LDec: One Pass Time Synchronous Decoder

Tomáš Pavelka and Václav Matoušek

Laboratory of Intelligent Communication Systems,
Dept. of Computer Science and Engineering,
University of West Bohemia in Plzeň, Czech Republic
E-mail: tpavelka | matousek@kiv.zcu.cz

Abstract

The search for the most probable word sequence is in automatic speech recognition called decoding and is usually carried out by the Viterbi algorithm, an efficient search strategy based on dynamic programming. The paper discusses implementation issues and methods to further reduce computational costs when performing recognition with large vocabularies and stochastic language models. Such methods can be divided into two categories: those that eliminate unnecessary computations (such as tree structured lexicons) and those that exclude computations according to the probability that those computations will lead to the desired result (pruning).

1. INTRODUCTION

In decoding the task is to find the most probable word sequence $W^* = w_1, w_2, \ldots, w_k$ given a set of acoustic *observations* $O = o_1, o_2, \ldots, o_T$, their respective *acoustic model* likelihoods $P(O|W)$ and the *language model* prior probability $P(W)$. In terms of statistics the goal is to find the word sequence W with maximal *a posteriori* probability:

$$W^* = \underset{\forall W}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)}. \tag{1}$$

Since the term $P(O)$ remains constant for all possible word sequences W it can be excluded from the equation.

The main problem of decoding is how to efficiently search for this word sequence. The most widely used approach to this problem is to model each word as a sequence of hidden Markov model (HMM) states, constructing a HMM representing all possible sentences and searching for the most likely sequence of states $Q^* = q_1, q_2, \ldots, q_T$. The maximum a posteriori search criterion can be expressed as

$$W^* = \underset{\forall W}{\operatorname{argmax}} P(W) \sum_{Q} P(O|Q)P(Q|W). \tag{2}$$

Usually the suboptimal but nevertheless well working *Viterbi* criterion is employed because it allows easy extraction of the optimal state sequence:

$$W^* \simeq \underset{\forall W}{\operatorname{argmax}} P(W) \max_{Q} P(O|Q)P(Q|W). \tag{3}$$

A typical decoder in automatic speech recognition is an implementation of the *Viterbi algorithm* (see (Rabiner, 1989) for details), an efficient search strategy based on dynamic programming. While the Viterbi algorithm is (in terms of computational complexity) much more efficient than e.g. testing every possible word sequence, the computational cost can still be enormous when searching with large vocabularies and statistical language models. The following pages will discuss further attempts to reduce the search costs.

2. DECODING STRATEGIES

Wide variety of ways how to implement the Viterbi algorithms have emerged in recent years. According to a classification scheme proposed in (Aubert, 2002) today's decoders can be divided along several dimensions:

- **One pass vs. multi pass decoding**. In one pass decoder the desired word sequence is known after a single run of the decoding algorithm. Multi pass strategies try to reduce the computational cost by first running with rough (and fast) acoustic and language models and refining the results during the second run with more precise (and more computationally expensive) models.
- **Time synchronous vs. time asynchronous search**. In time synchronous search all computations for a given time frame are done before moving to the next time frame. Asynchronous search tries to take advantage of the fact that when "looking forward" in time, one can eliminate those search paths that are unlikely to lead to the most likely word sequence and thus speeding up the process.
- **Static vs. dynamic network expansion**. The "network" here refers to the search variables stored in memory during the execution of the Viterbi algorithm. In the case of static network expansion there could be variables for all the HMM states for each time frame. As will be shown in the next sections the HMM can be quite large and may not even fit in the memory of most of todays computers. On the other hand dynamic network expansion generates the search variables "on the fly" and allows pruning of those that are unlikely to lead to the result.

Despite an enormous amount of research there is currently no "winning" strategy and many completely different decoders developed by various research institutions are surprisingly on the par when tested on standard testing data (see (Aubert, 2002)). Even the less favored methods (such as static network expansion) have advantages in certain specific niches of automatic speech recognition. The following section will describe the development of our decoder and the supported search strategies.

3. LDEC - LASER DECODER

The decoder has been under development within the framework of the LASER (LICS Automatic Speech Extraction/Recognition) automatic speech recognition system. According to the previous classification scheme all our decoders are one pass and time synchronous. It is because these search strategies are (in our opinion) easier to implement.

