IMPLEMENTATION OF WORD BASED STATISTICAL LANGUAGE MODELS

Frank Wessel, Stefan Ortmanns and Hermann Ney

Lehrstuhl für Informatik VI, RWTH Aachen, University of Technology,
D-52056 Aachen, Germany, Phone/Fax: +49 241 {8021616/8888219},
E-mail: wessel@informatik.rwth-aachen.de

Abstract. In this paper we present an efficient data structure for storing trigram, bigram and unigram counts. The amount of memory required has been reduced by 53% compared to straightforward approaches. The average access time for retrieving information from the data structure has also slightly been reduced. Based upon this special data structure we have implemented several types of language models and applied them to the North American Business (NAB '94) recognition task. We show that both, the perplexity and the error rate could be reduced compared to the official NAB '94 trigram language model.

1 INTRODUCTION

The main task of statistical language modelling is to provide a speech recognition system with the a-priori probabilities for a word sequence $w_1...w_N$. In order to be able to compute the widely used bigram and trigram language models, we have to count how often a trigram or bigram, i.e. a word triple or a word pair, has been seen in a training corpus. We can then compute the probability estimate for the trigram u, v, w as:

$$p(w|u,v) \cong \frac{N(u,v,w)}{N(u,v)} \quad ,$$

with $N(u,v,w)$ denoting the number of times the trigram u, v, w has occured in the training corpus. To overcome the well-known zero frequency problem, some sort of discounting must be applied to the relative freqencies. The words are usually replaced by word indices which correspond to their position in a lexically sorted vocabulary. Using this text representation, there are several approaches to compute the relative frequency of a trigram:

- The whole training corpus can then easily be stored in a one-dimensional array. Whenever the probability of a specific trigram is needed, its frequency can simply be calculated by counting the occurrences of the event in the corpus. Assigning two bytes for each word we need 480 MByte to store the whole NAB '94 Corpus consisting of about 240 million words. It is useful to introduce another array, in which the next occurence of a word in the training corpus is stored. The memory cost then rises up to 1.4 GByte.

Table 1. Statistics for the NAB'94 Corpus.

corpus size	240 million words
bigrams: $N(u, v) = 1$	51 888 422
$1 < N(u, v) < 256$	4 848 006
$N(u, v) > 255$	100 779
trigrams: $N(u, v, w) = 1$	41 885 919
$1 < N(u, v, w) < 256$	17 956 245
$N(u, v, w) > 255$	71 907

- The second approach computes the smoothed relative frequencies of all trigrams beforehand and stores these probability estimates, herewith avoiding the time consuming computations during the speech recognition process. The trigram probabilities can be easily retrieved using the word indices of the words in the trigram. As the main disadvantage, experiments which require modifications of the counts $N(u, v, w)$ (e.g. computing Leaving-One-Out-probabilities on the training corpus) can no longer be performed. Another disadvantage is the still large amount of memory needed, when no cut-offs are applied to the counts.

- The third approach which we have decided to follow is to store the counts of the trigrams instead of their probabilities. Nevertheless, the memory requirement is very high as presented below. We will show that by using the special structure of the counts the memory cost and the average access time can be reduced.

2   STORING THE COUNTS

A rather straigthforward solution to the storing problem is the following: Unigrams, bigrams and trigrams are stored in one array each. Starting with the first word u in the trigram u, v, w, we look for the second word v in the list of successors which make up a certain part in the bigram list. Applying the same scheme to the trigram list, we can easily find the word w and the corresponding count $N(u, v, w)$ using binary search. With this approach the memory cost sums up to 420 MByte. Assuming, that the trigram frequencies are almost identical for training and testing, we can easily compute the average number of accesses to the data structure which is needed to find a specific trigram:

$$\sum_u \sum_v \frac{N(u, v)}{N} \cdot [\mathrm{ld}\,(\mathrm{SUC}(u) + 1) - 1 + \mathrm{ld}\,(\mathrm{SUC}(u, v) + 1) - 1] \quad , \qquad (1)$$

