A TIME-SYNCHRONOUS, TREE-BASED SEARCH STRATEGY IN THE ACOUSTIC FAST MATCH OF AN ASYNCHRONOUS SPEECH RECOGNITION SYSTEM

Ellen M. Eide and Lalit R. Bahl
I.B.M. T.J. Watson Research Center
P.O. Box 218 Yorktown Heights, NY 10598 U.S.A.

1 INTRODUCTION

A common tactic in large vocabulary automatic speech recognition systems is to quickly provide short lists of likely candidate words from the vocabulary of several thousand possibilities. Subsequently, a detailed model for each of the words in a given short list is used to match that word to the acoustic signal. The process of determining the short list, called the fast match [1] reduces to a manageable number the hypotheses investigated by detailed acoustic models.

The flow of our existing decoding algorithm alternates between calls to the detailed match and the fast match modules. A detailed match of a candidate word is performed and a probability distribution of possible ending times of that word is calculated. From this distribution a fast match computation is performed and a list of candidate words to follow the current word is computed. The likelihood of each of these words is computed by the detailed match, along with the end-time probability distribution. This distribution is then used in the fast match computation to determine the next set of candidate words.

In this paper, we describe a change to the flow of the overall algorithm from one which alternates between detailed match and fast match to one which computes all fast match candidates for an entire sentence and stores them in a table accessible for look-up by the detailed match phase which follows the FM calculations. This change is enabled by adding arcs from each leaf node back to root in the fast-match tree. We use a Viterbi beam search to find the most-likely path through the state-space; any word along a path within a threshold of the most-likely path is included as a fast match candidate. We have simplified the phoneme topology from a three-state, left-to-right model to just a single state with a self-loop, which reduces the number of states in the search procedure. We enforce the desired minimum stay of three frames in each phoneme by agglomerating three frames into a "frame triplet" by multiplying together the probabilities of each of the three frames for each phoneme.

During the asynchronous, detailed match phase of decoding [4] when we need candidates over a particular time window, we retrieve them from the previously-computed hypotheses lists associated with those starting times. The acoustic scores returned by the fast match are weighted by a triangular window centered around the most-likely word boundary as determined by the previous detailed match, combined with the language model scores, sorted, and truncated to produce a short list of candidate words to be evaluated in the next call to the detailed match.

2 ADVANTAGES OF THE HYBRID ARCHITECTURE

The hybrid architecture of a synchronous fast match and an asynchronous detailed match is more attractive than either a fully-synchronous or fully-asynchronous system, for reasons outlined in this section.

An asynchronous fast match was efficient for isolated speech, since regions of silence were easily detected and fast match computations could be reliably restricted to speech onsets. However, in a continuous speech domain, the same frame of speech may be included in several fast match computations.

As an example, assume that the candidate list returned by the fast match contains the words ANNE and ANT. In the old procedure, the detailed match would extend the word ANNE and find a non-zero probability of ending between some times t_1 and t_3. A fast match calculation would then be performed in search of words starting in this time region. Also assume that a DM extension of the word ANT returned a non-zero probability of ending between some times t_2 and t_4 and that t_3 is between t_2 and t_4. The fast match calculation performed on this start-time distribution would repeat the calculations done in the FM

calculation due to the end of ANNE for all frames between t_2 and t_3.

This overlap, which is costly in terms of speed when there are many words with overlapping end-time distributions, is eliminated by the synchronous search; performing a fast match calculation on the same frame more than once will not happen, resulting in less time being spent in the fast match module of the decoder.

Furthermore, in the dynamic programming search, the quality of the path history leading up to the beginning of a word weights the score for that word, while in the asynchronous search acoustic path information is not directly represented in a particular word's fast match score, but is indirectly carried through the detailed match. By performing the synchronous fast match backwards in time and the detailed match forwards in time we are able to construct candidate lists which consider complete paths through the utterance.

Conversely, in the detailed match phase of decoding, an asynchronous search is more attractive than a synchronous one, as it allows full word extensions of active paths, giving rise to simple implementations of complex whole-word modeling strategies and ngram language models of any order.

The attractive aspects of each of these searches are retained by using a hybrid decoding architecture.

