

We now define a reduced-range level threshold as $M_T \cdot \phi_{\ell-1}$ where M_T is a defined parameter, and search the range $m_1(\ell-1) \leq m \leq m_2(\ell-1)$ to find the indices S_ℓ^1 and S_ℓ^2 such that

$$S_\ell^1 = \arg \max_{m_1(\ell-1) \leq m \leq m_2(\ell-1)} \left[\frac{\bar{D}_{\ell-1}^B(m)}{m} > M_T \cdot \phi_{\ell-1} \quad \forall m \leq S_\ell^1 \right] \quad (7.31)$$

$$S_\ell^2 = \arg \max_{m_1(\ell-1) \leq m \leq m_2(\ell-1)} \left[\frac{\bar{D}_{\ell-1}^B(m)}{m} > M_T \cdot \phi_{\ell-1} \quad \forall m \geq S_\ell^2 \right]. \quad (7.32)$$

To see what is achieved by the beginning range reduction, consider Figure 7.14, which shows a sequence of levels and the resulting ending ranges with and without range reduction (top graph), and the way in which the reduced range is determined (bottom graph). For the level shown, the best normalized distance score, $\phi_{\ell-1}$, is determined as the smallest value of the curve, and the threshold $M_T \cdot \phi_{\ell-1}$ is shown as a level above $\phi_{\ell-1}$ (clearly $M_T \geq 1.0$). The initial reduced range point, S_ℓ^1 , is the last index (beginning from $m = m_1(\ell-1)$) where the curve is always above the threshold; similarly the index S_ℓ^2 is the last index (beginning from $m = m_2(\ell-1)$) where the curve is always above the threshold. The reduced range means that the computation at the next level is smaller as seen at the top of Figure 7.14. Depending on the value of M_T , the reduced range can get smaller (as $M_T \rightarrow 1$) or larger (as $M_T \rightarrow \infty$). It should be clear that too small a value of M_T will allow the best path to be prematurely eliminated; hence proper choice of M_T is essential.

2. Global range reduction— ϵ

The idea behind global range reduction is to reduce the search range along the reference axis, for each test frame, by tracking the global minimum, and allowing only a range around the global minimum. Thus for each test frame, m , at each level ℓ , and for each reference pattern, $\mathcal{R}_v$, we determine the local minimum, $c(m)$, as

$$c(m) = \arg \min_{c(m-1)-\epsilon \leq n \leq c(m-1)+\epsilon} [D_\ell^v(m-1, n)] \quad (7.33)$$

in which $D_\ell^v(m-1, n)$ is defined to be the best distance at level ℓ using reference $\mathcal{R}_v$ at test frame $m-1$ to reference frame n (as determined by the local alignment path) and with $c(1)$ defined to be 1. Figure 7.15 illustrates the global range reduction for a typical level building search. For global range reduction to be effective we must have the reduced range, $2\epsilon + 1$, be smaller than the typical reference pattern width.

3. Test pattern ending range— δ_{END}

The idea here is to allow a range of test pattern ending frames, rather than restricting the ending frame to $m = M$. This feature provides a measure of robustness to test pattern endpoint errors. If we extend the end of the test pattern by δ_{END} frames, the global level building solution is modified to be

$$D^* = \min_{1 \leq \ell \leq L_{\max}} \min_{M-\delta_{END} \leq m \leq M} [\bar{D}_\ell^B(m)]. \quad (7.34)$$

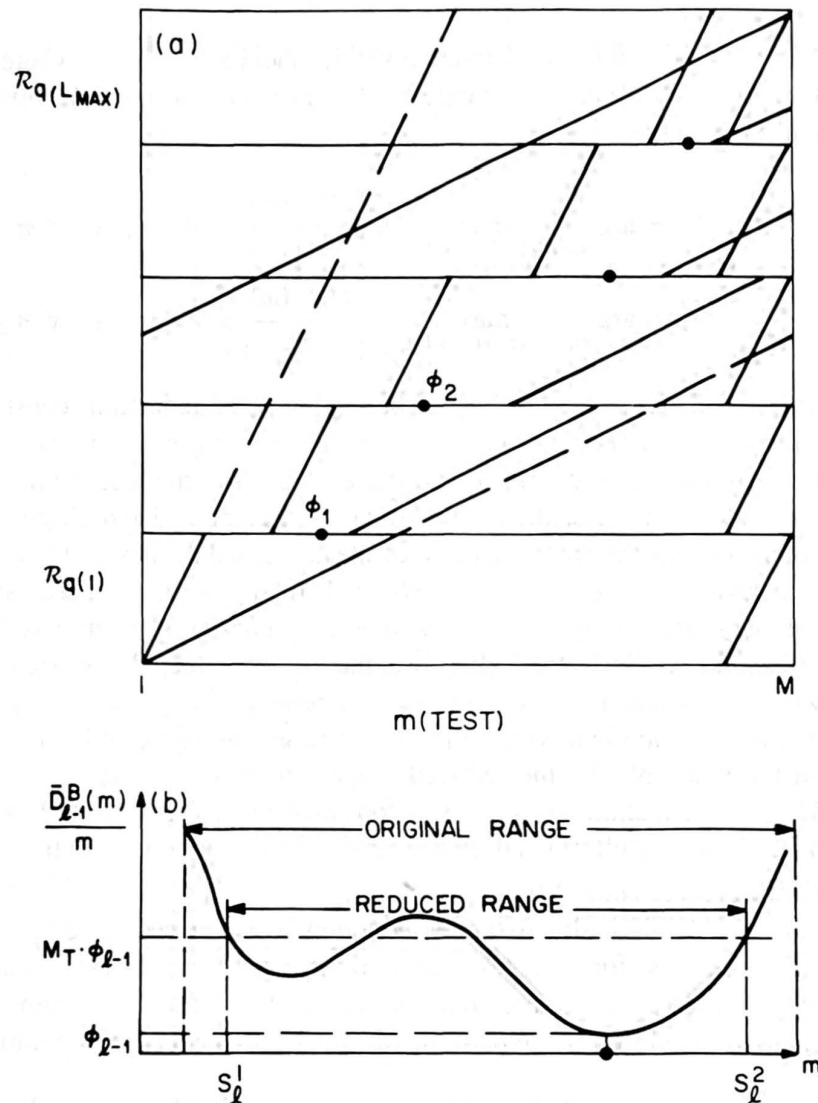


Figure 7.14 Illustration of beginning range reduction (after Myers and Rabiner [4]).

4. Reference pattern uncertainty regions— δ_{R_1} , δ_{R_2}

To account for coarticulation of words across word boundaries, the level building algorithm allows a range of beginning and ending frames of the reference pattern. In this manner, at any level, the path can begin over the range $1 \leq n \leq 1 + \delta_{R_1}$ (where n is the reference pattern index), and end at any frame in the range $N_v - \delta_{R_2} \leq n \leq N_v$. For appropriate values of δ_{R_1} and δ_{R_2} , it is possible to have a path which skips $(\delta_{R_1} + \delta_{R_2})$ frames at highly coarticulated word boundaries (e.g., the boundary between six and seven in the string “six-seven”). Figure 7.16 illustrates the use of reference pattern uncertainty regions in level building.

A summary of the four implementation aspects of level building, as discussed in this section, is shown in Figure 7.17 in which all four features are combined in a single

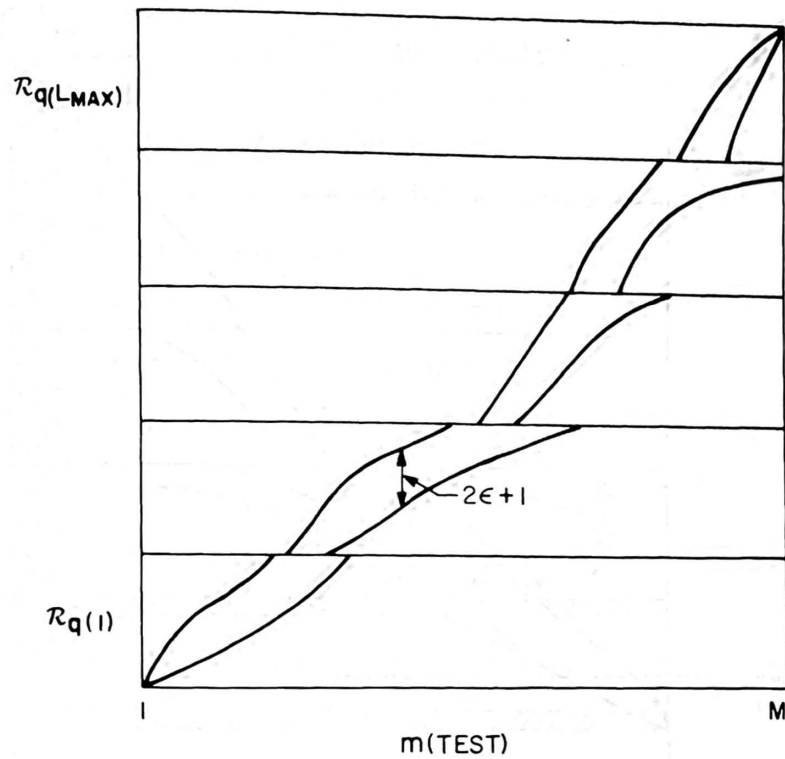


Figure 7.15 Illustration of global range reduction (after Myers and Rabiner [4]).

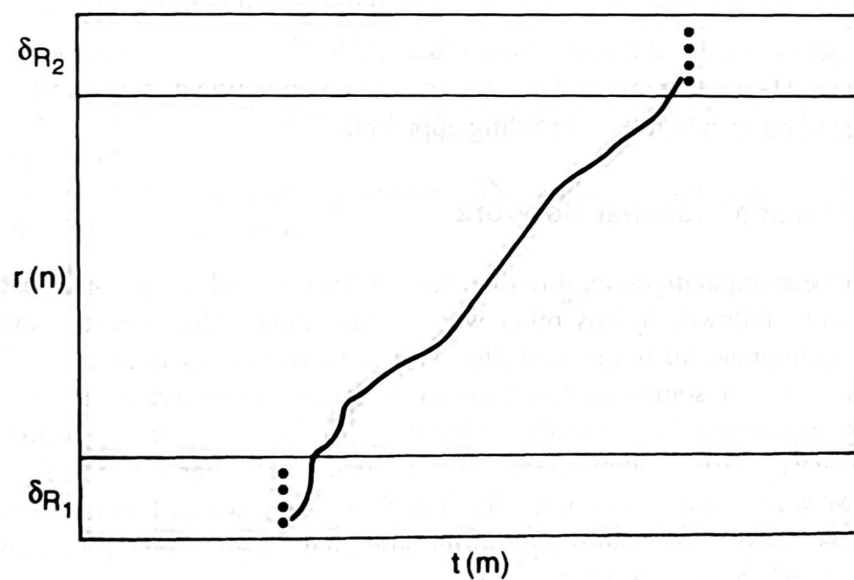


Figure 7.16 Illustration of the use of reference pattern uncertainty regions (after Myers and Rabiner [4]).

