Fast N-Gram Language Model Look-Ahead for Decoders With Static Pronunciation Prefix Trees

Marijn Huijbregts, Roeland Ordelman and Franciska de Jong

Department of Electrical Engineering, Mathematics and Computer Science,
University of Twente, the Netherlands
{m.a.h.huijbregts, ordelman, fdejong}@ewi.utwente.nl

Abstract

Decoders that make use of token-passing restrict their search space by various types of token pruning. With use of the Language Model Look-Ahead (LMLA) technique it is possible to increase the number of tokens that can be pruned without loss of decoding precision. Unfortunately, for token passing decoders that use single static pronunciation prefix trees, full n-gram LMLA increases the needed number of language model probability calculations considerably. In this paper a method for applying full n-gram LMLA in a decoder with a single static pronunciation tree is introduced. The experiments show that this method improves the speed of the decoder without an increase of search errors.

Index Terms: Automatic speech recognition, decoding, token pruning, language model look-ahead

1. Introduction

Decoding can be regarded as finding the optimal path in a large search space. When the token-passing algorithm is applied, each token represents a possible path in this search space. Especially with large lexicons and language models, the number of possible paths and therefore the number of tokens needed to run an exhaustive search can be dramatically high and the search can easily take up near infinite computational efforts. Therefore, it is very important that the search space is somehow managed and that the number of used tokens is restricted. Typically this is achieved by deleting or *pruning* the tokens that represent the least promising paths.

The risk of token pruning is that the optimal path, i.e. the token with the highest score after all feature vectors have been processed, could be deleted from the search space during one of the pruning runs. In order to prevent this, it is best to perform pruning based on both acoustical knowledge as well as language model information. Because in most decoders, the entire language model score is added to each token at the leaves of the tree, there will be a big gap between the scores of the tokens just before adding the LM priors and just after adding them. In combination with enthusiastic token pruning, this gap might cause too many tokens to be pruned away. This problem can be solved by incorporating language model information at every node of the tree instead of only at the leave nodes. Language Model Look-Ahead (LMLA, [1]) makes it possible to incorporate approximations of the language model probabilities into the search tree at an earlier stage, before pruning is performed. This means that pruning is not only based on acoustical knowledge but also on linguistic knowledge. In [1] it has been shown that LMLA will allow for tighter pruning without loss of recognition accuracy.

For decoders that use copies of their Pronunciation Prefix Trees (PPT) to handle n-gram history, the LMLA probability approximations can be stored directly in the tree copies. Unfortunately, for decoders that don't apply PPT copying but instead use a single static tree, it is not possible to store the LMLA probability approximations directly into the tree [1]. Without taking special measures, for these decoders the LM probabilities need to be looked-up at every time frame. The time needed to do so will diminish the advantage of the smaller search space. In this paper a method will be provided that enables an efficient implementation of LMLA and reduces the LM probability look-up time for decoders that use static PPTs. This method is implemented in a decoder called SHoUT that was developed at the University of Twente.

The remainder of this paper is organized as follows. First, in section 2, the architecture of the SHoUT decoder is discussed. Next, in section 3, the token pruning methods applied by the SHoUT decoder will be discussed and in section 4 an overview of other systems using LMLA is given and the method used by the SHoUT decoder is described. The paper is concluded with experiments on the broadcast news domain (section 5) and with a discussion (section 6).

2. The SHoUT decoder

The Viterbi search of the SHoUT decoder is implemented using the token passing paradigm. Hidden Markov Models (HMM) applying Gaussian Mixture Models (GMM) and n-gram backoff language models are used to calculate acoustical likelihoods of context dependent phones and to calculate language probabilities respectively. The HMMs are organized in a single PPT as described among others in [2, 3]. Instead of copying PPTs for each possible linguistic state (the LM n-gram history), each token contains a pointer to its LM history (see figure 1). Tokens coming from leaves of the PPT are fed back into the root node of the tree after their n-gram history is updated. Only for tokens with the same LM history, token collisions will occur. This means that each HMM state of each node in the single PPT can contain a list of tokens with unique n-gram histories. These lists are sorted in descending order of the token probability scores.

