An Empirical Study of Smoothing Techniques for Language Modeling

Stanley F. Chen

Joshua Goodman

Why smoothing in language modeling?

- Language Model – Attempts to reflect the frequency with which each string s occurs as a sentence in natural text.

$$P(s) = \prod_{i=1}^{l} P(w_i|w_1^{i-1}) \approx \prod_{i=1}^{l} P(w_i|w_{i-n+1}^{i-1})$$

Using relative frequencies as a way to estimate probabilities is one example of the technique of Maximum Likelihood

$$\begin{aligned} P_{\mathrm{ML}}(w_i|w_{i-1}) &= \frac{P(w_{i-1}w_i)}{P(w_{i-1})} \\ &= \frac{c(w_{i-1}w_i)/N_S}{c(w_{i-1})/N_S} \\ &= \frac{c(w_{i-1}w_i)}{c(w_{i-1})} \end{aligned}$$

Why smoothing in language modeling?

Problem: What is the MLE for (the | burnish)?

If the bigram "burnish the" doesn't appear in our training corpus then P(the| burnish) = 0. But no word sequence should have 0 probability, this can't be right.

Furthermore, MLE produces poor estimates when the counts are non-zero but still small

Solution: Smoothing

Comparing Smoothing Techniques

We now know that we need to apply smoothing to our language model. But there are many smoothing techniques, which one should we use?

Previous only a small number of methods have been compared (usually 2) on a single corpus using a single training data size.

Goal of Chen and Goodman: An extensive comparison of multiple smoothing techniques on various corpora on many training sizes for both bigrams and trigrams. Also introduce two new smoothing techniques

Criteria for judging smoothing techniques

Entropy(X): The expected number of bits needed to encode a randomly drawn example from X. Informally, entropy can be thought of as a mathematical measure of information or uncertainty.

In general assign short encodings for more probable events and longer encodings for less probable events. If we can encode with fewer bits this implies less uncertainty and more information
But what if we don't know the probability distribution p that generated some data? Use Cross Entropy

$$\frac{1}{N_T}\sum_{i=1}^{l_T} -\log_2 P_m(t_i)$$

Cross entropy is an upper bound on the entropy. For any model m: H(p) <= H(p,m). Between two models m1 and m2 the more accurate model will be the one with the lower cross-entropy.

Smoothing techniques

- A simple smoothing technique: Add 1

$$P_{+1}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)+1}{c(w_{i-1})+|V|}$$

We simply pretend that each unique bigram appears once more than it actually did. We do this by adding one to each count before normalizing into probabilities

|V| stands for vocabulary size, which is the total number of word types in the language. Since we add one for each word type, we've effectively added |V| bigrams starting with wi-1 to our corpus, so we must add |V| when normalizing

Solves 0 bigram probability problem, but doesn't work well in practice. Same probability for (the | burnish) as (thou | burnish) if both originally had P(0).

A slightly more complex technique

- Interpolation

$$P_{\text{interp}}(w_i|w_{i-1}) = \lambda P_{\text{ML}}(w_i|w_{i-1}) + (1-\lambda)P_{\text{ML}}(w_i)$$

Allows us to take into account the unigram probabilities. Bigrams involving common words are assigned higher probabilities

Smoothing techniques used in practice

- Additive Smoothing

$$P_{\text{add}}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i})+\delta}{c(w_{i-n+1}^{i-1})+\delta|V|} \qquad (2)$$

Instead of adding 1, we'll add delta. A problem with add1 smoothing, besides not taking into account the unigram values, is that too much or too little probability mass is moved to all the zeros by just arbitrarily choosing to add 1 to everything. By adding delta we can fix this problem.

Jelinek and Mercer

- Use linear interpolation

$$P_{\text{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} P_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}}) P_{\text{interp}}(w_i|w_{i-n+2}^{i-1})$$

Intuition:use the lower order n-grams in combination with maximum likelihood estimation. The probability for ever n-gram is estimated using a (n-1)-gram.

