Language modeling

Keith Trnka

CISC882 Oct 8, 2009

What is a language model?

- input: some representation of context (possibly empty)
  - usually context to the left (sequential processing)
- output: a mapping of words to probabilities
- abstractly written: P(word | history)

Representing history

- How could we represent the history/context?
  - the previous two words (trigram model)
  - bag-of-words of all preceding words (some topic-based models)
  - no context at all (unigram model)
  - the previous two part of speech tags (trigram pos model)

Probabilities

- ngram models: a big lookup table, indexed by the previous (n-1) words
  - each value in the table is a table mapping words to frequencies (during training) and probabilities after training
- How else could you get probabilities?
  - more formal machine learning (decision trees, maximum entropy, neural networks)

Applications

- Note: language models are just tools, not the end goal (VERY common theme in NLP/CL)
- (Automatic) Speech Recognition (ASR)
  find the sentence that maximizes
  P(sounds | sentence) * P(sentence)
  - use a language model to compute P(sentence) using chain rule
  - process word-by-word (harder) or sentence-by-sentence

Applications (cont'd)

- Machine Translation (MT)
  find the (English) sentence that maximizes
  P(French sentence | sentence) * P(sentence)
  - the first component "does translation"
  - the second component "does generation"
  - very similar model to ASR

Applications (cont'd)

- surface generation - make a sensible sentence from underspecified semantics (Halogen/ Nitrogren project)
- spell checking - most systems just combine a dictionary/unigram model with a model of errors
  - bigram/etc models can be used for fixing things like "the nand flash" vs "then and flash"

Applications (cont'd)

- ambiguous keyboards
  - cell-phone typing - does 843 mean "the", "vid", "tie", "uid"?
  - can fit it into the usual model:
    find the word that maximizes
    P(keys | word) * P(word)
    - P(keys | word) is either 1 or 0 unless you model typos
    - P(word) might really be P(word | previous word)

Applications (cont'd)

- document clustering - cluster documents using similarity of ngrams
- information retrieval - many methods use much more than unigrams
- author identification

Applications (cont'd)

- Word prediction
  - can fit it into the usual framework:
    P(prefix | word) * P(word | history)
  - like ambiguous keyboards, P(prefix | word) isn't interesting - it's 1 or 0 unless you correct typos
  - used for augmentative and alternative communication (AAC) and also machine-aided translation (and somewhat on cellphones)

Notation sidetrack

- Say you write $P(w \mid w_{-1}, w_{-2})$
- What could this refer to?
  - plain old MLE trigram model
  - trigram model with Katz backoff
  - trigram model with Witten-Bell smoothing
  - a decision tree using the previous two words as input
  - a trigram model interpolated with bigram/unigram (note on weights)

Not equal

- Not all applications are equal
  - word prediction doesn't require non-zero probabilities for words not in the ngram model
    - What should we predict for OOVs?
  - speech recognition DOES require non-zero probabilities for words not in the ngram model
    - What text should we return for OOVs?

Smoothing

- Tasks like ASR require non-zero probabilities for words not in the training vocabulary
  - (trigram) need to estimate the probability of words that never occurred after the previous two words in the training data
  - (unigram) need to estimate the probability of words we've never seen at all
- smoothing examples will use unigrams at first

Add one

- useful for
  - making people angry
- just add one to the frequency of all words:

$$P(w) = \frac{c(w) + 1}{c(*) + |V|}$$

Keith's lazy notation:
c(*) - wildcard, count of all words
|V| - size of the vocabulary set

Add one

- Where does everything come from?
  - c(w) we just count in training
  - c(*) we can just add up after training
  - |V| ... <*crickets chirping*>
    - For this to work, this must be at least the size of the total vocabulary in both training AND testing
    - Ideally the size of the language's vocabulary
- So... add-one isn't accurate (see Kathy's lecture) and we can't compute it accurately anyway...

