TTIC 31110
Speech Technologies

May 19, 2020

Announcements

- HW4 due Monday 5/25 7pm
- Project proposal due Friday 5/22 7pm
- Code base for end-to-end ASR available via canvas site
- Code base for acoustic word embeddings is coming...
- Revised project timeline for graduating seniors has been posted
    - If you are graduating and haven't received an email from me about this, please email me!

Outline

Reminder: The “fundamental equation of ASR”

$$\begin{aligned}\mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{w}|\mathbf{O}) \\ &= \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{O}|\mathbf{w})p(\mathbf{w})\end{aligned}$$

- where $\mathbf{O}$ are the acoustic feature frames for an utterance, $\mathbf{w}$ is a word sequence
- $p(\mathbf{O}|\mathbf{w})$ is the acoustic model
- $p(\mathbf{w})$ is the language model
- Both are too complex to model directly; we factor them into manageable chunks

History-based statistical LMs

- $p(\mathbf{w})$ can be expanded using the chain rule:

$$p(\mathbf{w}) = \prod_{i=1}^{K} p(w_i|w_1,\ldots,w_{i-1}) = \prod_{i=1}^{K} p(w_i|h_i)$$

  where $h_i = (w_1, \ldots, w_{i-1})$ is the *history* for word w_i.
- Note: First & last words typically assumed to be a sentence boundary marker, $w_1 = w_K = <>$
- Too many possible histories $\Rightarrow$ reduce to equivalence classes $\phi(h_i)$, such that $p(w_i|h_i) \approx p(w_i|\phi(h_i))$
- Good equivalence classes maximize the information about the current word given the class $\phi(h_i)$
- (LMs requiring the full word sequence $\mathbf{w}$ can be used, but usually in a "rescoring" setting – more later...)

n-gram language models

- In *n*-gram LMs, the history equivalence class is the previous $n-1$ words: $\phi(h_i) = (w_{i-1}, \ldots, w_{i-n+1})$
- For example:
  - bigram LM $p(w_i | w_{i-1})$
  - trigram LM $p(w_i | w_{i-1}, w_{i-2})$
- Trigrams were for a long time the dominant LM in large-vocabulary recognition research, but longer histories now being used with large training sets (even arbitrarily long histories)

Where do the probabilities come from?

Maximum-likelihood estimate of *n*-gram probabilities given some training set of text:

$$\hat{p}(\text{quick}| <>, \text{the}) = \frac{\text{count}(<>,\text{the, quick})}{\text{count}(<>,\text{the})}$$

Evaluating language models

- One way: Qualitatively (How good do random sentences generated from the LM look?)
- The ultimate way: Task performance (e.g., word error rate)
- Would like some intermediate way...
  - LMs are much quicker to estimate and run than entire speech systems
  - LMs can be estimated independently from text and then used in different tasks
  - Would like a way to test a LM independently of the final task

Evaluating LMs: Cross-entropy

- If X is a discrete random variable taking one of N values with probabilities $p_1, \ldots, p_N$, respectively, then the entropy of X is
$$H(X) = -\sum_{i=1}^{N} p_i \log_2 p_i$$
- The *cross-entropy* of a model $\hat{p}(X)$ with respect to some data set $\mathbf{X} = \{x_1, \ldots, x_n\}$ is $H_{\hat{p}}(\mathbf{X}) = -\frac{1}{n} \log_2 \hat{p}(\mathbf{X})$
- For an *n*-gram LM $\hat{p}(\cdot)$ on a test set $\mathbf{w} = (w_1, \ldots, w_n)$,

$$\begin{aligned} H_{\hat{p}}(\mathbf{w}) &= -\frac{1}{n} \log_2 \hat{p}(\mathbf{w}) \\ &= -\frac{1}{n} \log_2 \prod_{i=1}^{n} \hat{p}(w_i | \phi(h_i)) \\ &= -\frac{1}{n} \sum_{i=1}^{n} \log_2 \hat{p}(w_i | \phi(h_i)) \end{aligned}$$

