Speech Recognizer Optimization under Speed Constraints

Ivan Bulyko
Raytheon BBN Technologies, Cambridge MA 02138, USA
ibulyko@bbn.com

Abstract
We present an efficient algorithm for optimizing parameters of a speech recognizer aimed at obtaining maximum accuracy at a specified decoding speed. This algorithm is not tied to any particular decoding architecture or type of tunable parameter being used. It can also be applied to any performance metric (e.g. WER, keyword search or topic ID accuracy) and thus allows tuning to the target application. We demonstrate the effectiveness of this approach by tuning BBN’s Byblos recognizer to run at 15 times faster than real time while maximizing keyword search accuracy.

Index Terms: speech recognition, optimization, keyword search

1. Introduction
Interactive speech applications require fast speech recognition technology, one that can run in real time. In many instances speech-to-text is only part of the application and other sophisticated technologies (e.g. named entity detection, information extraction, machine translation) make use of the speech recognition output. This places further constraints on the recognition speed in order to leave time for the additional processing to complete in real time. Even non-interactive applications often require increases in recognition speed over time as the growth in the volume of data can outpace the rate of increase in total CPU bandwidth.

Modern speech recognizers are complex systems, that even in fast configurations may perform multiple passes of decoding, adaptation, and/or rescoring (e.g. [1], [2], [3]). Such systems typically have weights for combining different model scores and parameters that control pruning of the search space. The number of adjustable parameters can be large (e.g. 10 or 20) which makes tuning a challenging task.

Systems are typically optimized by hand to run at some particular speed using one set of models and a tuning set. This configuration is then fixed and applied to other models and tasks. In our experience, there was considerable variance in speed when we applied a fixed system configuration to multiple languages. We also found that settings optimized for one language did not give optimal performance on other languages. These discrepancies were particularly apparent with fast sub-real-time systems. Furthermore, systems that are tuned to minimize WER may not be optimal for all applications. For instance, one may be particularly interested in the detection performance of rare words, or the topic ID accuracy. In the environment where speed requirements and/or target applications change frequently, manual tuning of the recognizer is no longer practical, which motivates the work presented in this paper.

Recent work in speech decoder optimization includes the use of genetic algorithms [4] and linear programming [5], although in both papers the speed constraints are ignored. In [6] the authors devise an iterative algorithm to optimize a cost function that represents the tradeoff between speed and accuracy. The tradeoff is controlled by a parameter $\lambda$ that is adjusted in each iteration for either maximum performance or greater speed, depending on the current operating point. In [7] both speed and WER are optimized jointly with an algorithm designed to track the curve of best performance over a range of speeds starting with the unconstrained (i.e. slowest) configuration. In this paper we propose an algorithm that performs joint speed and performance optimization, however, unlike [7] we tune to a single target speed and do not track the full curve, which leads to faster optimization. We apply the optimization algorithm to the keyword search task, which is a departure from the traditional WER-based optimization.

2. The Algorithm
Current methodology is inspired by the work described in [6], where the authors developed an iterative algorithm for minimizing WER at the desired speed. In [6] the authors make a key assumption, that small changes in speed and accuracy associated with slight variations of individual parameters are additive when multiple parameters are updated. In the current work we make an additional assumption that small changes in speed and accuracy are linear with respect to the parameter values in a short interval. We were able to empirically verify both of these assumptions.

The optimization algorithm works as follows:
1. Initialize the set of $N$ decoder parameters $\theta(0)$ with some default values and assign some value $\Delta$ for each parameter that corresponds to a small change in speed and accuracy.
2. For each iteration $i$ (0 … K)
   a. Decode using the current set of parameters $\theta(i)$ as well as sets where individual parameters are changed by $+/-\Delta$, one parameter change at a time. This step results in $2N+1$ decodes that can be run in parallel, where each parameter configuration differs from $\theta(i)$ only by one parameter.
   b. Compute a new set of parameters $\theta(i+1)$ aimed at maximizing projected accuracy at the desired speed. This step will be described in detail below.
3. Return the parameter configuration that produced the highest accuracy and satisfied the desired speed in step 2.

Step 2(b) is the heart of the algorithm as it is responsible for updating parameters between iterations by taking into account speed-accuracy tradeoffs of the individual parameters as measured in step 2(a). A hypothetical plot of the speed-accuracy tradeoff is shown in Figure 1 where point X is the initial operating point in the speed-accuracy coordinate space and each of the individual vectors corresponds to one parameter change. The shaded area covers the space of operating points reachable by taking a weighted linear combination of the vectors with each weight in the interval $[W_{min}, W_{max}]$. Our goal is to choose a set of vectors and
weights (i.e. a set of decoding parameters) that maximize accuracy at the target speed, e.g. point A for the target speed Sa. If the target speed is not reachable (e.g. Sb), then we choose the maximum achievable speed (point B).

Figure 1: A hypothetical illustration of the logic used in updating parameters in the optimization algorithm (step 2b).

