Analysis of Smoothing Techniques in Language Model Synthesis

Timothy Liu
University of California at Berkeley
CS294-19
timothytliu@berkeley.com

ABSTRACT

This paper presents various implementations of language models differing in smoothing techniques from the most basic to the most widely used. The purpose of these techniques will be presented along with test results detailing their performance. Statistical analysis of such smoothing techniques will center on evaluating errors encountered and how they were corrected. The analysis does not focus on choosing the best, but on which methods to optimize performance work best. Finally, an analysis of the limitations of language models in minimizing errors will be presented.

General Terms

Language Models, Probability Distributions, Smoothing

Keywords

Kneser-ney, Good-Turing, perplexity, WER

1. INTRODUCTION

Information processing and understanding the human language serve as the basis for language modeling. However, language does not conform to specific standards or rules but rather increases in complexity especially in the context of translation. Therefore, statistical language models enable computer scientists to represent words, phrases, and sentences as probability distributions to simplify and accurately draw inferences about language.

Three aspects of statistical language models serve as this paper's measurements for accurately comparing various implementations of language models.

Perplexity - Given a language (probability) model, how confident this model predicts an unknown sample sentence is the perplexity calculated by using log-probabilities.

$$2^{-\sum_{i=1}^{N} \frac{1}{N} \log_2 q(x_i)}$$

q is the *probability distribution*

Two categories of WSJ Perplexity and HUB Perplexity are presented in this paper. WSJ means a perplexity generated from news text from the Wall Street Journal(Newswire), while HUB contains news from a different source. The subtle difference is that the HUB vocabulary does not contain any unknown words while the WSJ vocabulary does.

Word Error Rate (WER) - Given a language model, how close sequences of words generated by this model are to the reference sequences.

$$WER = \frac{S + D + I}{N}$$

S is the # of substitutions, D is the # of deletions, I is the # of insertions, and N is the # of words.

Coherency - Given a language model, how well the generated sequences of words match the English language whether the scope of the model is general or specific.

EX: [house, of, representatives, has, sued, to, stop]
EX: [october, it, buy, 's, through, for , percent, expects, the]

2. SYNTHESIS OF LANGUAGE MODELS

An Empirical Unigram with Add-One Smoothing was the given baseline language model for testing purposes. An Empirical Bigram and Trigram were the first language models to be synthesized using a similar implementation of Add-One Smoothing from the baseline. These models were chosen to show the difference between a growing order in language models and whether they would behave according to convention of a lower perplexity.

2.1 Empirical Language Models

The implementation of a bigram and trigram encountered one large obstacle: dealing with unknown words. The first method of implementation focused on tallying unigrams, bigrams, and trigrams that appeared once and distributing the counts to UNK tokens, representing unknown words. A separate Empirical Unigram model accounted for UNK tokens, letting UNK appear.

However, for bigram and higher, a problem of distributing the counts between UNK|KNOWN, KNOWN|UNK, KNOWN|KNOWN and UNK, KNOWN|UNK and KNOWN, etc occured. If not taken care of correctly, after generating an UNK token, the possible words that would occur would be UNKNOWN and would potentially loop forever. The loop structure to generate the next word is as follows:

```
double sample = Math.random()
double sum    = 0.0;
for (String word : bigram.getCounter(prev).keySet()) {
            sum += getWordProbability(prev);
            if (sum > sample) return word;
}
return UNK;
```

Table 1: Perp. and WER for Empirical Models

Model	WSJ	HUB	WER	Example
Baseline	1587.6	1541.2	0.091	[close, businesses, far, in, base, UNK, yes, state, new, the, UNK]
Bigram	1106.5	1106.3	0.076	[the, lure, of, the, UNK, funds, now, drexel, UNK, budget, process]
Trigram	18534.0	24018.5	0.098	[e., s., p., n., 's, UNK, UNK, said, that, UNK, a, tax, credit, hundred, million, dollars]

If given previous history, whether it is unigram, bigram, or trigram, summing over all possible probabilities of words that follow must be checked. This total should sum up to one because UNK tokens have already been inserted and because a distribution over the previous history should accurately reflect the training data and sum to one.

A trend regarding the generated sentences began to appear. The difference between Unigram, Bigram, and Trigram is the increase of *Coherency*. **See Table 1**. When generating sentences, a trigram is more likely to generate an 'a' after [u., s.] whereas the unigram would choose a high probability word like 'the.' However, higher order models lose generality and overfit the training data because it is only assigning probabilities to the phrases in training.

