An Implementation of a Token Pass Decoder

Janne Pylkkönen

Laboratory of Computer and Information Science
Helsinki University of Technology, Finland
janne.pylkkonen@hut.fi

Abstract

The concept of token passing for speech recognition was introduced over 15 years ago. The most popular decoding technique for large vocabulary continuous speech recognition (LVCSR) nowadays is the one-pass time-synchronous beam search strategy, which is still based on that same principle. The key advantage of token passing is the conceptually simple approach, which makes it possible to extend the strategy to handle many advanced problems in speech recognition, such as cross-word contexts and early language model pruning.

The goal of this project was to implement a token pass decoder to the existing CIS-HUT LVCSR framework. The key design issues were to have a rather simple decoder based on lexical prefix tree and beam search, which could be extended and modified along time depending on research needs. To have the decoder perform well enough, it was necessary to implement language model lookahead. The performance of the decoder was compared against the existing stack decoder, showing that the new decoder performs almost equally well. The advantage with the new decoder is its simple structure and architecture, which allows it to be extended so that its performance could be made to somewhat equate to those of the state-of-the-art decoders.

1. Introduction

A decoder is the part of the speech recognizer which does the actual recognition, that is, finds the word sequence which best matches the input signal. This operation can be described with a simple equation (see [1]):

$$\hat{W} = \underset{W}{\operatorname{argmax}}\{P(O \mid W)P(W)\}, \qquad (1)$$

where O is the acoustic observations and W is a sequence of words.

The naive approach, according to (1), is therefore to try all combinations of words in the vocabulary. It is easy to show that this becomes very quickly infeasible. Assume we have a relatively small vocabulary of 10000 words. Further suppose we have a fast computer which can evaluate 10^7 word combinations in a second. Even with these restrictions, decoding a four-word sentence would require evaluating 10^{16} word combinations, which would then take 10^9 seconds, or about 31.7 years!

From this kind of consideration it is clear that the main difficulty which a decoder has to face is how to reduce the search space so that the decoding task becomes feasible. Fortunately there are several powerful design principles which allow us to do decoding with even larger vocabularies and slower machines than was assumed in the previous example, and still achieve near real-time performance. The speed up does not come without a cost, so another issue is the decoding accuracy. With good design, these both can be combined fairly well.

2. Decoder designs

2.1. Different approaches

In [1] Aubert classifies different decoding methods to three broad classes. The first class is a static network, which includes all the knowledge sources (acoustic and language models) in the same search network. The most popular approach using this design is the weighted finite state transducer (WFST) method.

The other two classes are based on a dynamically expanded search network, in which the knowledge sources are not statically combined. Usually this means building a static lexical network, and explicitly handling the language model during the search. The difference between the two classes using this approach is whether the search proceeds in a time-synchronous or asynchronous manner. The former can be further divided to re-entrant tree and start-synchronous tree approaches.

All of these decoder designs still share some mutual characteristics, which allow them to reduce the search space to feasible extents. One is the use of redundancies in the knowledge sources. WFST method can reduce the search network to surprisingly compact format by utilizing redundancies in language models, lexicon and acoustic models. The approaches based on dynamically expanded network usually only concentrate on the latter two. By building the search network as a lexical prefix tree, the words sharing the same beginning can be combined together, reducing the search effort by almost an order of magnitude [2].

Another common characteristic between the decoder architectures is the search strategy. Even with a compact search network it is not possible to search through all the paths. Instead, only the most promising paths are kept active, resulting in a so called beam search strategy. The most common method of implementing this is the token pass algorithm [3].

2.2. The selected architecture

Of all the designs described in the previous section, the most popular appears to be the dynamically expanded time-synchronous search based on a re-entrant lexical tree (see e.g. [4]). It's popularity is probably due to conceptual simplicity and the possibility to address several advanced problems, such as early language model pruning and the utilization of cross-word triphone contexts.

This same design was also selected as the architecture for the decoder in this project. As the decoder is mainly aimed for research purposes, the simplicity and easy extendibility are important issues. The actual search is done as a beam search based on token passing, which has been proved to be an efficient decoding paradigm.

The large vocabulary continuous speech recognition (LVCSR) framework used at the laboratory of computer and information science of Helsinki university of technology has previously included a stack decoder, which used start-synchronous search based on time-synchronous lexical trees. This can be considered as the baseline for the new decoder, so that there is something to compare the performance to. However, it must be noted that in such a short time period this project was completed, it is difficult to optimize all the parameters of the decoder. Therefore the performance of the new decoder may not be quite the optimal, whereas the parameters of the stack decoder has been optimized for years for best performance.

One of the reasons for implementing a new decoder is the restrictions implied by the architecture of the stack decoder. One example is the possibility to use cross-word triphone contexts, a feature which could not be implemented to the stack decoder but which is a feasible extension to a basic token pass decoder. The possibility to implement this feature was considered during the project, but the actual implementation was not made.