The task of finding the weighted linear combination of vectors that maximize accuracy at the target speed (i.e. point A in Figure 1) does not have a simple analytic solution; therefore we employ an iterative algorithm where vectors and weights are updated to make small tradeoffs between speed and accuracy at each iteration. The parameter selection algorithm works as follows:

1. Define a small interval [Tmin, Tmax] around the target decoding time, e.g. [0.99T, 1.01T] designed to constrain the parameter updates in each iteration.
2. Initialize the set of available weights W with the maximum allowable weight wj for each parameter. Set current operating point OP = X.
3. For each iteration i (0 … MaxIters)
   a. if (OP.time > T)
      choose available parameters to obtain decoding time Tmin at minimal cost in accuracy (vectors with slope closest to 180).
   else
      choose available parameters that increase accuracy with minimal loss in speed (i.e. vectors with slope closest to 90), as long as the decoding time does not surpass Tmax.
   b. Update the set of available weights W by subtracting values used in step a.
   c. Update the current operating point OP with the projected values of decoding time and accuracy computed in step a. Stop if improvement in accuracy is below threshold or the set of available parameters is exhausted.
4. Return the optimal set of vectors and weights and the corresponding projected operating point OP.

3. System Description

We use BBN’s Byblos speech recognizer [1] in a sub-real-time configuration, designed to run at 0.1xRT or 10 times faster than real time (10FTRT). This configuration employs a twopass decoding strategy, as shown in Figure 2. The “fast match” pass (also called forward pass, because it runs through the data in the forward direction) is designed to quickly eliminate improbable hypotheses. This pass uses coarse and inexpensive models: 1) a state-tied mixture (STM) acoustic model, where each phoneme state is assigned a Gaussian mixture with mixture weights clustered by triphone context, and 2) a bigram language model. The result of the forward pass is a set of likely word-ends with their corresponding scores, which are used to guide the “beam search” (or backward pass), effectively reducing the search space. The backward pass employs a set of more detailed models: 1) a non-crossword state-clustered tied mixture (SCTM) acoustic model, where states are clustered using quinphone context and each cluster is assigned a Gaussian mixture, and 2) a trigram language model. The backward pass produces a pruned word lattice which is then transformed into a consensus network [8] with approximate time marks.

Each decoding pass takes its own set of parameters: language and acoustic model weights and various search pruning thresholds. The backward pass also has a setting for pruning the output word lattice. We fixed the acoustic model weights to 1 and let the other parameters (total of 11) vary in our optimization experiments.

The keyword search (KWS) system used in this work is similar to BBN’s STD-06 system [9]. It indexes words with their locations and confidences from consensus networks into an Oracle database. The search operates by querying the database and outputting results sorted by confidence.

Detection accuracy is scored using a variant of the mean average precision (MAP) metric with normalization factor N (Equation 1) equal to the total number of possible correct hits instead of the number of returned correct hits (as in the NIST MAP scoring). This metric is often referred to as the “area under curve” or AUC, and unlike NIST MAP, it penalizes queries with no returned hits.

\[
AUC = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{|R_q|} \sum_{r \in R_q} P(q,r) \quad (1)
\]

where \(Q\) is the set of queries with non-empty references, \(N\) is the number of possible hits, \(R_q\) is the set of ranks of correct hits for query \(q\), and \(P(q,r)\) is precision for query \(q\) at rank \(r\).
The accuracy of search, as measured by AUC, is determined by the ranking of hits. AUC, which takes on values in the range $[0,1]$, is high when correct hits are ranked high in the list of results. Note that AUC is not weighted by the word frequency, i.e. both rare and common words contribute equally to the overall score.

### 4. Experiments

In these experiments we used the 3.75-hour Spanish CallHome Eval97 test set. The recognition models were trained with 250 hours of speech which include Spanish Fisher, CallHome, CallFriend and Ricardo. The detection performance was measured by computing AUC on a query set that included all words in the reference transcript of the test set. The experiments were run on a cluster of compute nodes with dual quad-core 2.8GHz Intel Xeon CPUs and 32GB of RAM.

Figure 3 illustrates the progression of the optimization algorithm that was set to the target speed of 0.067xRT, or 15 times faster than real time (15FTRT). The initial set of parameters resulted in the speed of 0.113xRT (or 8.86FTRT) and AUC of 0.4501. Each iteration performed 23 runs of decoding, indexing and search, with each run taking about 1 hour to complete. Since these runs were executed in parallel, the full iteration required only 1 hour of wall clock time. The algorithm performed 8 iterations, producing a set of parameters that achieve AUC of 0.448 at the target speed of 15FTRT.

The relative change in the parameter values between the initial and the final iterations of optimization (Table 1) reveals significant tightening of the search beams in the forward pass of decoding (C, D, E, which represent thresholds applied at different stages of recognition) plus a more aggressive variable frame rate window (F). Note that increasing the log10 beam widths leads to more aggressive pruning of the search space. There is a slight tightening of the backward pass search beam (J is tightened while I is relaxed). Also note the lowered 3-gram language model weight (H) and the relaxed lattice pruning threshold (K), both of which have the effect of increasing the lattice size, and therefore improving detection recall at some cost in decoding speed.