Training of the lambdas should be done on data disjoint from the data used to estimate maximum likelihood. Two methods for this.
Held-out interpolation: Reserve a section of the training data for each.
Deleted interpolation: Rotate different parts of the training data for each and then average the results

Jelinek and Mercer Smoothing

- Even with held-out interpolation or deleted interpolation you never have enough data to train every lambda value.
- Solution: Use bucketing
- Bucketing: Divide into disjoint groups where each group is characterized independently through a set of parameters and every group receives the same value. For Jelinkek and Mercer we divide into groups based on the count.

Good Turing Estimate

- Not used directly for smoothing, but combined with other methods

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

States that an n-gram that occurs r times should be treated as if it has occurred r* times. Nr is the number of n-grams that occur exactly r times in the training data

Intuition: Consider an n-gram that has occurred 0 times. When it does occur it will be the first time we see this new N-gram. So the probability of a zero frequency N-gram can be modeled by the probability of seeing an N-gram for the first time. Good Turing builds on this intuition to allow us to estimate the probability mass assigned to n-grams with lower counts by looking at the number of n-grams with higher counts.

Katz smoothing

Katz smoothing on a trigram model

$$P_{Katz}\left(w_i \mid w_{i-2}^{i-1}\right) = \begin{cases} C\left(w_{i-2}^{i}\right)/C\left(w_{i-2}^{i-1}\right), & \text{if } r > r_T \\ d_{r,3}C\left(w_{i-2}^{i}\right)/C\left(w_{i-2}^{i-1}\right), & \text{if } 0 < r \le r_T \\ \alpha\left(w_{i-2}^{i-1}\right)P_{Katz}\left(w_i \mid w_{i-1}\right), & \text{if } r = 0 \end{cases}$$

Intuition: Combine Good-Turing with interpolation. Outperforms Good-Turing by redistributing different probabilities to different unseen units. (the | burnish) would have higher probability than (thou | burnish)
Models are defined recursively in terms of lower order models.

Church and Gale

- Combine Good-Turing estimate with bucketing.
- Like Kats models are defined recursively in terms of lower-order models
- Each n-gram is assigned to one of serveral buckets based on its frequency predicted from lower-order models

$p_{ML}(w_{i-1})p_{ML}(w_i)$

For instance, the value of this would determine into which bucket a bigram falls

- Good Turing estimation is performed within each bucket

Novel Smoothing Techniques

- Average-Count
- Variation of Jelinek-Mercer, the criteria for bucketing has been changed. Instead of bucketing based on the count, they bucket based on the number of counts per non-zero element
- Intuition: The less sparse the data the larger lambda should be. The more accurate counts we have, the more trustworthy the n-gram is, and the higher we can make lambda. The count propposed by Jelinek-Mercer generally corresponds to less sparse distributions but ignores the allocation of counts between words. The average number of counts per word gives us a better correlation with sparseness than the total count for all the words.

$$\frac{c(w_{i-n+1}^{i-1})}{|w_i : c(w_{i-n+1}^{i}) > 0|}$$

Novel Smoothing techniques

- One count

$$P_{\text{one}}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) + \alpha P_{\text{one}}(w_i|w_{i-n+2}^{i-1})}{c(w_{i-n+1}^{i-1}) + \alpha}$$

Alphas represents the number of counts being added to the distribution with the new counts distributed as in the lower order distribution.