Add one

- Can modify it to add a small constant instead, but it's still just as bad

Discount ratios

- Most smoothing methods can be interpreted as a discount ratio:
  new count = old count * discount ratio
- Compute probabilities based on the new counts and the old sum of counts
- Easy to get the amount of probability leftover for unseen words - just add all the probabilities and diff with 1

Discount ratios

- Most methods come up with pretty principled amounts of unseen probability mass
- Most methods guess at the number of words that occurred zero times in training (N_0)
  - divide the unseen probability mass by this to get the probability of a particular unseen word
  - sketchy but common: $N_0 = N_1$

Witten-Bell smoothing

- Discount ratio is (tokens) / (tokens + types)
- Not to be confused with
  P(seen word) = (tokens - types) / tokens
  - yeah I made this mistake before

Good-Turing smoothing

- key insight: why are we taking away the same percent of probability for infrequent and frequent words alike?
  - words that occurred once in training may actually be equally probable (in the language) as some words that didn't occur
  - words that occurred a lot probably occurred a lot for a good reason

Good-Turing smoothing

- discounts more from infrequent words than frequent words
- uses the ratio of the number of words that occurred x times to x+1 times
  - zeros in that N_x function make it a pain in the butt

Smoothing conditional ngrams

- How do you compute the probability of a conditional ngram, say P(w | w_{-1})?
- Remember: probabilities must sum to 1, no more, no less
- MLE version in most papers/books is slightly off:

$$P(w \mid w_{-1}) = \frac{c(w_{-1}\ w)}{c(w_{-1})}$$

Smoothing conditional ngrams

- Not all single-word occurrences are legitimate multi-word contexts:

$$P(w \mid w_{-1}) = \frac{c(w_{-1}\ w)}{c(w_{-1}\ *)} = \frac{c(w_{-1}\ w)}{\sum_i c(w_{-1}\ w_i)}$$

Smoothing conditional ngrams

- If we just use smoothing on a trigram
  - unseen mass is spread evenly over words that didn't follow the previous two words in training
  - these could (potentially) all be equally unlikely
    the nice book
    the nice attendant
    the nice the
    the nice lavender
    the nice qq[j230rjq23p9rthpqendjk!*mvm
  - can we make it so that they have probability more in order with their likelihood?

Backoff

- key insight: not all words that haven't been seen in a context are equally unlikely
- distribute the unseen probability mass for trigrams to the bigram distribution
- distribute the unseen probability mass for bigrams to the unigram distribution
- unigrams still have the same old smoothing though

Backoff (cont'd)

$$P'(w \mid w_{-1}, w_{-2}) = \begin{cases} P_{smooth}(w \mid w_{-1}, w_{-2}) & if P(w \mid w_{-1}, w_{-2}) > 0 \\ \alpha_{w_{-1}, w_{-2}} * P_{smooth}(w \mid w_{-1}) & if P(w \mid w_{-1}) > 0 \\ \alpha_{w_{-1}} * \alpha_{w_{-1}, w_{-2}} * P_{smooth}(w) & otherwise \end{cases}$$

Backoff (cont'd)

$$P'(w \mid w_{-1}, w_{-2}) = \begin{cases} P_{smooth}(w \mid w_{-1}, w_{-2}) & if P(w \mid w_{-1}, w_{-2}) > 0 \\ \alpha_{w_{-1},w_{-2}} * P_{smooth}(w \mid w_{-1}) & if P(w \mid w_{-1}) > 0 \\ \alpha_{w_{-1}} * \alpha_{w_{-1},w_{-2}} * P_{smooth}(w) & otherwise \end{cases}$$

- What is alpha$_{w\text{-}1,w\text{-}2}$?
  - the unseen/held-out probability mass from the trigram distribution for the context (w_{-1}, w_{-2})
- What if (w_{-1}, w_{-2}) never occurred? *dun dun dun*
  - then we just say that the unseen mass is 1.0

Backoff (cont'd)

$$P'(w \mid w_{-1}, w_{-2}) = \begin{cases} P_{smooth}(w \mid w_{-1}, w_{-2}) & if P(w \mid w_{-1}, w_{-2}) > 0 \\ \alpha_{w_{-1},w_{-2}} * P_{smooth}(w \mid w_{-1}) & if P(w \mid w_{-1}) > 0 \\ \alpha_{w_{-1}} * \alpha_{w_{-1},w_{-2}} * P_{smooth}(w) & otherwise \end{cases}$$

- Does it sum to 1?
  - nope!
  - thought experiment: what if there were only one word in the language? (Assume the discount ratio is neither 1 nor 0)
  - if you want to sum to 1, you add instead of if/then

Backoff (cont'd)