with $\mathrm{SUC}(u, v)$ denoting the number of different words w succeeding the word-pair u, v in the training and N being the training corpus length. With this approximation we obtain the following results: Finding a trigram takes 17.8 search accesses on average and 29 accesses in the worst case.
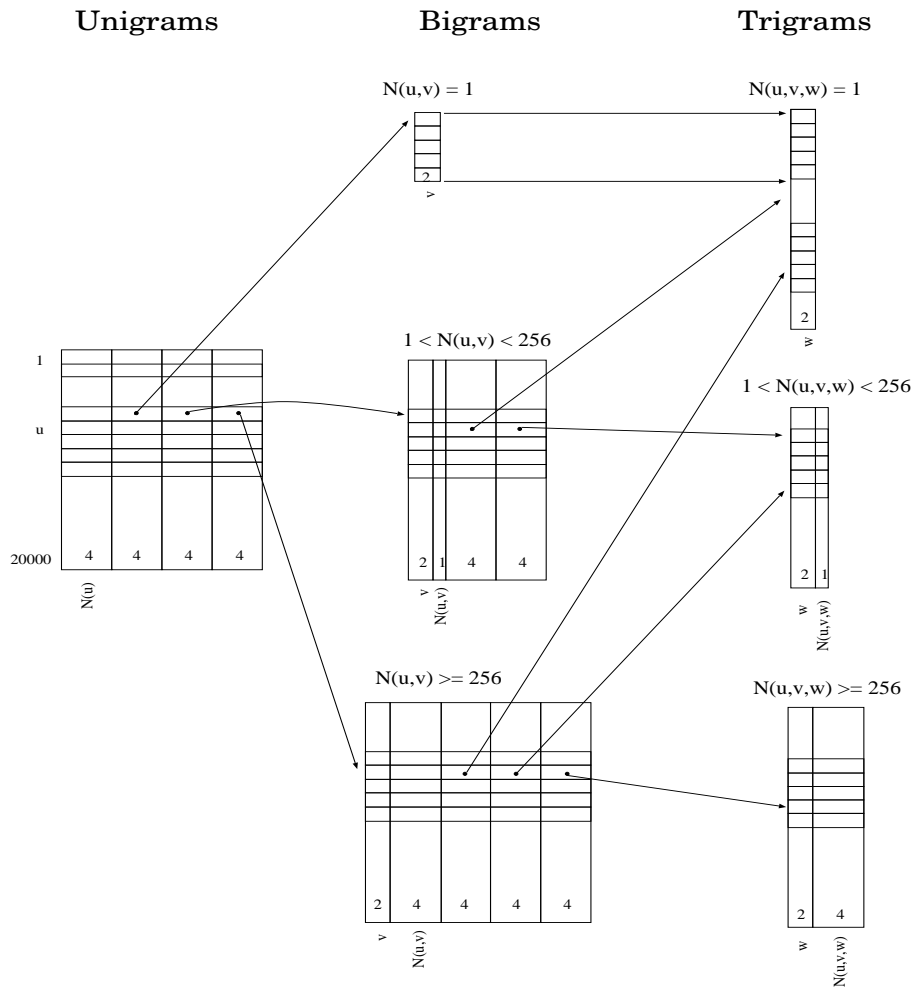
Fig. 1. Data structure used for storing trigram, bigram and unigram counts.

The drastic reduction of memory in our approach is based on the usage of different data types for the counts. When analysing the corpus statistics in Table 1 it becomes obvious that a lot of memory is wasted on singleton events, i.e. events seen only once in the trainging corpus. Furthermore, storing the counts of all events, which have been observed less than 256 times we only need one byte for each count instead of four. Bigram and trigram singletons are stored without their counts and events, whose frequency ranges from 2 to 255 are stored in arrays of the type *unsigned char*, i.e. one byte only. For counts larger than 255 we have to use arrays of the type *unsigned long*, i.e. four bytes. The gain is most dramatic for all events u, v, w, with: $N(u, v) = N(u, v, w) = 1$. In this

special case there only exists one successor w following v and the position of the word w in the trigram singleton list exactly corresponds with the position of v in the bigram singleton list. Using the different arrays for the counts, the memory requirement has been reduced from 420 to 198 MByte. The links into the successor lists are stored as array indices and not as pointers, so that the whole data structure can be created once in the memory and then be written to storage media. The time spent on initializing any of the count based language models which we have implemented, can be kept very low this way. Searching the data structure for a trigram is, of course, more complicated now. On the other hand, the average access time could be reduced, regarding the unigram count $N(u)$: if u is observed only once during the training, the search for the succeeding word v can be restricted to the upper two arrays in the bigram level of the data structure. The same applies, of course, to searching for trigrams. In the average case, we need 14.6 accesses before a trigram is found and 48 accesses in the worst case.