Although the *perplexity* and *WER* decreased for the bigram, it increased again for the trigram, showing that add-one smoothing did not function correctly and neither did adding the UNK tokens. Adding UNK tokens did not accurately represent the different probabilities of words as low perplexities occurred, but the distribution summed to greater than one. The results show a model that summed to one, but lacked a good technique to deal with unknown's. The changing *perplexity* shows that it is not accurately estimating the likelihoods of words as they appear, and low probabilities that multiply together produce high *perplexity's*. It is more likely to see new and unknown words, and thus miniscule probabilities keep multiplying together.

2.2 Smoothing Techniques in Models

To better handle the higher order models' sparsity without forsaking *coherency*, different methods of smoothing were implemented. Several smoothing techniques exist such as additive-smoothing, Jelinek-Mercer, Good-Turing Estimate, Katz smoothing, Witten-Bell smoothing, and Kneser-Ney. Smoothing techniques allow for a more accurate representation of unknown words and their probabilities. Good smoothing algorithms decrease the spread of the probability distribution while still allowing the distribution to add to one. In this way, the branching factor of the model or *perplexity* decreases.

Of the many smoothing techniques an N-gram additive smoothing and a bigram Good-Turing Estimate model were implemented as the next stepping stone. These models utilized a better technique to handle unknown words and form better estimates. Furthermore, the models would serve as valuable comparison and could provide valuable results up to as many as 7-gram.

Table 2: Perp. and WER for Smoothing Models

Model	WSJ	HUB	WER	Example
2-gram	867.7	704.7	0.076	[total, reduction, credits, result, of, macmillan, 's, board, will, some, see, charleston]
3-gram	1052.4	908.5	0.074	[it, pays, twenty, three, cents, a, share]
Good	693.0	835.6	0.080	[slightly, smaller technology, stocks were, a, weak]

The Good-Turing Estimate portrayed unknown words as the number of words that appeared once and by reweighting the counts, a more accurate representation of words resulted. The probability of an unknown word is large especially in a higher order-context.

The Good-Turing Estimate utilizes the principle that the number of unknowns is approximated by the number of words that appear once. This is extrapolated to all words of counts r in the reweighting: $r^* = (r+1) * \frac{n_{r+1}}{n_r}$ while maintaining the same overall mass.

Although Good-Turing had good *perplexities*, by itself, it was not able to produce a good *WER*. This showed that at handling UNK tokens Good-Turing could provide a reasonable probability to deal with word sequences by reweighting. However, for choosing a specific word over another by probabilities, the choices were just wrong and obvious for any human reader. **See Table 2**.

GUESS: AM: -9.66E02 LM: -2.51E01 Total: -9.91E02 [j., could, n't, be, reached, for, comment]
GOLD: AM: -9.67E02 LM: -2.83E01 Total: -9.95E02 [she, could, n't, be, reached, for, comment]

The N-gram additive smoothing model served multiple purposes, as it allowed for a broad-spectrum analysis of higher order tests with better smoothing than add-one. The algorithm to generate the recursive N-gram model is as follows:

$$p_{add}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) + \Phi_k * p_{add}(w_i|w_{i-n+2}^{i-1})}{\Phi_k + \sum_{w_i} c(w_{i-n+1}^{i})}$$

The additive-smoothing algorithm handles unknowns by using linear interpolation to back off to lower order models when $c(w_{i-n+1}^{i})$ equals zero. By recursively calling itself until the base case $\frac{1}{|V|}$, this algorithm allows for both a hyper-parameter, Φ_k, to optimize for *perplexity* or *WER*, and an elegant way to take care of unknown's. As more unknowns arise in higher-orders, their probabilities are no longer impossibly small, but higher with linearly interpolation.

Table 3: Perp. and WER for Kneser-Ney

Model	WSJ	HUB	WER	Example
KN Bigram	408.4	413.0	0.073	[their, own, people these, of, american, it, 's, of, money, laundering, aichi, committed, to, clean, forty, three, million, shares]
KN Trigram	352.7	376.3	0.074	[for, instance, may, goods, and, automotive, illegal, acts, 's, weak, its, durability]

