ELSEVIER

Speech Communication 14 (1994) 61–70

SPEECH COMMUNICATION

Books on tape as training data for continuous speech recognition **

G. Boulianne *, P. Kenny, M. Lennig †, D. O'Shaughnessy, P. Mermelstein

INRS-Télécommunications, 16, place du Commerce, Verdun (Ile-des-Soeurs), Québec, Canada H3E 1H6

(Received 25 January 1993; revised 20 August 1993)

Abstract

Training algorithms for natural speech recognition require very large amounts of transcribed speech data. Commercially distributed books on tape constitute an abundant source of such data, but it is difficult to take advantage of it using current training algorithms because of the requirement that the data be hand-segmented into chunks that can be comfortably processed in memory. In order to address this problem we have developed a training algorithm which is capable of handling unsegmented d ıta files of arbitrary length; the computational requirements of the algorithm are linear in the amount of data to be processed and the memory requirements are constant.

Zusammenfassung

Trainingsalgorithmen für Systeme zur Erkennung natürlicher Sprache benötigen eine sehr große Anzahl etikettierter Daten. Eine unerschöpfliche Quelle für solche Daten sind leicht erhältliche, auf Tonbänder aufgenommene Bücher. Es ist jedoch schwierig, von dieser Datenquelle zu profitieren, da die herkömmlichen Trainingsalgorithmen nur dann benutzt werden können, wenn die Daten zuvor manuell in kleinere Teile segmentiert worden sind, um im Speicher des Rechners verarbeitet werden zu können. Um dieses Problem zu lösen, haben wir einen Trainingsalgorithmus entwickelt, der unsegmentierte Daten arbiträrer Länge verarbeiten kann. Die Anforderung an die Rechenleistung steigt linear im Verhältnis zu der Menge der Daten die Speicheranforderung konstant bleibt.

Résumé

Les algorithmes d'apprentissage pour la reconnaissance de la parole continue ont besoin de très grandes quantités de données sous forme de parole transcrite. Les livres sur cassette, disponibles commercialement,

* Corresponding author.

** This work was supported by the Natural Sciences and Engineering Research Council of Canada.

† Also with Bell-Northern Research, Montreal, Canada.

SSDI 0167-6393(93)E0079-B

représentent une source de telles données, abondante mais difficile à exploiter avec les algorithmes d'apprentissage actuels, car ceux-ci exigent que les données soient d'abord segmentées, à la main, en blocs assez petits pour être traités en mémoire. Pour résoudre ce problème, nous avons mis au point un algorithme d'apprentissage capable de traiter des fichiers de données de longueur arbitraire; les besoins en calculs de cet algorithme sont linéairement proportionnels à la longueur des données et la quantité de mémoire requise est constante.

Key words: Hidden Markov model; Continuous speech recognition; Training algorithm; Speech segmentation; Speech labeling; Viterbi decoding

1. Introduction

It is a truism to say that large vocabulary continuous speech recognition applications require very large amounts of acoustic training data. An indication of the amount of data needed can be obtained by extracting some statistics from a large dictionary. For instance, the *Moby Pronunciator* contains transcriptions of 167 000 words and phrases (one transcription in each case). We translated the transcriptions using a standard phoneme inventory of 40 symbols and counted the number of triphones that would have to be modelled if this vocabulary were to be used in a continuous speech recognizer. We found that there were about 17 000 within-word triphones and about 51 000 cross-word triphones (assuming that words can appear in any order). This suggests that tens or even hundreds of thousands of words of training data will be needed to obtain respectable recognition accuracies.

The current generation of speech processing algorithms require that training utterances be spoken in isolated-sentence mode so data collection is a very expensive operation.

Although books on tape constitute an abundant source of natural speech data whose transcriptions are readily available, they cannot be used for training purposes without manual segmentation of the data. Note that this is not simply a matter of automatically identifying where pauses occur in the training data since a pause may appear between any two words in the training text. Even the much simpler task of automatic sentence detection in the text generates errors for general purpose, large vocabulary English (Paul and Baker, 1992). In this paper we will describe a variant of the Viterbi algorithm that we have developed to deal with this problem.