- backoff is attributed to
  - Slava M. Katz. (1987) *Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer*. In IEEE Transactions on Acoustics Speech and Signal Processing (ISASSP).
  - Katz' backoff uses Good-Turing smoothing in a very specific manner
  - caution: don't call something Katz' backoff if it's not the same smoothing method

Linear interpolation

$$P(w \mid w_{-2}w_{-1}) = \lambda_3 * \frac{c(w_{-2}\ w_{-1}\ w)}{c(w_{-2}\ w_{-1}\ *)} + \lambda_2 * \frac{c(w_{-1}\ w)}{c(w_{-1}\ *)} + \lambda_1 * \frac{c(w)}{c(*)}$$

- Instead of all the backoff mess, we could just do a weighted average of the trigram, bigram, unigram probabilities
- The weights must sum to 1
- Can add a weighted even-distribution at the end for speech recognition and such

Linear interpolation

$$P(w \mid w_{-2}w_{-1}) = \lambda_3 * \frac{c(w_{-2}\ w_{-1}\ w)}{c(w_{-2}\ w_{-1}\ *)} + \lambda_2 * \frac{c(w_{-1}\ w)}{c(w_{-1}\ *)} + \lambda_1 * \frac{c(w)}{c(*)}$$

- Where do the weights come from?
  - can just guess and it'll be okay for some tasks
  - in general $\text{weight}_{\text{trigram}}$, $\text{weight}_{\text{bigram}} > \text{weight}_{\text{unigram}}$
  - can learn them (see Jelinek's deleted interpolation)

Kneser-Ney smoothing

- a specialized combination of backoff and smoothing, like Katz' backoff
- key insight: some zero-frequencies ***should be*** zero, rather than a proportion from a more robust distribution
  - example: suppose "Francisco" and "stew" have the same frequency, and we're backing off from "expensive" - which would you pick?

Kneser-Ney smoothing

- more or less accepted as the best method

Backoff finished

- inherent tradeoff in ngrams:
  good context vs. robustness
- backoff is a general method for ordering models in
  - decreasing strength of conditioning information
  - increasing robustness
- data sparseness is a core topic with ngrams

Keith's notes

- We don't need probabilities for words we've never seen for some tasks
- For simple implementations of word prediction, can just follow this method:
  - check the trigram distribution - add any words that match the context and prefix in descending order
  - if list isn't full, do the same thing for bigrams (but don't add the same word twice!)
  - if list still isn't full, do the same thing for unigrams
  - if the list STILL isn't full, add words from a large word list (see Yet Another Word List = YAWL)

Keith's notes (cont'd)

- Use a special token to signify start-of-sentence (really does help)
- Can delete words that only occur once (can hurt performance, but cuts memory usage in half or so)
- Use a dictionary at the end of backoff (it helps more than weeks of work on smoothing)

Trigrams

- we've been using variations on trigrams as "state of the art" for about 30 years....
  - Frederick Jelinek. (1991) *Up from trigrams! - the struggle for improved language models*. In proceedings of Eurospeech.
  - Joshua Goodman (2001) *A Bit of Progress in Language Modeling, Extended Version*. Microsoft Research Technical Report MSR-TR-2001-72.
    - if you plan to do research on language modeling, READ THIS
  - improvements over trigrams are possible, but each individual improvement is somewhat small

Cache modeling

- sometimes called cache, sometimes called recency
- key insight: words tend to be repeated in a document (especially content words)
- simple method
  - maintain a unigram model of the current document
  - fill predictions from cache after the normal unigram model, but before dictionary

Cache modeling (cont'd)

- more advanced variation: if the first letter of the prefix is capitalized, use a special cache for that
  - give priority to the proper name cache over the base ngram model
  - Jianhua Li and Graeme Hirst (2005) *Semantic Knowledge in Word Completion*. In proceedings of ACM SIGACCESS Conference on Computers and Accessibility (ASSETS).

Cache modeling (cont'd)

- iteratively train a full-fledged ngram model on testing data
  - Jelinek, F. and Merialdo, B. and Roukos, S. and Strauss, M. (1991) *A dynamic language model for speech recognition*. In proceedings of the DARPA Workshop on Speech and Natural Language.
  - trigrams, bigrams, and unigrams trained on the most recent C words (e.g., 100-1000)
  - used linear interpolation with a static ngram model
  - reduced perplexity and WER pretty well

Cache modeling (cont'd)

- if you don't reset the cache between documents, that can work too
  - Tonio Wandmacher and Jean-Yves Antoine. (2006) *Training Language Models without Appropriate Language Resources: Experiments with an AAC System for Disabled People*. European conference on Language Resources and Evaluation (LREC)
  - it's modeling something a little different though
  - beware! Now the order of processing documents is significant!