Evaluating LMs: Cross-entropy (2)

- Intuition: This is the number of bits per word needed to encode this data set using the model
- For English texts, cross-entropy ranges from around 6 to 10 bits/word

Evaluating LMs: Perplexity

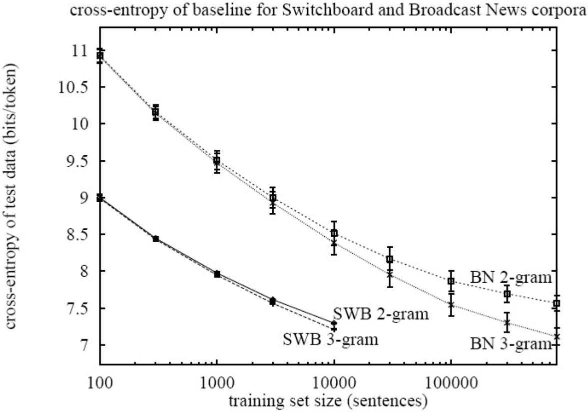
- Perplexity is related to the cross-entropy via
  $PP_p(\mathbf{w}) = 2^{H_p(\mathbf{w})}$
- For most purposes, lower entopy/perplexity $\Rightarrow$ lower uncertainty about the following word $\Rightarrow$ better language model
- For English text, perplexity ranges from around 25 to several 100s.
- Exercise: What is the perplexity of a uniform LM?
- Answer: the vocabulary size N
- Intuition: Perplexity is the average number of words possible after a given history (the average *branching factor* of the LM)

Evaluating LMs on different domains

Domain	Size	Type	Perplexity
Digits	11	All word	11
Resource Management	1,000	Word-pair	60
		Bigram	20
Air Travel Understanding	2,500	Bigram	29
		4-gram	22
WSJ Dictation	5,000	Bigram	80
		Trigram	45
	20,000	Bigram	190
		Trigram	120
Switchboard Human-Human	23,000	Bigram	109
		Trigram	93
NYT Characters	63	Unigram	20
		Bigram	11
Shannon Letters	27	Human	~ 2

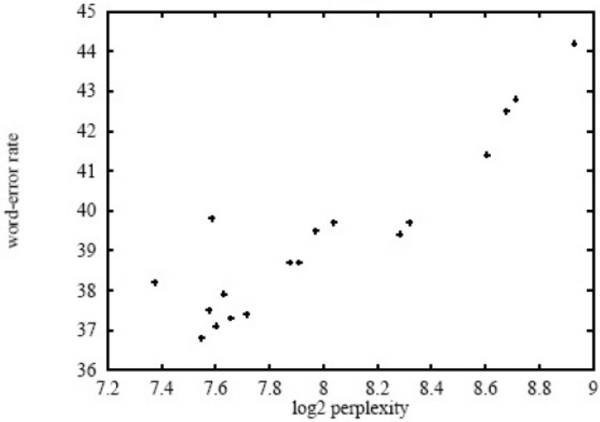
Evaluating LMs with different training set sizes

Cross-entropies of a particular LM on varying data set sizes (Chen and Goodman 1998):



How good is perplexity/entropy at predicting WER?

Different model types and domains (Chen *et al.*, 1998):



The sparse data problem in LMs

- Example [Jelinek 1997]: Given a text corpus of IBM patent descriptions
  - Training set: 1.5 million words
  - Test set: 300,000 words
- 23% of trigrams occurring in test corpus were unseen in training corpus
- In general, a vocabulary of size V has V^n possible *n*-grams, e.g. 20,000 words $\Rightarrow$ 400 million bigrams, 8 trillion trigrams
- To alleviate the unseen n-gram problem, we use *smoothing*: We re-distribute probability mass from frequently seen to unseen/rare events
  - Increase probability of unseen/rare *n*-grams
  - ... and therefore, decrease probability of frequently seen *n*-grams

Is it really OK to take away probability from frequently seen n-grams?