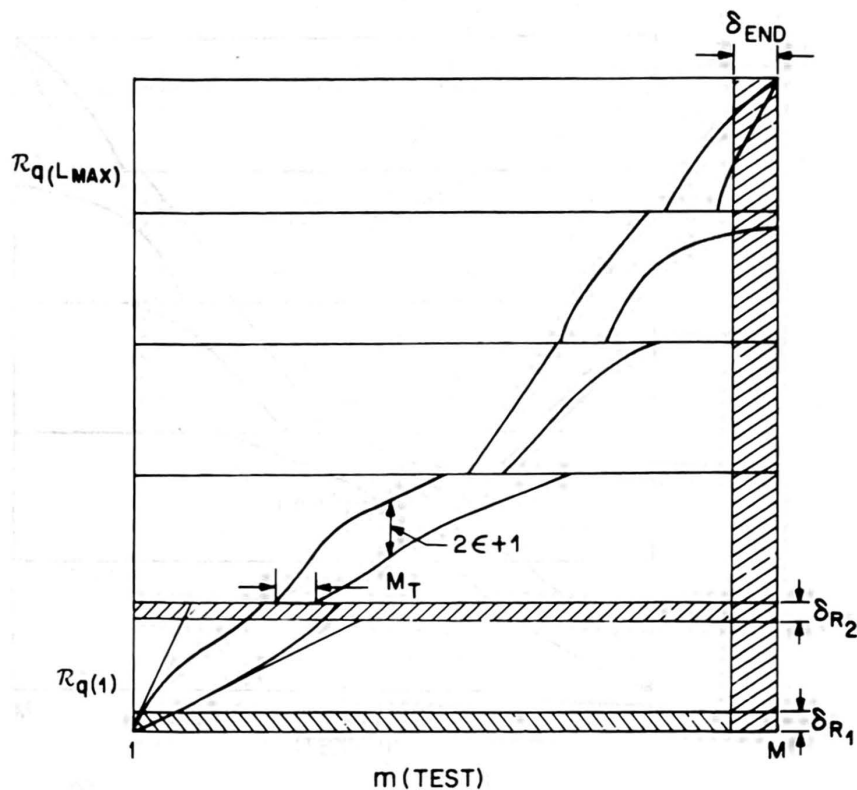


Figure 7.17 Summary of level building implementation of computation reduction methods (after Myers and Rabiner [4]).

level building search. It can be shown that with judicious choice of the implementation parameters, M_T , ϵ , δ_{END} , δ_{R1} and δ_{R2} , the overall computation can be substantially reduced from that of the standard level building approach.

7.4.5 Integration of a Grammar Network

We have been implicitly assuming that, for connected word recognition, each word in the string can be followed by any other word in the string. This implicit form of grammar is most appropriate for things like digit strings in which any digit can follow any other digit. However, for some connected word recognition tasks there is an explicit set of rules (grammar) governing which words can logically follow other words to form valid sentences in the language [10]. Although the form of such a grammar can be of several different types (we will discuss this in more detail in Chapter 8), we will restrict ourselves here to those tasks in which we can represent the grammar by a finite state network (FSN) or a finite state automata (FSA) of the form

$$G = A(Q, V, \delta, q_0, Z) \quad (7.35)$$

where

Q = set of states

V = set of vocabulary words

δ = set of transitions

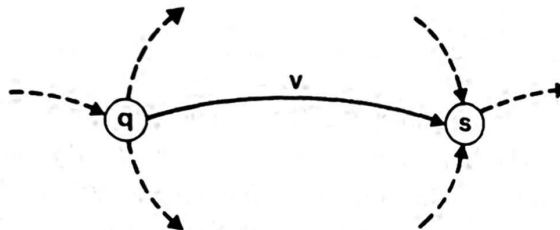
$q_0 \in Q$ = initial state

$Z \subseteq Q$ = set of terminal states

and the set of transitions obeys the rule

$$\delta(q, v) = s \quad (7.36)$$

meaning that word v drives the state from q to s



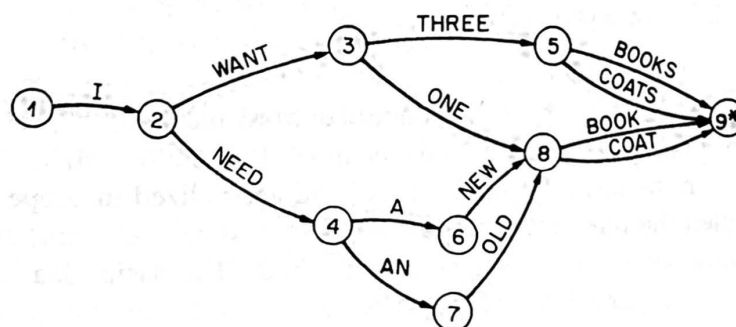
To integrate the FSN grammar network into the following level building algorithm we must do the following:

1. Identify levels with states rather than word position so that word candidates at the ℓ^{th} level need not be temporally contiguous to those at the $(\ell + 1)^{\text{st}}$ level
2. Partition the vocabulary so that only the reference patterns for words leaving the ℓ^{th} state are matched at the ℓ^{th} level
3. Retain state backtracking pointers for recovering the best matching string.

(It should be noted that for the most efficient computation, the states in Q should be topologically sorted.)

To illustrate how a simple grammar FSN can be integrated into the LB algorithm, consider the following example:

1. I	5. ONE	9. BOOKS	13. OLD
2. WANT	6. A	10. COAT	
3. NEED	7. AN	11. COATS	
4. THREE	8. BOOK	12. NEW	



Current State	Words Used	Predecessor State	Current Level	Predecessor Levels
2	I	1	1	0
3	WANT	2	2	1
4	NEED	2	3	1
5	THREE	3	4	2
6	A	4	5	3
7	AN	4	6	3
8	ONE	3	7	2
8	NEW	6	8	5
8	OLD	7	9	6
9*	BOOK, COAT	8	10	7,8,9
9*	BOOKS, COATS	5	11	4

If we study this simple example, we see that levels 2 and 3 both pick up from level 1; similarly both levels 4 and 7 pick up from level 2. By building up the computation by levels (keeping track of the correct predecessor level), and by backtracking the final result by states (according to the grammar FSN) we can essentially use all the techniques described to efficiently find the best grammatically correct string.

7.4.6 Examples of LB Computation of Digit Strings

Figures 7.18 and 7.19 show two examples of connected digit strings matched using the LB algorithm. Figure 7.18 is for the string “51560” and shows the computation building up level by level. For this example, the locally best path at each level (shown by the digit to the right of the last test frame) is actually the globally best digit. At the end of level four the algorithm provided the four best choices with the string “5157” having the lowest score of 0.553. At the end of level 5 there were six string choices with the correct string “51560” having the lowest average accumulated distance of 0.333.

The example in Figure 7.19 is for the string “99211,” which did not provide a match nearly as good as the previous example. We see here that the locally best string at each level, namely “90111” is not globally best, and actually is incorrect in two positions. Although the algorithm gets the correct string as the best score, the second best string is the string “901,” which has two digit deletions, and one digit-substitution error.

7.5 THE ONE-PASS (ONE-STATE) ALGORITHM

The third general approach to the connected word recognition problem is a procedure which was originally proposed by Vintsyuk in 1971 [5] and which has been “rediscovered” several times in the last two decades [6–8] and generalized in scope [9]. The algorithm has been called the one-pass procedure or the one-state algorithm, or most recently, the frame-synchronous level building (FSLB) method. The basic idea behind the algorithm is illustrated in Figure 7.20, which shows a grid with the test pattern, $\mathcal{T}$, mapped to

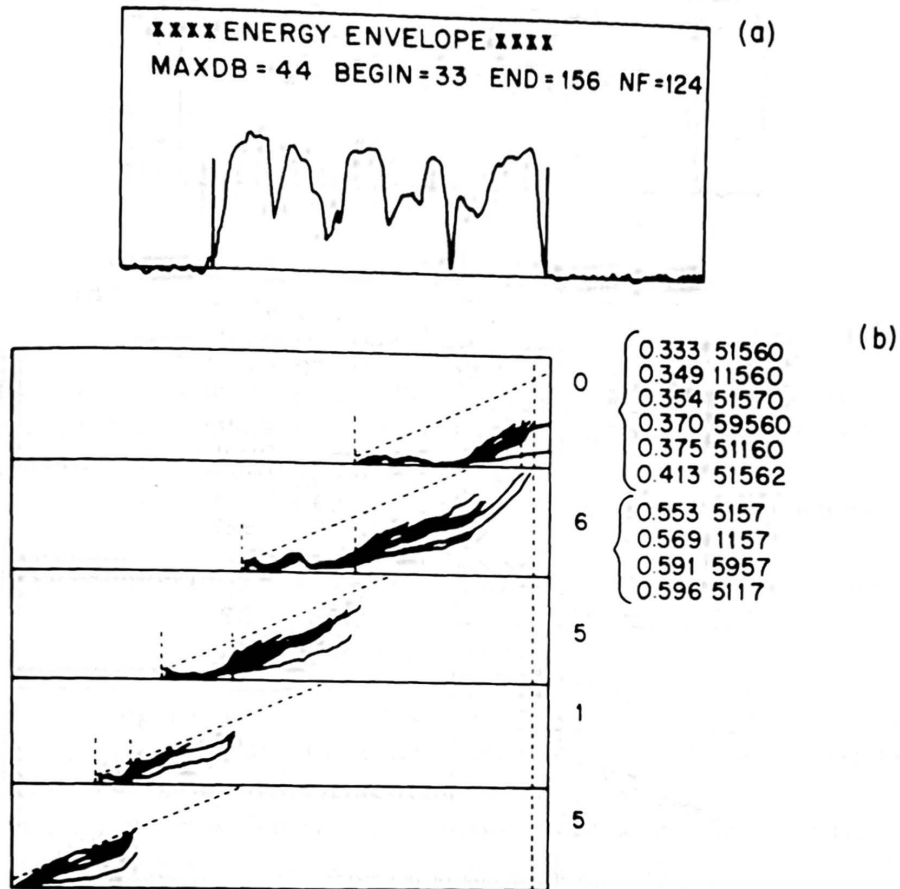


Figure 7.18 Level building of the string "51560" (after Myers and Rabiner [4]).

the horizontal axis, and the set of reference patterns, $\{\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_V\}$ mapped to the vertical axis.