Cache modeling (cont'd)

- many researchers use decayed caches
  - linear: most recent word has 100% weight, least recent has 0% weight
  - exponential: multiply all counts in the cache by say 0.95 before adding a new token
- caches are effective in a part of speech model
- beware: cache modeling + typos will cause you to predict typos

Pruning

- key insight: ngram models are susceptible to over-fitting, just like other ML methods
- Andreas Stolcke. (1998) *Entropy-based pruning of backoff language models*. In proceedings of DARPA Broadcast News Transcription and Understanding Workshop.
- can prune backoff ngram models just like decision trees
- SRI LM toolkit has an implementation

Pruning (cont'd)

- personal experiences
  - reduces language model size effectively
  - doesn't improve testing on the same type of data
  - does improve testing on a different type of data
    - it's a more generic language model

Other tricks

- variable-length language models - Google has used 9-grams without much trouble with this trick (somewhat like pruning)
- skip ngrams - you could condition on the w_{-2} but not w_{-1} for instance, interpolate many such ngrams

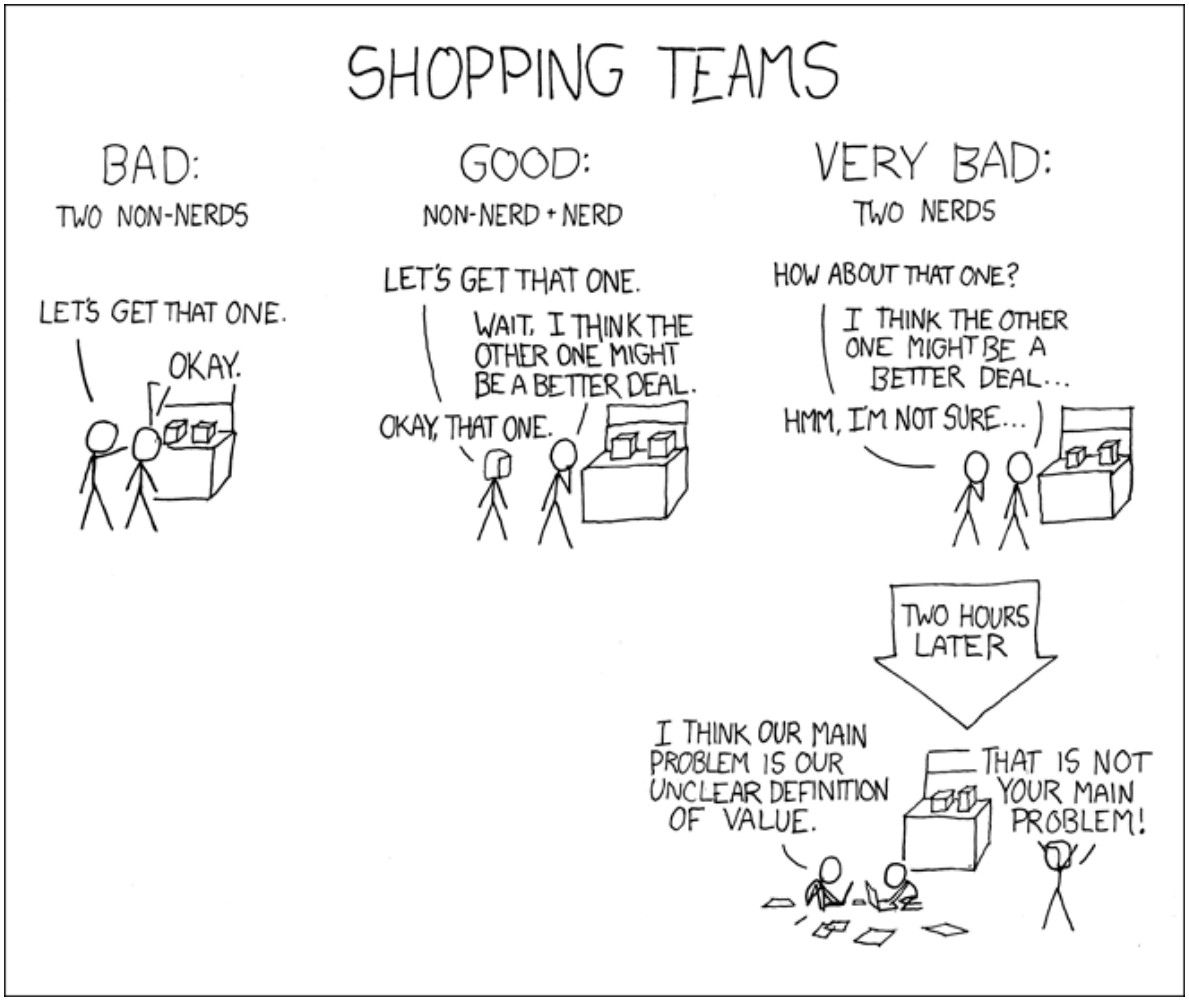
Personal experiences

- how much data a model "needs" is proportional to its parameter space (e.g., trigrams = $|V|^3$, bigrams = $|V|^2$)
- unigram vs. bigram difference is night vs. day
- bigram vs. trigram difference is very subtle (especially without a ***lot*** of data)

Personal experiences

- implementing word prediction is completely different than the equations
  - if you actually sorted or filtered the whole vocabulary at every prediction, it takes waaaaaay too long to finish
  - unigram pre-sort and store them pre-filtered to a few letters
  - conditional distributions are small, so you can sort and filter them on demand

Evaluation methods



http://www.xkcd.com/309/

Evaluation methods

- application evaluations
  - word prediction - keystroke savings (percentage reduction in key presses)
  - ASR - word error rate
- intrinsic evaluations
  - geometric mean word probability
  - perplexity (the inverse of last bullet)

Evaluation methods

- Why I hate keystroke savings
  - it takes a long time to compute
  - need to do heavy optimization (the code looks very little like the equations in the end)
  - doesn't range from 0-100% - each word takes at least on keystroke
- But... it's as close as I can get to user evaluation

Evaluation methods

- Why I hate perplexity
  - doesn't always correlate with keystroke savings or WER
  - is sensitive to choice of N_0
