EFFICIENT N-GRAM DECODING AND WORD-GRAPH GENERATION IN LVCSR

Jie Zhao

Department for Electrical and Computer Engineering
Mississippi State University, Mississippi State, MS 39762
zhao@isip.msstate.edu

ABSTRACT

N-gram decoding and word-graph generation are expensive processes in large vocabulary continuous speech recognition (LVCSR). Efficient storage and access of the language model (LM) are critical issues in these processes. In this paper, an efficient N-gram decoding and word-graph generation approach is proposed.

1. INTRODUCTION

Large vocabulary continuous speech recognition requires the use of a language model or grammar to select the most likely word sequence from the relatively large number of word hypotheses produced during the search process. Typically, one-pass decoding using an N-gram language model is called N-gram decoding. Alternatively, we can use N-gram language model to generate a word-graph which contains a large number of possible hypotheses first — typically referred to as word-graph generation, then rescore this word-graph with more complex acoustic model or language model in the following pass and get the final hypothesis.

For larger vocabularies, the N-gram language model provides a relatively compact representation of the linguistically probable word sequences, since it provides estimates of the likelihood of the occurrence of a word based on the previously observed $N-1$ words. If the vocabulary size is M words, then to provide complete coverage of all possible word sequences the language model needs to consist of M^N N-grams (*i.e.*, sequences of N words). This is prohibitively expensive (*e.g.*, a bigram language model for a 40,000 words vocabulary will require 1.6×10^9 bigram pairs), and many such sequences have negligible probabilities. However, even though N-gram language models store only a small subset of all the possible sequences of N words, they are still significantly large for large vocabulary applications. Many systems have been devised which try to improve the efficiency of N-gram decoding [1, 2]. An alternative to the one-pass N-gram decoding is word-graph generation. The process of word-graph generation is very similar to N-gram decoding, but instead of keeping only the 1-best hypothesis, it keeps multiple hypotheses at each word end, and outputs them as a word-graph. This word-graph incorporates a large number of hypothesis and defines which words can follow which words. It then can be used as the grammar constraints in a second pass decoding, which is typically referred to as the word-graph rescoring. Because the search space is strictly constrained at the word level by the word-graph, word-graph rescoring is much more efficient than the N-gram decoding. Although the process of word-graph generation is time-consuming, once a word-graph is generated, it can be rescored using more complex acoustic models and language models. This will give an overall better performances in terms of both speed and accuracy. For both one-pass N-gram decoding and word-graph generation, memory issue is a big problem towards efficient decoding.

In this paper, we propose an efficient algorithm to implement N-gram decoding and word-graph generation. In our implementation, we applied carefully designed data structures, memory management and advanced pruning techniques.

2. N-GRAM LANGUAGE MODEL

Every language consists of a sequence of words. Language model is to provide the probability of next word given preceding words. The dominantly used LM in speech recognition is N-gram LM [3], which uses the history of the $n-1$ immediately preceding words to compute the occurrence probability P of the current word. The value of N is typically limited to 2 (bigram model) or 3 (trigram model) for feasibility. Obviously, it is not possible for an N-gram language model to estimate probabilities for all possible word pairs. Typically an N-gram lists only the most frequently occurring word pairs, and uses a backoff mechanism to compute the probability when the desired word pair is not found.

For instance, in a bigram LM, given w_i, the probability of the next word is w_j:

$$\hat{p}(w_j|w_i) = \begin{cases} p(w_j|w_i) & (w_i, w_j) \text{ exists} \\ b(w_i)p(w_j) & \text{otherwise} \end{cases} \quad (1)$$

where $b(w_i)$ is the back-off weight for the word w_i,

$p(w_i)$ is the unigram probability of the w_i

The backoff weight $b(w_i)$ is calculated to ensure that the total probability

$$\sum_j \hat{p}(w_i, w_j) = 1 \quad (2)$$

Similarly, for a trigram,

$$\hat{p}(w_j|w_h w_i) = \begin{cases} p(w_j|w_h w_i) & (w_h, w_i, w_j) \text{ exists} \\ b(w_h w_i)\hat{p}(w_j|w_i) & \text{otherwise} \end{cases} \quad (3)$$

N-gram language models have been very effective in reducing WERs in LVCSR significantly. In a typical SWB type application, a bigram could have on the order of a couple of hundred thousand unique bigrams with sufficient data to estimate their probabilities. This number could grow significantly for an application like broadcast new where a few million unique bigrams could be used. With higher order N-grams, the overall number of unique LM probabilities could be much higher.

