CS 288: Kneser-Ney Language Modeling

Anting Shen
23566738
`antingshen@berkeley.edu`

1 Introduction

This report describes an implemenation of Kneser-Ney trigram language model. The model stores counts in optimized hashmaps and calculates probabilities at decode time. It is trained on a training corpus of nine million sentences, and tested as part of a machine translater. The model's implementation details, performance (BLEU, memory usage, decode speed), and areas for improvement follow.

2 Trading accuracy for memory

Most counts in the trigram counter are very small and waste space as they are allotted 32 bits. In the training data, only around 150 trigrams appeared more than $2^1$6 times. Because of this, I experimented with using 16-bit shorts to (no longer accurately) store trigram counts, and the results showed a 100MB reduction in memory usage. This was accompanies by a drop in BLEU from 24.8 to 24.7, which is a small amount to give up for the amount of memory saved. A small implementation was then tested to check for counter overflows and stop the counter at $2^1$6 instead of overflowing, which would bring the counts closer to the actual count that the short was not capable of storing. However, this proved insignificant both in terms of additional decoding time to perform such check as well as BLEU, most likely due to some trigram counts reaching as high as around $2^1$9, so using $2^1$6 instead of a random short made little difference in the calculations.

The same modification was then made to the two bigram fertility counters as the two bigram fertility counters each had less than 50 overflows. Running the test with these modifications, there was another 50-100MB drop in memory use with no drop in BLEU. But when the bigram counts array was switched to shorts, the BLEU dropped to 23.5, which is an unacceptable tradeoff (despite still being above the cutoff).

3 Additional potential improvements

An alternate solution to overflowing the short would be to store an overflow bit as the highest bit of the counter. The overflowed bits would then be stored in the hashmap under the negative of the original key (since the sign bit is unused with $\leq$ 21 bits per type in the trigram/bigram). This doubles the decode time and memory use of the ngrams that appear most frequently, but preserves the accuracy of the counter. Another implementation would be to store overflowed counts in a separate hashmap, and use the max short to signify overflow. The advantage of this method is that it allows 16 bits of counts before overflowing as opposed to 15.