The language model is stored in lookup tables. The first table contains unigram probabilities and backoff values for all words of the lexicon. The statistics for all available bigrams, trigrams and 4-grams are stored in minimal perfect hash tables [4]. These hash tables contain the probability and backoff values of exactly one n-gram item in each slot of the table. This means that no extra memory is needed except for storing the hash function and for the key of each data structure, the n-gram history. The key is needed during lookup to ensure that the cor-

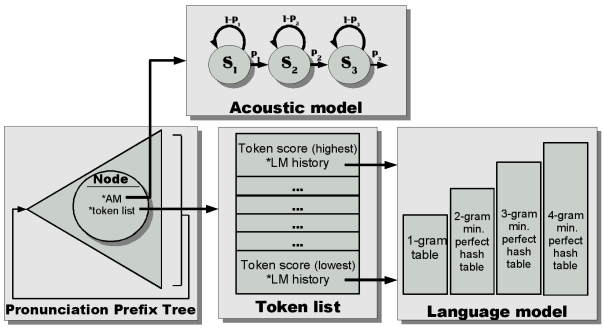

Figure 1: *The modular architecture of the SHoUT decoder. The acoustic model, language model and the pronunciation prefix tree are each implemented in their own module and communicate through a straightforward API. For the token passing administration, token lists are needed.*

rect n-gram is found as the hash function will map queries for non-existing n-grams to random slots. The algorithm proposed in [5, 4] is used to generate the hash functions. The use of a minimal perfect hash table has the advantage that all n-gram probabilities can be found with a single lookup. Therefore, in practice the procedure is as fast as using a cache for often occurring words.

3. Token pruning

3.1. Common pruning methods

Two types of token pruning methods are commonly used in PPT based decoders: *beam pruning* and *histogram pruning* [6]. With beam pruning, tokens with a probability value between the best found probability and the best probability minus a constant *beam* are retained at each time-frame. All tokens that are not within this *beam* are deleted. The SHoUT decoder uses two beam pruning methods. During *global beam pruning* all tokens of the entire PPT are compared to the best scoring token and pruned if necessary. *Word-end beam pruning* is done on all tokens that are at the leaves of the PPT and for which the LM probabilities are incorporated into their probability scores. This pruning method is used to limit the number of tokens that is fed back into the root node of the PPT.

Histogram pruning is also implemented in the SHoUT decoder. Here, only the best N tokens are retained when the number of tokens exceeds a maximum N which significantly restricts required memory [6]. This method is called histogram pruning because for sorting the tokens on their score, often a histogram is used [6]. Simular to beam pruning, histogram pruning is performed both globally (*global histogram pruning*) and in the leaves of the tree (*word-end histogram pruning*).

3.2. Pruning in SHoUT

Global pruning and word-end pruning, both by applying beam pruning or histogram pruning, are commonly used in PPT based decoders. For decoders that use static trees such as the SHoUT decoder, a third pruning method can be used that is not explicitely mentioned in the literature. Instead of only pruning tokens that are in the word-end nodes, pruning is performed in each single node of the PPT. This pruning method, referred to as *single-state pruning*, restricts the length of each token-list. In the SHoUT decoder these token-lists are sorted on token score, which allows those lists to be pruned very efficiently (both for single-state beam pruning and single-state histogram pruning). Although the length of the token-lists does not influence processor load needed for calculating Gaussian Mixtures, it does take a lot of processing time to merge long lists and to calculate LM probabilities for all tokens. When LMLA is used, the lists need to be re-ordered in each compressed node (see below). Therefore restricting the length of the lists by using these two pruning methods should speed up the search considerably.

4. Language Model Look-ahead

Language model knowledge is added to the hypothesis score at the PPT leaf nodes. Beam pruning is done earlier in the tree solely on the basis of acoustic evidence. Incorporating the LM model in an early stage into the tree will make it possible to compare and prune hypotheses on both linguistic and acoustic evidence. LMLA [1] achieves this by calculating for each token in the tree the LM probabilities of all words that are reachable from that token and temporarily adding the best one to the token's score. When the token reaches a leaf node, the temporary LM probability is replaced by the probability of the word represented by the leaf node. Following this procedure, sharper beams can be applied during pruning so that less tokens need to be processed and decoding is speed up considerably. On the down side, calculating all possible LM probabilities for all tokens takes a lot of time. In the literature a number of methods to manage these calculations is proposed. First, these solutions will be discussed and then the solution developed for the SHoUT decoder will be described.

