ANLP Lecture 6
N-gram models and smoothing

Sharon Goldwater
(some slides from Philipp Koehn)

26 September 2019

School of informatics

Recap: N-gram models

- We can model sentence probs by conditioning each word on $N-1$ previous words.
- For example, a bigram model:

$$P(\vec{w}) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

- Or trigram model:

$$P(\vec{w}) = \prod_{i=1}^{n} P(w_i|w_{i-2}, w_{i-1})$$

MLE estimates for N-grams

- To estimate each word prob, we could use MLE...

$$P_{\text{ML}}(w_2|w_1) = \frac{C(w_1, w_2)}{C(w_1)}$$

- But what happens when I compute $P(\text{consuming}|\text{commence})$?
  - Assume we have seen commence in our corpus
  - But we have never seen commence consuming

MLE estimates for N-grams

- To estimate each word prob, we could use MLE...

$$P_{\text{ML}}(w_2|w_1) = \frac{C(w_1, w_2)}{C(w_1)}$$

- But what happens when I compute $P(\text{consuming}|\text{commence})$?
  - Assume we have seen commence in our corpus
  - But we have never seen commence consuming
- Any sentence with commence consuming gets probability 0

> The guests shall commence consuming supper
> Green inked commence consuming garden the

The problem with MLE

- MLE estimates probabilities that make the observed data maximally probable
- by assuming anything unseen cannot happen (and also assigning too *much* probability to low-frequency observed events).
- It **over-fits** the training data.
- We tried to avoid zero-probability sentences by modelling with smaller chunks (n-grams), but even these will sometimes have zero prob under MLE.

Today: **smoothing** methods, which reassign probability mass from observed to unobserved events, to avoid overfitting/zero probs.

Today's lecture:

- How does add-alpha smoothing work, and what are its effects?
- What are some more sophisticated smoothing methods, and what information do they use that simpler methods don't?
- What are training, development, and test sets used for?
- What are the trade-offs between higher order and lower order n-grams?
- What is a word embedding and how can it help in language modelling?

Add-One Smoothing

- For all possible bigrams, add one more count.

$$P_{\text{ML}}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

$$\Rightarrow \qquad P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1})} \qquad ?$$

Add-One Smoothing

- For all possible bigrams, add one more count.

$$P_{\text{ML}}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

$$\Rightarrow \qquad P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1})} \qquad ?$$

- NO! Sum over possible w_i (in vocabulary V) must equal 1:

$$\sum_{w_i \in V} P(w_i|w_{i-1}) = 1$$

- True for P_{ML} but we increased the numerator; must change denominator too.

Add-One Smoothing: normalization

- We want: $$\sum_{w_i \in V} \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + x} = 1$$

- Solve for x:

$$\begin{aligned} \sum_{w_i \in V} (C(w_{i-1}, w_i) + 1) &= C(w_{i-1}) + x \\ \sum_{w_i \in V} C(w_{i-1}, w_i) + \sum_{w_i \in V} 1 &= C(w_{i-1}) + x \\ C(w_{i-1}) + v &= C(w_{i-1}) + x \end{aligned}$$

- So, $P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1},w_i)+1}{C(w_{i-1})+v}$ where v = vocabulary size.

Add-One Smoothing: effects

- Add-one smoothing:

$$P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + v}$$

- Large vobulary size means v is often much larger than $C(w_{i-1})$, overpowers actual counts.

- Example: in Europarl, $v = 86,700$ word types (30m tokens, max $C(w_{i-1}) = 2$m).

Add-One Smoothing: effects

$$P_{+1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + v}$$

Using $v = 86,700$ compute some example probabilities:

$C(w_{i-1}) = 10,000$

$C(w_{i-1}, w_i)$	$P_{\text{ML}} =$	$P_{+1} \approx$
100	1/100	1/970
10	1/1k	1/10k
1	1/10k	1/48k
0	0	1/97k

$C(w_{i-1}) = 100$

$C(w_{i-1}, w_i)$	$P_{\text{ML}} =$	$P_{+1} \approx$
100	1	1/870
10	1/10	1/9k
1	1/100	1/43k
0	0	1/87k

The problem with Add-One smoothing

- All smoothing methods "steal from the rich to give to the poor"

- Add-one smoothing steals way too much

- ML estimates for frequent events are quite accurate, don't want smoothing to change these much.

Add-α Smoothing

- Add $\alpha < 1$ to each count

$$P_{+\alpha}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha v}$$

- Simplifying notation: c is n-gram count, n is history count

$$P_{+\alpha} = \frac{c + \alpha}{n + \alpha v}$$

- What is a good value for α?

