Scalable Language Model Look-Ahead for LVCSR

Dominique Massonié, Pascal Nocera, Georges Linares

LIA – CNRS, University of Avignon, France
{dominique.massonie, pascal.nocera, georges.linares}@lia.univ-avignon.fr

Abstract

In this paper a new computation and approximation scheme for Language Model Look-Ahead (LMLA) is introduced. The main benefit of LMLA is sharper pruning of the search space during the LVCSR decoding process. However LMLA comes with its own cost and is known to scale badly with both LM n-gram order and LM size. The proposed method tackles this problem with a divide and conquer approach which enables faster computation without additional WER cost. The obtained results allowed our system to participate in the real-time task of the ESTER Broadcast News transcription evaluation campaign for French.

1. Introduction

Due to the linguistic constraints and following the Bayes formula, the decoder evaluates sentence hypotheses at the word level. For Large Vocabulary Continuous Speech Recognition (LVCSR) systems, a subword level is used (generally the chosen unit is the phoneme) in order to share the common part of words, for training and decoding. In practice the lexicon is generally organized as a Pronunciation Prefix Tree (PPT). When the decoder extends the hypotheses at the subword level, at some point – frame $t$ – it has to compare well formed hypotheses (a sequence of words ending at frame $t$) with hypotheses ended by a partial word (a sequence of words followed by a partial word ending at frame $t$). Language Model Look-Ahead (LMLA) aims at evaluating partial hypotheses by anticipating the expected probability of possible continuations.

Figure 1 shows the extension at the subword level of one hypothesis (eg. from frame $t$ to $t+1$). If a word boundary is reached, the true hypothesis score is computed. If the hypothesis remains a partial word, then LMLA is used to evaluate its probability. Thus the decoder is able to compare hypotheses at each decoding step, as the acoustic and linguistic information are now synchronized. The system can then reorder the hypotheses and use sharper pruning.

This work presents an efficient and scalable LMLA computation method based on a divide and conquer approach.

2. Language model look-ahead

LMLA appears in different forms and their integration into the decoding process depends on the decoder search space and strategy (see [1] for an overview of decoding techniques). LMLA is described in [2] and more recently here [3], the two following subsections summarize LMLA techniques. Our computation paradigm is introduced in section 2.4, then results and discussion are presented in 3.3 and 3.4.
2.4. Proposed LMLA computation method

The LMLA is a factored language model access (equation 1). The best probability \( \pi_h(s) \) for each word \( w \) of the wordlist \( W(s) \) is computed according to the word history \( h \).

\[
\pi_h(s) = \max_{w \in W(s)} p(w|h)
\]  

(1)
Let $L'(h)$ be the list of available existing n-grams, $L'(h) = L(h) \cap W(s)$ and $L''(h)$ be the list of n-grams with required backoff, $L''(h) = W(s) \setminus L'(h)$.

When the sublist $L'(h)$ is empty, the same process is repeated at the lower order (down to bigram). The additional cost of intersection computation is largely compensated by the spared search time at an n-gram order without matching word candidates, where LM backoff is needed for each word (results in section 3.3). As a side effect of intersection computation, the PPT wordlist is split into sublists of existing trigrams or bigrams in the LM. Thus LMLA computation $\pi_n(s)$ is done as follow:

$$\pi'_n(s) = \max_{w \in L'(h)} p(w|h)$$

$$\pi''_n(s) = \text{backoff}(h) + \max_{w \in L''(h)} p(w|h) - 1$$

$$\pi_n(s) = \max\{\pi'_n(s), \pi''_n(s)\}$$

Where $\pi'_n(s)$ is recursively computable down to bigram, using $L''(h)$ as $L(h)$ at the lower order n-gram and $h$ is always reduced by its first word (ie. head). Approximations introduced in this paper are based on LMLA computation for existing n-grams only $\pi'_n(s)$ and avoid $\pi''_n(s)$ computation or use a precomputed score, depending on the size of $W(s)$.