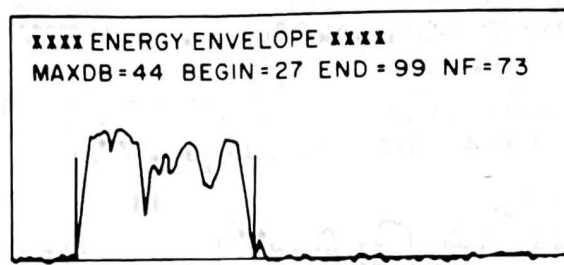
Using the standard notation of m to represent the test frame index, $1 \leq m \leq M$, v to represent the reference pattern ($\mathcal{R}_v$) index, $1 \leq v \leq V$, and n to represent the reference frame index of pattern $\mathcal{R}_v$, $1 \leq n \leq N_v$, then for each test frame we calculate the accumulated distance, $d_A(m, n, v)$ as:

$$d_A(m, n, v) = d(m, n, v) + \min_{n-2 \leq j \leq n} (d_A(m-1, j, v)). \quad (7.37)$$

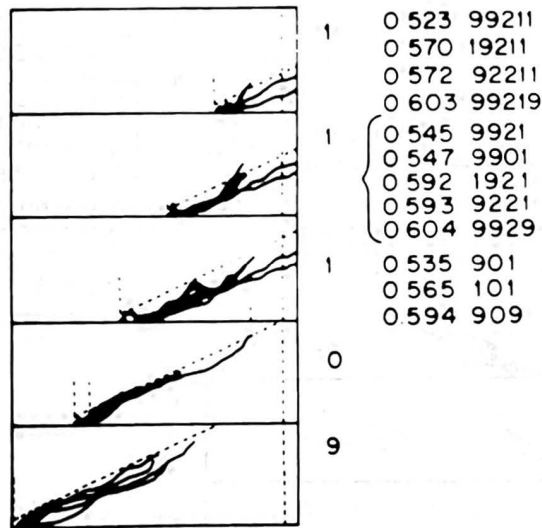
For $2 \leq n \leq N_v$, $1 \leq v \leq V$, where $d(m, n, v)$ is the local distance between test frame $t(m)$ and reference from $r_v(n)$, and we assume a maximum path expansion of 2 to 1 (hence we search back by 2 reference pattern frames in the combinatoric stage). The recursion of Eq. (7.37) is carried out for all internal frames of each reference pattern (i.e., $n \geq 2$). At the reference pattern boundary, i.e., $n = 1$, we have the simple recursion

$$d_A(m, 1, v) = d(m, 1, v) + \min \left[\min_{1 \leq r \leq V} [d_A(m-1, N_r, r)], d_A(m-1, 1, v) \right]. \quad (7.38)$$

Thus the combinatorics for internal and boundary frames are as shown in Figure 7.21. Figure 7.21a shows that for internal frames the combinatorics choose the best internal path



(a)



(b)

Figure 7.19 Level building of the string "99211" (after Myers and Rabiner [4]).

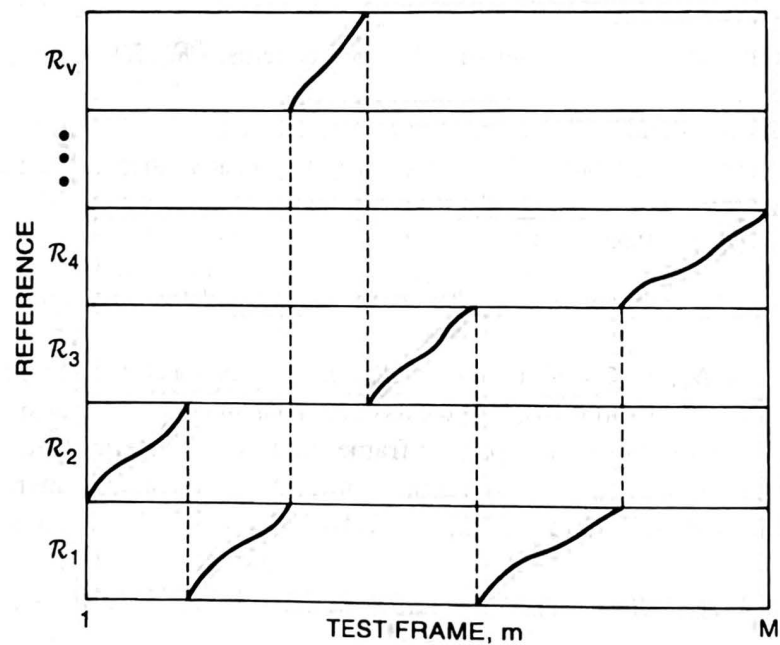


Figure 7.20 The one-pass connected word recognition algorithm (after Bridle et al. [6]).

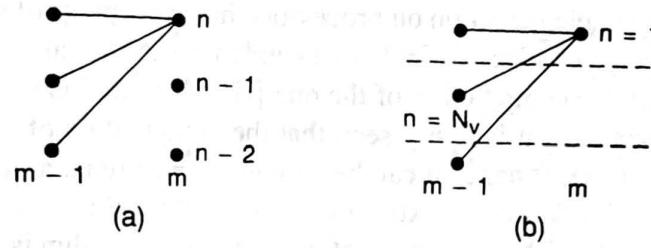


Figure 7.21 Combinatorics for the one-pass algorithm.

within the reference pattern, whereas at boundary frames the combinatorics choose either a straight (horizontal) path from within the reference pattern (subject to the constraint that the path cannot remain constant for more than one frame) or the best ending frame of any reference pattern. (Within the dynamic programming framework, of course, incorporation of a set of *local constraints* that differ from those of Eq. (7.38) is possible, subject to pragmatic considerations.)

The final solution for the best path (corresponding to the best word string) is

$$D^* = \min_{1 \leq v \leq V} [d_A(M, N_v, v)]. \quad (7.39)$$

Thus the one-pass algorithm computes best paths to every reference pattern frame at every test frame and eventually is able to backtrack the best score (from Eq. (7.39)) to give the best word sequence, as shown in Figure 7.20.

The major problem with the one-pass algorithm is that no mechanism is provided for controlling the resulting string length—that is, for giving a best path for a string of arbitrary length. The algorithm inherently finds a single best path whose string length is whatever it turns out to be. Thus there is no simple way of exploiting given constraints on string length within the fundamental procedure.

There is, however, a simple and straightforward way of incorporating level (i.e., string-length) constraint in the computation. We do this by extending the accumulated distance to include level information—that is, we extend the recursion to compute the accumulated distance at level ℓ as

$$d_A^\ell(m, n, v) = d(m, n, v) + \min_{n-2 \leq j \leq n} [d_A^\ell(m-1, j, v)] \quad (7.40)$$

where the computation is for $1 \leq \ell \leq L_{\max}$, $2 \leq n \leq N_v$, $1 \leq v \leq V$, $1 \leq m \leq M$. At each boundary frame the computation now becomes

$$d_A^\ell(m, 1, v) = d(m, 1, v) + \min \left[\min_{1 \leq r \leq V} d_A^{\ell-1}(m-1, N_v, r), d_A^\ell(m-1, 1, v) \right] \quad (7.41)$$

with

$$D^* = \min_{1 \leq \ell \leq L_{\max}} \min_{1 \leq v \leq V} [d_A^\ell(M, N_v, v)]. \quad (7.42)$$

The key difference is in Eq. (7.41), which only allows a path to an ending frame at level $(\ell-1)$ to become a path at a beginning frame at level ℓ .

The main advantage of the one-pass algorithm is that the computation for a given test frame, m , can be done frame synchronously; hence the one-pass algorithm is well

suited to real-time implementation on processors that are capable of doing all the necessary computations of Eqs. (7.40) and (7.41) in a single frame interval. Although at first glance it seems as though the computation of the one-pass algorithm is significantly greater than that of the LB approach, it is easily seen that the computation of $d(m, n, v)$ of Eq. (7.40) is independent of level, ℓ ; hence it can be computed once (e.g., at level 1) and stored, and used in subsequent levels with no extra computation. Because of its suitability for real-time implementation the level-based version of the one-pass algorithm is generally the one used for connected word recognition tasks.

7.6 MULTIPLE CANDIDATE STRINGS

In the previous sections we have been discussing ways of determining the globally best path through the extended grid, corresponding to the best match to the spoken word string. There are many ways of obtaining multiple candidate strings from which we can determine, at least in theory, the second-best string match, the third-best string match, etc. Multiple string candidates are particularly useful when using grammar networks in order to provide robust recognition decisions. The way in which we obtain multiple candidate strings is simple; namely, we keep track of both the *best* distance, and the *second-best* distance to each ending frame at each level (to get the second-best string match). Since we have computed all reference pattern distances already (as needed to determine the best distance), all that is required is additional storage (to keep track of the array of second-best distances) and the bookkeeping to determine the second best string. (In Chapter 4 we discussed the general problem of using dynamic programming methods to determine the N -best paths through a grid. What is described here is essentially what was called the parallel algorithm in Chapter 4, Section 4.7.5. The main difference is that here the path ranking is in terms of word candidate strings instead of frame-based time warping paths. This difference gives rise to some extra optimality considerations as discussed in this section.) We illustrate the procedure in Figure 7.22, which shows a simple two-level LB search with tracking of the two best strings at each grid point. At the end of level 2 there are two distinct paths to the last frame of the test pattern—namely, the best distance (labeled 1 and shown as a solid path) and the second best distance (labeled 2 and shown as a dashed line). The best path from level 2 branches off to two other paths in level 1, with one being a best path and the other being a second-best path. Similarly, the second-best path from level 2 branches off to two other paths in level 1. Thus a total of four paths are followed—namely, the 11 path (best from level 2, best from level 1), the 12 path (best from level 2, second best from level 1), the 21 path (second best from level 2, best from level 1 to the beginning point of the level 2 path) and the 22 path (second best from level 2, second best from level 1). The overall best path is, by definition, the 11 path through the grid. The second-best path, according to this method, is (with some pathological exceptions, to be explained next) either the 12 path, or the 21 path; the 22 path cannot ever be as good as the 21 path.

The procedure described above can be extended trivially to the best three candidates at each level, in which case a total of 3^L scores are obtained after L levels. This is illustrated at the bottom of Figure 7.22 for a two-level match in which there are nine string candidates.