Because of the large mount of N-gram models, the representation of N-gram model data structure has direct effect upon the memory size. Figure 1 shows an efficient organization of the N-gram LM. Each order N-gram is stored as a list. Each list is an array of N-gram nodes. Each unigram node points to the head of a set of bigram successors, and each bigram node points to the head of its trigram successors. Each

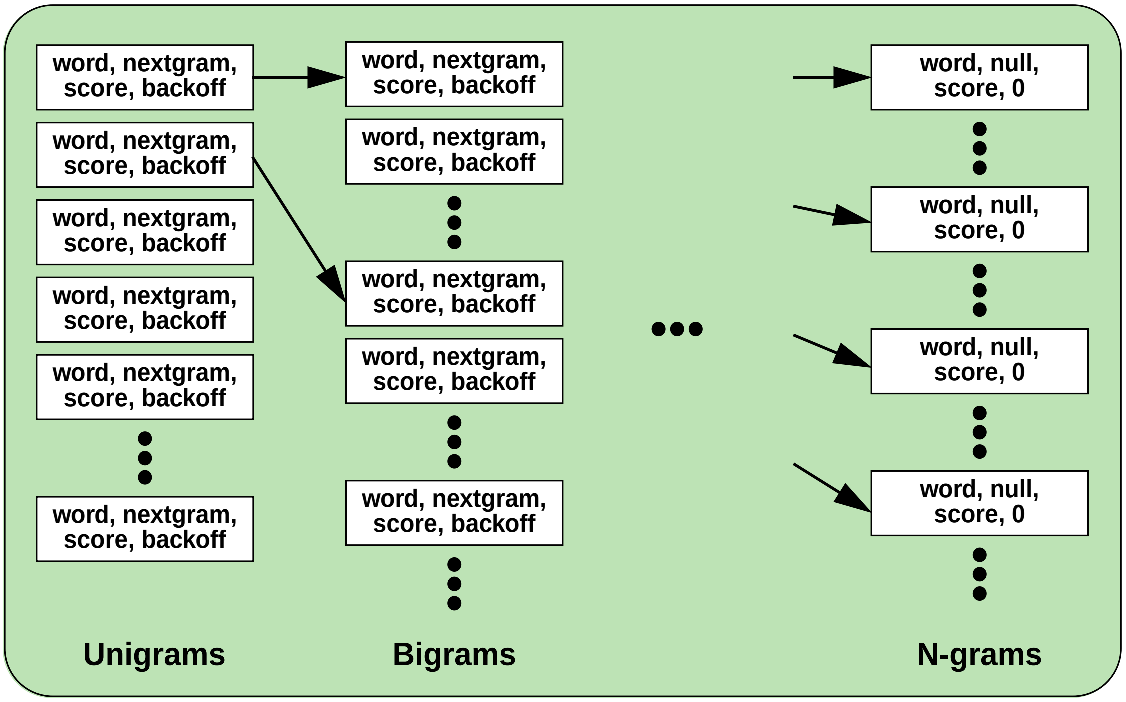

Figure 1: N-gram language model storage scheme

N-gram node has the following components: word identity, N-gram probability, a backoff weight, a pointer to the head of the next order N-gram successors. The word here only records the last word of an N-gram. For example, a bigram should be a word-pair. Since all the bigram followers of a single unigram are grouped under the unigram, it is only necessary to record the second word of this bigram in its data structure. This scheme can save a large amount of space to store the words.

The storage can be further minimized if we distinguish the highest-order N-gram list with others, because the highest-order gram doesn't have backoff weight and pointer to the next N-gram list, and usually the highest order LM has a very large amount of N-gram nodes. In addition, a lookup table can be used to minimize the total cost to store the LM and backoff score. That means to quantize those float numbers and limit them to a certain amount. Suppose we save 4bytes on each N-gram node, and there are total 500,000 N-gram nodes, then we can save 16MB memory.

3. N-Gram Lexical Tree

In LVCSR, the search space is usually organized as a lexical tree [1]. An example of the lexical tree is shown in Figure 2. Each lexical node contains a list of the words on that path covered by the monophone held in the lexical node. The dark circles represent starts and ends of words, but the word identity is unknown until a word-end lexical node is reached. This lexical tree is used to generate phone hypotheses. The scores for word transitions are computed based on the position in this lexical tree and the current N-gram history.

Typically, in lexical tree-based searches, the LM scores are stored in the lexical tree nodes. At each lexical node, we maintain a list of next words. When a word end is reached during the decoding process, the decoder constructs a new lexical tree encompassing all the possible next words which are used to generate the next phone hypotheses. For word-graph rescoring, constructing new lexical trees is necessary since the possible next words are different for each word, and the number of possible next words is relatively small. However in N-gram decoding and word-graph generation using N-gram LM, each word (except the sentence start word) can be followed by any other word, so for large vocabulary speech recognition, the lexical tree would be very large. Even a few copies of the complete lexical tree will quickly overshoot the available memory. Since we know that the N-gram lexical tree structure is the same for all occurrences of each word, and that only the LM scores vary according to their N-gram histories, making many trees with the same structure is a waste of both time and memory. Therefore, instead of making many copies of the lexical tree, we reuse the same N-gram lexical tree for each occurrence of a word, disconnect the history words and LM scores from the lexical tree.