2.4.5. Precomputed LMLA

In the case of LVCSR systems for Broadcast News, the LM size – number of bigrams and trigrams (see table 2) – is big enough to discourage complete LMLA precomputation. In this paper, the precomputation is limited to a small number of the biggest PPT wordlists (see table 3) and is only done at the bigram level. This corresponds to the $h-1$ n-gram order needed to compute equation 3. The precomputed LMLA could be initialized with approximated scores but in our case only exact scores are used. Depending on precomputation availability, the on-the-fly LMLA computation is whether exact or an approximation (ie. equation 3 is used if available already or skipped). The optional precomputed LMLA of bigram order is stored as a simple two dimension table of size vocabulary size $\times$ number of precomputed values.

3. Experiments

3.1. Experimental framework

The experiments are performed on a subset of the development corpus for the French Broadcast News evaluation campaign Es-

3.2. Baseline results

The baseline results show the decoding speed of two basic LMLA computation methods (see table 4 and 5).

- This is the worst case scenario for equation 1, where the tree structure of the LM is not used and each n-gram is recomputed from 1-gram up to $n$. The lines with a Skip column set to 1000 are the same as the one above except that LMLA for $W(s)$ wordlists larger than 1000 words is skipped (section 2.4.3). This only represents the 15 or 34 biggest wordlists depending on the lexicon size (table 3). However, these cases are the most time consuming. This is explained by the fact that the wordlists are not only the biggest but also the closest to the PPT root, thus systematically used for new hypothesis extension after any word boundary. The given sample cut introduces a significant loss in decoding precision. 1.7% WER increase for the 65k lexicon. This underscores the weight of LMLA both in terms of computation cost and precision in the decoding process. An intermediate LM called 27k+ generated without cutoff, thus bigger than the 27k LM was used to confirm that lower computation time between 27k and 65k lexicons was due to vocabulary size and not affected by the on disk data layout.

- This case takes advantage of the tree structure of the LM during LMLA computation.

3.3. Divide and conquer strategy results

The results integrating our scalable method into the decoding process are shown on table 6 (as an example, last line corresponds to figure 4 with thresholds 2, 5, 98, 1500).
Table 4: Baseline results with simple LMLA cut. The Skip ratio avoids LMLA computation according to wordlists size (eg. > 1000). The 27k+ is a larger LM version of 27k (see table 2), but overall system performance remains close.

<table>
<thead>
<tr>
<th>Lex</th>
<th>Skip</th>
<th>Corr</th>
<th>Sub</th>
<th>Del</th>
<th>Ins</th>
<th>WER</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>27k</td>
<td>0</td>
<td>71.6</td>
<td>19.0</td>
<td>9.4</td>
<td>2.5</td>
<td>30.9</td>
<td>89m8</td>
</tr>
<tr>
<td>27k</td>
<td>1000</td>
<td>70.5</td>
<td>19.8</td>
<td>9.7</td>
<td>2.3</td>
<td>32.0</td>
<td>49m30</td>
</tr>
<tr>
<td>27k+</td>
<td>0</td>
<td>71.6</td>
<td>19.0</td>
<td>9.4</td>
<td>2.4</td>
<td>30.8</td>
<td>89m36</td>
</tr>
<tr>
<td>27k+</td>
<td>1000</td>
<td>70.4</td>
<td>19.8</td>
<td>9.8</td>
<td>2.5</td>
<td>32.1</td>
<td>55m43</td>
</tr>
<tr>
<td>65k</td>
<td>0</td>
<td>72.8</td>
<td>17.4</td>
<td>9.8</td>
<td>1.9</td>
<td>29.1</td>
<td>157m54</td>
</tr>
<tr>
<td>65k</td>
<td>1000</td>
<td>71.2</td>
<td>18.5</td>
<td>10.3</td>
<td>2.0</td>
<td>30.8</td>
<td>62m25</td>
</tr>
</tbody>
</table>

Table 5: Improved Baseline with faster LMLA computation (but no approximation). Our scalable method starts from step Φ.

