EFFICIENT PRECALCULATION OF LM CONTEXTS FOR LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION

F. Javier Diéguez-Tirado, Antonio Cardenal-López

E.T.S.I. Telecomunicación
Dpto. Teoría de la Señal y Comunicaciones
Universidad de Vigo
36200 Vigo-Spain

Abstract

In a previous work we described a new method to speed up the computation of a Language Model look-ahead in a speech recognizer with a tree organized lexicon. Three different mechanisms were employed to avoid a redundant computation of the probabilities: a node level cache memory, a pre-calculation of active contexts and a perfect hash LM organization. This strategy allowed us to apply a trigram based LM to compute the look-ahead with important computational savings in comparison with the usual bigram or unigram approximation. In this paper we describe several improvements to the pre-calculation of active contexts that allow us to achieve further time reductions.

1. INTRODUCTION

It is a well known fact that the early incorporation of all knowledge sources in the decodification stage of a LVSR is a useful way to reduce the complexity of the search. When a tree-organized lexicon is used, a problem arises in the application of the language model probabilities. Since the identity of the word remains unknown until the path reaches the end of the tree, the incorporation of the exact LM probabilities must be delayed. In order to reduce the search effort, a LM look-ahead algorithm is usually incorporated in modern continuous speech recognizers. The goal is to obtain an approximated value of the true LM, which may be added to the global score of the path as soon as possible. A commonly used approximation is the maximum probability of all the words that may be reached from the actual node [1]. Although this strategy has been proved to be very effective [2], it has an important computational cost. The calculation of a single look-ahead value using a back-off based LM may comprise several thousands of accesses to the LM data structures. This extra load may seriously reduce the performance of the algorithm. For this reason, in many modern recognizers the look-ahead is computed using a simplified LM, with shorter and less expensive n-grams [3]. In this way the computational cost is decreased, but some potential reduction of the search effort is also lost.

In previous works [4] we presented a fast method to compute the maximum approximation using three mechanisms applied sequentially: a node level cache, a precalculation of the context probabilities, and a highly optimized LM access using perfect hash organization. This strategy has allowed us to apply a trigram based LM look-ahead in a complex large-vocabulary dictation task, with better overall results both in computation time and recognition rate than using a bigram approximation. In this paper we present several improvements to the context precalculation stage which allow us to achieve further computational savings.

The remainder of this paper is organized as follows: in section 2 a brief description of the three layer LM look-ahead algorithm is given; in section 3 we describe the new procedure used to compute the context probabilities; section 4 is dedicated to analyze the performance of the algorithm in a complex continuous speech task; finally some conclusions are presented in section 5.

2. EFFICIENT LM LOOK-AHEAD

Given a tree node \( N \) and a LM context \( \{w_1, w_2\} \), the LM look-ahead is calculated by obtaining the maximum of the probability \( Pr(w_3 \mid w_1, w_2) \), where \( w_3 \) can be any of the words \( U_N = \{w_1, \ldots, w_N\} \) that may be reached from \( N \). Using Good-Turing backoff based discounting, a database of observed bigrams and trigrams is employed, and each probability is defined by these expressions:

\[
\text{if } \exists (w_1, w_2, w_3) \rightarrow P = Pr(w_1, w_2, w_3) \\
\text{else if } \exists (w_2, w_3) \rightarrow P = Bo(w_1, w_2) + Pr(w_2, w_3) \\
\text{else } \rightarrow P = Bo(w_1, w_2) + Bo(w_2) + Pr(w_3)
\]

The first step is to identify the set \( U_N \). In our decoder each word is represented by a natural number, and this set...
is currently stored as an array linked to each node for faster calculation. However, this is a memory consuming approach and must be revised when the network size should grow.

The next and most expensive part of the algorithm is to perform the actual maximization. In a typical scenario, there are millions of LM probability queries in the recognition of a short speech segment, but most of them correspond to related data, so it proves useful to reutilize calculations, as well as using a convenient data structure to optimize each access. In our previous work [4], we presented a 3 layer architecture that addressed these issues:

- On top is a node cache indexed by the context identifier (a single integer), which stores the recently computed maximum for this node and context. This cache has usually got a hit rate of over 90% and is considerably faster than using two separate indexes for each word in the context.

- Every time a LM context \(w_1, w_2\) is defined in the network, all the possible probabilities \(Pr(w_1, w_2, w_3)\) are calculated and stored in an array with length equal to the size of the vocabulary. These are shared and used for the look-ahead throughout the context life. To reduce the memory consumption, the number of active contexts per frame is limited using word pruning. All words completed at each frame are sorted by score and those with less probability are not propagated.

- To speed up the calculation of each single probability, LM data is accessed using perfect hashing [5]. The data structure is shared for both the look-ahead and the other LM queries, but the access for the look ahead is loop optimized for improved performance.

While context precalculation and loop optimized perfect hashing provide an important improvement over not using them (Sec. 4), the array filling step was still the most expensive one, so we identified it as a possible objective for future work. In the following section we present a new method to fill the array of probabilities that does not rely on perfect hashing and that gives better results.