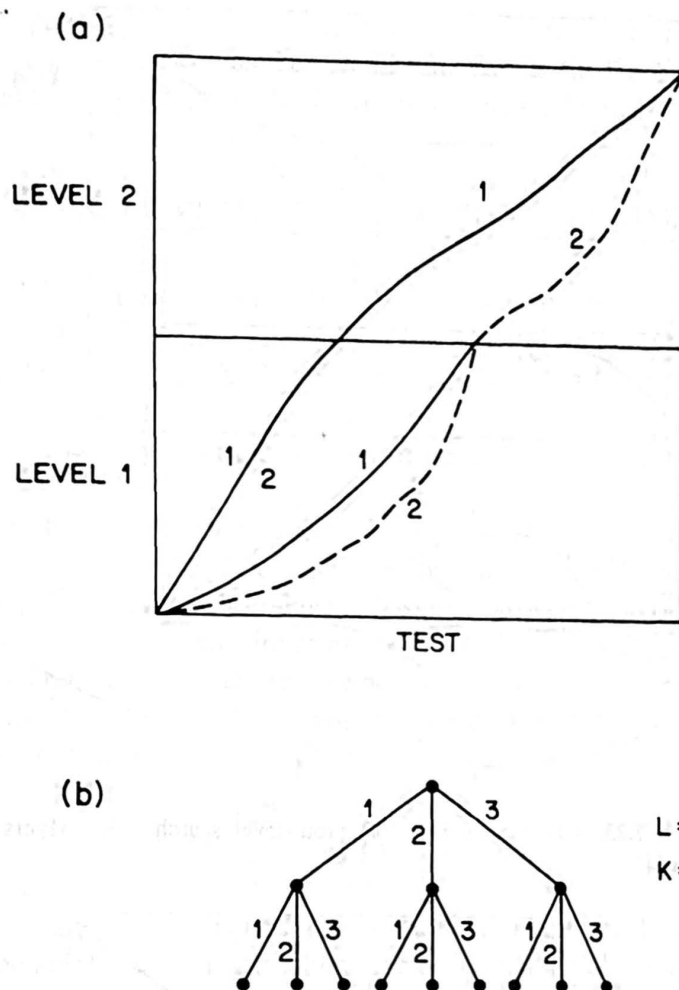


Figure 7.22 Description of procedure for determining multiple candidate scores (after Myers and Rabiner [4]).

Figure 7.23 shows a case of using $L = 4$ levels with two best candidates. There are now four possible choices for the second-best string, including the 1112 path, the 1121 path, the 1211 path and the 2111 path. A sorting algorithm can be used to order the K^L paths obtained when keeping track of K candidates at L levels.

It should be noted that the procedure described above, implemented using the level building algorithm, *cannot* guarantee that the computed "second-best" string is actually the true second-best string. This is because in keeping track of multiple strings to any ending frame the procedure inherently requires that the candidate strings come from different reference patterns. (Recall that in the LB algorithm, partial word decisions are made at each level before reaching the end of the test pattern.) The real requirement should be that they come from different overall strings (i.e., any word in the built-up strings can be different, not just the immediate last word). Hence, in theory, the two best paths to a given ending frame could indeed be from the same reference pattern. Figure 7.24 [10] illustrates the level building flaw via an example in which the true second-best string is the sequence

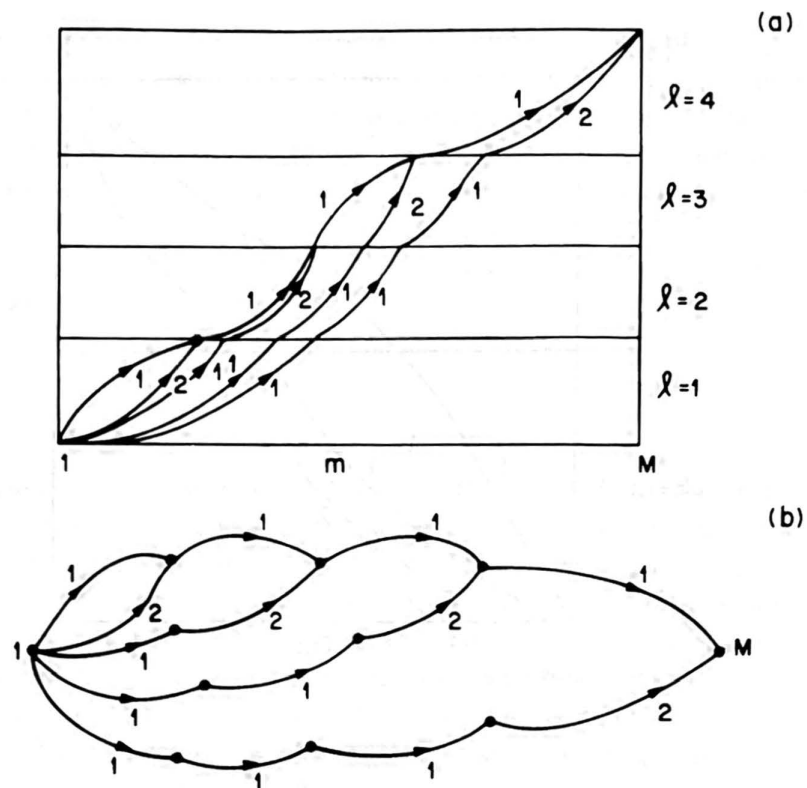


Figure 7.23 Candidate strings for a four-level search (after Myers and Rabiner [4]).

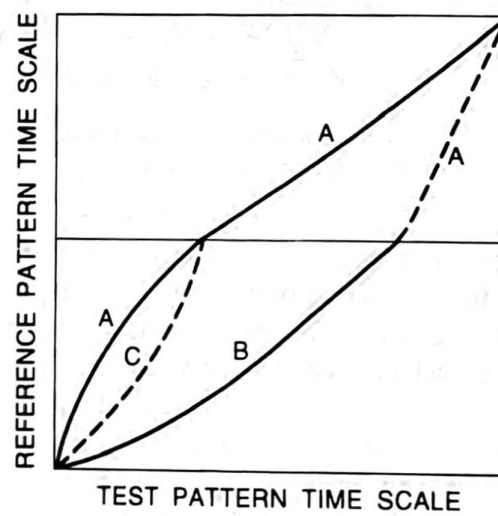


Figure 7.24 Illustration of level building flow for determining the second-best candidate string (after Myers and Rabiner [10]).

BA (i.e., reference pattern B followed by A); however, since the best string match is AA , the reference pattern A is not allowed as the second candidate to the last frame at level 2. This situation occurs extremely infrequently; however, it can and does occur from time to time.

7.7 SUMMARY OF CONNECTED WORD RECOGNITION ALGORITHMS

We have presented three approaches to solving the “connected word recognition” problem—namely, the two-level DP algorithm, the LB method, and the (frame synchronous) one-pass method. The algorithms all are fundamentally identical in that they provide the identical best matching string with the identical matching score. Basically the algorithms differ in computational efficiency, storage requirements, and ease of realization in real-time hardware.

Although we have concentrated primarily on word template patterns, it is easy to see that the basic procedures are virtually identical for statistical models like HMMs. To see this, consider the case of level building using N -state HMMs instead of N_v -frame reference patterns. If we denote the test frame index as t , $1 \leq t \leq T$, (rather than n as is conventionally done for HMMs), and we denote the test frame vector as $\mathbf{o}_t$ (rather than $\mathbf{t}_n$ as we have done throughout this chapter), then the local log likelihood for state j of reference model λ^v is (for an M -mixture density)

$$b_j^v(\mathbf{o}_t) = \log \left[\sum_{m=1}^M c_m \prod_{d=1}^D e^{-(\mathbf{o}_t(d) - \mu_{jm}(d))^2 / 2 U_{jm}(d)} \right]. \quad (7.43)$$

The level building computation is therefore a calculation of $P_\ell^v(t)$, $1 \leq t \leq T$, $1 \leq v \leq V$, $1 \leq \ell \leq L_{\max}$, the accumulated log likelihood to frame t , at level ℓ , for reference model λ^v along the best path along with $F_\ell^v(t)$, the backpointer indicating where the best path started at the beginning of the level. At the end of each level we compute the “level best” scores as

$$P_\ell^B(t) = \max_{1 \leq v \leq V} P_\ell^v(t), \quad 1 \leq t \leq T \quad (7.44)$$

$$N_\ell^B(t) = \arg \max_{1 \leq v \leq V} P_\ell^v(t), \quad 1 \leq t \leq T \quad (7.45)$$

$$F_\ell^B(t) = F_\ell^{N_\ell^B(t)}(t), \quad 1 \leq t \leq T \quad (7.46)$$

with solution

$$P^* = \max_{1 \leq \ell \leq L_{\max}} [P_\ell^B(T)]. \quad (7.47)$$

Figure 7.25 illustrates the grid for the level building computation for a set of N -state HMMs. The regularity of the grid (the trellis shape) is due to the lack of constraint about remaining in a state for more than one or two frames for statistical models.

Finally it should be clear that all three algorithms are equally amenable to inclusion (integration) of an FSN grammar network to constrain allowable word sequences. Consider

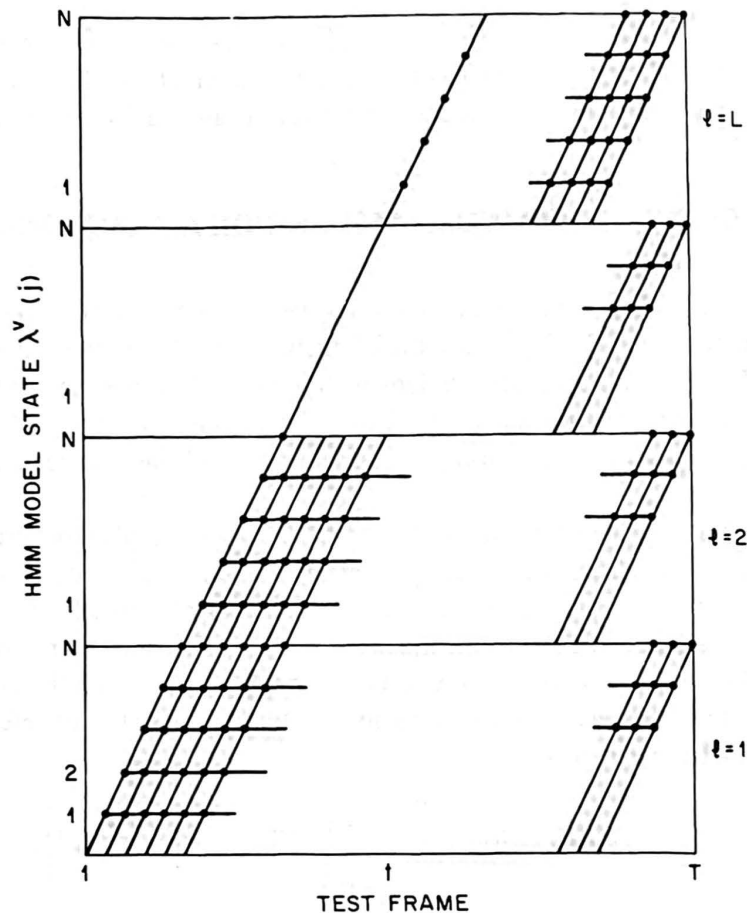


Figure 7.25 Use of HMMs in the level building procedure.

an arbitrary FSN grammar network with a typical node, g , as shown in Figure 7.26. The input to the grammar node is a series of word arcs (corresponding to words in the vocabulary) from predecessor grammar nodes $i-1$, i and $i+1$. Since all these inputs merge at grammar node g , the basic computation at this grammar node is to determine the maximum likelihood path over the set, $P(g)$, (which constitutes the paths of all words coming into g), and to propagate this path to all successor nodes, namely $j-1$, j , $j+1$ through the appropriate word arcs. The grammar node computation is iterated over all nodes in the grammar in an organized fashion (so that all computation necessary for node g is done before considering node g).

