TIME AND MEMORY EFFICIENT VITERBI DECODING FOR LVCSR USING A PRECOMPILED SEARCH NETWORK

Daniel Willett, Erik McDermott, Yasuhiro Minami, Shigeru Katagiri

Speech Open Lab, NTT Communication Science Laboratories
2-4, Hikaridai, Seika-cho, Soraku-gun, Kyoto, Japan
{willett,md.minami,katagiri}@cslab.kecl.nctt.co.jp

ABSTRACT

In this paper, we present our recently developed time-synchronous speech recognition decoder, which adopts the idea of representing the search space of Large Vocabulary Continuous Speech Recognition (LVCSR) in a single precompiled network. In particular, we outline our approaches for time and memory efficient Viterbi decoding in this scenario. This includes reducing the fast memory needs by keeping the search network on disk and only loading the required parts on demand. Evaluations are carried out on a difficult Japanese LVCSR task which involves a back-off trigram language model and full cross-word dependent triphone acoustic models. Time and memory efficiency enables the real-time Viterbi decoding of entire lecture speeches in a single time-synchronous pass with a search error of less than 1%.

1. INTRODUCTION

The organization and detailed time and memory efficient implementation of the Viterbi decoding procedure for LVCSR has long been a major issue of discussion [1]. Recently, it was proposed to make use of a single precompiled network in which all the different levels of statistical and phonological modeling are combined. This straightforward approach was before thought to be simply not feasible for large scale LVCSR task.

However, the steady increase of computational power and memory size of standard workstations combined with the idea of interpreting this search network as a weighted Finite State Transducer, which allows the application of several analyses of network processing from this field, proved to make this approach feasible in a couple of studies [2, 3].

2. DECODER OVERVIEW

Our recently developed time-synchronous decoder adopts exactly this concept. For further readings on how to set up and compose weighted Finite State Transducers that represent the various modeling levels of state-of-the-art LVCSR consult [4, 5] and referred literature. Figure 1 shows an example of a precompiled search network for a simple task of two words and a back-off bigram language model. In the terminology of weighted Finite State Transducers, the network arcs are labeled with weights, input and output symbols. In our case of a search network for LVCSR, the weights represent language model likelihoods and possibly HMM transition probabilities, the input symbols are HMMs or HMM states that are consumed along the paths and the output symbols represent the dictionary words that are the result output associated to a specific path.

In this type of precompiled network multiple search techniques can be applied. We chose a fully time-synchronous Viterbi beam-search. The organization of this search follows the principles of Token-Passing as introduced in [6]. Tokens serve as a means for tagging the activated nodes and arcs during search. In addition to that, they store the log-likelihood of the best path leading to the specific network position and the link into the result lattice. This lattice structure is set up during decoding in order to enable the traceback of the best matching word sequence after the final feature frame has been propagated. At the beginning of the search, only the entry node holds a token. Then, as the major loop of the Token-Passing procedure, with each time frame the tokens are updated and propagated through the network, while the appropriate transition and observation probabilities are added.

Figure 2 illustrates the decoding and lattice generation procedure. It shows excerpts of the recognition network and of the result lattice. For reasons of simplicity, in this figure the arcs are regarded as representing one-state HMMs only and it is assumed that the three visible tokens are the only ones currently active in the network.
3. TIME EFFICIENCY ISSUES

3.1. Network setup

Determination and minimization [4] proved to be useful approaches for reducing the size of the search network and thus for indirectly speeding up decoding using the network. In addition, the factorization of the network weights, often referred to as language model look-ahead, is of highest importance for good beam-search performance, just as it is when the network is not precompiled, but composed during search [7]. In [3], the usage of heterogeneous model units has been proposed in order to model any linear sequence of HMM states as a single model unit. Our decoder also makes use of these concepts. Here, we do not want to further focus on these topics, but on how to organize efficient Viterbi decoding using the precompiled network.

3.2. Beam reference

Concerning the application of the pruning beam of conventional beam-search, there are several alternatives regarding which log-likelihood threshold to refer to. One is to use the best score of the previous time-frame, another is to use the best of the current frame with a continuous update whenever a new best hypothesis is found. Our decoder is capable of both strategies and the evaluations show that the continuous update method is superior.

When continuously updating the pruning log-likelihood whenever a new best score is found, finding a good, possible the best, hypothesis early during the token propagation should lead to particularly good beam-pruning performance. Therefore, we propose a dedicated beam-pruning mode in which the token holding the best score of the previous frame is evaluated and propagated first. This token has high potential of resulting in the best or at least a good score at the current frame as well.