The first version (see (Pavelka, 2003)) had the search variables stored in a three dimensional array with the dimensions representing time, word from the dictionary and the HMM state of a given word. This architecture needs only a pronunciation dictionary and optionally a language model (limited to bigrams or word-pair grammars) to be supplied by the user and both the HMM and all the search variables are constructed by the decoder. Another possibility for language modelling is to construct a so called *word graph* which is an oriented graph with nodes representing words and links representing allowed transitions between words. The advantage here is that such a word graph can be constructed from a hand written regular grammar. This

requires only a slight modification of the decoding algorithm: the array dimension that represented words now represents nodes in the word graph. The user supplies dictionary and the word graph and the HMM is again constructed by the decoder.

One possible way to speed up the search is a *tree structured lexicon* which is based on the fact, that if two states which can follow a given state have the same acoustic model (i.e. belong to the same phonetic unit) the computation for those two states is exactly the same. This leads to a tree like HMM representing the pronunciation lexicon. Such a HMM has approx 40 % states in comparison with a linear lexicon. The organization of the search variables is now different: the array has only two dimensions, one representing time and the other a HMM state, where the HMM must be constructed outside the decoding algorithm.

If a search guarantees to find the best path with respect to the given criterion (i.e. a single path with the highest probability in the case of Viterbi criterion) the search is called *admissible*. A way to speed up the search is to exclude those states that are thought unlikely to be in the final optimal sequence. This method is referred to as *pruning*. It makes the search no longer admissible but has been shown to work well in practice. one of the simplest and most often used way of pruning is called *beam search*. It works as follows: first a state with the highest score in a given time frame is found and then all states having lower score than a certain percentage of the highest score are discarded.

The first pruning tests were carried out with a decoder with static network expansion. The results have shown that no matter how strict the pruning threshold the computational costs could not be reduced to more than about one half. This is due to the fact that when searching for the maximum score, all the HMM states (stored in an array) in each time frame had to be accessed even if most of those states were later to be discarded. To address this problem we have implemented dynamic network expansion where all the states are stored in a hash map. The decoding starts with one starting state which is stored in a so called *active state list*. Only those states that can be reached from the states in the active list in the current time frame are added to the active list for the next time frame, where some of the states are pruned. This way only the active state list is searched for maximum and pruned. Our experiments have shown that even though accessing data in a hash map is much slower compared to array, the pruned decoding process can be much faster with dynamic network expansion especially if the HMM representing all possible word sequences is very large.

When neural network acoustic model is used, each phonetic unit needs to be represented by only one HMM state which has the emission probability associated with and output neuron of the network. A simple technique for modeling a minimum duration of phonemes is to duplicate the hmm states, i.e. to use a left-right HMM model with several states for each phoneme, where all those states share the same emission probability.

4. LANGUAGE MODELING

Language modeling deals with estimating the probability $P(W)$ of the word sequence being recognized. Ideally a stochastic language model tries to compute the probability of the next word in the sequence as a product of the probabilities of the previous words:

$$P(w_1, w_2, \ldots, w_N) = \prod_{i=1}^{N} P(w_i | w_1, w_2, \ldots, w_{i-1}). \quad (4)$$

In practice the number of previous words included in the computation is limited leading to a so called *N-gram language model*:

$$P(w_i|w_1, w_2, \ldots, w_{i-1}) = P(w_i|w_{i-N+1}, \ldots, w_{i-1}) \tag{5}$$

For $N = 2$ the model is often called *bigram* language model, for $N = 3$ *trigram* language model and so on. The first problem is, that the N-gram language model violates the first order Markovian assumption, i.e. that the probability of being in a given state is only dependent on the previous state. In the case of N-gram language model, the probability of being in a state depends on its $N - 1$ word history. The power of the Viterbi algorithm lies in *recombination*: selecting the best path leading to a given state when it is certain that the other paths will not lead to the optimal solution. If the first Markovian assumption is valid, only the highest scoring path leading to a given state in a given time frame is stored, i.e. distinct HMM states belonging to the same time frame can be recombined.

We have chosen the following solution to the problem of stochastic language modeling: Suppose we are using tree structured lexicons and each state in the HMM representing the lexicon has a unique number i then if two states with the same number i in time frame t have the same $N - 1$ word history they can be recombined. This means that states with different $N - 1$ histories are stored in distinct trees and since the number of different histories as well as the number of states in each tree can be quite large, the whole recognition network would not even fit into memory of today's computers. This difficulty is addressed by dynamic network allocation with a tight pruning threshold leaving only a small percentage of all possible states active during the search. Our experiments show, that even though the search does not guarantee to find the optimal solution it works well in practice.