4.1. LMLA in other systems

The least complex way for reducing the number of LM look-ups while applying LMLA is to use unigram probabilities for the look-ahead. By using unigrams the approximation of the best final LM score will be less precise, but it becomes possible to integrate these look-ahead scores directly in the PPT. In this case, each node stores a single value: the difference between the best LM score from before and after entering the particular node. Because only unigrams are used, these look-ahead values can be applied for all tokens, no matter their n-gram history. Unfortunately, it was shown that unigram look-ahead is outperformed by higher order look-ahead systems [1, 4].

Another method to reduce the number of LM look-ups is proposed in [1]. All nodes of the PPT with only one successor node are skipped for calculating the LMLA values. The resulting compressed PPT will never require more nodes than twice the number of words from the PPT, reducing the number of needed LM lookups. The decoder in [1] uses tree copies in order to incorporate the LM probabilities. LMLA is performed on demand whenever a new copy is needed.

In [7] at each node in the compressed PPT a list is stored with all words that are still reachable from that node. For small word lists, the look-ahead value is calculated exactly (each trigram probability is calculated and the best is chosen). Huge word lists, at the root of the PPT, are skipped. For all other lists, the intersection with the n-gram lists are calculated before calculating the LMLA values. This saves a considerable amount of search time for those words that do not have a trigram or bigram LM value.

Similar to the systems described in [4, 8], the SHoUT decoder does not make tree copies. Instead, LM histories are stored in the tokens and the PTT is shared by all tokens. Therefore, Storing the LMLA values directly in the tree is not possi-

ble. To circumvent this problem, in [4] and [8] an LMLA cache is created in each node of the tree. These small caches contain LMLA values of earlier computed LM histories. Although the caches are highly optimized, the procedure takes more time than reading the single values directly when tree copying is applied. The LMLA data structure proposed below makes it possible to obtain a pre-calculated lookahead value in a static tree without searching in a cache.

4.2. LMLA in the SHoUT decoder

Figure 2 is a graphical representation of the data structure used to speed up language model look-ahead in the SHoUT decoder. Each node in the static PPT that has more than one successor or that is a leaf node (each node that is part of the compressed tree as described in [1]) is assigned a unique *LMLA index* value. LMLA probabilities are not stored directly in the nodes, but in *LMLA field* structures. Each structure contains the look-ahead values for tokens with one particular language model history in an array of probabilities *P*. The *LMLA index* of a node points to the corresponding LMLA probability in *P*. The probability array *P* is filled using the dynamic programming procedure described in [1]. Starting at the leaf nodes the LM probabilities are calculated. The probabilities are propagated backwards to the root of the tree and at each branch the maximum probability is selected and stored in *P*. Using this method, each candidate LM probability is calculated exactly one time. Each LMLA field structure is stored in a global hash table.

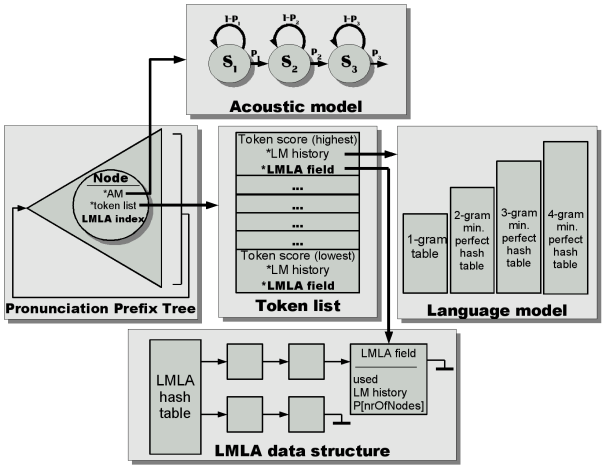

Figure 2: *The architecture of the SHoUT decoder including the module for language model lookahead (LMLA). Also an LMLA-index is added to each node in the PPT and a pointer to an LMLA field is added to each token list (marked in yellow).*

When a new token enters the root of the PPT, the LMLA field structure having the identical LM history as the token is looked-up in the hash table. If a field structure with the same LM history does not yet exist, a new one is created and added to the hash table. A pointer to the LMLA field structure is stored in the token. Once the token is linked to a field, obtaining the LMLA probabilities is straightforward. When a token is propagated to a new node, it uses this node's LMLA index on the probability array of the LMLA field structure. This action only takes two look-ups: retrieving the LMLA index and retrieving the probability. The search through the hash table is needed only once. After that, the time needed for look-ups is negligible.