  - reducing perplexity on words that are already predicted right away....
- But... it's fast (most papers use it, many use it ***exclusively***)

Evaluation methods

- Why I hate evaluation in general
  - unclear/inconsistent definitions of value
  - many ways to compute keystroke savings (is an uppercase letter two keys? how about punctuation? what about a newline?)
    - some of the questions depend on whether it's word prediction for AAC vs mobile text entry
  - subtle variations in perplexity
    - really need to include a token for the ending of a sentence (but few people say whether they do or don't)

Evaluation methods

- rule of thumb:
  - if your model is too complex to complete testing before the deadlines,
    (or if you want to compare word prediction and ASR)
    (or if you don't wanna work too hard)
    use perplexity
  - else,
    use application-specific evaluation

Cross-validation

- split a corpus into *k* sets
- iterate through the sets:
  - for a given set *i*, train the model on all other sets
  - test on set *i*
  - combine the results of testing on all *k* sets afterwards
    - what's the subtle problem here?

Cross-validation

- cross-validation is a lifestyle...
  - cause now all your code takes 10x longer to run or so
  - results will be more resilient against over-fitting your data (it's like a bike helmet for research)

Balancing sets...

- (thought experiment time)
- imagine we have a text message corpus (very small documents)
- imagine we're doing 10-fold cross-validation
- how do you assign messages to sets?
  - randomly?
  - what happens if you cluster messages by similarity?
  - what's the opposite of that?

Balancing sets...

- very important for topic adaptation
  - why: if there are topics in testing that weren't in training (or vice versa)
- important for style adaptation
- still relevant even without categorization (simple = balance sets to minimize OOVs)
- also balancing # of documents **and** # of words is takes work

Domain-variations

- ngrams are ***very*** sensitive to the difference between training and testing data
  - make a clear distinction between in-domain and out-of-domain
  - but it's not so clear...
    - broadcast news transcription
    - newspaper
    - email
    - blogs

Spoken vs. written

- designing your system for written language is ***completely*** different than spoken language
- corpus normalization (and nightmares)
  - do all your corpora capitalize the first word in a sentence?
  - do they all have punctuation?
  - (spoken especially) “don’t know” vs. “dunno”
  - spoken may have speech repairs, written may have typos