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 needed (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.

Figure 1: For each hypothesis extension at the subword level, the LMLA enables acoustic and linguistic synchronization without waiting for a word boundary to be reached.

2.1. Decoder integration

Less efforts are needed for systems based on weighted finite state transducers (WFST) [4]. The full search network expansion is done statically, exploiting sparsity of the knowledge sources and taking advantage of redundancies. The LMLA appears (with AM/LM synchronization as conjectured here [5]) to be an optimization of the weight distribution in the network structure. The Weight pushing algorithm moves the weights along the paths to the head of the graph without changing the structure nor the overall results. The new network is equivalent but discriminant information are encountered earlier in the search process, thus enabling more efficient pruning.

Systems with a dynamic network expansion use two search strategies, time-synchronous or asynchronous. The former is considered a breadth-first search with all hypotheses ending at the same time. The latter is a depth-first search with all hypotheses ordered but not ending at the same time. Both take advantage of LMLA but without the same precision need, leading to different computation strategies.

In the case of a time-synchronous decoding strategy the LMLA helps to reorder the current hypotheses with anticipated linguistic information. This avoids early dropping of hypotheses with low score despite good expected possibilities. The rescoring hypothesis list is then shortened and all remaining hypotheses are extended. The LMLA precision is less important when hypotheses are selected according to their ranking, the reordering with LMLA needs only to be discriminant enough.

In the case of an asynchronous decoder, also known as stack based decoder, a probe function is used to estimate the cost of the remaining path. To be an admissible heuristic, the probe must not overestimate the true minimal remaining cost.
The LMLA is included in each hypothesis score as part of this probe, which combines both expected linguistic and acoustic scores. The ranking of all hypotheses is done using the whole path cost (including probe), only the best one is extended. The probe quality and its LMLA precision is a determining part of the search efficiency. In this paper, the stack based decoder Speeral [6] developed at LIA is used to evaluate different LMLA computation strategies. The presented method satisfies the stack decoding conditions but is applicable to others. The two following subsections extend system description to two relevant components for LMLA computation.

2.2. Language model organization

The Language Model – LM structured as a tree of lists $L(h)$, see figure 2 – is directly mapped from disk to memory. The LM memory cache is thus managed by the operating system page-cache (ie. 4kB blocks with prefetching). Moreover a simple hash-based cache is used to store last asked trigrams. In [3] fast access to LMLA is based on a perfect hashing (more place consuming) to store the LM.

The LMLA is based on the LM structure and duplicates the hash-based cache to store last LMLA calls. The current ondisk LM postpones the choice to optimize the data layout for embedded systems (LM index compression), reduced vocabulary size (short integers) or SIMD computation (as the bunch method discussed in [7]). Moreover the same models and data layout are used for any ‘faster’ or ‘slower’ decoding configurations.

2.3. Lexicon structure and compression

The lexicon is stored as a Pronunciation Prefix Tree (PPT) for search efficiency [1]. Each node $s$ (as a state reached during the decoding process) knows the wordlist of reachable words $W(s)$ (see figure 3). The wordlist size decreases along the PPT to reach one single word at leaves. No other information like subtree dominance [8] is used. Due to pronunciation variation or tree sparsity, different nodes can share the same wordlist. As shown on table 1 most nodes are leaves. On average each wordlist tends to be shared by 3 internal nodes. Other properties of the lexical tree compression are also discussed in [2].

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 $h$.

$$\pi_h(s) = \max_{w \in W(s)} p(w|h)$$  \hspace{1cm} (1)
The LMLA computation strategy:

Wordlist size

<table>
<thead>
<tr>
<th>Exact</th>
<th>Approximated</th>
<th>Pre-computed</th>
<th>Exact/Approx.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: The type of LMLA computation depends on the number of reachable words from a PPT node. LMLA is irrelevant for wordlist of a single element.

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:

$$\pi_n'(s) = \max_{w \in L'(h)} p(w|h)$$  \hspace{1cm} (2)

$$\pi_n''(s) = \text{backoff}(h) + \max_{w \in L''(h)} p(w|h - 1)$$ \hspace{1cm} (3)

$$\pi_n(s) = \max\{\pi_n'(s), \pi_n''(s)\}$$ \hspace{1cm} (4)

Where $\pi_n'(s)$ is recursively computable down to unigram, 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 * 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-
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>2.5</td>
<td>32.0</td>
<td>55m13</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 ①.

<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>2.5</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>60m21</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>57m39</td>
</tr>
</tbody>
</table>

Table 6: LMLA computation with approximation line ① (ie. equation 3 is skipped) and ② a Precomputed Bigram Size (PBS) of 400 is used (ie. equation 4 available for the 400 biggest wordlists).

<table>
<thead>
<tr>
<th>Lex</th>
<th>Skip/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>0/0</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>0/0</td>
<td>72.8</td>
<td>17.3</td>
<td>9.9</td>
<td>1.9</td>
<td>29.1</td>
<td>60m21</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 – ② to ① – 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 approximating 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