The boolean parameter *used* of the LMLA field structure is set to true each time the field object is used for a look-up. All LMLA fields that are not used (*used* parameter is false) during a fixed time window are deleted in order to safe memory.

5. Experiments

First, the efficiency of the pruning methods and LMLA is tested in three experiments on the broadcast news domain. Next, two additional experiments are conducted in order to determine if unigram LMLA is equally effective as using n-gram LMLA. All experiments are conducted on Dutch broadcast news recordings. The first set of experiments are performed using our broadcast news development set. The second set of experiments is conducted on the recently made available development set for the Dutch benchmark N-Best [9].

5.1. Pruning and LMLA

Three experiments are conducted to test the pruning methods and the LMLA efficiency. For the first experiment, only global beam pruning, global histogram pruning and word-end beam pruning is used. In the second experiment, state beam pruning and state histogram pruning is used as well. Finally, in the third experiment, also LMLA is applied. In order to ensure that it is possible to compare the experiments purely on basis of their needed processing time, the pruning configuration of each experiment is chosen so that the word error rate is the same for all experiments (29.5%).

The pruning values for the three experiments shown in table 1 are obtained by first setting the global beam threshold so that the WER is 29.5% and then setting the other thresholds as tight as possible without influencing the word error rate.

ID	global beam, hist.	word-end beam	state beam, hist.	LMLA
exp-1	210, 125000	45	no, no	no
exp-2	210, 125000	45	75, 250	no
exp-3	150, 40000	45	50, 250	yes

Table 1: *The pruning configuration of each experiment. The first number in the global end state columns represent the beam value. The second number is the histogram value.*

For decoding the broadcast news recordings, a dictionary containing $65K$ words was used. At the time of these experiments, the decoder was not yet able to handle 4-grams and therefore a trigram LM was used. All experiments are conducted on a machine with a $3.4GHz$ Intel Xeon processor. For each experiment the number of search errors [10], the WER and the real-time factor of the system are calculated. Finally, also the average and maximum number of active nodes and number of tokens needed to decode each sentence are stored. For the third experiment also the maximum and average number of LMLA lookup tables are stored.

5.1.1. Results

For all three experiments, the word error rate was 29.5% and 3.5% WER was due to search errors, showing that the system performance was the same for each experiment. The real-time factor (RTF) of the first experiment (only global and word-end pruning) is 27.4 while the RTF of the third experiment is 10.5. This improvement is obtained because of the drastic reduction of active nodes and tokens due to LMLA. In table 2 all mea-

sured statistics of the three experiments are listed. The unused fields of the LMLA hash table are not deleted every time-frame ($10ms$) but every 25 frames. This means that the actual average *active* fields is less than the 51 mentioned in table 2. The number of tokens is measured each time-frame before histogram pruning. Therefore it is possible to have an average number of tokens that is higher than the histogram pruning threshold.

		Average number of		
ID	RTF	nodes	tokens	LMLA fields
exp-1	27.4	43314	126782	n.a.
exp-2	24.9	43730	124839	n.a.
exp-3	10.5	11395	20686	51

Table 2: *The measured statistics of the three experiments. For all experiments the WER was* 29.5%.

As can be seen from table 2, the second system (with state pruning) is roughly 9% faster than the first system (without state pruning) while the average number of tokens was only reduced by 1.5%. This is explained with the optimization that is possible when single-state pruning is applied: single-state pruning can be applied directly during the merging of two token-lists coming from different HMM states into the same state. Without single-state pruning, all tokens of both lists need to be placed into the merged list, whereas when using single-state pruning, the merged list is finished as soon as the maximum number of tokens is reached (because all token-lists are sorted on token score).

5.2. Unigram LMLA

As discussed before, one solution for reducing the needed number of LM look-ups is to apply unigram LMLA instead of full n-gram look-ahead. In order to prove that this method is not as efficient as full LMLA, two extra experiments were conducted. First the system with full LMLA is run and after that a system that only uses unigram LMLA is evaluated. As before, for these two experiments the pruning parameters are determined so that the WER is equal for both experiments (29.6%) and the performance can be measured by the real-time factor. The experiments are performed on the development set of the NBest task. In table 3 the results of the two experiments are listed. Although in the second experiment, less time was spend in querying LM n-grams, more tokens and token-lists were needed. The need of these tokens slowed down the decoder considerably.