Figure 1: Word Error Rate

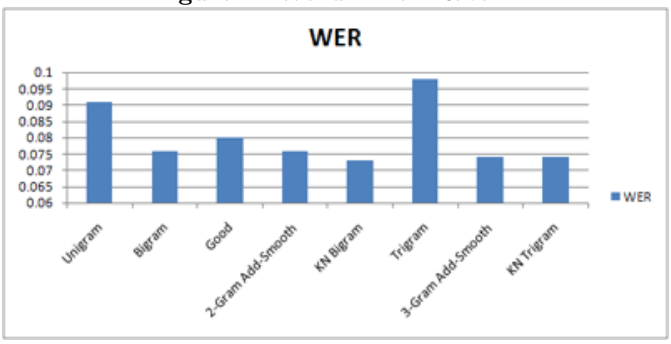
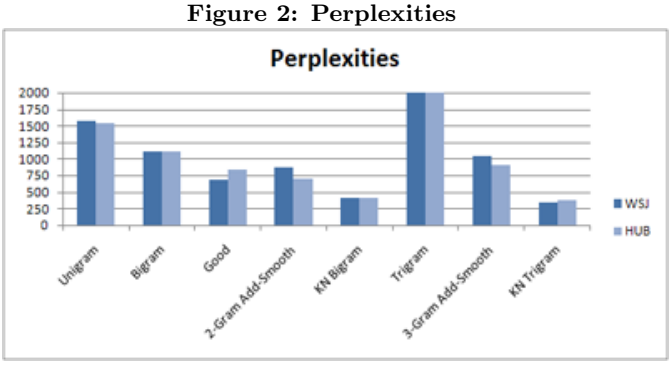


Figure 2: Perplexities



The improved *perplexity* and WER is coming from a better model of both representing unknowns in a dynamic manner. Linear interpolating in the N-gram case shows that words can have multiple contexts extending from the previous word to the entire history. The word error rate is decreased and with simple optimization of Φ_k, the *perplexity* is improved. This mix of weights allows customization for other possible features to better represent sequences of words that the current model is bounded by like correct grammar or context. Some words look the same, have different meanings, but more importantly occur in different frequencies and can only be differentiated by context.

To tune the parameter of Φ_k, a simple algorithm can be used to minimize the *perplexity*:

$$argmax_{\Phi} \sum_{w} p(w|w_{-1}, \Phi_k) * \log p(w_{-1}, w|\Phi_k)$$

Although in Chen and Goodman[1], Φ_k is bounded by 0 and 1, the optimal value tuned on the validation or held out data is always the upper threshold up until over 50. Therefore, a value of 5 was chosen as a compromise between 1 and 50. The results of this are shown in the table.

2.3 Kneser-Ney

The next language model implemented was a Bigram Kneser-Ney with Linear Interpolation and a Trigram was later implemented, but not fully tested. The algorithm to generate this is similar to the additive-smoothing one:

$$p_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{maxc(w_{i-n+1}^{i}) - D, 0 + D * N_{1+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})}$$

$$* \frac{maxc(w_{i-n+1}^{i}) - D, 0 + D * N_{1+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})} * \frac{1}{N_{1+}(w_{i-n+1}^{i-1}\bullet)}$$

The Kneser-Ney model was able to minimize both *perplexity* and *WER* due to absolute discounting and linearly interpolating lower-order models for unknown words. **See Table 3**. The discount for the Kneser-Ney model is approximated from Chen and Goodman[1] to be $D = \frac{r_{n+1}}{r_{n+1} * r_{n+2}}$.

The reweighting of counts fixes the overestimation of new words while letting the probability distribution still sum to one. Although not entirely sure as to why Kneser-Ney works the best, the linearly interpolation due to lower-models helps to increase the probability of words that are successful in completing sentences like in the bigram, unigram, and uniform distribution.

[1]http://cs.berkeley.edu/∼klein/cs294-5/chen_goodman.pdf

3. ERROR ANALYSIS

The analysis of *perplexity*, *WER*, and *coherency* between the various models began with a cursory comparison of numbers and their trends, which can be seen on **Figures 1 & 2**. However, an in depth look at the errors encountered and whether or not they could be corrected serve as a better model for testing.

3.1 Acoustic Model Example

The situation that fully tests the performance of these models is to see how they behave in real life. This is tested by having an acoustic model generate sentences with all possible acoustic variations as if recorded. The language models will then return the sentence with the highest probability. This is more valuable than the *WER* and *perplexity* because errors that occur can now be classified, evaluated, and possibly even corrected. There were several types of errors encountered.

3.2 Errors Encountered and Corrected

As previously mentioned specific word errors such as abbreviation or specific coherency mistakes were encountered in the baseline model. However, as the order of the model increased, this mistake was never made again. The specificity of the models increased as the order increased and was more

'fitted' towards the training data. It would be interesting to test multiple abbreviation combinations such as u.s.a versus u.s.s.r.