We have applied this algorithm to 6 readily available books on tape. Around two hours of speech (roughly 17 000 words) from each book were used for training purposes. The training data was partitioned into files of 2 to 10 minutes (so that they conveniently fall on chapter boundaries) although training has been done successfully on files of up to one hour of uninterrupted speech. Recognition results on these books are reported in (Kenny et al., 1992, 1993).

2. Searching with a sliding window

The major computational burden in training phonetic hidden Markov models consists of finding the Viterbi alignment of training data with a hidden Markov model constructed from a training script in the following way. For each of the words in the training script, a phonetic graph (Kenny et al., 1993) is constructed with the property that there is a 1–1 correspondence between possible paths through the graph and phonetic transcriptions of the word. Graph reduction can be applied when branches are shared by phonetic variants, but always preserving the 1–1 correspondence property. Next a phonetic graph corresponding to the entire training script is constructed in such a way as to allow for the possibility that a speaker may choose to leave a pause between any two successive words. Each of the branches in this graph (except for null branches) carries a phoneme label; the hidden Markov model referred to above is constructed by replacing each of the branches by the phonetic HMM corresponding to the branch label. (An example of this construction will be given later in Fig. 3.)

The most widely used method for aligning speech data with such a hidden Markov model is

the frame-synchronous Viterbi algorithm (Rabiner, 1989). In a naive implementation, the computational complexity of this algorithm is proportional to the square of the utterance length. (The size of the trellis is $T \times S$, where T is the length of the utterance and S the number of states in the HMM; S is proportional to T.) In practice, when the algorithm is applied iteratively in training, it is possible to use segmentation found on each iteration to guide the alignment in the next iteration so that the computational complexity is linear in the utterance length. (This is the idea of semi-relaxed training described in (Deng et al., 1991), also described in (Pieraccini, 1991) as the Viterbi piecewise alignment. An extension to the multiple transcription case is described in Section 3.) However, since the final decision about the optimality of a path through the model cannot be made until all of the data has been processed, the memory requirements of the algorithm (in both cases) are proportional to the utterance length so that the data has to be pre-segmented into chunks of manageable size. The alignment algorithm that we present in this paper has constant memory requirements (independent of the utterance length) and so it can handle utterances of unlimited length.

In order to perform Viterbi alignments using a constant amount of memory (independent of the utterance length) it is necessary to make some decisions concerning the optimality of partial paths through the model before the data has been processed in its entirety. This can be accomplished by a *sliding window algorithm* which can be summarized as follows:

Starting with existing ancestor paths at time t, prolong the paths up to time $t + L$ by a Viterbi search. At time $t + L$, choose a limited number N of candidate paths; take these N candidate paths as the ancestors for a new window starting at time $t + L$ and discard the others.

Here L is the window length. Pruning to a constant number of candidates at each window makes memory use constant (computation is still proportional to the utterance length). Care has to be taken in pruning candidate paths at window boundaries since the algorithm will only find the optimal path if it is always included in the candidate lists. This general procedure gives rise to a number of variants according to the way the candidate paths are chosen at the window boundaries.

One can choose the best path according to a heuristic value computed from the scores. This is in fact the algorithm of Kriouile et al. (1990). This procedure is never optimal, since there is no way to ensure that the globally optimal path will be chosen at every window.

Another possibility is to prune those paths that have a score which differs from that of the best path by a preset threshold; this is the idea of a beam search. (To be precise, what we are describing differs from the standard beam search (Lee, 1990) in that thresholding is applied only at window boundaries and not elsewhere.) Note that although in principle this procedure is not optimal, in practice a beam search can be made as close as desired to optimality by using a large enough threshold (Pieraccini et al., 1990).

Our previous experience with *two-phone lookahead* (Kenny et al., 1993), suggests that looking a few phones ahead in the utterance could give valuable information about the optimality of the current partial paths. The next section presents a new pruning rule using this kind of information.