LMLA method	global beam, hist.	word-end beam	state beam, hist.	RTF
Trigram	150, 40000	50	65, 160	14.0
Unigram	160, 85000	50	75, 210	18.9

Table 3: *The settings used for the unigram and trigram LMLA experiments. To obtain a WER of* 29.6%*, for unigram LMLA, wider pruning settings are needed resulting in an increase of the real-time factor of* 35%.

6. Discussion

The experiments discussed in section 5 show that the single-state pruning method reduced the real-time factor of the system considerably and also the architecture for performing LMLA efficiently in static tree based decoders helps increasing the decoder speed performance. It has also been shown that in the SHoUT decoder, this LMLA architecture outperforms unigram look-ahead.

Although the unigram LMLA system was 35% slower than the full LMLA system, it must be noted that the system without LMLA was even 2.4 times as slow as the optimal system. The fact that unigram LMLA already provides a considerable speed-up, and that it is less complex to implement than full LMLA, could be a consideration to chose for unigram LMLA. Also, note that the reported real-time factor results are closely related to the implementation of the SHoUT decoder and that it is possible that, if implemented in another decoder, the RTF gain of the full LMLA system compared to the unigram LMLA system is less distinct. Given this caveat, the experiments with the SHoUT decoder are very promising and it is our believe that full LMLA using the proposed data architecture will also improve the real time factor of other token passing decoders.

Some decoders use pre-compiled caches for LM probability look-up of the most occurring words. This helps because these words are used considerably more often than the remaining words and therefore have a high probability of being looked up. In the SHoUT decoder, no cache is being used, but instead a very efficient LM look-up method (discussed in section 2) is implemented that reduces a regular n-gram query to calculating the key for a minimum perfect hash table and using this key to directly access the probability. A cache might be useful for speeding up the calculation of the key, but the effect of this speed-up will be highly limited.

7. Acknowledgments

The work reported here was partly supported by the bsik-program MultimediaN which is funded by the Dutch government and the EU project MediaCampaign (IST-PF6-027413).

8. References

[1] S. Ortmanns, H. Ney, A. Eiden, and N. Coenen, "Look-ahead techniques for improved beam search," in *proceedings of the CRIM-FORWISS Workshop*, Montreal, 1996, pp. 10–22.

[2] M. Finke, J. Fritsch, D. Koll, and A. Waibel, "Modeling and efficient decoding of large vocabulary conversational speech," in *proceedings Eurospeech'99*, Budapest, Hungary, 1999, pp. 467–470.

[3] K. Demuynck, J. Duchateau, D. V. Compernolle, and P. Wambacq, "An efficient search space representation for large vocabulary continuous speech recognition," *Speech Commun.*, vol. 30, no. 1, pp. 37–53, 2000.

[4] A. Cardenal, J. Dieguez, and C. Garcia-Mateo, "Fast LM look-ahead for large vocabulary continuous speech recognition using perfect hashing," in *proceedings ICASSP 2002*, Orlando, USA, 2002, pp. 705–708.

[5] Z. J. Czech, G. Havas, and B. S. Majewski, "An optimal algorithm for generating minimal perfect hash functions," *Information Processing Letters*, vol. 43, no. 5, pp. 257–264, 1992.

[6] V. Steinbiss, B.-H. Tran, and H. Ney, "Improvements in beam search," in *Int. Conf. on Spoken Language Processing*, Yokohama, Japan, 1994, pp. 1355–1358.

[7] D. Massonie, P. Nocera, and G. Linares, "Scalable language model look-ahead for LVCSR," in *proceedings Interspeech 2005*, Lisbon, Portugal, 2005, pp. 569–572.

[8] H. Soltau, F. Metze, C. Fugen, and A. Waibel, "Efficient language model lookahead through polymorphic linguistic context assignment," 2002.

[9] J. Kessens and D. van Leeuwen, "N-best: The northern- and southern-dutch benchmark evaluation of speech recognition technology," in *Interspeech*, Antwerp, Belgium, August 2007.

[10] L. Chase, "Blame assignment for errors made by large vocabulary speech recognizers," in *proceedings Eurospeech '97*, Rhodes, Greece, 1997, pp. 1563–1566.