The way in which computation for the grammar FSN is integrated into the connected word algorithm is shown in Figure 7.27. For each test frame, t , the spectral vector is computed, and then the local likelihood (or distance) scores are computed for every reference pattern state (or frame). In parallel the local combinatorics (within reference patterns) are performed (the local distance scores are added at the end of the computation). The grammar network scores (corresponding to word transitions) are then computed and the procedure is iterated until the last test frame, at which point a backtracker is used to determine the best matching string.

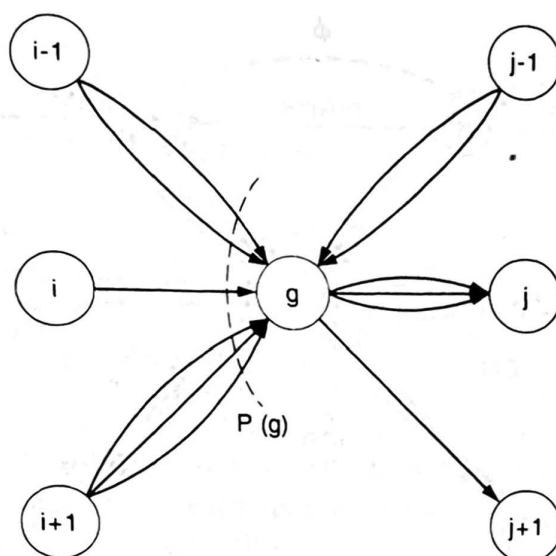


Figure 7.26 A typical grammar node of an FSN grammar network (after Lee and Rabiner [9]).

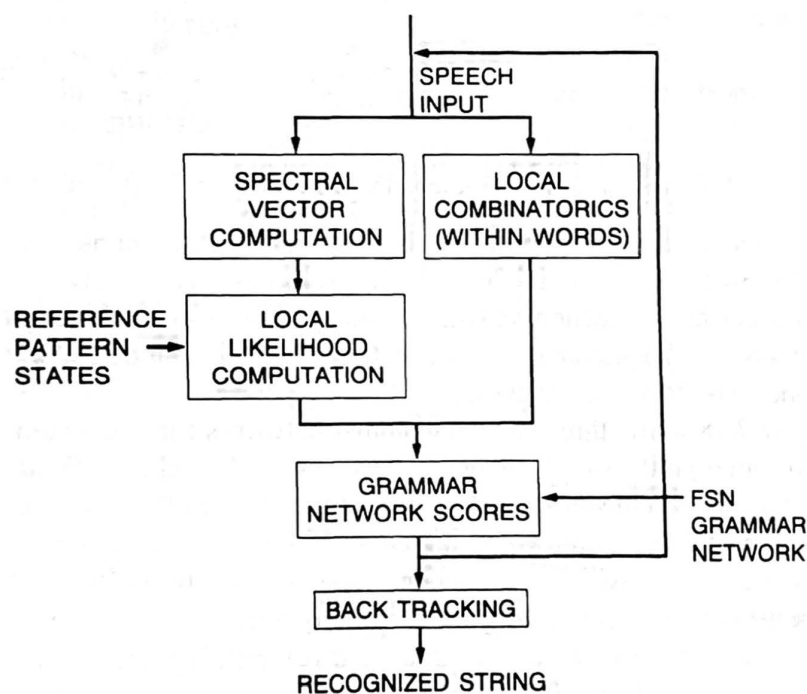


Figure 7.27 Block diagram of connected word recognition computation.

7.8 GRAMMAR NETWORKS FOR CONNECTED DIGIT RECOGNITION

One of the most important applications of connected word recognition is connected digit recognition because of its potential application to credit card entry, all-digit dialing of telephone numbers, personal identification number (PIN) entry, catalog ordering, and so on.

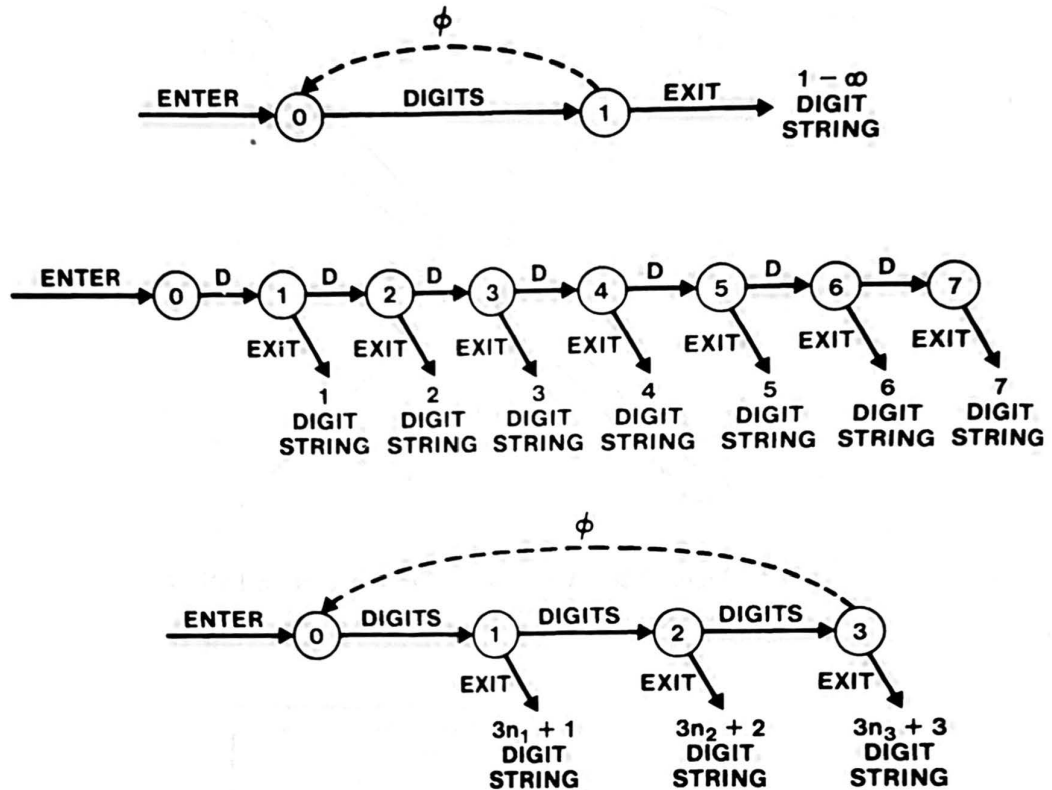


Figure 7.28 Three possible grammar networks for connected digit recognition (after Lee and Rabiner [9]).

Thus in this and the next section we will concentrate on aspects of this important application, including forms of the grammar network that are often used, and some implementation and performance aspects of current systems.

Figure 7.28 shows three general grammar networks for connected digit strings. The network of part (a) is the simple search of the one-pass algorithm *without* level information. Thus the network will always find the best string, but cannot control the string length, and therefore is unusable for known string length tasks (e.g., choosing the best seven-digit string to represent a spoken telephone number).

The network of Figure 7.28, part b explicitly breaks out digit strings of length one to seven digits, and therefore corresponds to the level building approach with $L_{\max} = 7$. As we have seen, the local combinatorics search requires about 7 times the computation of the simple grammar network of part a; however, local distance computation remains the same.

The network of Figure 7.28, part c represents a reasonable compromise between the networks of parts a and b in that it breaks out three distinct levels corresponding to string lengths of $3n_1 + 1$, $3n_2 + 2$, and $3n_3 + 3$ digits where n_1 , n_2 and n_3 are arbitrary integers. The idea here is that if the input is an n -digit string, the most likely errors are single-digit insertions or deletions, at which point one of the three outputs will most likely not have the single insertion or deletion, and will be the correct string. The computation is only three times the combinatorics search of part a, with again the same computation

for local distances.

7.9 SEGMENTAL K-MEANS TRAINING PROCEDURE

The discussion in the previous sections focused primarily on efficient and optimal solutions to the so-called decoding problem, in which the likelihood (or distance) of a given speech pattern (the unknown test pattern) corresponding to a *string of words* is evaluated. Of equal importance is the connected word training problem in which the object is to derive appropriate word reference patterns or models from a labeled training set of many connected word sequences. The major problem here is the lack of an exact (precise) correspondence between speech segments and the spoken words they correspond to. One could consider manually segmenting and labeling each spoken training utterance into the individually spoken words of each string. However, this process is a tedious one that is well known to be error prone because of inconsistencies in determination of the exact boundaries between adjacent words in the string. Hence what is required for connected word training is a fully automatic procedure for both segmentation of a connected word string into individual words, and model (pattern) training from the segmented strings. Such an algorithm is described in this section.

The way in which the individual digit models are derived from connected word training strings is a procedure called the segmental *k*-mean training procedure, which is a straightforward variation on the well-known *k*-means iteration (e.g., as used for vector quantizer design) [11–13]. The basic idea is to have a training set of labeled connected digit strings, and an initial set of digit models (e.g., isolated digit models). (The procedure to be described works even without an initial set of digit models, but we will not describe how this is accomplished here.)

The segmental *k*-means training procedure (as shown in Figure 7.29) works as follows (when used for training HMMs):

1. Given the initial model (the set of word pattern files) and the (labeled) training files, any of the connected word recognition procedures is used to segment each training string into individual digit tokens which are stored in appropriate files according to the identity of the digits. (This is the segmentation phase into words.)
2. Each file of word tokens (e.g., the file for all the 1s in the training set) is then segmented into states and within each state the parameters of the mixture density (the mixture weights, means, and covariances) are determined using a standard VQ clustering procedure. The result of this procedure (which we call the word pattern building algorithm) is an updated set of word models.
3. A test for convergence is made, either based on a set of testing files or based on the likelihood scores of the training set files. If the convergence test shows continuing improvement in performance, the procedure is iterated (i.e., steps 1 and 2 are repeated using the updated set of word models); otherwise the procedure is terminated and the updated set of models is the final set of models.