Good-Turing suggests that the number of these extra counts should be proportional to the number of words with exactly one count in the given distribution. Alpha is the number of words with one count plus some scaling by constants.

$$\alpha = \gamma\,[n_1(w_{i-n+1}^{i-1}) + \beta]$$

Data

- Used data from treebank and TIPSTER corpora:
- Brown Corpus - Treebank
- Associated Press - TIPSTER
- Wall Street Journal - TIPSTER
- San Jose Mercury News –TIPSTER

Division of Data

- Three segments of held out data
- One segment of training data
- One held out segment used for performance evaluation, other two for optimizing parameters

Implementation

- Baseline Smoothing: Jelinek-Mercer where all lambdas are equal to a single value for each n.
- Additive Smoothing: Two version. Version 1 delta = 1. Version 2 delta allowed to vary.
- Katz Smoothing: Use a different k for each n>1. And smooth the unigram distribution with additive smoothing
- Church Gale Smoothing: Bucketing done similar to Jelinek and Mercer. Had to extend the smoothing to trigrams while original paper only described bigrams. Some ambiguity here.

Implementation

- Jelinek-Mercer Smoothing: 2 versions, that differed by what data is used to train the lambdas. One trained using held-out interpolation and the other trained using relaxed deleted interpolation.
- Average-Count: Identical to interp-held-out except with the novel bucketing scheme
- New-one-count: Varies the constant parameters that act on alpha for each n.

Parameter optimization

- Parameters were chosen to optimize the cross-entropy.
- Searched only those parameters that were found to affect performance significantly
- Optimal values searched for using Powell's search algorithm
- For Katz and church-gale parameter search wasn't done for training sets over 50,000 sentences due to resource constraints, but were manually extrapolated

Results

- Poor performance from additive smoothing
- Consistently strong performance from Katz and interp-held-out.
- Church-Gale performs poorly except on large bigram training sets where it performs the best
- Novel methods perform well across training set sizes and are superior for trigram models.
- One-count yields marginally worse performance than average-count but is extremely easy to implement.

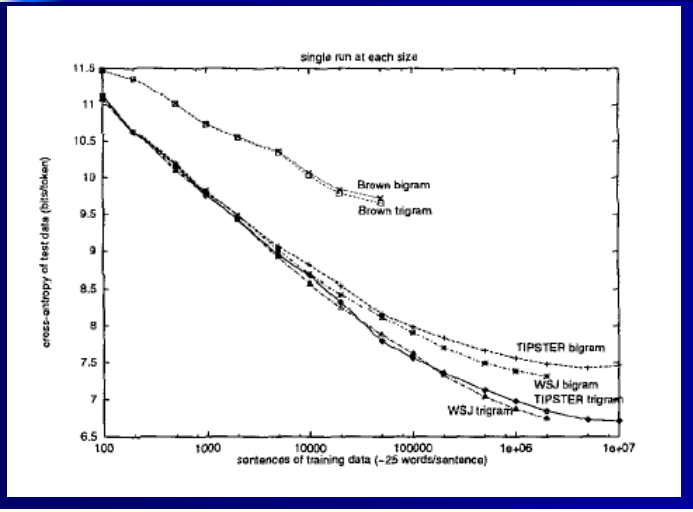
Results

- Interp-held-out significantly outperforms interp-del-int, however deleting larger chunks of words might yield better performance
- Poor parameter settings can greatly hurt the performance on Katz and new-avg-count

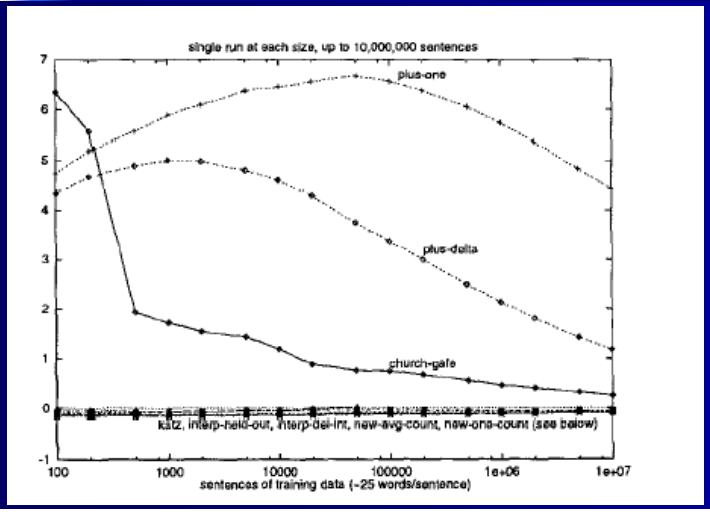
Results

- Performance relatively consistent across corpora but varies widely with respect to training set size and n-gram order