2.1. Looking ahead

In the field of digital communications, Viterbi decoding decisions have often to be made *before* a message has been received in its entirety. A commonly used technique (Lee and Messerschmitt, 1988) is based on the observations that

1. the optimal path will be included among the highest scoring paths at any given time (provided the beam threshold is sufficiently generous),
2. all of the highest scoring paths are descended from a common ancestor in the not-too-remote past.

The idea of a common ancestor or "immortal node" was also introduced for speech recognition in (Bridle et al., 1982). These observations sug-

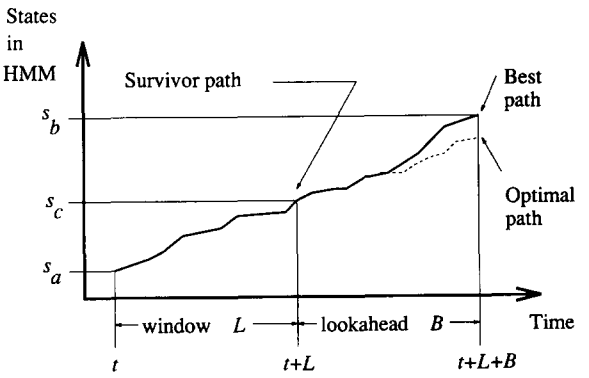

Fig. 1. Selection of survivor path by look-ahead.

gest the following pruning strategy, which reduces to one the number of paths which are passed from one window on to the next. Referring to Fig. 1:

1. Initial conditions: Assume that after searching the previous window (ending at time t) there is one surviving path. In the figure, s_a denotes the state of the model which is occupied at time t on this path.
2. Search: Extend the starting path (using the Viterbi algorithm) into all partial paths ending at time $t+L+B$. Record the partial path having the best score at this time. In the figure, this path is indicated by the solid line; the state on this path which is occupied at time $t+L+B$ is denoted by s_b and the state occupied at time $t+L$ is denoted by s_c.
3. Survivor selection: Backtrack along the path recorded in Step 2 to find its ancestor at time $t+L$. This path is selected as the only survivor path.
4. Loop: Set a new window beginning at time $t+L$ with the survivor as the starting path and repeat the steps (in other words, set $t=t+L$, $s_a=s_c$ and go to Step 1).

For this lookahead rule to be optimal we only require that the lookahead B be sufficiently large that the best path and the overall optimal path merge together at some time past $t+L$. In the limit, when B is large enough so that the window spans the entire data file, the search reduces to a full Viterbi search. The search can be made as close to optimal as desired by using a large B. Experimental results will show that convergence of the paths can be obtained for useful values of L and B.

2.2. Block Viterbi searching

Our approach to the alignment problem in training and the search problem in recognition is based on block Viterbi decoding (Kenny et al., 1993) which computes a Viterbi segmentation directly from a phonetic graph without explicit reference to the underlying phonetic HMMs. Although block Viterbi decoding is less efficient than standard Viterbi decoding (by about a factor of two), it has several attractive features. It can model non-Markovian segment-level features (such as constraints on phone durations) and it is particularly simple to implement semi-relaxed alignment constraints in this framework; furthermore, the block Viterbi algorithm can accommodate *any* method of scoring phone segments (whereas the standard Viterbi algorithm only works if phone segments are modelled with HMMs).

2.2.1. The phonetic trellis

Block Viterbi decoding uses a trellis built from a phonetic graph (Kenny et al., 1993) rather than an HMM. *Nodes* in the phonetic graph are joined by *branches* and each branch carries a phone label. Since there is a 1–1 correspondence between phonetic transcriptions and paths through the graph, a block Viterbi search of a phonetic graph yields a phonetic transcription together with a segmentation (i.e., a specification of entry and exit times for each of the phones in the transcription). Although any type of segmental model would be suitable to score the phone branches, in the following we assume that HMMs are used as underlying models. The search of the phonetic graph does not yield the alignment of the data with the internal states of the phonetic HMMs.