Optimizing α

- Divide corpus into **training** set (80-90%), **held-out** (or **development** or **validation**) set (5-10%), and **test** set (5-10%)
- Train model (estimate probabilities) on training set with different values of α
- Choose the value of α that minimizes perplexity on dev set
- Report final results on test set

A general methodology

- Training/dev/test split is used across machine learning
- Development set used for evaluating different models, debugging, optimizing parameters (like α)
- Test set simulates deployment; only used once final model and parameters are chosen. (Ideally: once per paper)
- Avoids overfitting to the training set and even to the test set

Is add-α sufficient?

- Even if we optimize α, add-α smoothing makes pretty bad predictions for word sequences.
- Some cleverer methods such as Good-Turing improve on this by discounting less from very frequent items. But there's still a problem...

Remaining problem

- In given corpus, suppose we never observe
  - Scottish beer drinkers
  - Scottish beer eaters
- If we build a trigram model smoothed with Add-α or G-T, which example has higher probability?

Remaining problem

- Previous smoothing methods assign equal probability to all unseen events.
- Better: use information from lower order N-grams (shorter histories).
  - beer drinkers
  - beer eaters
- Two ways: **interpolation** and **backoff**.

Interpolation

- Higher and lower order N-gram models have different strengths and weaknesses
  - high-order N-grams are sensitive to more context, but have sparse counts
  - low-order N-grams consider only very limited context, but have robust counts
- So, combine them:

$$\begin{aligned} P_I(w_3|w_1,w_2) = {} & \lambda_1\, P_1(w_3) && \mathrm{P}_1(\text{drinkers}) \\ & + \lambda_2\, P_2(w_3|w_2) && \mathrm{P}_2(\text{drinkers}|\text{beer}) \\ & + \lambda_3\, P_3(w_3|w_1,w_2) && \mathrm{P}_3(\text{drinkers}|\text{Scottish},\text{beer}) \end{aligned}$$

Interpolation

- Note that λ_is must sum to 1:

$$\begin{aligned} 1 &= \sum_{w_3} P_I(w_3|w_1,w_2) \\ &= \sum_{w_3} [\lambda_1\, P_1(w_3) + \lambda_2\, P_2(w_3|w_2) + \lambda_3\, P_3(w_3|w_1,w_2)] \\ &= \lambda_1 \sum_{w_3} P_1(w_3) + \lambda_2 \sum_{w_3} P_2(w_3|w_2) + \lambda_3 \sum_{w_3} P_3(w_3|w_1,w_2) \\ &= \lambda_1 + \lambda_2 + \lambda_3 \end{aligned}$$

Fitting the interpolation parameters

- In general, any weighted combination of distributions is called a **mixture model**.
- So λ_is are **interpolation parameters** or **mixture weights**.
- The values of the λ_is are chosen to optimize perplexity on a held-out data set.

Back-Off

- Trust the highest order language model that contains N-gram, otherwise "back off" to a lower order model.
- Basic idea:
  - discount the probabilities slightly in higher order model
  - spread the extra mass between lower order N-grams
- But maths gets complicated to make probabilities sum to 1.

Back-Off Equation

$$P_{BO}(w_i|w_{i-N+1},...,w_{i-1}) =$$

$$= \begin{cases} P^*(w_i|w_{i-N+1},...,w_{i-1}) \\ \qquad \text{if count}(w_{i-N+1},...,w_i) > 0 \\ \alpha(w_{i-N+1},...,w_{i-1})\ P_{BO}(w_i|w_{i-N+2},...,w_{i-1}) \\ \qquad \text{else} \end{cases}$$

- Requires
  - adjusted prediction model $P^*(w_i|w_{i-N+1},...,w_{i-1})$
  - backoff weights $\alpha(w_1,...,w_{N-1})$
- See textbook for details/explanation.

Do our smoothing methods work here?

Example from MacKay and Bauman Peto (1994):

> 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 used in interpolation or 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:

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

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

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

Kneser-Ney in practice

- Original version used backoff, later "modified Kneser-Ney" introduced using interpolation (Chen and Goodman, 1998).
- Fairly complex equations, but until recently the best smoothing method for word n-grams.
- See Chen and Goodman 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.

Bayesian interpretations of smoothing

- We contrasted MLE (which has a mathematical justification, but practical problems) with smoothing (heuristic approaches with better practical performance).
- It turns out that many smoothing methods are mathematically equivalent to forms of **Bayesian estimation** (uses priors and uncertainty in parameters). So these have a mathematical justification too!
  - Add-α smoothing: Dirichlet prior
  - Kneser-Ney smoothing: Pitman-Yor prior

See MacKay and Bauman Peto (1994); (Goldwater, 2006, pp. 13-17); Goldwater et al. (2006); Teh (2006).

