Backward Viterbi Beam Search for Utilizing Dynamic Task Complexity Information

Min Tang, Philippe Di Cristo

VoiceBox Technologies, Bellevue, Washington, USA
mint@voicebox.com, philipped@voicebox.com

Abstract

The backward Viterbi beam search has not received enough attentions other than being used in the second pass. The reason is that the speech recognition society has long ignored the concept of dynamic complexities of a speech recognition task which can help us to determine whether we should operate Viterbi decoding in forward or backward direction. We use the U.S. street address entry task as one example to show why the analysis of the dynamic task complexity information is important. We then describe the procedure to implement a backward Viterbi decoding system from an existing traditional ASR engine. We evaluated the backward Viterbi search using CMU’s PocketSphinx. The experimental results of the backward Viterbi Search show significant performance improvement over the forward Viterbi decoding system on the U.S. street address entry task.

Index Terms: backward Viterbi beam search, dynamic task complexity

1. Introduction

Ever since the Hidden Markov Model (HMM) dominated the Automatic Speech Recognition (ASR) area as the most powerful statistical method for modeling speech signals in the recent two decades[1], the Viterbi algorithm has become the most widely used search algorithm to decode a HMM network in most state-of-the-art ASR systems. As a dynamic programming algorithm, the Viterbi algorithm is performed in a time-synchronous fashion from left to right.

Time-synchronous Viterbi search can be viewed as a breath-first search with dynamic programming as illustrated in Figure1. Instead of expanding the search paths along a tree, it merges multiple paths that leads to the same search state and keeps only the best path. The standard Viterbi search guarantees to find the global optimal path because it searches through the whole search space. However, in real applications, the search space is very big (the search space of Viterbi algorithm is $O(NT)$, the complexity for the Viterbi algorithm is $O(N^2T)$, where $N$ is the number of total HMM states and $T$ is the speech duration[2]). Thus it is impractical to search the overwhelming number of possible paths especially when we have limited resources in embedded devices.

In order to reduce the computation load, people uses some heuristics to narrow down the search space. A simple way to prune the search space for breath-first search is using beams. Basically instead of keeping all paths at each time, the beam search only looks at a subset of paths whose scores are within a beam to the local-best path so as to achieve a balance between speed and accuracy.

Figure 1: the Forward Viterbi Algorithm example of two mono-phone words recognition

In [3], people proposed to do a two-pass search in order to further speed up the system; the first pass is a forward search which uses a simplified algorithm; the second pass is a backward Viterbi search, which uses information computed in the forward pass to perform a more complicated search in reverse with much less computation load. This approach has shown great speed-up with little performance degradation.

However, the backward Viterbi beam search has not been used in a speech recognition system as the main and standalone decoding approach. We believe the reason is that the dynamic task complexity information is missing in our comprehensive understanding of speech recognition systems, especially in the embedded speech recognition world.

In this paper, we propose an innovative approach to speed up Viterbi beam search, which is to use the forward or backward Viterbi beam search as required by the task based on the analysis of dynamic task complexity. Our experiments have demonstrated that in the U.S. street address entry task, which by its nature favors the backward Viterbi beam search because the complexity of task narrows down along the time line smoothly, we are getting great improvement in both accuracy and speed by using the backward Viterbi beam search. Compared to the conventional forward Viterbi decoding system, our new system was able to run 3 times faster with more than 90% errors reduction on that task.

The rest of this paper is organized as follows. Section 2 introduces the concept of dynamic task complexity. Section 3 describes the concept of the backward Viterbi decoding. Section 4 discusses the procedure to build a system that is able to do the backward Viterbi decoding. Experimental results are presented in Section 5.
2. Dynamic Task Complexity

The problem to evaluate the complexity of a speech recognition task is not trivial, because it not only involves the computation of $H(W)$, which is the language model entropy, but also involves the vocabulary acoustic similarity, speaker variability and the channel characteristics[4]. Theoretically the task complexity should be the estimate of average conditional entropy $H(W|O)$, where $W$ is what the speaker has spoken and $O$ is what the ASR system is hearing. However the direct comparison of estimated $H(W|O)$ could be affected by bad acoustic model estimation, which results in higher conditional entropy with no change in task complexity. Therefore for decades, the speech recognition society has fallen back to use $H(W)$ as the main task complexity measurement.

In this paper, we will adopt the same metric to evaluate the task complexity. Different with traditional task complexity estimation approach, we are not only looking at the task as one whole and static block, but also seeing it as the combination of several consecutive sub-tasks and focusing on how the task complexities of sub-tasks change as the search proceeds, which we called – dynamic task complexity.

We use the U.S. street address entry task as one example, where the speaker says the whole street address from street name, city name to state name in one utterance. Naturally we could divide this task into 3 sub-tasks – street recognition, city recognition and state recognition. Depending on how you proceed the Viterbi decoding, the sub-task complexity flow could be very different.