<table>
<thead>
<tr>
<th>Lex</th>
<th>Skip</th>
<th>Corr</th>
<th>Sub</th>
<th>Del</th>
<th>Ins</th>
<th>WER</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>27k</td>
<td>0</td>
<td>71.6</td>
<td>19.0</td>
<td>9.4</td>
<td>2.5</td>
<td>30.9</td>
<td>77m4</td>
</tr>
<tr>
<td>27k</td>
<td>0</td>
<td>71.6</td>
<td>19.0</td>
<td>9.3</td>
<td>2.5</td>
<td>30.9</td>
<td>63m56</td>
</tr>
<tr>
<td>27k</td>
<td>1000</td>
<td>70.5</td>
<td>19.8</td>
<td>9.7</td>
<td>2.3</td>
<td>32.0</td>
<td>49m30</td>
</tr>
<tr>
<td>65k</td>
<td>0</td>
<td>72.8</td>
<td>17.4</td>
<td>9.8</td>
<td>1.9</td>
<td>29.1</td>
<td>120m20</td>
</tr>
</tbody>
</table>

Φ – In this case equation 1 is computed in two steps as discussed in section 2.4.4, formula 4. The speedup shows the effectiveness of intersection computation before directly applying LMLA on the W(s) list. Thanks to the sublists, significantly less LM backoff computation is needed with this method. The computation burden remains for the biggest lists, but with Skip set to 1000, the system can now reach real-time performance with the 65k lexicon.

Φ – The LMLA is approximated, the computation is done for existing trigrams only, the equation 3 is not taken into account for lists larger than 5 words. This approximation does not lower the search efficiency, while significantly speeding up the decoding process.

Φ – The precomputed LMLA for bigrams is introduced at this step. Thanks to the wordlists splitting, equation 4 can partially be precomputed for a given set. The results are not changed but the running time is significantly reduced (close to real-time).

Table 6: LMLA computation with approximation line Φ (ie. equation 3 is skipped) and Φ a Precomputed Bigram Size (PBS) of 400 is used (ie. equation 3 available for the 400 biggest wordlists).

<table>
<thead>
<tr>
<th>Lex</th>
<th>PBS</th>
<th>Corr</th>
<th>Sub</th>
<th>Del</th>
<th>Ins</th>
<th>WER</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>65k</td>
<td>0</td>
<td>72.8</td>
<td>17.4</td>
<td>9.8</td>
<td>1.9</td>
<td>29.1</td>
<td>83m18</td>
</tr>
<tr>
<td>65k</td>
<td>0/00</td>
<td>72.8</td>
<td>17.3</td>
<td>9.9</td>
<td>1.9</td>
<td>29.1</td>
<td>67m40</td>
</tr>
<tr>
<td>65k</td>
<td>0/400</td>
<td>72.8</td>
<td>17.4</td>
<td>9.8</td>
<td>1.9</td>
<td>29.1</td>
<td>60m21</td>
</tr>
<tr>
<td>65k</td>
<td>1000/0</td>
<td>71.2</td>
<td>18.5</td>
<td>10.3</td>
<td>2.0</td>
<td>30.8</td>
<td>57m39</td>
</tr>
<tr>
<td>65k</td>
<td>1500/400</td>
<td>70.9</td>
<td>18.7</td>
<td>10.4</td>
<td>2.0</td>
<td>31.1</td>
<td>55m43</td>
</tr>
</tbody>
</table>

3.4. Discussion

The presented method with approximation is twice faster – Φ to Φ – than the ‘normal’ computation scheme without any WER difference. The last two lines of table 6 show that faster computation can be achieved with different parameters combinations, but with a significant WER increase. Moreover, real-time performance is achieved.

The lexicon compression as presented in section 2.3 was not discussed because no significant speedup was noticed for the real-time task described in this paper (due to sharp pruning of the decoder).

Different experiments were carried out with a fallback to a lower order n-gram (precomputed) LMLA, but none have been faster than the 3-gram method, with generally significantly worse WER, as expected.

The introduced approximations obtain the same WER as the ‘exact’ computation but results are not exactly the same (at the sentence level). Nevertheless the worst case scenario seems to get the same overall result. For different tests sets, results were slightly better with our approximation. A conjecture is that favoring existing trigrams during the decoding process works better with tests domain matching the trained LM.

More implementation efforts could be done, especially on efficient computation technique with SIMD instructions as used for neural network language modeling [7].

4. Conclusions

In this paper a new and scalable language model look-ahead computation method was introduced. The method is based on splitting the LMLA computation into parts which enables different cut or approximation techniques. This allowed our system to cope with large linguistic models adapted for Broadcast News transcription and real-time constraints. The end result is a twice faster decoding speed without any additional WER cost. More parameter tuning or code optimization could lead to further improvements for the next phase of Ester evaluation campaign.

5. References