in language modeling

- Early version: class-based language models (J&M 4.9.2)
  - Define classes c of words, by hand or automatically
  - $P_{CL}(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$ (an HMM)
- Recent version: **distributed** language models
  - Use neural networks to project words into a continuous space, so words that appear in similar contexts have similar representations (e.g., Mikolov 2012).

Distributed word representations

- Each word represented as high-dimensional vector (50-500 dims)

E.g., salmon is $[0.1, 2.3, 0.6, -4.7, \ldots]$

- Similar words represented by similar vectors

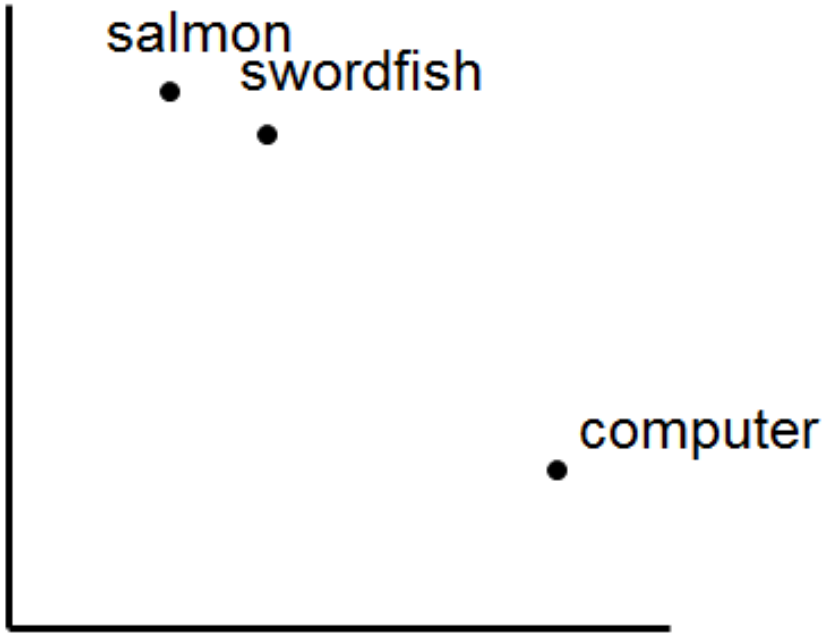
E.g., swordfish is $[0.3, 2.2, 1.2, -3.6, \ldots]$

- More about this later in the course.

Training the model

- Goal: learn word representations (**embeddings**) such that words that behave similarly are close together in high-dimensional space.

- 2-dimensional example:



Training the model

- N-gram LM: collect counts, maybe optimize some parameters
  - (Relatively) quick, especially these days (minutes-hours)
- distributed LM: learn the representation for each word
  - Use ML methods like neural networks that iteratively improve embeddings
  - Can be extremely time-consuming (hours-days)
  - Learned embeddings capture both semantic and syntactic similarity.

Using the model

Want to compute $P(w_1 \ldots w_n)$ for a new sequence.

- N-gram LM: again, relatively quick
- distributed LM: often prohibitively slow for real applications
- An active area of research for distributed LMs

Other Topics in Language Modeling

Many active research areas!

- Modeling issues:
  - Morpheme-based language models: *preempt* vs. *preregistration*
  - Syntactic language models (more later)
  - Domain adaptation: when only a small corpus is available in the domain of interest

- Implementation issues:
  - Speed: both to train, and to use in real-time applications like translation and speech recognition.
  - Disk space and memory: espcially important for mobile devices

Summary

- LMs are central to many NLP tasks.
- Smoothing is designed to handle unseen ngrams at test time.
- Different smoothing methods account for different aspects of sparse data and word behaviour.
  - Laplace and Good Turing: unseen ngrams equally likely.
  - Interpolation/backoff: leverage advantages of both higher and lower order N-grams.
  - Kneser-Ney smoothing: accounts for diversity of history.
  - Distributed representations: account for word similarity.