Fig. 2 represents the trellis constructed from a phonetic graph and acoustic observations made at times $t=1,\ldots,T$; the point (t, n) corresponds to a time t and a node n of the phonetic graph. Suppose that the nodes n and n' in the graph are

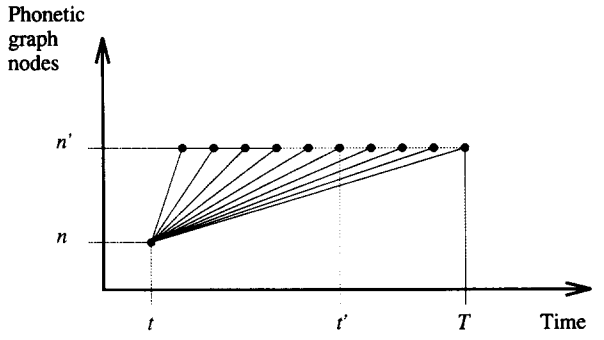

Fig. 2. Phonetic trellis for block Viterbi algorithm.

joined by a branch carrying a phone label f. In the trellis, there will be a branch joining (t, n) and (t', n') for each t' in the interval $[t+1, T]$. When using an HMM as the underlying model, the scores associated with this branch will be the Viterbi likelihoods of the data from time $t+1$ to time t' calculated using the f model. We will denote these likelihoods by $V([t+1, t'] | f)$ and refer to them as "point scores". For a given time t, all of the point scores $V([t+1, t'] | f)$ (for $t < t' \leqslant T$) can be calculated by the standard Viterbi algorithm on the HMM using a single trellis, and duration constraints can be imposed simply by setting $V([t+1, t'] | f)$ to zero for certain values of t'; in the case of minimum and maximum durations, for all t' for which $t' - t$ is less than the minimum duration or more than the maximum duration. Some branches, such as null branches, do not carry a phone label and have no cost or models associated with them. Their point scores are simply set to one for t' in the interval $[t, T]$.

Just as in the case of the standard Viterbi algorithm, the optimal path through the phonetic trellis can be found by a breadth-first search of the trellis (Kenny et al., 1993) and the entry and exit times of each of the branches on the optimal path give the Viterbi segmentation.

2.3. Block Viterbi decoding with lookahead pruning

The sliding window and lookahead principles can be modified to accommodate the block Viterbi algorithm in the following way (step numbers here mirror those of Section 2.1):

1. Initial conditions: Assume that after searching the previous window (ending at time t) there is only one surviving path in the phonetic trellis.

1a. Precomputation: Compute point scores for all phones in the inventory and for all times in the interval $[t, t+L+B]$.

2. Search: Extend the starting path by the block Viterbi algorithm into all possible partial paths ending at time $t+L+B$. Find the partial path having the best score at time $t+L+B$.

3. Backtracking: Backtrack along the path found in Step 2 until a trellis point (t', n') is found such that $t' \leqslant t+L$. Truncate the path at (t', n') and discard all other paths.

4. Loop: repeat from Step 1a with a new window extending from t' to $t'+L+B$, starting the search at the trellis point (t', n').

Note that, in this case, a phone boundary is hypothesized at time $t+L+B$ on all of the paths found in Step 2. (Unlike the standard Viterbi trellis, points in the phonetic trellis always correspond to hypothetical phone boundaries.) Since there is no reason why there should be a phone boundary at this time on the optimal path, it may appear that our lookahead procedure is no longer correct. Recall however that the correctness of the lookahead procedure only requires that the highest scoring path at time $t+L+B$ and the optimal path coincide up to time $t+L$. Obviously, provided B is sufficiently large, this condition will continue to hold even if a phone boundary is forced to occur at time $t+L+B$, so this is not a problem in practice.

Similar considerations dictate a slight modification to the backtracking procedure (Step 3). Since there is no reason to believe that a phone boundary should occur at time $t+L$, we have to backtrack until the first phone boundary *prior* to $t+L$ is found. As a result, the window advance is no longer deterministic. (The new window starts at time t' rather than time $t+L$.)

