Implementation of Kneser-Ney Tri-gram Language Model

Wei Peng
University of Pittsburgh
wpeng1@andrew.cmu.edu

Abstract

we implement a Kneser-Ney trigram language model (Chen and Goodman, 1999) and test how well it works by incorporating the language model into an MT system and measuring translation quality. Our best model achieves a BLEU score of 25.008 with a memory usage of 1.1G and a decoding time of 176.269s.

1 Introduction

Kneser-Ney Language Model was first introduced by (Kneser and Ney, 1995), and further explored by (Chen and Goodman, 1999). In most cases, we just have few observations, thus to improve the estimates, we suppose to modify observed counts slightly and also allocate small probability to unseen n-grams. General approaches are **back-off**, **interpolation** and **discounting**. For Back-off, Trigram should be used if we have good evidence, otherwise Bigram and otherwise Unigram. For interpolation, we combine estimates from related denser histories to approximate counts of n-grams. For discounting, we assign probability mass for unobserved events by discounting observed events. (Chen and Goodman, 1999) combines those ideas into one algorithm. The recursive formula of Kneser-Ney is following:

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{max(c(w_{i-n+1}^{i} - d, 0))}{\sum_{v \in V} c(w_{i-n+1}^{i-1}, v)} + d \cdot \frac{N_{1+}(w_{i-n+1}^{i-1} \bullet)}{\sum_{v \in V} c(w_{i-n+1}^{i-1})} P(w_i|w_{i-n+2}^{i-1}) \quad (1)$$

where $P(w_i|_{i-n+2}^{i-1})$ is the interpolation back-off. (Chen and Goodman, 1999) explained out the motivation for KN smoothing clearly that "We should select lower-order distribution such that the marginals of higher order smoothed distribution should match the marginals of the training data." Based on 1, we have

$$P(w_i|w_{i-n+2}^{i-1}) = \frac{N_{1+}(\bullet w_{i-n+2}^{i-1})}{N_{1+}(\bullet w_{i-n+2}^{i-1} \bullet)} \quad (2)$$

1.1 Trigram Language Model

We provide the formulas for Trigram, Bigram and Unigram explicitly. See (Chen and Goodman, 1999) on page 16 for more details.

Trigram

$$P_3(w_3|w_1w_2) = \frac{\max(c(w_1w_2w_3) - d, 0)}{c(w_1w_2)} + d \cdot \frac{N_{1+}(w_1w_2\bullet)}{c(w_1w_2)} P_3'(w_3|w_2)$$

$$\text{where } P_3'(w_3|w_2) = \frac{\max(N_{1+}(\bullet w_2w_3) - d, 0)}{N_{1+}(\bullet w_2\bullet))} + d \cdot \frac{N_{1+}(w_2\bullet)}{N_{1+}(\bullet w_2\bullet)} P_3'(w_3)$$

$$\text{and } P_3'(w_3) = \frac{N_{1+}(\bullet w_3)}{N_{1+}(\bullet\bullet)}.$$

Bigram

$$P_2(w_2|w_1) = \frac{\max(c(w_1w_2) - d, 0)}{c(w_1)} + d \cdot \frac{N_{1+}(w_1\bullet)}{c(w_1)} P_2'(w_2)$$

$$\text{where } P_2'(w_2) = \frac{N_{1+}(\bullet w_2)}{N_{1+}(\bullet\bullet)}.$$

Unigram

$$P(w_1) = \frac{c(w_1)}{\sum_v c(v)}.$$

If w_3 is never seen before, It is better to assign a small probability on w_3, say $1/\sum_v c(v)$. This will definitely hurt the integrity of the probability distribution, but the effect can be negligible as we get more observations.

2 Implement Details

In the project, we implement a Kneser-Ney trigram language model and test how well it works by incorporating the language model into an machine translation system and measuring translation quality. The MT decoder takes a French sentence and attempts to find the highest-scoring English sentence. The translation model was already well-established and we are here to build the language model. The resulting translations are then compared to the human annotated reference English translations using BLEU score. Particularly, to build *KneserNeyLanguageModel*, we store $C(w)$ by `long[]`, $N_{1+}(\bullet w)$, $N_{1+}(w\bullet)$ and $N_{1+}(\bullet w\bullet)$ by `int[]`. And we build *BigramOpenHashMap* to store keys by `long[]`, and $C(ww')$, $N_{1+}(\bullet ww')$, and $N_{1+}(ww'\bullet)$ by `int[]`. Also, we build *TrigramOpenHashMap* to store keys by `long[]` and $C(w'w''w''')$ by `int[]`.