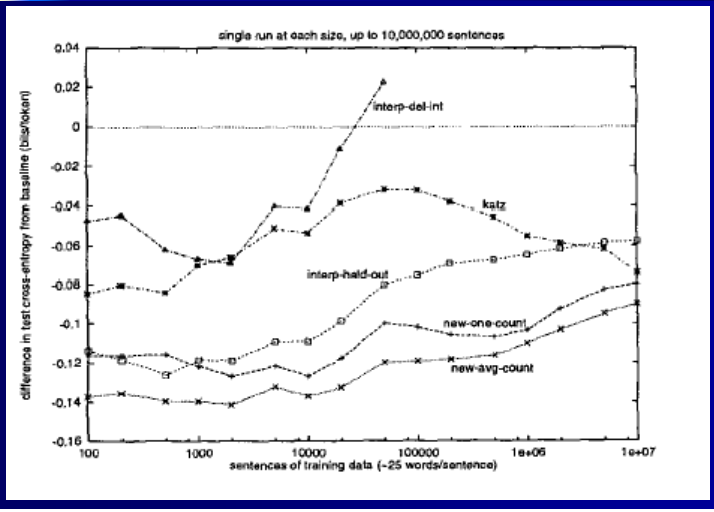
Baseline Results



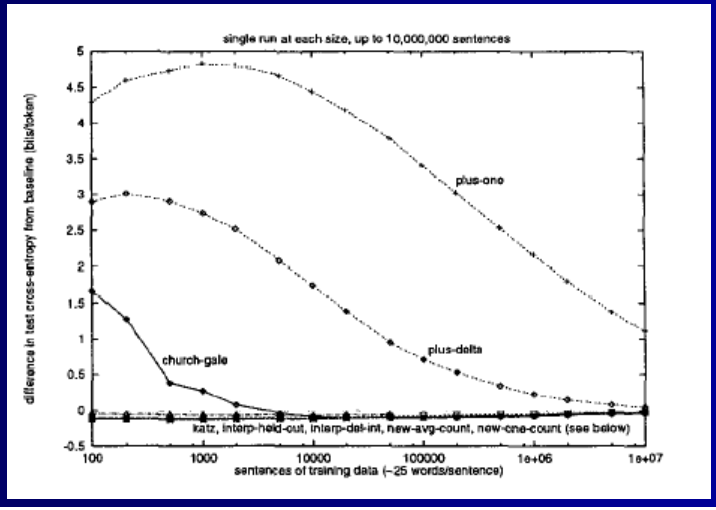
Trigram model performance



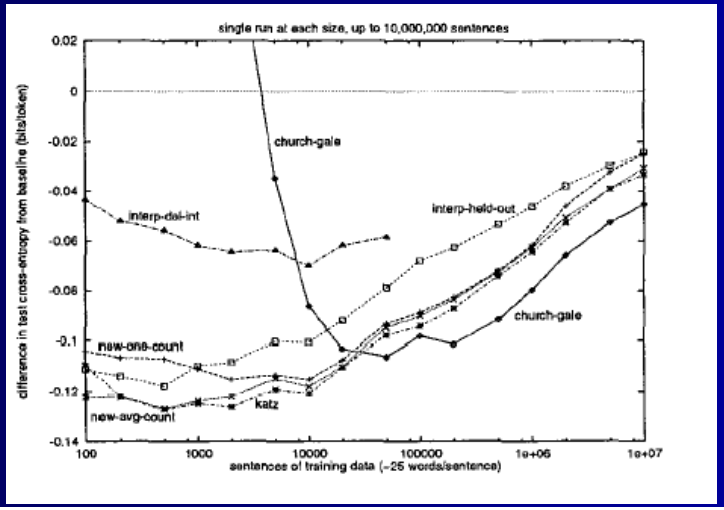
Trigram model performance



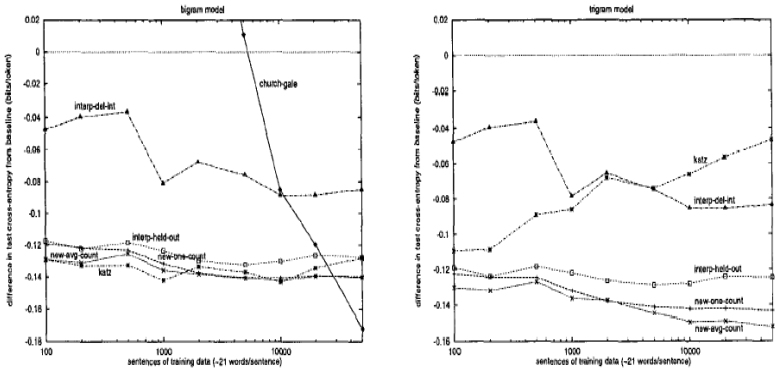
Bigram model performance