As mentioned in Section 2, a general sliding window search admits many different types of pruning at window boundaries. The particular

type of pruning that we have chosen (looking ahead and backtracking) is particularly suited to block Viterbi decoding since it has the property that only one partial path is passed from one window to the next. Had we implemented the usual beam search type of pruning, we would have had to retain multiple hypotheses not only at time $t + L$ but at every time in a neighborhood of $t + L$ which is sufficiently big that it is guaranteed to contain a phone boundary on the optimal path.

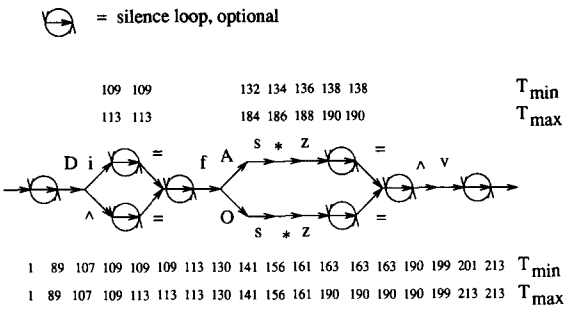

Fig. 3. Minimum and maximum entry times for nodes in phonetic graph.

3. Semi-relaxed training

The sliding window block Viterbi search that we have described can be used to perform a Viterbi search of an arbitrary phonetic graph using unsegmented data files of arbitrary length. As such, it can be used either for training or recognition. Our focus in this paper is on the training problem (we actually use quite a different approach in recognition (Kenny et al., 1992, 1993). Since training is an iterative procedure, it is natural to use information obtained in each iteration to guide the search in the following iteration. This is the idea of semi-relaxed training (Deng et al., 1991; Pieraccini, 1991; Boulianne et al., 1992). In this section we show how this type of information can be incorporated into our search algorithm and report some experimental results obtained when the algorithm was used in training HMMs for continuous speech recognition applications.

3.1. Use of prior segmentation knowledge

Fig. 3 illustrates how a training script is used to build a phonetic graph representing all the possible ways of uttering the string of words in the script. In this example the words uttered are "The forces of...". Pronunciation graphs are first constructed for the individual words using the transcriptions in the dictionary; a limited amount of graph reduction is performed. Note that the words "the" and "forces" each have two transcriptions. Optional silences between successive words are handled by means of the silence loop. Since the block Viterbi algorithm incorporates maximum duration constraints on phonemes, pauses have to be transcribed using strings of silence phonemes of arbitrary length. This is handled in the silence loop by associating a single silence phoneme with the top and middle branches; the bottom branch is a null branch (to allow for the possibility that there may be no pause in the utterance). Finally, the various components of the pronunciation graph are integrated using null branches (indicated by the label " = "). (We did not attempt to implement phonological rules applying across word boundaries.)

3.1.1. Node-time assignment

Suppose that on a particular training iteration we are aligning a given window in the training file with the corresponding pronunciation graph. In the absence of any prior information, the part of the trellis that has to be searched consists of all trellis points (t, n), where t ranges over all times in the window and n ranges over all nodes in the graph. In order to reduce the portion of the trellis that has to be searched we can argue as follows.

Fix a node n. If, in the path through the graph found on the preceding iteration, the node n was visited at time t, then it is reasonable to say that, on the current iteration, the node can only be visited at some time in the interval $[t - \Delta, t + \Delta]$ (for some suitably chosen constant Δ). A problem with this scheme as it stands is that the node n may *not* have been visited on the preceding iteration (because the phonetic transcription may

change from one iteration to the next). In such a case we can at least say that if n' is any node in the graph which precedes n and n' is visited at time t' in the preceding iteration, then, in the current iteration, n cannot be visited prior to time $t' - \Delta$. A somewhat tighter lower bound can be obtained by taking account of the minimum duration thresholds for the phone(s) on the path(s) joining n and n'. An upper bound on the times that n can be visited can be found in the same way. Thus for nodes n that are not visited on the preceding iteration we can assert that, on the current iteration, they can only be visited at times falling in an interval of the form $[t_{\min} - \Delta, t_{\max} + \Delta]$.