3. Project objectives

To restate the project's main objective, this work involved implementing a token pass decoder to the LVCSR framework used at the laboratory of computer and information science of HUT. The goal was to have a flexible decoder, which can be easily extended and used for research purposes. The chosen architecture for the decoder was the time-synchronous search based on a re-entrant lexical tree, implemented as a token pass beam search. If properly tweaked, it should be able to perform comparatively to the state-of-the-art decoders. This was not yet achieved, although one technique which can be considered as advanced, namely the language model lookahead, was already implemented to the decoder during the project.

For the new token pass decoder, some of the code from the stack decoder already included in the LVCSR framework could be reused. However, all parts specific to the token pass decoding, namely the construction of the lexical prefix tree and the actual token pass beam search, had to be made from scratch. The performance of the token pass decoder was tested against the existing stack decoder.

When the project objectives were first defined, it was stated that the basic implementation of decoder was the primary goal. Two specific extensions was given as optional features which were considered during the project. Only one of these, language model lookahead, was implemented. However, the architecture of the decoder is such that it does not restrict the implementation of the other extension, the cross-word triphone contexts. Due to time constraint, it was not implemented as a part of this project, but was left as a possible future improvement.

Next, some of the implementational issues are considered in more detail.

4. Lexical prefix tree

Lexical prefix tree means that the lexicon is constructed as a tree, where the words share their common beginnings. This greatly decreases the number of tree nodes required. Comparing to the usual approach of sharing only those triphones which have been completely tied together, that is, have identical HMM states, a slightly more general approach was taken. It is possible, for example, to have two triphones which only differ in their last HMM state. In these kind of cases, the first states of the triphones can still be shared. Hence, the tree construction is based on HMM states rather than triphones. This approach is convenient also because now it is not necessary to explicitly tie triphones with identical HMM states, as the triphones are defined only with means of the HMM states.

The procedure of tree construction is as follows. The words are added to the tree one after another. The triphone list of the word is expanded to the list of HMM states, and the tree is traversed according to this list. Once a node is found from which a branch to the next state doesn't exist, a new branch to a new node is created. Some additional processing is also performed to allow different HMM topologies. The tree construction requires the HMMs to have only left-to-right transitions, but it allows multiple transitions and skip states to exist. To the word end a dummy node is inserted, which includes the word ID and an arc to the beginning of the tree, but it does no have any HMM state associated with

it. In decoding, these kind of nodes are passed immediately, they do not consume an observation frame.

5. Beam search

The basic unit of the search is a token, which incorporates knowledge about current likelihood (both acoustic and language model), search position in the lexical tree and the word history of the speech recognized so far. At each frame every active token is propagated, which means moving them with each transition available at their current position. While propagating the tokens, a beam pruning is applied. This means that the likelihood of a token must be within defined limit compared to the best performing token at each frame. If this is not true, the token is pruned. Also the number of tokens is controlled so that if it exceeds a defined maximum level, the worst tokens are pruned away to meet the maximum limit. The former pruning is so called beam pruning, the latter is histogram pruning.

Two different beam limits have been defined. The other is the global beam, which is applied to every token. It is important to note that the sooner the pruning is applied, the more effective it is. That is why the beam pruning is applied already while propagating the tokens, even though the best token is not yet known. The pruning is then done using the best token likelihood encountered so far, and after all the tokens have been propagated, another pruning round is performed.

Another beam value is reserved for the word ends. This is important, because from word end position a token is propagated back to the root of the lexical prefix tree, the position which contains the most branches and is therefore the most costly to have a token in. The word end beam is compared to the best token in word end position, and the beam value is kept much lower than the global beam (about a half of it).

To have a little more performance, histogram pruning is avoided by adaptively lowering the beam limits if there were more tokens than the defined maximum number. If after that the maximum number of tokens is not met, the beam limits are gradually increased to their real values.

6. Language model

6.1. Morph language model

The main target language of the decoder was Finnish. Finnish is a highly inflectional language, which poses some problems to speech recognition as it is not possible to list all the possible words. To overcome this problem, the recognition is done in means of sub-word units instead of words. In the CIS-HUT LVCSR system units called morphs are used. These are morpheme-like units which have been found in an unsupervised manner from a large corpus.

The benefit of a morph language model is that the number of units can be quite low. The lexicon recently used for decoding contains about 26000 morphs. On the other hand, the language model order may need to be higher than the usual three to achieve the same level of language knowledge. The decoder must then be able to cope also with higher order language models, and the current implementation indeed does not restrict that order.