### 3. FAST LM CONTEXT FILLING

Our new approach for creating the array of probabilities \(Pr(w_3/w_1, w_2)\), where \(W\) is the size of the vocabulary, takes into account that only a few bigrams \((w_2, w_3)\) and and trigrams \((w_1, w_2, w_3)\) exist in the n-gram database, instead of all the possible combinations, thus equation (3) will apply for most \(w_3\). It would be desirable that in these cases, the query to the trigram and bigram would be avoided (Eqs. 1 and 2). We will show how to reduce this cases to zero work and to lower the cost the remaining array positions at the same time.

To achieve this, we decided to subtract the backoff component \(B = Bo(w_1, w_2) + Bo(w_2)\) from the probability array, storing \(P' = P - B\), and so the equations become:

\[
\begin{align*}
\text{if } &\exists (w_1, w_2, w_3) \rightarrow P' = Pr(w_1, w_2, w_3) - B \quad (4) \\
\text{else if } &\exists (w_2, w_3) \rightarrow P' = Pr(w_2, w_3) - Bo(w_2) \quad (5) \\
\text{else } &P' = Pr(w_3) \quad (6)
\end{align*}
\]

This does not affect the maximization of probabilities as long as the backoff component is added to the result, i.e.:

\[
LMLA = \max_{U_N}(P) = B + \max_{U_N}(P')
\]

The new discounting algorithm (Eqs. 4, 5 and 6), could be implemented as follows:

- Start with a preinitialized array containing the unigram probabilities \(Pr(w_3)\).
- Store (5) in each cell where a bigram \((w_2, w_3)\) exists.
- Fill with (4) the positions where a trigram \((w_1, w_2, w_3)\) exists.

Following this sequence of steps, the existence of a bigram would discard the unigram probability, and each trigram would in turn take precedence over any other value of the cell, as the discounting algorithm dictates.

The bigrams and trigrams related to each context can be stored consecutive in memory, so that only the first one must be found every time a LM context is created, without the need to do a query for each one individually. In our implementation, we exploit the existing perfect hashing data structure that is still used for every LM access other than look-ahead, taking advantage of its order preserving property. The position of the first bigram \((w_2, w_3)\) and trigram \((w_1, w_2, w_3)\) for each context \(\{w_1, w_2\}\) are found using an auxiliary array and auxiliary perfect hashing.

After the context has been created and should be deleted, it has proven useful to restore the original unigram probabilities of the cells that have been modified (cases 4 and 5), and
to return the array to a bank of preinitialized contexts that can be used in the next context creation, without the need to initialize the \( W \) cells again.

3.1. Maximizing the tree from backward to forward

We have tried to push this approach a bit further by developing the ideas that only a few bigrams and trigrams per context exist, and that the number of active contexts can be kept low thanks to word pruning.

The objective in this case is to store the actual LMLA value at each node every time a new context is defined. This would eliminate the need for a node cache, an array or probabilities, and the maximization step, so if it could be done in a faster way than the combined 3 layer approach, it would be more convenient.

To perform this task, we have allocated a fixed number of slots at each node to store the LMLA values for each possible active context, and have also incorporated a special slot to store the unigram maximum, the context independent value that would constitute the LMLA if only unigrams would exist (Eq. 6). This unigram maximum is intended to define the LMLA value in most cases, except were a bigram or trigram exists for a given context, that causes a leaf in the tree to achieve a higher probability (obtained by Eqs. 4 and 5), and that alters the LMLA value from that leaf backwards, until other n-gram or the unigram maximum is observed to be higher than that probability, so that it will define the LMLA value from that node backwards and we can stop traversing the tree (Fig. 1). For priority reasons, this process would first be repeated for each trigram related to each context and then for each bigram, using a n-gram data structure analogous to the one used for the discounting algorithm implementation (Sec. 3). This can be seen as an adaptation of the method described in [6] to single lexical tree decoders.

The fact that there are not many n-grams for each context, and that the tree must not be traversed completely for each n-gram, led us to believe that this would be a good approach. However, it has proven inferior to our previous 3 layer architecture for our tests (Sec. 4). The reason behind the poor performance can be attributed to performing the maximization for every single node, even those that won’t be used during the actual decoding because of pruning.

### Table 1. Successive LMLA improvements

<table>
<thead>
<tr>
<th>Look-Ahead</th>
<th>%RT LA</th>
<th>Cache</th>
<th>Maxim.</th>
<th>Context fill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unopt. phash</td>
<td>0.3090</td>
<td>0.1226</td>
<td>0.3864</td>
<td>-</td>
</tr>
<tr>
<td>Loop opt. phash</td>
<td>0.3058</td>
<td>0.1081</td>
<td>0.1977</td>
<td>-</td>
</tr>
<tr>
<td>Contexts</td>
<td>0.2178</td>
<td>0.0541</td>
<td>0.0360</td>
<td>0.1277</td>
</tr>
<tr>
<td>New contexts</td>
<td>0.0990</td>
<td>0.0555</td>
<td>0.0357</td>
<td>0.0078</td>
</tr>
</tbody>
</table>