4. MEMORY EFFICIENCY ISSUES

Despite all minimization techniques, precompiled search networks that include full cross-word dependency and trigram language models grow extremely large. The tremendous fast memory requirement is therefore probably the major drawback of the approach of performing the LVCSR search in a precompiled search network. The following two issues we found of particular importance in terms of the decoder’s memory efficiency.

4.1. Disk-based search network

In order to tackle the problem of large memory requirements needed for storing the precompiled search network, we propose and implemented an approach of keeping the network on disk and only loading the required parts on demand. This is inspired by an approach in [8] for reducing fast memory requirements when coping with large n-gram language models. The idea is to keep the search network on disk in a dedicated file structure, to load a node and its outgoing arcs only when a token reaches it and to unload it, as soon as it is not longer activated.

The structure of the involved files is illustrated in Figure 3. A file called nodes-file contains only a link per network node. This link, which actually is a file position mark, points to the starting position of the specification for this specific node which is stored in a second file, named arcs-file hereafter. This file contains for each node the node id, the information on whether it is a final node or not and the number of arcs leaving the node. The specification of these arcs follows straight after these entries. The arcs represent arbitrary linearly arranged sequences of HMM states. Thus, the input symbol specification of each arc consists of the number of HMM states represented by the arc, these HMM states’ ids and their self-transition probabilities. In
addition to that, the specification of each arc consists of the node id of the node that the arc is pointing to, which directly represents the node’s position in the **nodes.txt** file; the log-probability to add when traversing the arc and finally the output symbol. The output symbol is given simply as the word’s address (file-position) in a third file, **strings.txt**, which contains all the possible output symbols, and which in our realization is loaded completely into memory, because even for tasks with hundreds of thousands of output words, its size remains reasonably small (< 1 MB).

At the beginning of the search, the only starting node is loaded into memory and activated. Loading a node always includes loading the specification of the specific node’s outgoing arcs from the **arc.txt** file. Then, during search, a new node is loaded whenever a token is about to be propagated from an incoming arc into a node which cannot be found among the currently loaded ones. A loaded node is held in fast memory as long as the node itself or at least one of its outgoing arcs is activated, i.e., holds a token.

### 4.2 Result lattice garbage collection

Depending on utterance length and pruning beam width, the result lattice grows extremely large. Even with a medium sized beam, it easily grows to thousands of links per feature frame which results in lattices of tens of millions of links when decoding long utterances such as those of up to 30 minutes processed in the following section on evaluations. This circumstance is not specifically caused by the approach of using a precompiled search network, but it becomes prominent with the gained ability of performing a time-synchronous beam-search using the full detailed modeling structures, like cross-word triphones and such.

In order to limit the memory needs for storing the result lattice, we apply a garbage collection strategy which, in an adjustable frequency of every nth feature frame, traces back all currently active tokens and marks those lattice links that are not part of any of the traceback as dead, so that the memory they consume can be reused as the lattice continuously grows in time. This procedure hardly slows down decoding to any measurable extent.

From Figure 2, it is obvious that, with no link left from a token of the recognition network into the upper "on"-"new" branch of the result lattice, this part of the lattice will be found dead at the next garbage collection so that the memory it consumes can be reused for new nodes and links as the lattice grows in time.

### 5. Evaluation

The evaluations are carried out on a difficult Japanese task that consists of four full unsegmented lecture speeches with a total length of 97 minutes. The data is part of a Japanese spontaneous speech database that is currently being set up [9]. A 16k dictionary with a single pronunciation per word and a trigram language model trained on lecture speech transcriptions is applied. Dictionary and language model were provided within the Japanese Spontaneous Speech Research Program [11]. The three-state cross-word dependent triphone models use mixture distributions of 16 Gaussians per state. Tree-based clustering lead to 2000 distinct physical states. A feature vector of 25 elements, 12 MFCC coefficients, Δ and energy, is extracted every 10 ms. Starting from speaker-independent acoustic models trained on read speech, for each of the four speakers the models have been improved by means of unsupervised adaptation [10].

The size of the minimized networks with the trigram, dictionary and full cross-word dependency compiled into is shown in Table 1. The three rows show the network size when using different types of model units as input symbols of the network arcs. The step from HMMs to HMM states leads to a surprisingly moderate increase in network size. This is due to the use of tree-based clustering information which results in merging of HMM states that share the same output distribution. Regrouping linearly arranged arcs to single arcs that represent linear sequences of HMM states then leads to a good reduction of network size. As in our decoder, these heterogeneous model units can be handled more efficiently than linearly arranged single state arcs, only this network of row 3 is used in the following experiments.