One further issue for the decoder which has to be addressed when using a sub-word language model is the explicit inclusion of word boundaries. As the basic recognition unit is shorter than a word, there is no word boundary between all the units, but they have to be explicitly inserted. Unlike for the sub-word units, there may not be acoustical evidence (that is, silence) for these word boundaries, so they must have special treatment. This is currently handled by simply making two different path histories for tokens in word end nodes. For the other just the word is appended to the word history, for the other also the word boundary is appended. These word histories are assigned to different tokens, so they both continue propagating until one or both of them are pruned. It should be noted that at the point of this word boundary insertion, the only difference between these two tokens is in their language model score.

6.2. Language model lookahead

In its basic form, the token pass decoder achieved similar accuracy as the stack decoder, but was somewhat slower (about 1.5 times). To improve the performance, a technique called language model lookahead was implemented. At first, the method described in [5] was taken as the reference, but for the current implementation it was further simplified.

Language model lookahead means that the language model score is taken into account as early as possible. This is a problem with lexical prefix trees, where the word identity is only known at the final nodes of the tree. However, already early in the tree, the number of different word alternatives becomes quite small. The idea in the language model lookahead is then to compute the language model scores of these alternatives, and use the maximum of these as an estimate for the final language model score. This makes it possible to prune the tokens earlier.

Language model lookahead requires lots of computation, and therefore its implementation is crucial to the performance. The first approximation which was made is that the early language model scores are computed as bigram scores. This speeds up the computation and also reduces the number of different contexts, so that more tokens will be able to use the same lookahead scores. Another simplification, unlike done in [5], is that the lookahead scores are only computed after the first branch from the lexicon tree root, not in further nodes.

To avoid computing same language model scores

Table 1: Comparison of the two decoders.

Decoder	Word error	Phoneme error	RT factor
Stack	15.5%	2.58%	4.3
Token pass	15.8%	2.74%	3.4

again and again, a two level cache for the scores was implemented. The first level contains the actual scores for recent contexts, of which 500 was noted to be quite enough. This was done with a hash technique, so that the correct score table is found quickly. In addition, a queue was implemented on top of the hash, so that it is possible to remove the least referenced item from the cache. Another cache was implemented to the tree node level, which included the maximum language model score for different contexts for that particular sub-tree. This was implemented as a simple hash, so that a colliding item replaces the previous one.

7. Decoder evaluation

The new token pass decoder and the old stack decoder were compared in a speech recognition test. As both of these decoders were implemented in the same framework, it was easy to run the tests with the same settings, namely the same acoustic and language models.

The evaluation task was a speaker dependent LVCSR task. Preliminary testing and parameter optimization (beam parameters and the scaling of transitions, duration models and language model) was done with a separate 19-minute development set of the same material. The length of the actual evaluation set was 27 minutes.

As a language model, a morph lexicon of 26000 morphs was used with a 4-gram language model. Acoustic models were tied triphone HMMs, with 4094 states, each having a GMM of 8 Gaussians. These models were trained from a training set of about 12 hours.

The decoder pruning settings were adjusted so that the decoders performed about as accurately as possible. The parameters of the token pass search were as follows: maximum number of tokens: 30000, global beam: 240, word end beam: 120, language model scaling: 28, duration model scaling: 4.

The results of the evaluation are shown in Table 1. It can be seen that the accuracy of both decoders is almost the same, the token pass decoder had a phoneme error rate 6% worse than the stack decoder. However, its real time factor was 20% smaller. But to be fair, the stack decoder suffered some speed lost due to tied triphone states, which it wasn't designed to handle well.

8. Conclusions

During this project, a token pass decoder was implemented to the CIS-HUT LVCSR framework. After extending the basic implementation with language model lookahead technique, it achieved almost the same accuracy as the stack decoder, but with 20% decrease in decoding time. With the current implementation the token pass decoder did not add any modeling capabilities compared to the stack decoder, so no accuracy gain was even expected.

The performance of the implemented token pass decoder shows that the time synchronous beam search is an efficient search strategy for decoding and the current implementation is reasonable. Due to its simple design, the token pass decoder should be more flexible for future improvements than the existing stack decoder. One such extension is the capability to handle cross-word triphones, something that is almost impossible to be implemented to a stack decoder.

9. References

[1] Aubert, X. L., "An overview of decoding techniques for large vocabulary continuous speech recognition", Computer Speech and Language, 16(2): 89–114, 2002.

[2] Ney, H., Haeb-Umbach, R., Tran, B.-H., Oerder, M., "Improvements in beam search for 10000-word continuous speech recognition", Proc. ICASSP, 1992, pp. 9–12.

[3] Young, S. J., Russell, N. H., Thornton, J. H. S., "Token passing: a simple conceptual model for connected speech recognition system", Tech. Report, Cambridge University Engineering Department, 1989.

[4] Aubert, X. L., "One pass cross word decoding for large vocabularies based on a lexical tree search organization", Proc. EUROSPEECH, 1999, pp. 1559–1562.

[5] Ortmanns, S., Eiden, A., Ney, H., "Improved lexical tree search for large vocabulary speech recognition", Proc. ICASSP, 1998, pp. 817–820.