3 SYSTEM DESCRIPTION

An established method of reducing the number of states to be searched in the fast match procedure is to organize the phonemic representations of the words in the vocabulary in the form of a tree, with a common acoustic model for each phoneme, independent of its context; all phonemes shared from word beginning are tied together into a single node of the tree. Traversing the tree from the root to one of leaves spells out the word in the vocabulary indicated at the leaf. By using this graph for speech recognition we efficiently represent the vocabulary while maintaining constraints of recognizing phoneme sequences that constitute words in the vocabulary.

The work described in this paper requires converting such a fast match tree into a graph capable of representing arbitrary sequences of words by adding arcs which go from each leaf back to the root node. Transition probabilities are assigned to each each arc of the graph as $1/N$ where N is the number of arcs leaving the source node. We implicitly assume a self-loop probability of 0.5 for each node, which would scale the transition arcs out of each node by the same factor, and therefore need not be included in the computation. Furthermore, the transition probability from leaf to root is multiplied by the relative frequency of occurrence of the word in English.

The FM graph is used to constrain a dynamic programming search in which states correspond directly to nodes in the fast match graph. We use the Viterbi algorithm to find the most-likely path through the state-space; any word along a path within a threshold of the most-likely path is included as a fast match candidate.

In our existing decoder, each node of the fast match tree was expanded into a set of three states, with self-loops omitted on the first two states. That omission reduced greatly the number of possible paths through the model and therefore resulted in a faster search than a model with self-loops on each of the states as is used in the detailed match, while enforcing a minimum stay of three frames in each phoneme.

In this work we use a simplified, single-state with self-loop graph topology to reduce the number of active states relative to the previous three-state fast match topology, thereby decreasing the time needed to perform the candidate search. We allow transitions only every third frame which enables us to maintain the desired three-frame minimum duration while at the same time takes advantage of the simple graph topology.

We use a modified Viterbi search [3] to find the best sequence of states for a test sentence, where one state is occupied for each frame triplet in the utterance. At any point in time τ, the score s for a node n in the fast match tree is given by:

$$s(\tau, n) = max_{i}(s(\tau - 3, i)t(i, n))\Pi_{\delta=0}^{\delta=2} p_{n}(\tau - \delta) \quad (1)$$

where $t(i, n)$ is the transition probability associated with the arc connecting node i to node n, $p_n(\tau)$ is the probability of the phoneme associated with node n occurring at time τ as evaluated by context independent phoneme models, and $s(\tau - 3, i)$ is the score of node i for the previous time triplet, i.e. the set of three frames ending at time $\tau - 3$. We obtain context independent models by taking as the probability of each phoneme the maximum probability over all leaves associated with that phoneme in the detailed match models [2]. The maximum in equation 1 is taken over all nodes i which are predecessor nodes to n as defined by the fast match tree and are currently active, where an "active node" is defined as a node whose score is within a user-defined parameter D of the highest scoring node at a given time.

In a forward search, for each active node we store, in addition to its score, the time at which we exited the root of the tree along the best path to that node. For each active node which corresponds to a leaf of the fast match tree, we enter the word in the list of

candidates starting at the stored most-likely starting time of that word. Into the list of candidate scores we enter the difference in scores between its ending state and the best scoring node at time at which the word ended.

The implementation of the fast match in the decoder takes the following steps :

Having computed a set of phoneme probabilities for an utterance, we compute lists of candidate words and their scores over the entire utterance using the modified Viterbi search described above.

During the detailed match phase of decoding [4] when we need candidates over a particular time window, we retrieve them from the previously-computed hypotheses lists associated with those starting times. The acoustic scores returned by the fast match are weighted by a triangular window centered around the most-likely word boundary as determined by the previous detailed match, combined with the language model scores, sorted, and truncated to produce a short list of candidate words to be evaluated in the next call to the detailed match.

4 REVERSING DIRECTION OF CALCULATION

One weakness of the algorithm described above is inherent to the Viterbi search, in which only the best path to a given node at a given time is saved. In a forward Viterbi search, where the calculation begins with the first time triplet and progresses to the final triplet of the utterance, only the highest scoring path through each word is remembered. In some cases, however, the beginning of a word is not clear; for example, when the first of a pair of words along a given path ends with the same phoneme which begins the second word such as "this says," the boundary between the two words is not well defined. In a forward Viterbi search, the end of the word "this" would have a good score for a number of frames, but when the end of "says" is reached, only the single best starting point of the word is available. This could be a number of frames away from where the (forward-progressing) detailed match is asking for candidates.