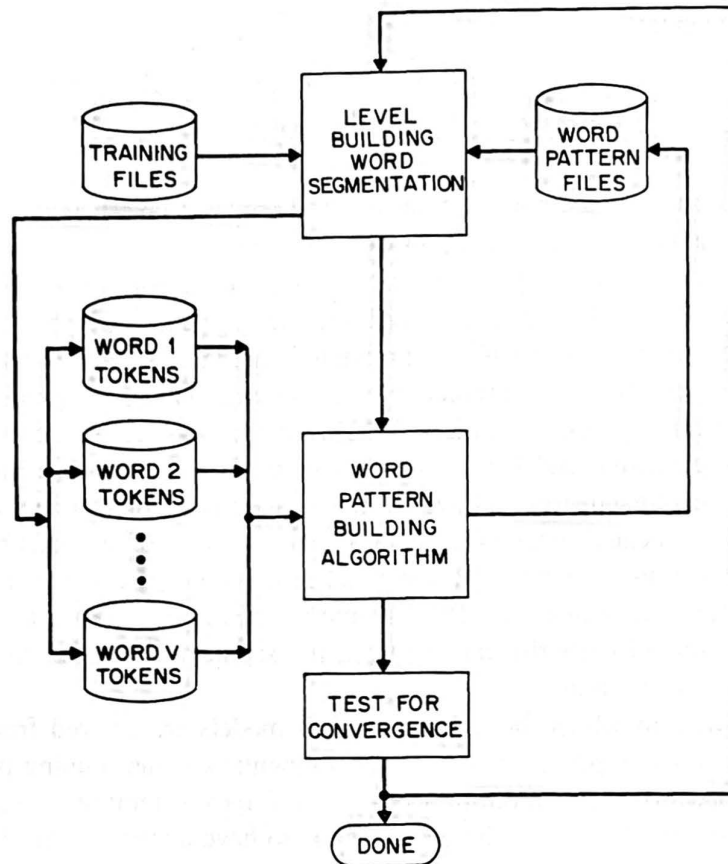


Figure 7.29 The segmental *k*-means training algorithm for connected word strings (after Rabiner et al. [13]).

7.10 CONNECTED DIGIT RECOGNITION IMPLEMENTATION

A block diagram of a canonic system for connected digit recognition is shown in Figure 7.30. There are three basic steps in the recognition process, namely:

1. **Spectral analysis**, in which the speech signal, $s(n)$, is converted to an appropriate spectral representation, e.g., filter-bank vector, LPC-based vector, ear model vector.
2. **Connected word pattern matching**, in which the sequence of spectral vectors of the unknown (test) connected digit string is matched against whole word (single digit) patterns using any of the algorithms discussed in this chapter. The output of this process is a set of candidate digit strings, generally of different lengths, ordered by distance (likelihood, probability) score.
3. **Postprocessing**, in which the candidate digit strings are subjected to further processing (e.g., based on digit durations, word stress, etc.) so as to eliminate unreasonable (unlikely) candidates. The postprocessor chooses the most likely digit string from the ordered list of candidates which passed the postprocessor tests.

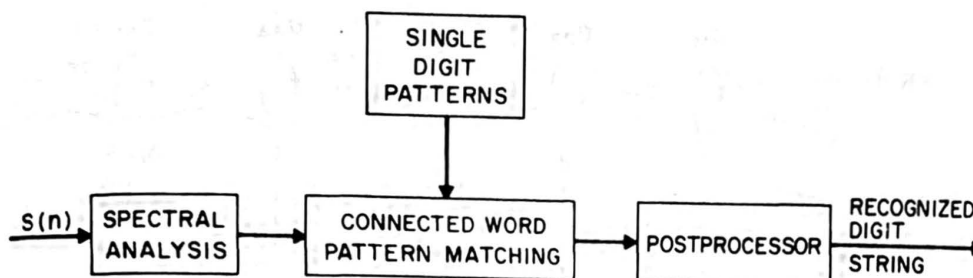


Figure 7.30 Block diagram of connected digit recognition method (after Rabiner et al. [13]).

In the remainder of this section we present an overview of the techniques that are currently used to provide the best performance (highest string accuracy) on this task.

7.10.1 HMM-Based System for Connected Digit Recognition

The system that provides the highest reported string accuracy on a standard testing set is one based on LPC cepstral analysis and HMMs. In particular, the spectral analysis uses an LPC front end with the following characteristics:

- sampling rate—6.67 kHz
- analysis window size—300 samples (45 msec)
- analysis window shift—100 samples (15 msec)
- LPC order—8
- cepstrum order—12
- delta cepstrum order—12
- delta-delta (second difference) cepstrum order—12
- cepstral window—raised, sinelike window

The observation vector used was a 38-component vector consisting of 12 cepstral coefficients, 12 delta cepstrum coefficients, 12 delta-delta cepstral coefficients, delta log energy, and delta-delta log energy. (The log energy was used directly in the evaluation of model likelihoods on the basis of measured histograms, rather than as a component of the observation vector.)

The hidden Markov models used for each digit model were left-to-right models of the type shown in Figure 7.31. Each model had N states (N varied from 5 to 10 for different digits), and within states a continuous mixture density was used to characterize the observation vector, where the number of mixture components per state was as few as 3 (for speaker-trained models) and as many as 64 (for speaker independent models). In addition to the spectral density, an empirically derived log energy probability density (e.g., a histogram) was used within each state (with appropriate weighting), as well as an empirically derived state duration density. For the postprocessor a single Gaussian digit duration density was used based on the measured mean duration and variance from the training set.

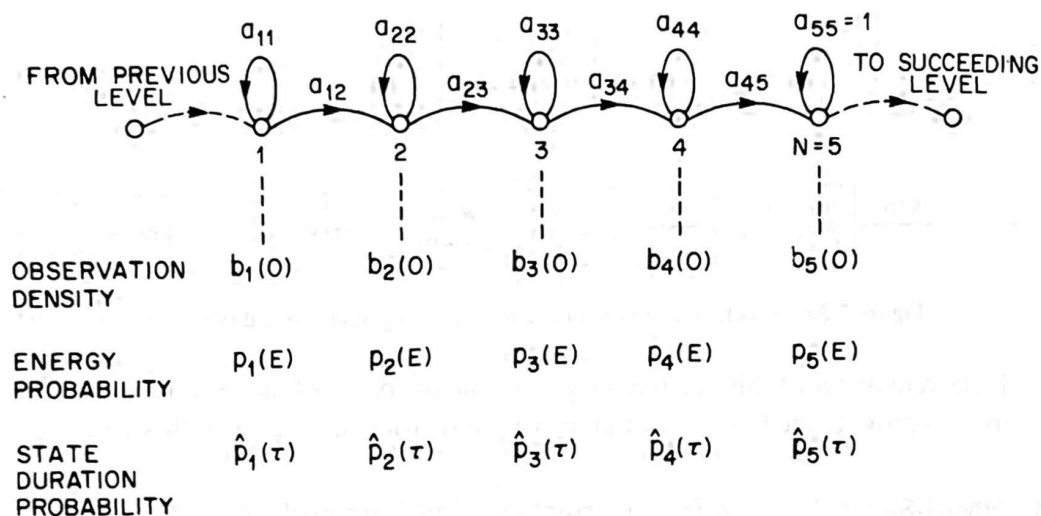


Figure 7.31 Connected digit HMM (after Rabiner et al. [13]).

7.10.2 Performance Evaluation on Connected Digit Strings

To evaluate the performance of connected digit recognition algorithms ([13–15]), two databases have been used:

1. **DB50**, consisting of 50 adult talkers (25 male, 25 female), from the local, nontechnical population of Murray Hill, New Jersey. Each talker provided from 600 to 1150 digit strings, for a total of 47,336 strings. The digit strings were recorded off of local, dialed-up, telephone lines (100–3200 Hz bandwidth), and were variable in length from 1 to 7 digits. Within each string the selection of digits was random from the set of zero to nine (oh was excluded). All digit strings were spoken fluently with no pauses.
2. **TI Set**, consisting of a training set of 112 talkers (55 male, 57 female) with 22 regional accents, and a testing set of 113 additional talkers (56 male, 57 female) from the same 22 regions with no overlap between training and testing set talkers. Each talker spoke 77 connected digit strings of length 1–5 or 7 digits. The input provided was wideband (100–7000 Hz) but was lowpass filtered to telephone bandwidth at Bell Labs for compatibility with the DB50 set. There was a random selection of digits within each string from the set of 11 digits including both zero and oh. Even though talkers were requested to speak each string fluently with no pauses, a few strings had internal pauses.

The first data base, DB50, was used for tests in a speaker-trained and a multispeaker mode, whereas the second database, TI Set, was used for speaker-independent tests. To contrast the spoken material in the two sets, Figure 7.32 shows a plot of the average speaking rate (measured in words per minute) of the two datasets as a function of the number of digits in the string. It can be seen that the speaking rate of the TI talkers is somewhat lower than that of the DB50 talkers (by from 10–20 wpm).

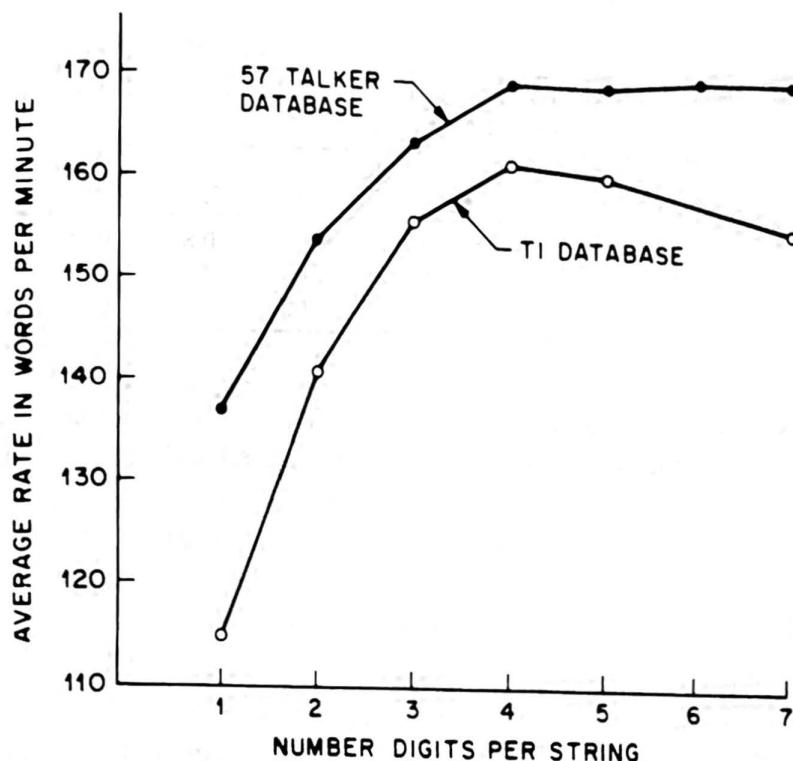


Figure 7.32 Average speaking rate of talkers in the two connected digit databases as a function of the number digits per string (after Rabiner et al. [13]).