In those open address hashmaps, we use linear probing with *MAX_LOAD_FACTOR* being 0.7. The provided `Long.hashCode()` in java seems to slow down the implementation process, thus we use a different hash function `hash6432shift` provided by `chttps://gist.github.com/badboy/6267743`. We have d , the discounting factor, equal to 0.9. Finally, to calculate the *getNgramLogProbability*, we use different formulas for different size of n-grams according to the equations derived in subsection 1.1.

3 Experiment Results

First, we give a descriptive summary of n-gram counts by table 1.

Unigram	Bigram	Trigram
495172	8374230	41627672

Table 1: n-gram count

Also, the best model we obtained is given in table 2.

Memory Usage	BLEU	Decoding Time
1.1 G	25.008	176.269s

Table 2: Best Performance

4 Investigation & Error Analysis

4.1 BLEU vs. TrainingDataSize

It would be interesting to see how the training data size n influences the translation performance. We build models with training data of size of one million, two million up to ten million for comparison with $d = 0.9$. Figure 1 indicates that the translation quality is getting better as training data size grows. Remarkably, 1 million training samples already roughly achieves the best performance of Kneser-Ney trigram language model.

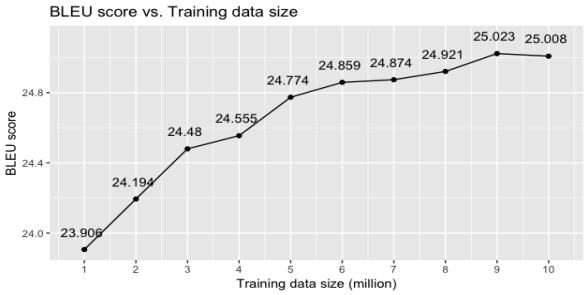

Figure 1: BLEU vs. Training Data Size

4.2 DecodingTime vs. TraingDataSize

We also recorded the decoding time under the previous setting. It is clear that the decoding time increases as the training data size increases. Since there are less unseen words, higher order n-gram probability needs to interpolate lower order n-gram probability, increasing the decoding time. Moreover, it takes more time to search in arrays with larger size.

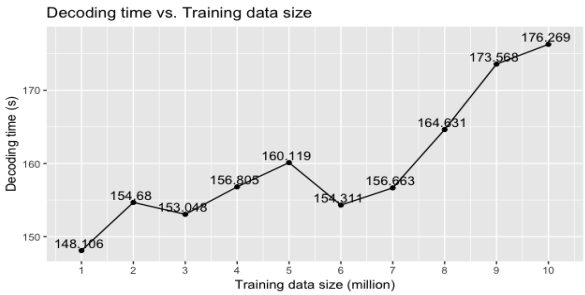

Figure 2: Decoding Time vs. Training Data Size

4.3 BLEU vs. Discounting

We also investigated how discounting effects the BLEU score. Figure 3 shows that the BLEU score increases steadily as d increases and achieves the maximum when $d = 0.9$.

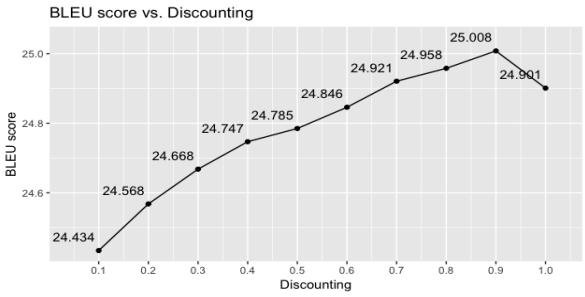

Figure 3: BLEU score vs. Discounting

Acknowledgments

The author would thank Taehee Jung for helpful conversation and feedback. The author also thank the online resources provided by the following links for inspiring on building efficient hashmaps.

- https://github.com/guilk/Algorithms_for_NLP
- https://github.com/tushardobhal/KneyserNey

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 *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184. IEEE.