The weakness of having only a single start time for each word can be overcome by computing the fast match candidates backwards in time. That is, we start the computation with the final time triplet of the utterance and compute toward its beginning. Reversing the calculation requires a reversal of the fast match tree, built by rewriting backwards the phonemic pronunciation of each word and proceeding to build the fast match tree in the normal way, but using the reversed spellings. The dynamic programming scores are computed from the end to the start of the sentence, with the paths constrained by the reversed tree. In the case of the reversed tree, a leaf corresponds to a word begin; when a leaf occurs in the beam search, the associated word is entered in the list associated with the time at which the leaf occurred. No carrying forward of word-begin times is required, which results in a slightly faster search requiring slightly less memory than the forward case.

5 REDUCING DELAY

One weakness of the backwards calculation is the fact that one must wait until the speaker finishes his sentence before beginning the decoding computations. However, if we could detect accurately specific speech events such as silence, we could segment the utterance into chunks of speech surrounded by silence and do the computation on each speech chunk independently, and thereby not be required to wait until the end of the entire sentence to begin computing. Alternatively, we could implement a forward-progressing fast match which stores ends of words and is used by a backward-going detailed match. This solves the problem of having only the single-best path available when tracing back to find word beginnings in a forward search, as well as reduces delay, since we can start the backwards detailed match at a time when the fast match is relatively confident in its top candidate. This embodiment would require a backwards language model.

6 SCORING

Unnormalized Viterbi scores cannot be used as the fast match score, since at each time frame the scores are multiplied with another observation probability which causes the scores at the end of the computation to be much smaller than those at the beginning. Several choices exist for scoring the candidates in the beam. One is the difference of the node's Viterbi score from the best Viterbi score seen at that time frame. This is a natural choice since that criterion was used to define the beam. For example, in the case of computing backwards in time, for a node at time t, this method of scoring takes into account all observations from time t to the end of the utterance, but does not consider observations from time 0 to time $t-1$. Thus, if a path indicates with a high score that a word begins at time t, that word will be given a high score, even if no continuation of the path exists to time 0. Because of this weakness, we have tested an alternative scoring

mechanism which takes into account complete paths from 0 to the end of the utterance for each hypothesis.

In the case of computing backwards in time, an alternative to normalizing the scores by taking the difference from the top of the beam is to keep track of the scores computed forwards in time by the detailed match and multiply the score of each node in the backward fast match search by the score of the root node in the forward detailed search. This normalization procedure scores a given word hypothesis according to its place along a complete path from start to end of the sentence.

7 RESULTS

The time-synchronous fast match algorithm was tested on an in-house continuous speech recognition task. The vocabulary consisted of 20,000 words from the business news domain. The test data consisted of 5294 words uttered by 14 speakers.

The algorithm reduced the computation time of the fast match by approximately 88% (from a total of 9516.3 seconds to 1130.4 seconds required for the fast match computation on a 133MHz processor) with approximately 6% relative increase in the word error rate of the speech recognition system (12.90 went to 13.68).

REFERENCES

[1] L.R. Bahl, S.V. De Gennaro, P.S. Gopalakrishnan, R.L. Mercer. "A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition", IEEE Transactions on Speech and Audio Processing, vol. 1, no. 1,pp 59-67. January 1993.

[2] L.R. Bahl et al. "Performance of the IBM Large Vocabulary Continuous Speech Recognition System on the ARPA Wall Street Journal Task." ICASSP 1995, vol 1, pp 41-44.

[3] G.D. Forney, Jr. "The Viterbi Algorithm," Proc. IEEE, vol 61, pp 268-278. 1973.

[4] P.S. Gopalakrishnan, L.R. Bahl, R.L. Mercer. "A Tree Search Strategy for Large Vocabulary Continuous Speech Recognition." ICASSP 1995, vol 1, pp 572-575.