3 LANGUAGE MODEL

We have implemented several language models, extending the idea of absolute discounting in several directions [Ney et al. 97]. Due to its simplicity and good performance we have decided to perform recognition experiments with the following model:

$$p(w|h) = \max \left\{ 0; \frac{N(h,w) - b}{N(h)} \right\} + b \cdot \frac{W - n_0(h)}{N(h)} \cdot \beta(w|\overline{h}) \quad , \qquad (2)$$

with h denoting the history of the word w, i.e. its predecessing words, b denoting the discount parameter, W the vocabulary size, $n_o(h)$ the number of words not seen following the history h and $\beta(w|\overline{h})$ a less-specific distribution with its generalized history $\overline{h}$.

3.1 EXPERIMENTAL RESULTS

The recognition experiments have been performed on all of the 310 sentences of the NAB '94 H1 development corpus with a closed vocabulary containing 20000 words. 199 spoken words are not part of this vocabulary. We have used the word graph method, as described in [Ortmanns et al. 97], comparing several variants of the above trigram language model ($RWTH$) and the official NAB '94 reference trigram language model (NAB) as defined in [Rosenfeld 95]. As table 2 shows, the error rate could be reduced with all models. In using a singleton backing off distribution, we have achieved the best result; when using a compact trigram, i.e. omitting all trigram singletons, the error rate has increased a little. The very low memory requirement of the reference model is due to higher cut-offs in the model. In [Rosenfeld 95] all trigrams, seen less than four times, and even all bigram singletons have been omitted from the language model. In order to verify the official results we have applied the same cutting scheme to the bigrams

Table 2. Experimental results for the word graph search with various trigram language models on the NAB '94 H1 development corpus (20 speakers = 310 sentences = 7387 words).

trigram language model		perplexity	memory cost [MByte]	recognition errors [%] del/ins	WER
NAB model		132.7	65	1.6/2.7	14.3
RWTH model		141.1	60	1.6/2.8	14.3
full RTWH model:	standard dist.	132.0	198	1.5/2.9	13.9
	singleton dist.	121.8	221	1.7/2.2	13.5
compact RTWH model:	standard dist.	135.8	109	1.5/2.7	14.0
	singleton dist.	124.3	132	1.7/2.5	13.7

and trigrams in our model, adding the probability mass of these events to the discounting mass. With this small model we have achieved identical results. Adding a bigram cache to the best of the above trigram language models, we have achieved another reduction of the word error rate down to 13.1%.

4 CONCLUSIONS

In this paper, we have presented an efficient data structure for storing trigram, bigram and unigram counts. We have achieved a drastic reduction of the memory cost and we have also slightly decreased the average search time within the data structure. Based on the implementation of this data structure we have implemented several extensions of the well-know absolute discounting models [Generet et al. 95, Ney et al. 97] and applied the best of them to the NAB '94 recognition task. Using the word graph search, the word error rate could be reduced from 14.3% to 13.5% and to 13.1% adding a bigram cache to the best trigram language model.

References

[Generet et al. 95] M. Generet, H. Ney, F. Wessel: "Extensions of Absolute Discounting for Language Modelling", Fourth European Conference on Speech Communication and Technology, Madrid, pp. 1245-1248, Sep. 1995.

[Ney et al. 97] H. Ney, S. Martin, F. Wessel: "Statistical Language Modeling Using Leaving-One-Out", to appear in Elsnet Utrecht Summer School Proceedings.

[Ortmanns et al. 97] S. Ortmanns, H. Ney, X. Aubert: "A Word Graph Algorithm for Large Vocabulary Continuous Speech Recognition", to appear in Computer Speech and Language.

[Rosenfeld 95] R. Rosenfeld: "The CMU Statistical Language Modeling Toolkit for Language Modeling and its Use in the 1994 ARPA CSR Evaluation", Proc. 'Spoken Language Systems Technology Workshop', Austin, TX, pp. 74-50, Jan. 1995.

This article was processed using the LaTeX macro package with LLNCS style