**Table 1. Relative change in the decoder parameters after 8 iterations of optimization. Shaded rows indicate tightening of parameters.**

<table>
<thead>
<tr>
<th>Decoder parameters</th>
<th>% ∆</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Maximum number of shortlists in forward pass</td>
<td>0</td>
</tr>
<tr>
<td>B Bigram LM weight in forward pass</td>
<td>+3.6</td>
</tr>
<tr>
<td>C Forward pass search beam width FW1 (log10)</td>
<td>+11</td>
</tr>
<tr>
<td>D Forward pass search beam width FW2 (log10)</td>
<td>+28</td>
</tr>
<tr>
<td>E Forward pass search beam width FW3 (log10)</td>
<td>+51</td>
</tr>
<tr>
<td>F Frame rate reduction window in forward pass</td>
<td>+43</td>
</tr>
<tr>
<td>G Maximum number of shortlists in backward pass</td>
<td>+15</td>
</tr>
<tr>
<td>H Trigram LM weight in backward pass</td>
<td>-14</td>
</tr>
<tr>
<td>I Backward pass search beam width BW1 (log10)</td>
<td>-5.8</td>
</tr>
<tr>
<td>J Backward pass search beam width BW2 (log10)</td>
<td>+8.6</td>
</tr>
<tr>
<td>K Final lattice pruning window</td>
<td>+54</td>
</tr>
</tbody>
</table>

Figures 4 and 5 illustrate the speed-AUC tradeoff of each individual parameter for the forward and backward passes respectively (taken from iteration 1 of the optimization experiment). Judging by the slope of these vectors, it appears that on average, the forward pass parameters are more effective in improving speed, while the backward pass parameters are better at increasing AUC. This may explain why the optimization led to a more severe tightening of the forward pass beams, while some of the backward pass parameters were actually relaxed.

**Figure 3:** Progression of the optimization algorithm. Each point is a run of decoding followed by indexing and search. The dashed line represents the target speed of 15 times faster than real time (15FTRT).

**Figure 4:** Speed-AUC tradeoff for the fast match (forward) pass parameters in decoder.

**Figure 5:** Speed-AUC tradeoff for the beam search (backward) pass parameters in decoder.

In the course of developing the experiment pipeline we discovered that in order to obtain reliable speed measurements
we must run decoding in isolation, i.e. with just one decoding process per machine. When multiple processes are running on a multi-core machine simultaneously, they compete for shared resources (memory, bus), which slows down the execution by an unpredictable amount. Since decoding speed is one of our optimization criteria, it is crucial to minimize the amount of noise when measuring the decoding speed.

Constraining the decoding to one process per machine can have a significant impact on the degree of parallelism for the optimization experiment, because machines with eight processors would utilize only one processor at a time. We solved this problem by extracting a random subset of the test data and decoding this subset on a dedicated machine. Three minutes of running time per decode is enough to measure decoding speed, thus all of these shortened decodes can be restricted to a single machine with only one CPU utilized and still complete in about one hour. In the meantime, the full test set can be decoded, indexed and searched using the full bandwidth of the compute cluster.

5. Conclusions and Future Work

This work presents an iterative algorithm for optimizing parameters of a speech decoder at a specified speed. The approach is not tied to any particular system architecture and can be applied to any speech decoder with tunable parameters. The optimization algorithm is general and it can use any performance metric as the objective function (e.g. WER, keyword detection accuracy, topic ID performance, etc.). This algorithm is particularly useful when tuning a speech recognizer for applications that require a certain decoding speed, especially for very fast sub-real-time applications, where the speed-accuracy tradeoff can be severe.

We demonstrated the effectiveness of the optimization algorithm by applying it to the sub-real-time (~0.1xRT) configuration of Byblos speech recognizer, which uses a two-pass decoding strategy with the total of 11 tunable parameters. We set the target speed to 0.067xRT (15 times faster than real time) and optimized for keyword detection accuracy measured by the MAP (AUC) score over all words in the test set. Starting with the decoding speed of 0.113xRT and AUC of 0.4501, we were able to achieve the target speed of 0.067xRT and nearly match the initial performance after 8 iterations of the optimization algorithm.

The efficiency of the algorithm can potentially be improved by allowing the set of parameter update steps \( \Delta \) and the corresponding maximum allowable weights \( w_j \) to adjust dynamically in each iteration. Such adjustment could be useful in cases where parameter sensitivity varies significantly between two operating points. Also if the target speed differs from the initial operating point by a large amount (e.g. starting with 1xRT and optimizing for 0.1xRT), it may turn out to be more efficient to introduce one or more intermediate targets (e.g. 0.7xRT, 0.5xRT, 0.3xRT) and run several iterations of optimization toward each consecutive target, allowing for a more gradual exploration of the speed-accuracy tradeoff.

In the current approach we select a new set of parameters at each iteration of optimization independently of the data points collected during the previous iterations. Future work can attempt to make the optimization algorithm be aware of the results from previous iterations by potentially exploiting dependencies between different parameters.

6. Acknowledgments

The author would like to thank Owen Kimball and Steve Lowe for their comments.

7. References