Baseline: [adds, that, the, four, be, believing, members, who]
Bigram: [adds, that, the, four, b., w., a., members, who]
CORRECT: [adds, that, the, four, b., w., a., members, who]

Another kind of error similar to this that occurred was that of unknown word combinations. In the baseline case, the error does not exist because the frequency of 'sony' to 'sunny is more than tenfold. However, once the previous history becomes introduced, the phrase 'no sony' does not exist in the vocabulary and a higher-order model hurts rather than helps. The Good-Turing, Bigram, Trigram, additive-smoothing Bigram and Trigram all fail. The specific solution is the combination of accurate representation of unknown word combinations and the ability to back off to lower order models. This is found in the kneser-ney model for both bigram and trigram. Other cases exist where higher-order models fail and only baseline, KN, and additive-smoothing will be able to correctly identify the correct word.

Baseline: [were, no, sony, buy, orders, and, tokyo]
Good-Turing: [were, no, sunny, buy, orders, in, tokyo]
BiGram: [were, no, sunny, buy, orders, and, tokyo]
TriGram: [were, no, sunny, buy, orders, and, tokyo]
2-Gram: [were, no, sunny, buy, orders, and, tokyo]
3-Gram: [were, no, sunny, buy, orders, and, tokyo]
2-KN: [were, no, sony, buy, orders, and, tokyo]
3-KN: [were, no, sony, buy, orders, and, tokyo]
CORRECT: [were, no, sony, buy, orders, in, tokyo]

There is a key relationship between *perplexity* and WER that can be seen from these examples. Though certain models may have an optimal complexity, it does not produce a correct result. Perplexity only optimally eliminates choices and smooths probabilities estimating with confidence, while WER focuses on the best way to produce correct comparisons between sequences of words. One example is that of the n-gram additive smoothing, where the perplexity can range from 1000 to 600 with a different hyperparameter but the WER changes less than .001.

3.3 Errors Encountered and Uncorrectable

Despite the low WER and perplexity, Kneser-Ney as well as the other models cannot entirely fix errors based on the context of a sentence. If the context can be found within the N-gram long sentence, then it is possible, but a model loses generality the more it extends.

3-Gram: [that, might, lift, the, yen, which, has, been, moving, in, tandem, with, the, market, recently]
3-KN: [that, 's, might, lift, the, yen, which, has, been, moving, in, tandem, with, the, market, recently]
CORRECT: [that, might, lift, the, yen, which, has, been, moving, in, tandem, with, the, mark, recently]

The probability of 'the market recently' is a fairly likely group of words, but the sentence is about exchange rates and so the sentence completely misses the comparison to the German Mark and its reference the Japanese Yen. A possible solution to this problem would be an added linearly interpolated weight that classified sentences and gave a score on how well the sentence matched the previous sentences' class. To generalize the error, these language models have extremely difficulty dealing with the disambiguation of words in language or accurately ascertaining the senses of words.

The langauge models although dealing progressively better with the concept of unknown words still have errors when trying to estimate a word that has no information. It is not simply an unknown word, but one that has no related context in the sentence.

3-Gram: [manufacturers, have, so, many, orders, already, in, the, pipeline, says, another, u., s., bank, manager, in, cell]
2-KN: [manufacturers, have, so, many, orders, already, in, the, pipeline, says, another, u., s., bank, manager, in, cell]
3-KN: [manufacturers, have, so, many, orders, already, in, the, pipeline, says, another, u., s., bank, manager, in, cell]
3-KN: [manufacturers, have, so, many, orders, already, in, the, pipeline, says, another, u., s., bank, manager, in, cell]
CORRECT: [manufacturers, have, so, many, orders, already, in, the, pipeline, says, another, u., s., bank, manager, in, seoul]

As 'seoul' is not in the vocabulary and the sentence has nothing related to it, few options remain. However, this error can be generalized to an error that points back to the purpose of language modeling. A flaw in processing any kind of information is that the lack of prior information hampers even the best models. In the same manner, these models and even more advanced models are capped by their training and validation corpora. This decidedly affects WER because if the models do not know the word, most likely a replacement word will be used. This is verified by results from the data where the training set collection was decreased by half, three-quarters, etc. The WER kept increasing despite using the most optimal techniques out of these models. Therefore, the best solution for almost all errors is to have the largest corpora available.

4. CONCLUSION

There are many ways to make language models and various implementations of the most important aspect: smoothing. Smoothing and language models in general must address issues of sparsity, specificity vs. generality, and the ever looming problem of cataloging all the text in the world as data. Despite this, results have shown that with a small margin of error, models such as Kneser-Ney with linear interpolation work well enough for use in the real world.

5. REFERENCES

[1] S. Chen and J. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. Harvard University. August 1998. Center for Research in Computing Technology.