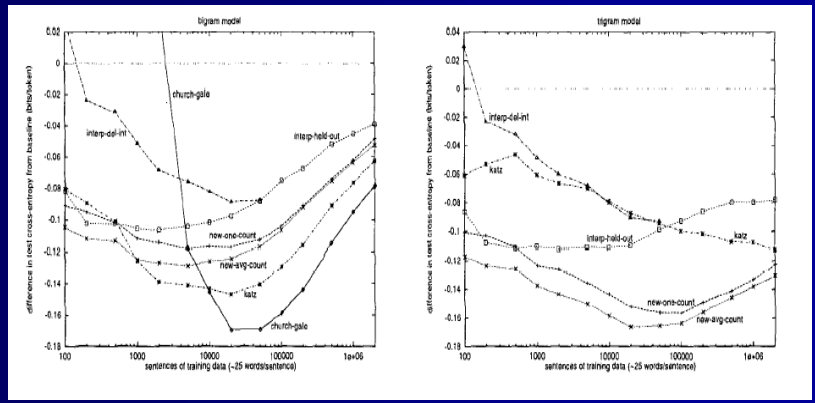
Bigram Model Performance



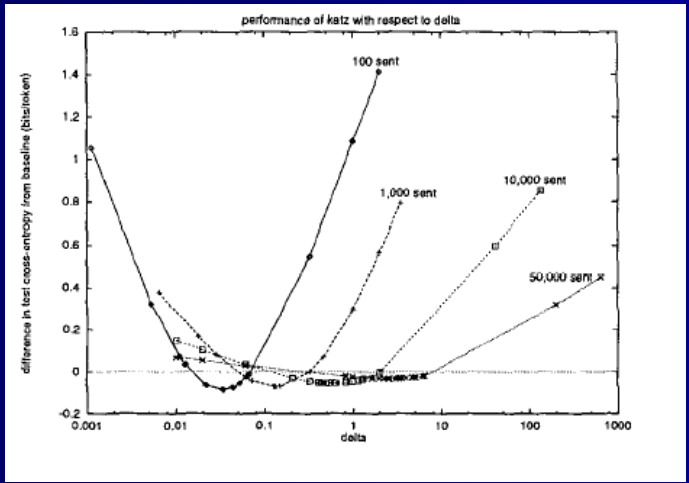
Performance on Brown Corpus



Performance on WSJ corpus



Performance of Katz with respect to delta



Performance of new-avg-count with respect to Cmin