The conditions used in the performance evaluations were as follows (when used for training HMMs):

1. **Speaker-Trained Mode (50 Talkers)**—For each talker, half the strings (randomly selected) were used for training, the other half for testing. A single HMM per digit was used. (For this test the observation vector did not use either energy or delta-delta cepstral components.)
2. **Multispeaker Mode (50 Talkers)**—One-fourth of the training set, as in the speaker-trained mode, was used for training 6 models per digit (using clustering techniques); the same testing set was used as in the speaker trained mode. (Again the delta-delta cepstral components or energy were not used in the observation vectors.)
3. **Speaker-Independent Mode (225 Talkers)**—The specified training and testing sets were used to create a single HMM per digit (based on the 38 parameter observation vectors). No silence model was created to account for pauses within digit strings.

The resulting performance of the connected digit recognizer is shown in Table 7.1. For each of the three testing conditions, results are given for self-test (namely, testing on the training data) and for the true testing set, in terms of average *string* error rates (%) for unknown length (UL) strings (i.e., allowing insertions and deletions as well as substitutions), as well as for known length (KL) strings (i.e., only allowing strings of the correct length). Performance in all three modes is comparable with string error rates around 1–3% for unknown length strings, and 0.4–1.7% for known length strings. (Performance

TABLE 7.1. Average String Error Rates (%) for Connected Digit Recognition Tests

Mode	Training Data		Testing Data	
	UL	KL	UL	KL
Speaker Trained	0.4	0.16	0.8	0.35
Multi-Speaker	1.7	1.0	2.85	1.65
Speaker Independent	0.3	0.05	1.4	0.8

scores for speaker-trained and multispeaker modes are underbounds, since the tests were not performed with the extended observation vector used in the speaker-independent mode tests.)

7.11 SUMMARY

In this chapter we have shown how the results presented in earlier chapters on time alignment of single words and phrases can be extended to the problem of aligning an input consisting of a sequence of words to a concatenation of individual word patterns. We have shown how an optimal matching can be achieved using one of several different procedures, and we have also shown how the basic ideas can be applied to either templates or statistical models. By way of example, we showed how the techniques described have been applied to the problem of recognizing a fluently spoken string of digits with high performance.

REFERENCES

- [1] H. Sakoe, "Two-Level DP Matching—A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-27 (6): 588–595, December 1979.
- [2] H. Sakoe, "A Generalized Two-Level DP-Matching Algorithm for Continuous Speech Recognition," *Trans. of the IEE of Japan*, E65 (11): 649–656, November 1982.
- [3] C.S. Myers and L.R. Rabiner, "A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-29 (2): 284–297, April 1981.
- [4] C.S. Myers and L.R. Rabiner, "Connected Digit Recognition Using a Level-Building DTW Algorithm," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-29 (3): 351–363, June 1981.
- [5] T.K. Vintsyuk, "Element-Wise Recognition of Continuous Speech Consisting of Words From a Specified Vocabulary," *Kibernetika (Cybernetics)*, No. 2: 133–143, March–April 1971.
- [6] J.S. Bridle, M.D. Brown, and R.M. Chamberlain, "An Algorithm for Connected Word Recognition," *Proc. ICASSP 82*, Paris, 899–902, May 1982.

- [7] J.S. Bridle, M.D. Brown, and R.M. Chamberlain, "Continuous Connected Word Recognition Using Whole Word Templates," *The Radio and Electronic Engineer*, 53 (4): 167-175, April 1983.
- [8] H. Ney, "The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-32 (2): 263-271, April 1984.
- [9] C.H. Lee and L.R. Rabiner, "A Frame-Synchronous Network Search Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, 37 (11): 1649-1658, November 1989.
- [10] C.S. Myers and L.R. Rabiner, "A Comparative Study of Several Dynamic Time-Warping Algorithms for Connected-Word Recognition," *Bell System Tech. J.*, 60 (7): 1389-1409, September 1981.
- [11] B.H. Juang and L.R. Rabiner, "The Segmental K -Means Algorithm for Estimating Parameters of Hidden Markov Models," *IEEE Trans. Acoustics, Speech, Signal Proc.*, 38 (9): 1639-1641, September 1990.
- [12] L.R. Rabiner, J.G. Wilpon, and B.H. Juang, "A Segmental K -Means Training Procedure for Connected Word Recognition," *AT&T Tech. J.*, 64 (3): 21-40, May 1986.
- [13] L.R. Rabiner, J.G. Wilpon, and B.H. Juang, "A Model-Based Connected Digit Recognition System Using Either Hidden Markov Models or Templates," *Computer Speech, and Language*, 1 (2): 167-197, December 1986.
- [14] L.R. Rabiner, J.G. Wilpon, and F.K. Soong, "High Performance Connected Digit Recognition Using Hidden Markov Models," *IEEE Trans. Acoustics, Speech, Signal Proc.*, 37 (8): 1214-1225, August 1989.
- [15] J.G. Wilpon, C.H. Lee, and L.R. Rabiner, "Improvements in Connected Digit Recognition Using Higher Order Spectral and Energy Features," *Proc. ICASSP 91*, Toronto, Ontario, Canada, May 1991.

Chapter 8

LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION

8.1 INTRODUCTION

Throughout this book we have developed a wide range of tools, techniques, and algorithms for attacking several fundamental problems in speech recognition. In the previous chapter we saw how the different techniques came together to solve the connected word recognition problem. In this chapter we extend the concepts to include issues needed to solve the large vocabulary, continuous speech recognition problem. We will see that the fundamental ideas need modification because of the use of subword speech units; however, a great deal of the formalism for recognition, based on word units, is still preserved.

The standard approach to large vocabulary continuous speech recognition is to assume a simple probabilistic model of speech production whereby a specified word sequence, W , produces an acoustic observation sequence Y , with probability $P(W, Y)$. The goal is then to decode the word string, based on the acoustic observation sequence, so that the decoded string has the maximum a posteriori (MAP) probability, i.e.,

$$\hat{W} \ni P(\hat{W}|Y) = \max_W P(W|Y). \quad (8.1)$$

Using Bayes' Rule, Equation (8.1) can be written as

$$P(W|Y) = \frac{P(Y|W)P(W)}{P(Y)}. \quad (8.2)$$

Since $P(Y)$ is independent of W , the MAP decoding rule of Eq. (8.1) is

$$\hat{W} = \arg \max_W P(Y|W)P(W). \quad (8.3)$$

The first term in Eq. (8.3), $P(Y|W)$, is generally called the acoustic model, as it estimates the probability of a sequence of acoustic observations, conditioned on the word string. The way in which we compute $P(Y|W)$, for large vocabulary speech recognition, is to build statistical models for subword speech units, build up word models from these subword speech unit models (using a lexicon to describe the composition of words), and then postulate word sequences and evaluate the acoustic model probabilities via standard concatenation methods. Such methods are discussed in Sections 8.2–8.4 of this chapter.

The second term in Eq. (8.3), $P(W)$, is generally called the language model, as it describes the probability associated with a postulated sequence of words. Such language models can incorporate both syntactic and semantic constraints of the language and the recognition task. Often, when only syntactic constraints are used, the language model is called a grammar and may be of the form of a formal parser and syntax analyzer, an N -gram word model ($N = 2, 3, \dots$), or a word pair grammar of some type. Generally such language models are represented in a finite state network so as to be integrated into the acoustic model in a straightforward manner. We discuss language models further in Section 8.5 of this chapter.

We begin the chapter with a discussion of subword speech units. We formally define subword units and discuss their relative advantages (and disadvantages) as compared to whole-word models. We next show how we use standard statistical modeling techniques (i.e., hidden Markov models) to model subword units based on either discrete or continuous densities. We then show how such units can be trained automatically from continuous speech, without the need for a bootstrap model of each of the subword units. Next we discuss the problem of creating and implementing word lexicons (dictionaries) for use in both training and recognition phases. To evaluate the ideas discussed in this chapter we use a specified database access task, called the DARPA Resource Management (RM) task, in which there is a word vocabulary of 991 words (plus a silence or background word), and any one of several word grammars can be used. Using such a system, we show how a basic set of subword units performs on this task. Several directions for creating subword units which are more specialized are described, and several of these techniques are evaluated on the RM task. Finally we conclude the chapter with a discussion of how task semantics can be applied to further constrain the recognizer and improve overall performance.

8.2 SUBWORD SPEECH UNITS

We began Chapter 2 with a discussion of the basic phonetic units of language and discussed the acoustic properties of the phonemes in different speech contexts. We then argued that the acoustic variability of the phonemes due to context was sufficiently large and not well understood, that such units would not be useful as the basis for speech models for recognition. Instead, we have used whole-word models as the basic speech unit, both for

isolated word recognition systems and for connected word recognition systems, because whole words have the property that their acoustic representation is well defined, and the acoustic variability occurs mainly in the region of the beginning and the end of the word. Another advantage of using whole-word speech models is that it obviates the need for a word lexicon, thereby making the recognition structure inherently simple.

The disadvantages of using whole-word speech models for continuous speech recognition are twofold. First, to obtain reliable whole-word models, the number of word utterances in the training set needs to be sufficiently large, i.e., each word in the vocabulary should appear in each possible phonetic context several times in the training set. In this way the acoustic variability at the beginning and at the end of each word can be modeled appropriately. For word vocabularies like the digits, we know that each digit can be preceded and followed by every other digit; hence for an 11-digit vocabulary (zero to nine plus oh), there are exactly 121 phonetic contexts (some of which are essentially identical). Thus with a training set of several thousand digit strings, it is both realistic and practical to see every digit in every phonetic context several times. Now consider a vocabulary of 1000 words with an average of 100 phonetic contexts for both the beginning and end of each word. To see each word in each phonetic context exactly once requires $100 \times 1000 \times 100 = 10$ million carefully designed sentences. To see each combination 10 times requires 100 million such sentences. Clearly, the recording and processing of such homogeneous amounts of speech data is both impractical and unthinkable. Second, with a large vocabulary the phonetic content of the individual words will inevitably overlap. Thus storing and comparing whole-word patterns would be unduly redundant because the constituent sounds of individual words are treated independently, regardless of their identifiable similarities. Hence some more efficient speech representation is required for such large vocabulary systems. This is essentially the reason we use subword speech units.

There are several possible choices for subword units that can be used to model speech. These include the following:

- **Phonelike units (PLUs)** in which we use the basic phoneme set (or some appropriately modified set) of sounds but recognize that the acoustic properties of these units could be considerably different than the acoustic properties of the “basic” phonemes [1–7]. This is because we define the units based on linguistic similarity but model the unit based on acoustic similarity. In cases in which the acoustic and phonetic similarities are roughly the same (e.g., stressed vowels) then the phoneme and the PLU will be essentially identical. In other cases there can be large differences and a simple one-to-one correspondence may be inadequate in terms of modeling accuracy. Typically there are about 50 PLUs for English.
- **Syllable-like units** in which we again use the linguistic definition of a syllable (namely a vowel nucleus plus the optional initial and final consonants or consonant clusters) to initially define these units, and then model the unit based on acoustic similarity. In English there are approximately 10,000 syllables.
- **Dyad or demisyllable-like units** consisting of either the initial (optional) consonant cluster and some part of the vowel nucleus, or the remaining part of the vowel nucleus and the final (optional) consonant cluster [8]. For English there is something on the

order of 2000 demisyllable-like units.