### Table 2. N-grams used for the lookahead

<table>
<thead>
<tr>
<th>Look-Ahead</th>
<th>RT LA</th>
<th>RT Total</th>
<th>Tokens</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigram</td>
<td>0.0609</td>
<td>1.632</td>
<td>2729.7</td>
<td>19.72</td>
</tr>
<tr>
<td>Bigram</td>
<td>0.0763</td>
<td>1.469</td>
<td>2296.6</td>
<td>19.43</td>
</tr>
<tr>
<td>Trigram</td>
<td>0.0990</td>
<td>1.552</td>
<td>2159.1</td>
<td>19.38</td>
</tr>
</tbody>
</table>

4. EXPERIMENTAL RESULTS

All results presented in this section were extracted using the LVCSR developed in the University of Vigo. The recognition is performed in two stages: a Viterbi synchronous pass, with a single lexicon tree, and a N-best stage using word-graphs. All available knowledge sources are included in the first pass: context-dependent gender-independent HMMs, cross-word models, and trigram based LM.

We have performed several experiments using a multi-speaker telephone database in Galician language. The test data consists on 364 files from 59 male and female speakers, with a total of 15891 uttered words. Each speaker was asked to read a short journalistic text. The LM was extracted from a newspaper macrotext and contains 21,216 unigrams, 267,806 bigrams and 218,050 trigrams. The perplexity of the task is 149.68. There are no out-of-vocabulary words. For the acoustic modelling, we have used 497 demiphones with 2 states per model and 4 mixtures per state trained from the SpeechDat database for Galician. The experiments were performed using a 930 MHz Pentium III processor.

Our reference test was performed using these parameters: pruning 180, word pruning 50, and a cache size of 20. Using these values, the test took an average of 1.25% RT time to complete, excluding the LMLA. Table 1 shows the time consumed by the LMLA routines, reflecting the successive improvements that we have achieved. It can be seen that our new approach for context creation is much superior to the previous strategy, causing the LMLA portion to halve and to represent just the 7.3% of the total decoding time. The context creation is thus relieved from being the most expensive part of the look-ahead, making the 3 layer architecture more balanced.

With this faster look-ahead, basing it on trigrams proves much more convenient than using bigrams or unigrams, because the more expensive look-ahead in the first case is compensated by a decrease in the number of active tokens. This result is shown in Table 2 and supports the conclusions of our previous work.

As shown in Table 3, the time required for every portion of the decoder, including the LM look-ahead, is heavily influenced by the pruning thresholds. On one hand, lowering the global pruning threshold helps to decrease the complexity at the expense of a higher word error rate. This affects the look-ahead in that there are less active contexts, thus the node cache works better and less probability arrays must be
Table 3. Some statistics for the test described in section 4

<table>
<thead>
<tr>
<th>Pruning level</th>
<th>RTF</th>
<th>Search Complexity (values per frame)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Active Tokens</td>
</tr>
<tr>
<td>Global</td>
<td>Word Cache Maxim. Context LMLA Total</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>50</td>
<td>0.0225</td>
</tr>
<tr>
<td>160</td>
<td>50</td>
<td>0.0356</td>
</tr>
<tr>
<td>180</td>
<td>50</td>
<td>0.0422</td>
</tr>
<tr>
<td>200</td>
<td>50</td>
<td>0.0486</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0560</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0915</td>
</tr>
</tbody>
</table>

created. Also, as less tokens exist and they will traverse a smaller portion of the tree, the maximization will be performed less times.

On the other hand, while word pruning helps to reduce the complexity in the same manner as described for global pruning, it has a significant effect in the recognition quality if it’s level is not adequately chosen. For example a word pruning level of 30 is inadequate in this case, as an execution using P=160 would be faster and more accurate. But a word pruning of 50 proves very interesting as it reduces a lot of complexity and memory usage compared to not using it, while not degrading the quality in a significant manner.

5. CONCLUSIONS AND FURTHER WORK

In this paper we have presented several improvements to our LM look-ahead implementation, which have allowed us to simplify the previously most expensive context precalculation layer. Particularly, we have taken advantage of the low number of bigrams and trigrams related to each context in a typical LM, and have separated the backoff part of the probabilities so that most of them can be shared. We have also avoided the use of hash for every probability except the first one, accessing them sequentially by exploiting the order preserving property of our data structure.

With this improvement, the total time required for the LM look-ahead is approximately halved, and it constitutes about 7% of the total program time for the tests we have performed. Consequently, the advantages of reducing the time even more should not be very significant.

Future work should be centred in the validation of the developed procedure in more complex tasks. An increment in the number of bigrams and trigrams may have an important impact in the time consumption of the algorithm, modifying substantially the values obtained, specially in the context precalculation stage. Also a general increase of the time expended in the LM look-ahead computation should be expected for larger vocabulary sizes. We are currently compiling additional journalistic text in Galician language to extend our LM and perform new experiments.

6. REFERENCES