- Church and Gale [1992] split a 44 million word data set into two halves
- For a bigram that occurs, e.g., 5 times in the first half, how many times does it occur in the second half?
- Maximum-likelihood prediction: 5
- Actual: $\sim$ 4.2
- What's going on?

Laplace (add-one) smoothing

- Simplest idea: Add count of 1 to each event (seen or unseen)
- Example: unigram
  - Unsmoothed: $p(w_i) = \frac{C(w_i)}{N}$, where N is the total number of training word tokens and $C(A)$ is the count of event A
  - Smoothed: $p(w_i) = \frac{C(w_i)+1}{N+V}$, where V is the vocabulary size
- Example: bigram
  - Unsmoothed: $p(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$
  - Smoothed: $p(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)+1}{C(w_{i-1})+V}$
- A simple extension is "delta smoothing": Add some value δ instead of 1

Good-Turing estimate for 0-count events

- Suppose you are fishing, and have caught 10 carp, 3 cod, 2 tuna, 1 trout, 1 salmon, 1 eel; in addition, the pond also contains flounder and bass
- What is the probability that the next fish you catch is a new species?
- Intuition: It's as probable as the fish you've only seen once, i.e. $p(\text{new species}) = 3/18$!
- Probability that the next fish is a bass: $1/2 \cdot 3/18 = 3/36$
- Probability that the next fish you catch is an eel: $< 1/18$, since we just "stole" some probability mass

Good-Turing estimate (2)

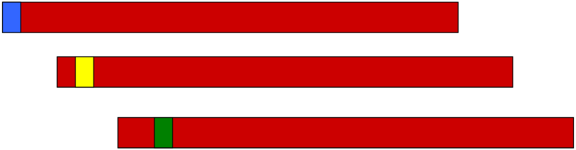
- Good-Turing estimate:
  - Let N = total number of items, n_r = number of items with count of at least r (seen r times)
  - Adjusted count: $r* = (r+1)\frac{n_{r+1}}{n_r}$
  - Adjusted probability estimate: r^*/N
  - Adjusted probability for unseen items: n_1/N
- Applied to a corpus of 22 million bigrams from AP newswire text:

r	n_r	r^*
0	74,671,100,000	0.0000270
1	2,018,046	0.446
2	449,721	1.26
3	188,933	2.24
4	105,668	3.24
5	68,379	4.22

Good-Turing estimate: Some intuition

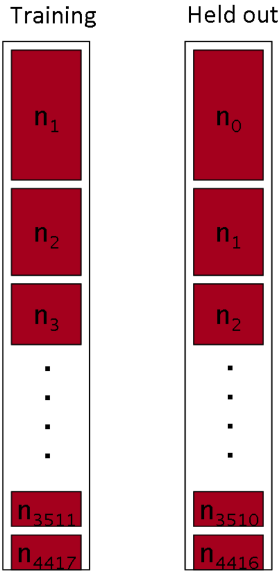
(Based on Ney *et al.*, "On the estimation of 'small' probabilities by leaving-one-out," *IEEE Trans. PAMI*, **17**:12,1202—1212, 1995. Slides from Dan Jurafksy.)

- A leave-one-out "thought experiment"
- Take each of the N training words out one at a time
- Form N training sets of size $N-1$, held-out sets of size 1



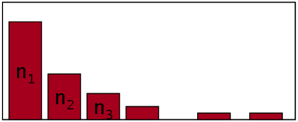
Good-Turing estimate: Some intuition (2)

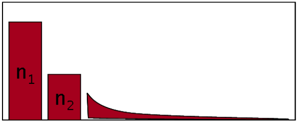
- Fraction of held-out words unseen in training $= n_1/N$
- Fraction of held-out words seen r times in training $= (r+1)n_{r+1}/N$
- So we expect $(r+1)n_{r+1}/N$ of future words to be those with training count r
- There are n_r words with training count r
- So each should occur with probability $\frac{(r+1)n_{r+1}/N}{n_r}$
- ...i.e., with expected count $r* = \frac{(r+1)n_{r+1}}{n_r}$



Good-Turing smoothing needs smoothing...

