Experimental Exploration of Trigram Kneser-Ney Language Models

Chirag Nagpal
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania
chiragn@cs.cmu.edu

Abstract

Kneser-Ney is a popular technique for building Ngram language models. In this paper we explore the performance of an exact Kneser-Ney with discounting Language Model on a machine Translation task and determine its sensitivity to parameters like Training Data size, discounting. We also explore techniques to improve decoding and storing such a language model.

1 Introduction

Kneser-Ney language model was first proposed by (Kneser and Ney, 1995) and further explored in (Chen and Goodman, 1999) aimed to improve language model estimation by incorporating discounting along with back-off.

For all our experiments, we train a Trigram Kneser-Ney language model on an English data set consisting of around 9 Million sentences. In-order to measure the performance of the Language Model, we employ the LM for a French to English Machine Translation task consisting of 2000 French Sentences and estimate the BLEU Score. Detailed corpus statistics are presented in.

There are various techniques that can be carried out in order to improve the computational overhead both in terms of memory required to build and store the lanaguage model, as well as the time required to carry out decoding. We have explored two such techniques in this study, the use of caching as described in (Pauls and Klein, 2011) and rank tables to store Ngram counts.

2 Corpus Statistics

Total Number of Trigrams	41736000
Total Number of Bigrams	8374231
Total Number of Unigrams	495172

Table 1: Corpus Statistics

3 Training Dataset Size

It is found that Kneser-Ney Model, is sensitive to training dataset size. In general, in our experiments, we found that the model prediction improves as the training dataset size increases.

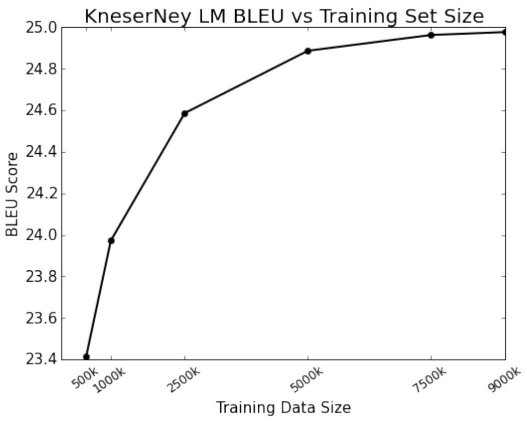

Figure 1: The Performance of the LM increases with dataset size.

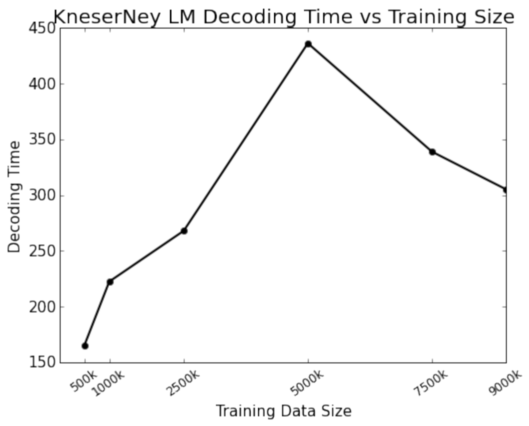

Figure 2: Time taken to Decode LM vs. dataset size.

It is interesting to note how the decoding time changes with change in training dataset size. From

Figure 2 it is clear that the decoding time increases with the increase in training data and then decreases. This behaviour can be explained easily by the use of cache table in our model.

For models built on small datasets, large number of trigrams are unseen, and the model backsoff to a lower order model, the decoding for which is fast. One the other hand as the training dataset increases, the model not only has to take into account the higher level ngram model, but also recursively decode lower level model, increasing the decoding time. As the dataset size increases, so does the repetition of certain trigrams, thus the model would have higher number of cache table entries and the probability of a cache miss will decrease, speeding the decoding time for the model.

4 Discounting

Discounting refers to the amount of probability mass subtracted from the higher order language model and redistributed to the lower order language models. Generally, discount, d is kept $0 < d < 1$ discount value of 0.75 is considered to work well on most datasets.