5. EXPERIMENTAL RESULTS

The accuracy of recognition is computed by a dynamic programing match of the referential (i.e. correct) sentence and the recognized sentence leading to N - the number of words in the referential sentence, H - the number of correct words in the recognized sentence, S - the number of word substitutions, I - the number of word insertions into the referential sentence, and D the number of word deletions from the referential sentence needed for the two sentences to match. We use the two measures to express the recognition accuracy:

$$\%Corr = \frac{H}{N} \tag{6}$$

$$\%Acc = \frac{H - I}{N} \tag{7}$$

The first measure is simply the percentage of correctly recognized words. The second one is useful because the recognized sentence can have more words than the referential sentence leading to a high percentage of correctly recognized words in a result that is not as useful in practice because of a high number of redundant words not present in the spoken sentence.

The first experiment was carried out to test the efficiency of using tree structured lexicons with pruning and dynamic network expansion. Table 1 shows the result. It can be seen that while the tree lexicon can reduce the total number of states by a bit more than one half, when pruning is applied, it can reduce the number of active states (and thus speed up the search) by an order of magnitude. The next experiment tests state duplication for different testing corpora: Chess corpus, containing voice commands for a chess game (table 2), Numbers corpus containing

	Avg. number of active states	Total number of states	active/total
Tree	537	12543	4.28 %
Linear	5903	30507	19.35 %
Tree/Linear	9.10 %	41.12 %	

Table 1: Number of total and active states when decoding with tree and linear lexicons. Tested on the same data with the same pruning threshold.

spoken numbers between one and one million (table 3) and Trains corpus containing queries about train schedules (table 4).

States/phoneme	%Corr	%Acc	%SCorr	Average response time [s]	Total no. of states
1	95.73	94.33	70.50	0.20	747
2	96.69	96.49	77.75	0.19	1478
3	97.18	97.18	80.50	0.19	2209
4	96.65	96.65	77.50	0.19	2940

Table 2: Chess corpus having 93 distinct words. A grammar describing all possible utterances was available.

States/phoneme	%Corr	%Acc	%SCorr	Average response time [s]	Total no. of states
1	95.55	94.12	63.55	0.47	2222
2	96.26	95.97	73.62	0.41	4407
3	91.29	91.09	54.75	0.39	6592

Table 3: Numbers corpus having 40 distinct words. A grammar describing all possible utterances was available.

States/phoneme	%Corr	%Acc	%SCorr	Average response time [s]	Total no. of states
1	79.21	55.37	7.85	1.35	4347
2	80.23	61.26	10.63	0.88	8445
3	81.05	65.73	14.18	0.65	12543

Table 4: Trains corpus having 1469 distinct words. Tested with tree structured lexicon and without grammar or language model.

It can be seen from all the experiments that when pruning is used the number of active states does not increase with the number of states per phoneme. This means that state duplication can increase performance without increasing computational cost. When tree lexicons are used, the computational cost actually decreases with the number of states per phoneme as can be seen from table 4. Duplicating states can be seen as assuming a minimum duration for phonemes, it can be seen from tables 2 and 3 that when the assumed duration is longer than the actual duration in the data (i.e. there are too many states per phoneme) the accuracy decreases.

Our first experiments with language modeling led to worse results than when no language modelling was done at all. This was probably due to the lack of training data since many N-grams present in testing data were not seen in the training data at all and thus had zero probability. A working solution was found to be a class based language model, where words are automatically clustered into classes (the algorithm can be found e.g. in (Allen, 1988)) and the N-gram model is built on those classes rather than words. Figure 1 shows recognition accuracy for different number of classes. Bigram language model gave best results for 300 classes. The poorer performance of the trigram language model can probably again be attributed to the low amount of training data. It can be seen that the number of word classes has also a major impact on the time response of the system.