- The counts n_r are very noisy for high r
- E.g., suppose "the" occurs $r = 3220$ times
- What do you think n_{3220} is? What about n_{3221}?
- "Simple Good-Turing": replace n_r with a smoothed power-law estimate for high r





Back-off *n*-grams

- [Katz 1987]: Use maximum likelihood estimate if we have enough examples; otherwise, back off to a lower-order model:

$$
\begin{aligned}
p_{\text{Katz}}(w_i|w_{i-1}) &= p_{\text{ML}}(w_i|w_{i-1}) \text{ if } C(w_{i-1}w_i) \geq 5 \\
&= p_{\text{GT}}(w_i|w_{i-1}) \text{ if } 1 \leq C(w_{i-1}w_i) \leq 4 \\
&= \alpha_{w_{i-1}} p_{\text{Katz}}(w_i) \text{ if } C(w_{i-1}w_i) = 0
\end{aligned}
$$

- Choose $\alpha_{w_{i-1}}$ so that the probabilities sum to 1

Interpolation

- Rather than backing off, interpolate higher- and lower-order *n*-grams:

$$p(z|xy) = \lambda \frac{C(xyz)}{C(xy)} + \mu \frac{C(yz)}{C(y)} + (1 - \lambda - \mu) \frac{C(z)}{N}$$

  where $x = w_{i-2}, y = w_{i-1}, z = w_i$
- Optimize λ, μ on held-out data
- Interpolated models have been very successful
- Basic model (*held-out interpolation*): Optimize interpolation parameters on a held-out data set
- Alternatively (*deleted interpolation*): Rotate among M different divisions of training and held-out data; average the M resulting models

Interpolation with non-constant weights

Basic interpolation:

$$p(z|xy) = \lambda \frac{C(xyz)}{C(xy)} + \mu \frac{C(yz)}{C(y)} + (1-\lambda-\mu)\frac{C(z)}{N}$$

- Consider $p(\text{Francisco}|\text{eggplant})$
- Suppose "eggplant Francisco" never occurred in the training data, so we interpolate with lower-order (unigram) model
- Francisco is a common word, so smoothed LM might say it is likely!
- But, it occurs almost exclusively after "San", so the bigram model actually modeled it well!
- So: When interpolating, take into account how frequent the *context* is, i.e. use λ_{xy}, μ_y
- But that's again a lot of parameters to estimate... OK, OK: use $\lambda_{C(xy)}, \mu_{C(y)}$

Interpolation with discounting

Interpolation for a bigram with non-constant weights:

$$p(y|x) = \lambda_{C(x)} \frac{C(xy)}{C(x)} + (1 - \lambda_{C(x)}) \frac{C(y)}{N}$$

Could replace counts with Good-Turing estimates

$$p(y|x) = \lambda_{C(x)} \frac{C_{GT}(xy)}{C(x)} + (1 - \lambda_{C(x)}) \frac{C(y)}{N}$$

Or, save some trouble and note that Good-Turing discounts by about a constant amount $d \approx 0.75$

$$p(y|x) = \lambda_{C(x)} \frac{C(xy) - d}{C(x)} + (1 - \lambda_{C(x)}) \frac{C(y)}{N}$$

Kneser-Ney smoothing [1995]