Without loss of generality, let’s assume we have 50 states, each state has 800 unique cities (by unique we mean no 2 states can have cities with the same name) and each city has 225 unique streets (cities shares no street names). And to make our life easy, let’s assume all cities and streets are in uniform distribution.

The dynamic task complexities when we employ the Viterbi search in normal direction from street to state are:

$$H_f(street) = -\sum \frac{1}{50 \times 800 \times 225} \log \frac{1}{50 \times 800 \times 225} = 23.1 \text{ bit}$$

(1)

$$H_f(city | street) = 0 \text{ bit}$$

(2)

$$H_f(state | street, city) = 0 \text{ bit}$$

(3)

$$H_f(street, city, state) = 23.1 + 0 + 0 = 23.1 \text{ bit}$$

(4)

The dynamic task complexities when we employ the Viterbi search in reverse direction from state to street are:

$$H_r(state) = -\sum \frac{1}{50} \log \frac{1}{50} = 5.6 \text{ bit}$$

(5)

$$H_r(city | state) = -\sum \frac{1}{800} \log \frac{1}{800} = 9.7 \text{ bit}$$

(6)

$$H_r(street, city, state) = -\sum \frac{1}{225} \log \frac{1}{225} = 7.8 \text{ bit}$$

(7)

$$H_r(street, city, state) = 5.6 + 9.7 + 7.8 = 23.1 \text{ bit}$$

(8)

As seen from Figure 2, although the whole task complexity is the same no matter whether we proceed the Viterbi decoding in forward direction or backward direction, their inner dynamic task complexities are completely different. When we employ the backward Viterbi search, at each one of the three stages – State recognition, City recognition and Street recognition, the maximum required entropy is 9.7 bits. Compared with the forward Viterbi search, which requires 23.1 bits entropy at the Street recognition stage, the backward Viterbi search requires much smaller beam so as to be able to keep a better balance between speed and accuracy.

3. Backward Viterbi Decoding

The Viterbi beam search is basically a network search. So it doesn’t matter whether we are searching from the beginning or from the end of the network. Theoretically when we perform the Viterbi decoding in a backward direction as shown in Figure 3, we should get the same result as what we do in Figure 1 if we set the beam width to be infinite.

However in a real system, the beam width is never infinite. The same beam could have different influence on the pruning process of the forward search and the backward search because of different dynamic task complexities.
4. Build Backward Viterbi Decoding ASR

It is fairly easy to build a Backward Viterbi Decoding system by making some minor changes to an existing forward Viterbi decoding system.

4.1. On Top of Traditional ASR Engines

4.1.1. Reverse Acoustic Model

The Gaussian Mixture Model (GMM) could be reversed by simply remapping the monophone/triphone to senones as shown in Figure 4:

Original triphone:

![Original triphone diagram]

Reversed triphone:

![Reversed triphone diagram]

Figure 4: Triphones that should map to the same senone in the original and reverse acoustic models.

The state transition matrix could be reversed by modifying the matrix so that it generates the same acoustic score without re-estimating the state transition probabilities.

As shown in Figure 4, using the original state transition matrix, the acoustic score of one path is equal to:

\[ S = n_0 t_{00} + n_1 t_{11} + t_{12} + n_2 t_{22} + t_{23}, \]

where \( n_0, n_1 \) and \( n_2 \) correspond to the number of times that the path stays at state \( S_0, S_1, \) and \( S_2. \)

After reversal, the acoustic score is kept the same:

\[ S' = n_0 t'_{02} + t'_{12} + n_1 t'_{11} + t'_{10} + n_2 t'_{20} + t'_{23} = S, \]

where \( n_0, n_1 \) and \( n_2 \) correspond to the number of times that the path stays at state \( S_0, S_1, \) and \( S_2. \)

4.1.2. Reverse Feature Vector

Simply reversing the audio cannot generate the same feature vectors (the static Cepstral coefficients will be the same, but delta-coefficients are opposite to each other). We need to reverse the feature vectors directly as shown in Figure 5.

4.1.3. Reverse Finite State Machine

Grammar is reversed by simply reversing the order of all rules and terminals.

For example, the reverse of:

\[ S\text{Root} = $Intro This Word $Exit; \]

will be:

\[ S\text{Root} = $Exit Word $Intro; \]

4.1.4. Reverse Language Model

Basically we need to reverse the training text and then re-train the language model. As long as we are using the same training configuration, the reversed LM should generate the same perplexity value on the test set (reversed).