This type of node-time assignment (which incidentally is used in critical path scheduling and PERT (Siddall, 1972) for completion times of tasks in a project) is illustrated in Fig. 3. The "bottom" path through the graph was the path found on the preceding iteration of training. For nodes on this path (other than nodes in the silence loops) $t_{\min}$ and $t_{\max}$ coincide and their common value is the visitation time on the preceding iteration; these times are indicated in the lines marked $T_{\min}$ and $T_{\max}$ situated under the graph. For the nodes which are not visited on the preceding iteration, times are assigned based on the assignments for nodes on the bottom path.

Once the node-time assignments have been determined for each of the nodes in the phonetic graph, it is a simple matter to determine which nodes can be visited in the course of searching a given window.

Fig. 4 indicates, for a data file which was not among the training files used to estimate the models, the subsets of the phonetic trellis ("uncertainty regions") that have to be searched in each window in order to find the optimal path.

The size of the uncertainty regions is controlled by the parameter Δ whose value has to be determined empirically so as to ensure that the uncertainty regions are guaranteed to contain the optimal path.

As well as limiting the portion of the phonetic trellis that has to be searched, the node-time assignment can also be used to reduce the point score calculations in each window. Firstly there is

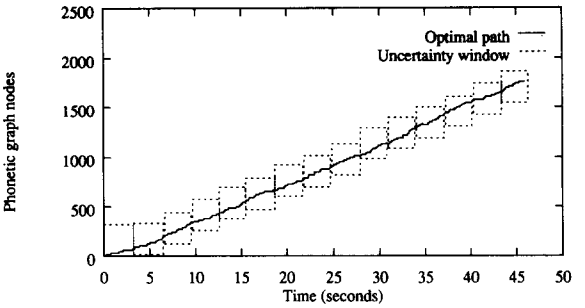

Fig. 4. Typical uncertainty regions obtained by node-time assignment.

phone pruning: the point scores need only be calculated for phones f with the property that there is a branch in the phonetic graph carrying the f label which joins two nodes both of which can be visited in the course of the window. Secondly there is *time pruning*: in order to score such a branch, the only point scores $V([t+1, t'] | f)$ that are needed are those defined by times t and t' satisfying the conditions

1. t is among the times that the node n can be visited,
2. t' is among the times that the node n' can be visited,
3. constraints on the duration of the f phone are respected.

3.2. Experimental results

Data for our experiments consisted of 105 minutes of continuous speech uttered by a single male speaker, taken from a commercially distributed book on tape. The speech rate was about 170 words/minute. The data was stored in twelve files (2 or 3 files per chapter) and was sampled at 16 kHz and blocked into frames of 30 ms, spaced 10 ms apart. The first 8 MFCC coefficients (Davis and Mermelstein, 1980) and their first difference (Deng et al., 1991) were used to form one 15-dimensional vector for each frame (the loudness coefficient C_0 was used only for the difference coefficients).

The text of the book was read using an optical character reader and manually verified to correct for scanning errors (about 2%) and speaker er-

rors (less than 1%). Phonetic graphs corresponding to the training scripts were constructed using a 60 000-word pronunciation dictionary which contains an average of 2.2 phonetic transcriptions per word. Typically each training script was about 1100 words long; the corresponding phonetic trellises had 14 000 nodes and were 36 000 frames long.

A total of 139 context-dependent phone models were trained. Each of the models was a left-to-right Gaussian mixture HMM having 25 mixture components per transition and a single covariance matrix per phoneme. (That is, the covariance matrices associated with the mixture components used to model the various allophones of a given phoneme were tied.)

On each iteration of training, a phonetic transcription and segmentation of the data in each file was found using the sliding window block Viterbi search. This search does not keep track of the alignment of the individual frames with the internal transitions of the phone models (memory requirements would be greatly increased if it did). These internal alignments are therefore calculated in a (relatively inexpensive) second pass through the data and new models were generated by the standard reestimation procedures.

The object of the experiments was to demonstrate the validity of the lookahead procedure, determine suitable values for window length L and lookahead interval B, and estimate the computational and memory requirements when the sliding window block Viterbi search on each iteration is implemented in such a way as to take advantage of segmentation information extracted from the preceding iteration.