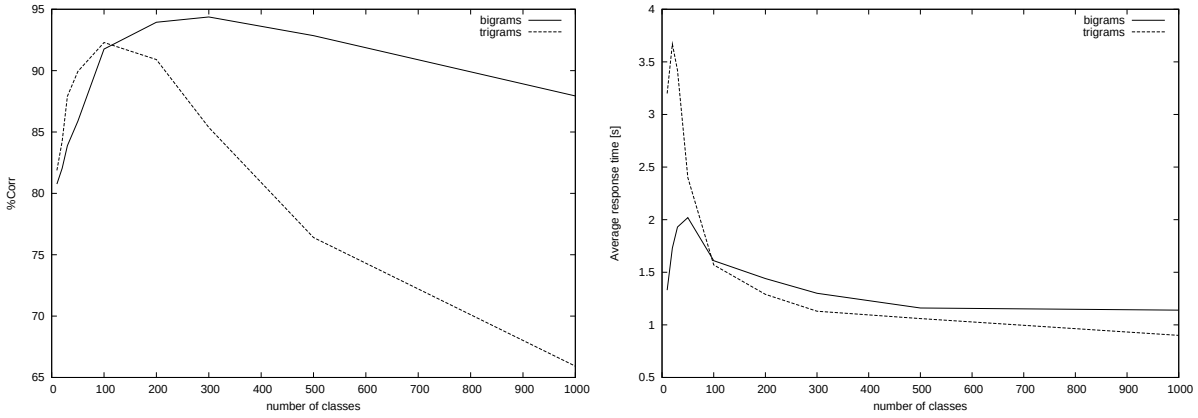

Fig. 1: Percentage of correctly recognized words and average time response for class based bigram and trigram language models.

Another known problem with language modeling is that the language and acoustic model probabilities have different dynamic ranges, since acoustic model probability is included in the computation every time frame, but language model probability is used on word ends only. The practical solution is to scale the language model probability, the scaling constants are usually in the form

$$W^* = \underset{\forall W}{\operatorname{argmax}}\, P(O|W)\alpha P(W)^{\beta}. \tag{8}$$

We will call α the *word insertion penalty* and β the *language model scale*. Their values have to be determined empirically which is shown in figure 2. Language model scale tuning has not led to any performance increase with our testing data.

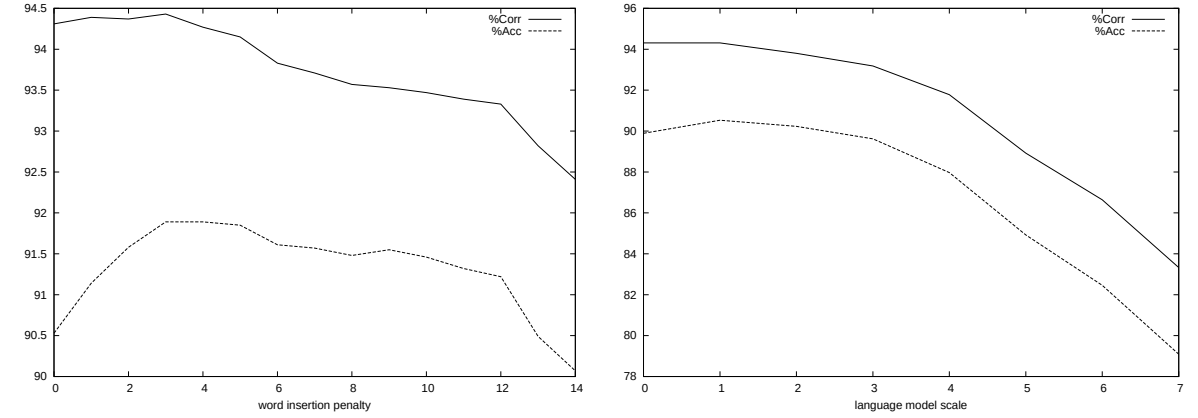

Fig. 2: Word insertion penalty and language model scale for 300 class bigram language model.

6. CONCLUSION

The paper describes our first attempts at large vocabulary speech recognition with the train schedule corpus. Even though there was an evident lack of data for the language model training, the tests have shown that the conception of pruned single pass search with tree structured lexicon is feasible and we are planning to use the same architecture for experiments with larger dictionaries. The best recognition results were obtained with 300 class bigram language model leading to $\%Corr = 94.43$ and $\%Acc = 91.89$. We have also shown that the method of state duplication can lead to accuracy increase and even to increase in recognition speed.

ACKNOWLEDGEMENT

This work was supported by grant 2C206009 Cot-Sewing.

REFERENCES

Allen, J.F. (1988). *Natural Language Understanding*. The Benjamin/Cummings Publishing Company.

Aubert, X.L. (2002). An overview of decoding techniques for large vocabulary continuous speech recognition. In: *Computer Speech and Language*. Vol. 16. pp. 89–114.

Pavelka, T. (2003). Hybrid speech recognizer implementation. Master's thesis. Universisty of West Bohemia.

Rabiner, L.R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. In: *Proceedings of the IEEE*. Vol. 77.