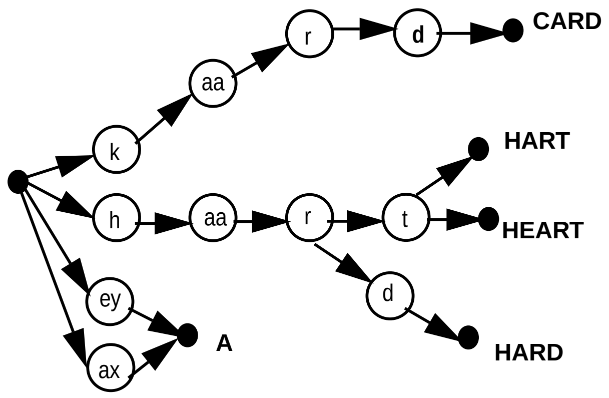

Figure 2: An example of a lexical tree

A data structure called instance is introduced to represent the unique position in the search space which is determined by the corresponding history word(s), the lexical node and the phone. LM lookahead is performed at each lexical node in order to apply the language model as early as possible [4]. The highest language model score is acquired based on the current word and the history word(s), and is stored in the instance. Two instances can be merged if they have the same word history, the same lexical node and phone, i.e. at the same position in the search space regardless of time. Now when a word end is reached, this word and its N-2 previous history words (if any) will become the new history words for the coming word. Since the instances are reused in the decoder, the score calculation needs to be done only once. This minimize the total access times to the N-gram.

4. PRUNING

To constrain the computing and memory resources space, it's imperative to identify less-likely partial paths and stop them from growing further. Multiple pruning criteria are used to remove such paths from decoding process [5].

Firstly, beam pruning is used. A path is pruned away if its path score is below the current best hypothesis score minus a specified beam width. Beam width can be different at each level in the search hierarchy. Usually heavier pruning is applied at word end because there are much more possible next words at word end than possible next phones at phone end. Secondly, we set a maximum number of words allowed at each frame. Thirdly, instance pruning is applied at each frame, which is limiting the number of model instances active at a given time. The instances are sorted according to their scores. Only the instances having better scores can survive.

Pruning has dramatic impact on the search speed and memory cost. Heavy pruning with less lose of accuracy is important for efficient N-gram decoding and word-graph generation.

5. EVAULATION

The performance has been evaluated using by word-graph generation followed by a word-graph rescoring on the SWB part of Hub-5 evaluation data set. The Language model being used is a pruned trigram backoff language model provided by SRI [6]. It was trained from Switchboard, Callhome and Broadcast news. A bigram version was used for word-graph generation, and a trigram version for rescoring word-graphs. Acoustic models are 16-mixture cross-word triphones models trained on 60 hours of SWB-I and 20 hours of English Call Home data.

The overall recognition WER is 42.9%. The memory usage varies with the length of the utterance. On a 5-second utterance, the maximum memory usage during decoding is about 500M.

6. SUMMARY

We introduced an N-gram decoding and word-graph generation algorithm. Comparing with our previous implementation [7], we have made improvement on performance in terms of both accuracy and memory usage. Further enhancement can be made by using disk-based LM [8], where chunks of the LM are brought into memory on-demand from the disk. This requires the LMs to be organized efficiently in both memory and disk to avoid long access times.

REFERENCES

[1] J. Odell, *The Use of Context in Large Vocabulary Speech Recognition*, Ph.D. Thesis, Cambridge University, 1995.

[2] J. Odell, V. Violative and P. Woodland, "A One-Pass Decoder Design for Large Vocabulary Recognition," *Proceedings of the DARPA Human Language Technology Workshop*, March 1995.

[3] R. Rosenfeld, "*Adaptive Statistical Language Modeling*," Ph.D thesis, Carnegie Mellon University, Pittsburgh, PA, USA, April. 1994.

[4] S. Ortmanns, H. Ney and A. Eiden, "Language Model Look-ahead for Large Vocabulary Speech Recognition", *Proceedings of the Fourth International Conference on Spoken Language Processing*, pp. 2095-2098, October 1996.

[5] Deshmukh, A. Ganapathiraju and J. Picone, "Hierarchical Search for Large Vocabulary Conversational Speech Recognition," *IEEE SignalProcessing Magazine*, vol. 16, no. 5, pp. 84-107, September 1999.

[6] SRI Language Modeling toolkit, *http://www.speech.sri.com/projects/srilm*, SRI international Corporation, CA.

[7] A. Ganapathiraju, N. Deshmukh, and J. Picone, "ISIP Public Domain LVCSR System," *Proceedings of the Hub-5 Conversational Speech Recognition (LVCSR) Workshop*, Linthicum Heights, Maryland, USA, June 1999.

[8] Mosur K. Ravishankar, *Efficient Algorithms for Speech Recognition*, Ph.D. Thesis, Carnegie Mellon University, 1996.