Table 2 lists the recognition performance, run-times and the average number of active tokens per frame measured for different beam-widths and beam-pruning strategies. Run-times are given as Real-Time-Factors (RTF), measured on a 667 MHz Compaq-Alpha machine. The average number of active tokens reflects the average number of active network nodes and arcs. The method of pruning hypotheses relative to the current best (ref) slightly outperforms beam-pruning relative to the best score of the previous frame (prev). In (ref+), token propagation first propagates the token which holds the best hypothesis of the previous propagation, as proposed in Section 3.2. This results in another slight improvement over (ref) where this is not taken into consideration. In this configuration, real-time performance comes along with only 1% of additional search error. The overall performance of 93.5% compares well to other approaches of automatic transcription of this task [12, 13, 14].

In the experiments conducted for Table 2, result lattice garbage collection has been applied in order to limit the

<table>
<thead>
<tr>
<th>type of model unit</th>
<th>states per unit</th>
<th>number of nodes</th>
<th>number of arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMMs</td>
<td>3</td>
<td>1,500,000</td>
<td>6,300,000</td>
</tr>
<tr>
<td>HMM states</td>
<td>1</td>
<td>1,500,000</td>
<td>7,500,000</td>
</tr>
<tr>
<td>Linear seq. of HMM states</td>
<td>1 to 87</td>
<td>600,000</td>
<td>6,900,000</td>
</tr>
</tbody>
</table>

**Table 1:** Size of the precompiled search networks

<table>
<thead>
<tr>
<th>beam ref.</th>
<th>beam width</th>
<th>activated tokens per frame</th>
<th>recognition accuracy [%]</th>
<th>run-time [RTF]</th>
</tr>
</thead>
<tbody>
<tr>
<td>prev</td>
<td>small</td>
<td>4,700</td>
<td>56.3</td>
<td>1.1</td>
</tr>
<tr>
<td>prev</td>
<td>wide</td>
<td>12,400</td>
<td>55.5</td>
<td>3.3</td>
</tr>
<tr>
<td>ref</td>
<td>small</td>
<td>4,700</td>
<td>56.6</td>
<td>1.1</td>
</tr>
<tr>
<td>ref</td>
<td>wide</td>
<td>11,500</td>
<td>55.5</td>
<td>3.0</td>
</tr>
<tr>
<td>ref+</td>
<td>small</td>
<td>4,700</td>
<td>56.6</td>
<td>1.0</td>
</tr>
<tr>
<td>ref+</td>
<td>wide</td>
<td>11,500</td>
<td>55.5</td>
<td>2.9</td>
</tr>
</tbody>
</table>

**Table 2:** Performance with different beam configurations
size of the result lattice, and so to enable the decoding of the full lecture speeches of up to 30 minutes in a single pass without presegmentation. Table 3 shows the result lattice size with and without result lattice garbage collection for the different beam widths. Only the shortest of the four speeches is used here. It is about 12 minutes long which results in 72k feature frames. Even with the small beam, the lattice grows to a size of around 50 million arcs which can be reduced to around 4% by result lattice garbage collection.

Table 4 finally evaluates the proposed disk-based recognition mode. In the experiments above, the whole recognition network has been loaded into fast memory. This consumes around 200 MB on the Alpha machine that the tests were run on. The table shows the maximum number of nodes in memory and decoding times measured using the disk-based mode with different beam widths. It is obvious that the disk-based mode leads to a drastically decreased number of network nodes in memory. The fast memory required for storing at most 5,500 nodes (and their outgoing arcs) is less than 10MB in our implementation. The reduced decoding speed that is due to disk-access operations and additional overhead might be tolerable depending on application and available hardware.

6. CONCLUSION

The paper has outlined the principal design of our recently developed speech recognition decoder and has given a couple of details of implementation. Furthermore it has described our first efforts and experiences in using it for decoding in the LVCSR domain. The decoder achieves real-time performance with a very small search error on a difficult LVCSR task, for which we managed to compile the trigram language model as well as full cross-word context dependent models into the search network. Lattice garbage collection enables the decoding of complete lecture speeches in a single pass without presegmentation. The approach of keeping the search structure on disk and only loading parts of it into fast memory on demand, proved to be an effective mean for vastly reducing the fast memory needs of this approach to LVCSR decoding. Our future work will focus on how to set up search networks that offer particularly good beam-search performance and on memory efficiency issues.

Acknowledgement. We thank the Japanese Science and Technology Agency (Priority Program "Spontaneous Speech: Corpus and Processing Technology") for providing the speech data, language model and dictionary, and also the IPA project for providing the acoustic models. Furthermore, we thank MIT's SLS group for providing the tools that network minimization was performed with.

7. REFERENCES