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

- Two words with $C(w_1) \gg C(w_2)$
  - salmon
  - swordfish
- Can $P(\text{salmon}|\text{caught two})$ tell us something 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
  - Current models have better perplexity than MKN.
  - Ongoing research to make them more efficient.
  - Examples: Log Bilinear LM (Mnih and Hinton, 2007), Recursive Neural Network LM (Mikolov et al., 2010), LSTM LMs, etc.

Distributed word representations

(also called **word embeddings**)

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

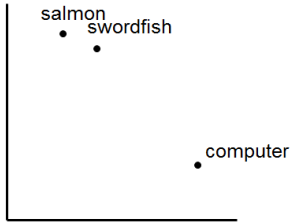
  E.g., salmon is $[0.1, 2.3, 0.6, -4.7, \ldots]$

- Similar words represented by similar vectors

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

Training the model

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



We'll come back to this later in the course...

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
  - Solved with machine learning methods (e.g., neural networks)
  - Can be extremely time-consuming (hours-days)
  - Learned embeddings seem to encode both semantic and syntactic similarity (using different dimensions) (Mikolov et al., 2013).

Using the model

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

- N-gram LM: again, relatively quick
- distributed LM: can be slow, but varies; often LMs not used in isolation anyway (instead use end-to-end neural model, which does some of the same work).
- An active area of research for distributed LMs

Other Topics in Language Modeling

Many active research areas in language modeling:

- Factored/morpheme-based/character language models: back off to word stems, part-of-speech tags, and/or sub-word units
- Domain adaptation: when only a small domain-specific corpus is available
- Time efficiency and space efficiency are both key issues (esp on mobile devices!)

Summary

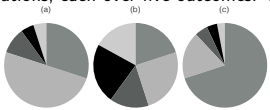
- We can estimate sentence probabilities by breaking down the problem, e.g. by instead estimating N-gram probabilities.
- Longer N-grams capture more linguistic information, but are sparser.
- Different smoothing methods capture different intuitions about how to estimate probabilities for rare/unseen events.
- Still lots of work on how to improve these models.

Announcements

- Assignment 1 will go out on Monday: build and experiment with a **character-level N-gram model**.
- Intended for students to work in **pairs**: we strongly recommend you do. You can discuss and learn from your partner.
- We'll have a signup sheet if you want to choose your own partner.
- On Tue/Wed, we will assign partners to anyone who hasn't already signed up with a partner (or told us they want to work alone).
- You may not work with the same partner for both assessed assignments.

Questions and exercises (lects 5-6)

1. What does sparse data refer to, and why is it important in language modelling?
2. Write down the equations for the Noisy Channel framework and explain what each term refers to for an example task (say, speech recognition).
3. Re-derive the equations for a trigram model without looking at the notes.
4. Given a sentence, show how its probability is computed using an unigram, bigram, or trigram model.
5. Using a unigram model, I compute the probability of the word sequence the cat bit the dog as 0.00057. Give another word sequence that has the same probability under this model.
6. Given a probability distribution, compute its entropy.
7. Here are three different distributions, each over five outcomes. Which has the highest entropy? The lowest?



8. What is the purpose of the begin/end of sentence markers in an n-gram model?
9. Given a text, how would you compute $P(to|want)$ using MLE, and using add-1 smoothing? What about $P(want|to)$? Which conditional probability, $P(to|want)$ or $P(want|to)$, is needed to compute the probability of "I want to go" under a bigram model?
10. Consider the following trigrams: (a) private eye maneuvered and (b) private car maneuvered. (*Note:* a *private eye* is slang for a detective.) Suppose that neither of these has been observed in a particular corpus, and we are using backoff to estimate their probabilities. What are the bigrams that we will back off to in each case? In which case is the backoff model likely to provide a more accurate estimate of the trigram probability? Why?
11. Suppose I have a smallish corpus to train my language model, and I'm not sure I have enough data to train a good 4-gram model. So, I want to know whether a trigram model or 4-gram model is likely to make better predictions on other data similar to my corpus, and by how much. Let's say I only consider two types of smoothing methods: add-alpha smoothing with interpolation, or Kneser-Ney. Describe what I should do to answer my question. What steps should I go through, what experiments do I need to run, and what can I conclude from them?

References

Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Center for Research in Computing Technology, Harvard University.

Goldwater, S. (2006). *Nonparametric Bayesian Models of Lexical Acquisition*. PhD thesis, Brown University.

Goldwater, S., Griffiths, T. L., and Johnson, M. (2006). Interpolating between types and tokens by estimating power-law generators. In *Advances in Neural Information Processing Systems 18*, pages 459–466, Cambridge, MA. MIT Press.

MacKay, D. and Bauman Peto, L. (1994). A hierarchical Dirichlet language model. *Natural Language Engineering*, 1(1).

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751.

Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM.

Teh, Y. W. (2006). A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 985–992, Syndney, Australia.