3.2.1. Path convergence and lookahead interval

Fig. 5 is typical of what is observed when backtracking the N best paths at the end of a window. In this example, B is 4 sec and L is made equal to zero to allow backtracking over the entire interval from 0 to 4 sec. The 75 best paths (at 4 sec) merge in a single path in less than 1.50 sec. Again the speech file used here had not been seen during the training. This suggests that the optimal path could merge with the best path in less than 1 sec most of the time, i.e., that a

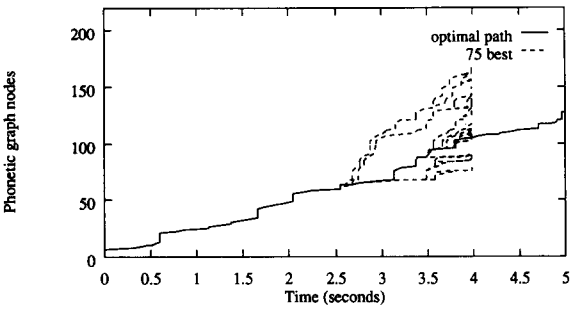

Fig. 5. Traces of 75 best scoring paths at window end.

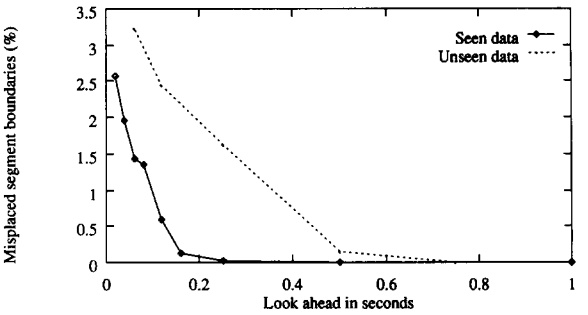

Fig. 6. Segment errors as a function of lookahead interval.

lookahead interval B of a second or less could guarantee an optimal search.

An experiment was run to determine exactly how large the lookahead interval B has to be for optimality. Fig. 6 shows estimated errors in segment boundary location as a function of the lookahead interval B, for speech data seen during previous iterations of training training and for speech that has never been seen before. The total of window length (L) plus lookahead interval (B) was kept constant at 4 sec, while lookahead interval B was varied from 0.06 to 2.0 sec. There were 3786 and 3873 segments in the seen and unseen speech data, respectively.

Errors were estimated by comparing with a reference segmentation obtained with a 3 sec lookahead interval. As pointed out in Section 2.1, the search becomes optimal when a large enough value of B is reached; we found no change in segmentation for any B beyond 0.8 sec (up to 3 sec). Thus the 3 sec segmentation can be assumed

to be optimal, and we considered differences from this segmentation as errors. The figure shows that using a lookahead interval of more than about 0.8 sec is not necessary, even for speech that has not been seen before.

The window length L is made as large as permitted by computation and memory limits. In practice, we use a length of 3 sec. The overhead for one second lookahead then accounts for only one-fourth of total processor and memory use.

3.2.2. Computation and memory requirements

For a typical speech file of 6 minutes, the phonetic graph has about 14000 nodes and is 36000 frames long. In order to guarantee that the uncertainty regions always contain the optimal path we set the parameter Δ to be 2 sec. (This is more generous than is really necessary. After a few iterations of training have been performed, large shifts in segment boundaries between successive iterations are rarely observed.) With this value of Δ, the number of nodes that can be visited in an interval of length 4 sec (recall that we took L to be 3 sec and B to be 1 sec) is about 300.

We found that under these conditions, phone pruning reduced the point score calculations for each window by about a factor of 2 and that time pruning reduced the calculation of the remaining point scores by another factor of 2. The total computation for the sliding window block Viterbi search was about 15 times real time on an HP Apollo 9000/720 workstation. A full training iteration took an additional 0.5 times real time (to perform the within-model Viterbi alignments and the reestimation procedures).

Under the same conditions, the memory required to search a window is about 10 Mbytes; the trellis occupies about a third of this and the precomputed point scores occupy the remainder.