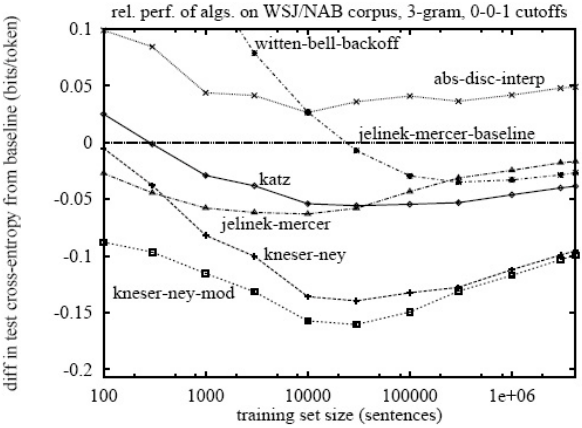
- Combines a number of above ideas
- Subtract a constant d from all counts
- Interpolate against a lower-order model that measures the number of *contexts* the word occurs in:
  $p(z|xy) = \frac{C(xyz)-d}{C(xy)} + \lambda \frac{C(\cdot z)}{\sum_z C(\cdot z)}$
- First term: Simulates the approximately constant discounting of Good-Turing
- Second term: Models probability of the unigram being a novel word
- Modified K-N smoothing (Chen and Goodman): Make d also a function of $c(xyz)$

Summary of n-gram smoothing techniques

- Usually, interpolation is better than back-off
- A dependable choice is smoothed n-gram is modified Kneser-Ney smoothing [Chen & Goodman 1998]
- But, differences are not huge: On broadcast news recognition, difference between best and worst trigram models is on the order of 1% WER
- Differences between best and worst can differ as model order is changed

Comparing different smoothing techniques

Entropy difference between a classic type of smoothing (Jelinek-Mercer) and various other techniques [Chen & Goodman 1998]:



What model order to use?

- Everything from bigrams to 5-grams is “common”
- Lower order $\Rightarrow$ less sparse data $\Rightarrow$ better estimates, and less sensitive to smoothing type
- With best smoothing, there is little or no degradation if the model is too large
- Given a lot of data, significant gains from higher-order models
- Google’s publicly available 5-gram: 10^{12} words in training set, 10^9 5-grams occurring $\geq$ 40 times, 13 million unique “words” occur $\geq$ 200 times

n-grams work surprisingly well...

- Probabilities are based on data, and the more, the better
- *n*-grams implicitly incorporate local syntax, semantics, pragmatics
- Many languages have a strong tendency toward a standard word order
- *n*-grams are relatively easy to integrate into standard Viterbi search

Problems with (basic) *n*-grams...

- Can't incorporate long-distance relationships
- Less well-suited to flexible word order languages
- Hard to incorporate new words, adapt to new domains, adapt to dynamic changes in topic
- Less good than humans at predicting following words, correcting recognizer errors
- Do not capture meaning for "downstream" tasks like sentence understanding

Other types of LMs: Class-based *n*-grams

Intuition: Consider the following sentences

- The class meets on Tuesdays.
- The class meets on Thursdays.
- The class meets on Fridays.
- The reading group meets on Tuesdays.
- The reading group meets on Thursdays.
- This class meets on Thursdays.
- That class meets on Thursdays.

Other types of LMs: Class-based *n*-grams

Intuition: Share parameters among words from similar classes

- Class-based bigram:
  $p(w_i|w_{i-1}) = p(w_i|\mathrm{class}(w_i))\, p(\mathrm{class}(w_i)|\mathrm{class}(w_{i-1}))$

Example classes: days of the week, numbers, first names, determiners, ... or automatically learned clusters!

Other types of LMs: Cache-based LMs

Main idea: Raise probability for previously seen words in the history. Examples:

- John suggested meeting on **Monday**, but on **Monday** I am busy.
- **John** suggested meeting on Monday, but I'd rather not meet **John**.

But also...

- John suggested meeting on **Monday**, but **Wednesday** is better.
- **John** suggested meeting on Monday, but I'd rather not meet **him**.