4.1.5. Reverse Dictionary

Dictionary is reversed by simply reversing the order of all phonemes. For example,

“Washington” pronunciation:

Pron 1: W AA SH IH NG T AH N
Pron 2: W AO SH IH NG T AH N

“Washington” reversed pronunciations:

Pron 1: N AH T NG IH SH AA W
Pron 2: N AH T NG IH SH AO W

4.2. On Top of Finite-State-Transducer based ASR Engines

For this kind of ASR engine, since all the grammar, dictionary, acoustic model indices have already been compiled into one network, we could do the trick by simply reversing the network without making any modifications to the acoustic model, grammar and dictionary.

5. Experiments

We evaluate the backward Viterbi decoding technique in 2 recognition domains using PocketSphinx[5][6] -- CMU’s open-source real-time continuous speech recognition system for hand-held devices. In order to make a fair comparison between the forward Viterbi decoding system and the backward Viterbi decoding system, we used the acoustic models that came with the PocketSphinx package as is and didn't make any changes to the Gaussian Mixture Model, the search procedure and the parameter settings. All we did is just reversing the acoustic model, the feature vector, the grammar or language model and the dictionary as described in section 4.1. Since we don't need to re-train or adapt the acoustic model, it requires little effort to make the conventional ASR engine to be able to do the backward Viterbi decoding.
5.1. TI-Digits Recognition Task

The TI-Digits acoustic model that came with the PocketSphinx package contains 670 state-clustered senones. The test set is the standard TI-Digits adult test set which contains 8700 sentences. We have tried both the Finite State Grammar and the statistical language model. The Finite State Grammar is simply a digram loop with the maximum digits length set to 7. The statistical language model is simply a unigram with uniform distribution.

Table 1 shows the speed and accuracy of the conventional Viterbi decoding system and the backward Viterbi decoding system on TI-Digits task. We can see from this table that the difference between the conventional Viterbi decoding and the backward Viterbi decoding is almost ignorable. Obviously this is what we expected because the TI-Digits task is symmetric thus the Viterbi direction makes no difference.

5.2. NSCS Address Entry Task: State and City Recognition

NSCS stands for address Number, Street name, City name and State name. In a navigation system, it is also referred as One Step Address Entry task because the user gives the whole address in one utterance. NSCS is a very challenging task for the embedded world and it is impractical to conquer this task in one pass because of the huge vocabulary (there are more than 9 million U.S. street-city-state entries and the vocabulary size is about 400,000). So here we adopt a multi-pass strategy – state recognition first, then city recognition followed by house number and street name recognition. The acoustic model that we used is the WSJ0 acoustic model that came with the PocketSphinx package. It contains 4210 state-clustered senones. The test set is collected in VoiceBox and consists of 396 sentences from 6 native American speakers.

5.2.1. State Recognition

We used the following simple grammar to recognize states:

\[
\text{\$Root ::= \$garbage* \$States}
\]

It works like a keyword spotting grammar. In order to make fair comparison, we used the same configuration settings for both systems.

Table 2. Comparison between the conventional Viterbi and the backward Viterbi on NSCS task: state recognition

<table>
<thead>
<tr>
<th>System</th>
<th>Real-Time Factor</th>
<th>State Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward Viterbi</td>
<td>0.09xRT</td>
<td>43.94%</td>
</tr>
<tr>
<td>Backward Viterbi</td>
<td>0.03xRT</td>
<td>0.51%</td>
</tr>
</tbody>
</table>

As seen from Table 2, the system that uses the backward Viterbi decoding technique is not only far more accurate than the forward Viterbi decoding system; it is also 3 times faster.

5.2.2. City Recognition

After getting the state information, we used the following simple grammar to recognize cities:

\[
\text{$Root ::= $garbage* $Cities_in_RecoState RecoState}
\]

In order to make fair comparison, both systems are using the same configuration settings.

Table 3. Comparison between the conventional Viterbi and the backward Viterbi on NSCS task: city recognition

<table>
<thead>
<tr>
<th>System</th>
<th>Real-Time Factor</th>
<th>City Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward Viterbi</td>
<td>0.164xRT</td>
<td>82.50%</td>
</tr>
<tr>
<td>Backward Viterbi</td>
<td>0.053xRT</td>
<td>9.60%</td>
</tr>
</tbody>
</table>

Once again, as seen from Table 3, the backward Viterbi system is 3 times faster than the conventional system with 90% errors reduction.

6. Conclusions

In this paper, we have proposed an innovative approach to speed up Viterbi beam search, which is, based on the analysis of dynamic task complexity, using the forward or backward Viterbi beam search as required by the task. Our experimental results have shown that in the situation where the dynamic complexity of the task narrows down along the time line smoothly, we can get great improvements in both the accuracy (more than 90% errors reduction) and speed (3 times faster) by doing the backward Viterbi beam search. It is easy to modify a traditional ASR engine to perform Viterbi search in the reverse direction. And the improvement on accuracy and speed makes this technique suitable for ASR systems running on embedded devices.

7. References