4. Conclusion

Extending the sliding window idea, we developed a training algorithm requiring a fixed amount of memory, that can be used on unsegmented, unlimited length speech utterances. In our experiments selection of a single survivor path could be made optimal by looking at its future less than 1 sec ahead. Retaining only one survivor at each window is particularly efficient when performing a block Viterbi search which incorporates phoneme duration constraints in a natural way, as well as segmentation information derived from prior training iterations. This algorithm does not require training utterances to be spoken in isolated-sentence mode, so it makes available abundant sources of natural speech data. We were able to train HMM models from 6 books on tape partitioned into speech files of 6 minutes on average, varying from 2 to 57 minutes.

5. References

G. Boulianne, P. Kenny, M. Lennig, D. O'Shaughnessy and P. Mermelstein (1992), "HMM training on unconstrained speech for large vocabulary, continuous speech recognition", *Proc. ICSLP*, Banff, October 1992, pp. 229–232.

J.S. Bridle, M.D. Brown and R.M. Chamberlain (1982), "An algorithm for connected word recognition", *Proc. IEEE Internat. Conf. Acoust. Speech Signal Process.*, Paris, May 1982, pp. 899–902.

S.B. Davis and P. Mermelstein (1980), "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. ASSP-28, pp. 357–366.

L. Deng, P. Kenny, M. Lennig, V. Gupta, F. Seitz and P. Mermelstein (1991), "Phonemic hidden Markov models with continuous mixture output densities for large vocabulary word recognition", *IEEE Trans. Signal Processing*, Vol. 39, pp. 1677–1681.

P. Kenny, R. Hollan, G. Boulianne, H. Garudadri, Y.-M. Cheng, M. Lennig and D. O'Shaughnessy (1992), "Experiments in continuous speech recognition with a 60000 word vocabulary", *Proc. ICSLP*, Banff, October 1992, pp. 225–228.

P. Kenny, R. Hollan, V. Gupta, M. Lennig, P. Mermelstein and D. O'Shaughnessy (1993) "A* – Admissible heuristics for rapid lexical access", *IEEE Trans. Speech and Audio Processing*, Vol. 1, No. 1, pp. 49–58.

P. Kenny, G. Boulianne, H. Garudadri, S. Trudelle, R. Hollan, M. Lennig and D. O'Shaughnessy (1994), "Experiments in continuous speech recognition using books on tape", *Speech Communication*, Vol. 14, No. 1, February 1994, pp. 49–60.

A. Kriouile, J.F. Mari and J.P. Haton (1990), "L'algorithme VITERBI-BLOC pour la reconnaissance de la parole continue", *XVIIIèmes Journées d'étude sur la Parole*, pp. 207–211.

E.A. Lee and D.G. Messerschmitt (1988), *Digital Communication* (Kluwer Academic Publishers, Boston).

K.F. Lee (1990), "Context-dependent phonetic hidden Markov models for speaker-independent continuous speech recognition", *IEEE Trans. Acoust. Speech Signal Process.*, Vol. 38, No. 4, pp. 599–609.

D.B. Paul and J.M. Baker (1992), "The design for the Wall Street Journal-based CSR corpus", *Proc. DARPA Speech and Natural Language Workshop*, February 1992, pp. 357–362.

R. Pieraccini (1991), "Speaker independent recognition of Italian telephone speech with mixture density hidden Markov models", *Speech Communication*, Vol. 10, No. 2, pp. 105–115.

R. Pieraccini, C.H. Lee, E. Giachin and L.R. Rabiner (1990), "Implementation aspects of large vocabulary recognition based on intraword and interword phonetic units", *Proc. DARPA Speech and Natural Language Workshop*, June 1990, pp. 311–318.

L.R. Rabiner (1989), "A tutorial on hidden Markov models and selected applications in speech recognition", *Proc. IEEE*, Vol. 77, No. 2, pp. 257–286.

J.N. Siddall (1972), *Analytical Decision-making in Engineering Design* (Prentice-Hall, Englewood Cliffs, NJ).