A very low discount value, would be suboptimal, as it would effectively reduce the Kneser-Ney model to a Trigram Language model, on the other hand very high discounting would reduce it to a lower order model.

In Figure 3 we experiment with different discount values and determine the ideal discount value with respect to the BLEU score.

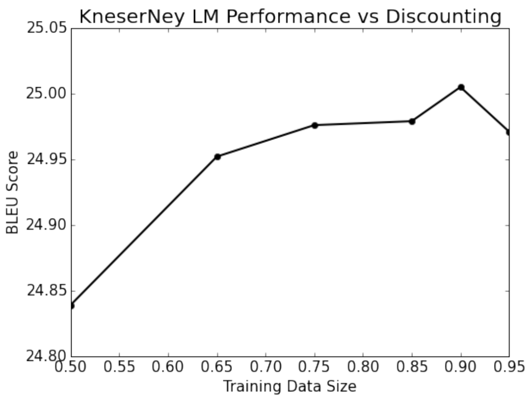

Figure 3: Time taken to Decode LM vs. dataset size.

From the figure it is clear that the performance of the model is the best for a discount of 0.9, after this, the performance decreases. It appears as if the ideal value of discount heavily depends on the given dataset. The nature of this parameter is such that an appropriate dicount value, can only be determined experimentally by estimating LM performance on some ground truth.

5 Cache Table

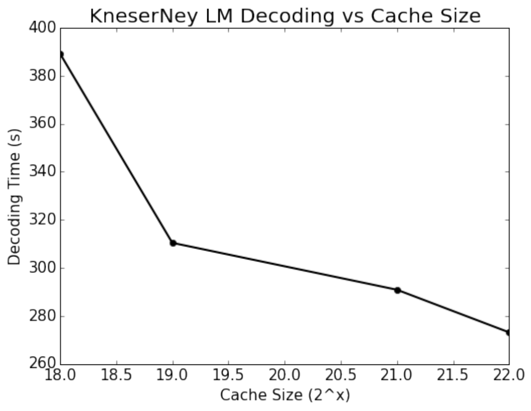

Figure 4: Time taken to Decode LM vs. Cache size.

Our exact Kneser-Ney model takes around 556 seconds to decode the Trigram probabilities for the 2000 Test Sentences. We experiment with a cache table in order to speed up the decoding process. Caching helps because instead of recursive lookup into multiple hashtables, caching allows the LM access to the Trigram probability in a single lookup.

In our cache table implementation we find that a table of size 2**21 performs reasonably well, and reduces the lookup time to 290 seconds, which is a speed up of a factor of ~ 1.9.

6 Rank Table

Inorder to reduce the memory overhead of storing the language model, we utilise a rank table to store the Trigram counts. In our experiments we found that while there are 41736000 unique trigrams, there are only about 10240 unique trigram counts.

Thus, we store the rank counts in an `int` array along with a mapping from the trigram encoding to the rank table, in another `int` array. This further reduces the memory consumption of our model by around $\sim$ 80 MB.

7 Conclusion

In this study we implement a full Trigram Kneser-Ney Model with discounting, and test its performance on a dataset. We tune parameters of our model in order to ensure maximum performance of our implemented model for the Machine Translation task.

We find that the model performs well with a discount value of 0.90, with a BLEU score of around, 25.00, incomparison a simple Trigram model had a BLEU score of only $\sim$ 22.34. On our testbed, decoding this model, takes around 556 seconds and a RAM of around 1.1 GB.

We further improve the decoding time by the use of a cache table of the size of 2**21. This speeds up the decoding by a factor of almost 2.

Inorder to reduce the memory footprint we implement a ranking table, and also cleverly use Java's `char` data type which allows us to store counts as a 16-bit unsigned value and reduce the overall memory footprint to about 960MB (**including the Cache Table**).

Acknowledgments

We would like to convey our gratitude to Professor Taylor Berg-Kirkpatrick, Wanli Ma and Kartik Goyal for all the help for this assignment.

References

Stanley F Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE.

Adam Pauls and Dan Klein. 2011. Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 258–267. Association for Computational Linguistics.