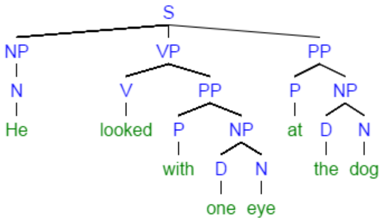
Other types of LMs: Structured LMs

Language can have very long-range dependencies...

- He **looked** with one eye **at** the dog.
- The **dog** that he looked at with one eye **was barking**.
- **Who** was he looking **at**?

Other types of LMs: Structured LMs

Structured language models factor the sentence probability based on parsing the sentence (or just the history):



Can be used in N-best rescoring framework:

- 1st pass: Using standard *n*-gramLM, output N best sentence hypotheses
- 2nd pass: Compute $p(\mathbf{w})$ or $p(\mathbf{O}, \mathbf{w})$ for each of the N best hypotheses using structured LM; output the best one

More...

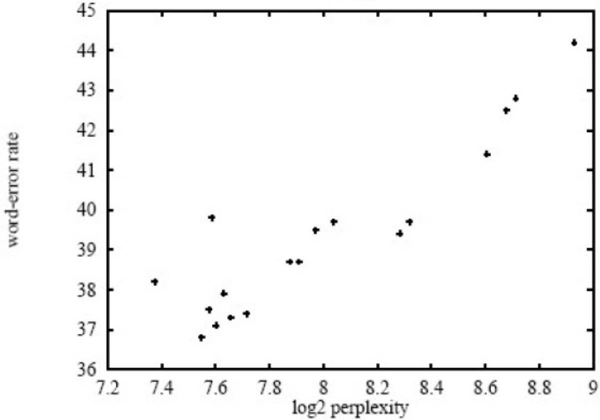
- Topic-based: Mixture of language models trained on data from different topics
- Discriminatively trained: Maximize a measure of error more closely connected to the task, rather than likelihood

Summary:

- Historically, class-based and discriminative LMs have been used a lot, and to some extent topic models
- The rest, less so

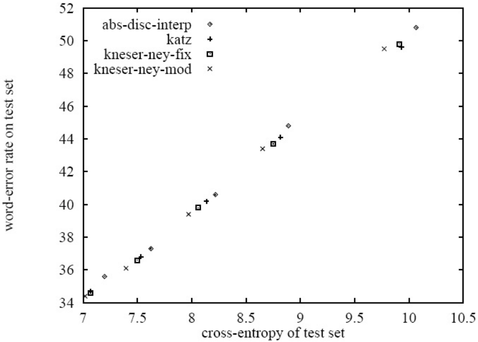
Other issues: How good is perplexity/entropy at predicting WER?

Recall: Different model types and domains [Chen+ 1998]



Now how good is perplexity/entropy at predicting WER?

Same model type/domain, different smoothing [Chen & Goodman 1998]:



Note: Different papers report different results on perplexity vs. WER... Bottom line: Take perplexity results with some caution!

Other issues: Language model weight

- A weight is typically applied to the language model: $\mathbf{w}^* = \operatorname{argmax}_{\mathbf{w}} p(\mathbf{w})^\alpha p(\mathbf{O}|\mathbf{w})$
- Empirically, this improves performance, typically with $\alpha \in [10, 20]$; why?
- One theory: This is a fix to our modeling errors
  - We don't have "correct" estimates of $p(\mathbf{w})$ and $p(\mathbf{O}|\mathbf{w})$
  - E.g., acoustic model typically assumes observation frames are independent given the states
  - Language model makes a similar locality assumption, but over a larger span of the observations
  - $\Rightarrow$ acoustic model gets too much "say" in the output

Other issues: Language model weight

- Another theory: Acoustic model probabilities have higher dynamic range
- E.g., for continuous digits, $|\log p(\mathbf{O}|\mathbf{w})| \approx 1000, |\log p(\mathbf{w})| \approx 20$
- This effect is also noted for transition probabilities vs. observation density in HMMs