- **Acoustic units**, which are defined on the basis of clustering speech segments from a segmentation of fluent, unlabeled speech using a specified objective criterion (e.g., maximum likelihood) [9]. Literally a codebook of speech units is created whose interpretation, in terms of classical linguistic units, is at best vague and at worst totally nonexistent. It has been shown that a set of 256–512 acoustic units is appropriate for modeling a wide range of speech vocabularies.

Consider the English word *segmentation*. Its representation according to each of the above subword unit sets is

- **PLUs**: /s/ /ɛ/ /g/ /m/ /ə/ /n/ /t/ /eʃ/ /ʃ/ /ə/ /n/ (11 units)
- **syllables**: /seg/ /men/ /ta/ /tion/ (4 syllables)
- **demisyllables**: /sɛ/ /ɛg/ /mə/ /ən/ /teʃ/ /eʃ/ /ʃə/ /ən/ (8 demisyllables)
- **acoustic units**: 17 111 37 3 241 121 99 171 37 (9 acoustic units).

We see, from the above example, that the number of subword units for this word can be as small as 4 (from a set of 10,000 units) or as large as 11 (from a set of 50 units).

Since each of the above subword unit sets is capable of representing any word in the English language, the issues in the choice of subword unit sets are the context sensitivity and the ease of training the unit from fluent speech. (In addition, for acoustic units, an issue is the creation of a word lexicon since the units themselves have no inherent linguistic interpretation.) It should be clear that there is no ideal (perfect) set of subword units. The PLU set is extremely context sensitive because each unit is potentially affected by its predecessors (one or more) and its followers. However, there is only a small number of PLUs and they are relatively easy to train. On the other extreme are the syllables which are longest units and are the least context sensitive. However, there are so many of them that they are almost as difficult to train as whole-word models.

For simplicity we will initially assume that we use PLUs as the basic speech units. In particular we use the set of 47 PLUs shown in Table 8.1 (which includes an explicit symbol for silence –h#). For each PLU we show an orthographic symbol (e.g., aa) and a word associated with the symbol (e.g., father). (These symbols are essentially identical to the ARPAPET alphabet of Table 2.1; lowercase symbols are used throughout this chapter for consistency with the DARPA community.) Table 8.2 shows typical pronunciations for several words from the DARPA RM task in terms of the PLUs in Table 8.1. A strong advantage of using PLUs is the ease of creating word lexicons of the type shown in Table 8.2 from standard (electronic) dictionaries. We will see later in this chapter how we exploit the advantages of PLUs, while reducing the context dependencies, by going to more specialized PLUs which take into consideration either the left or right (or both) contexts in which the PLU appears.

One problem with word lexicons of the type shown in Table 8.2 is that they don't easily account for variations in word pronunciation across different dialects and in the context of a sentence. Hence a simple word like "a" is often pronounced as /ey/ in isolation (e.g., the

TABLE 8.1. Set of basic PLUs for speech.

Number	Symbol	Word	Number	Symbol	Word
1	h#	silence	26	k	kick
2	aa	father	27	l	led
3	ae	bat	28	m	mom
4	ah	butt	29	n	no
5	ao	bought	30	ng	sing
6	aw	bough	31	ow	boat
7	ax	again	32	oy	boy
8	axr	diner	33	p	pop
9	ay	bite	34	r	red
10	b	bob	35	s	sis
11	ch	church	36	sh	shoe
12	d	dad	37	t	tot
13	dh	they	38	th	thief
14	eh	bet	49	uh	book
15	el	bottle	40	uw	boot
16	en	button	41	v	very
17	er	bird	42	w	wet
18	ey	bait	43	y	yet
19	f	fief	44	z	zoo
20	g	gag	45	zh	measure
21	hh	hag	46	dx	butter
22	ih	bit	47	nx	center
23	ix	roses			
24	iy	beat			
25	jh	judge			

TABLE 8.2. Typical word pronunciations (word lexicon) based on context-independent PLUs.

Word	Number of phones	Transcription				
a	1	ax				
above	4	ax	b	ah	v	
bad	3	b	ae	d		
carry	4	k	ae	r	iy	
define	5	d	iy	f	ay	n
end	3	eh	n	d		
gone	3	g	ao	n		
hours	4	aw	w	axr	z	

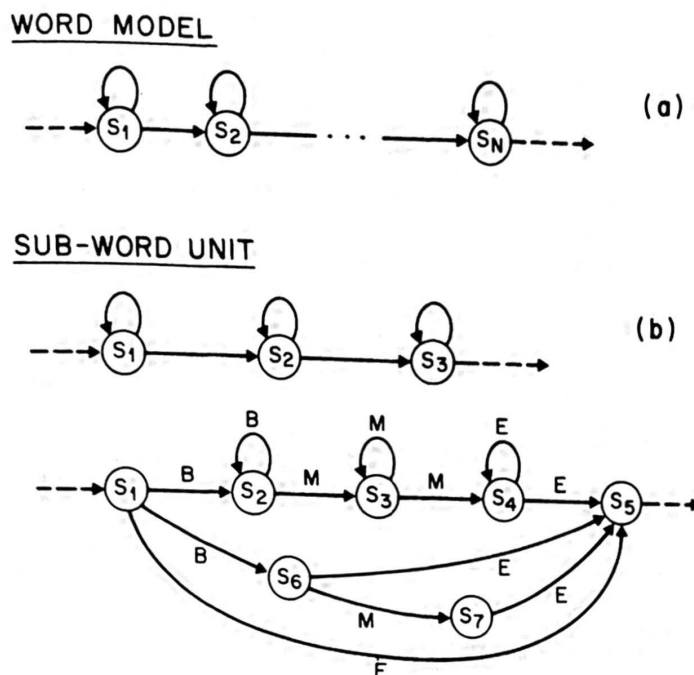


Figure 8.1 HMM representations of a word (a) and a subword unit (b).

letter A), but is pronounced as /ax/ in context. Another example is a word like “data,” which can be pronounced as /d ey t ax/ or /d ae t ax/ depending on the speaker’s dialect. Finally words like “you” are normally pronounced as /y uw/ but in context often are pronounced as /jh ax/ or /jh uh/. There are several ways of accounting for word pronunciation variability, including multiple entries in the word lexicon, use of phonological rules in the recognition grammar, and use of context dependent PLUs. We will discuss these options later in this chapter.

8.3 SUBWORD UNIT MODELS BASED ON HMMs

As we have shown several times in this book, the most popular way in which speech is modeled is as a left-to-right hidden Markov model. As shown in Figure 8.1a, a whole-word model typically uses a left-to-right HMM with N states, where N can be a fixed value (e.g., 5–10 for each word), or can be variable with the number of sounds (phonemes) in the word, or can be set equal to the average number of frames in the word. For subword units, typically, the number of states in the HMM is set to a fixed value, as shown in Figure 8.1b where a three-state model is used. This means that the shortest tokens of each subword unit must last at least three frames, a restriction that seems reasonable in practice. (Models that use jumps to eliminate this restriction have been studied [2].)

To represent the spectral density associated with the states of each subword unit, one of three approaches can be used. These approaches are illustrated in Figure 8.2. Perhaps the simplest approach is to design a VQ-based codebook for all speech sounds (as shown in part a of the figure). For this approach the probability density of the observed

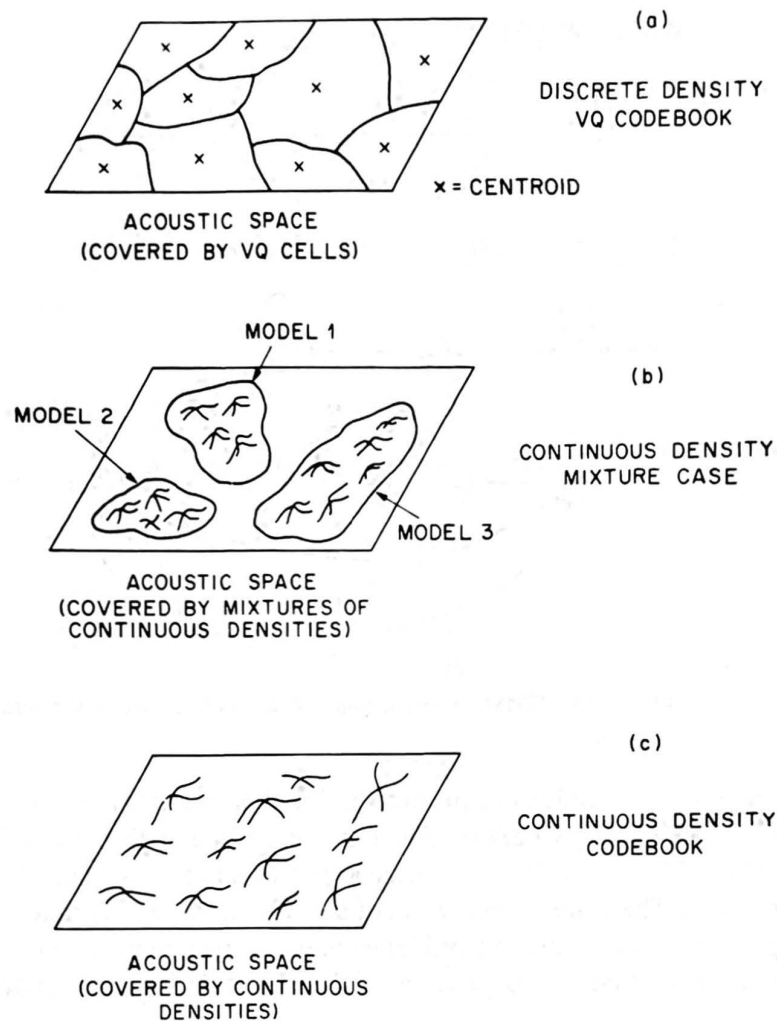


Figure 8.2 Representations of the acoustic space of speech by (a) partitioned VQ cells, (b) sets of continuous mixture Gaussian densities, and (c) a continuous-density codebook (after Lee et al. [7]).

spectral sequence within each state of each PLU is simply a discrete density defined over the codebook vectors. The interpretation of the discrete density within a state is that of implicitly isolating the part of the acoustic space in which the spectral vectors occur and assigning the appropriate codebook vector (over that part of the space) a fixed probability for spectral vectors within each isolated region regardless of its proximity to the corresponding codebook vector. A second alternative, illustrated in part b of Figure 8.2, is to represent the continuous probability density in each subword unit state by a mixture density that explicitly defines the part of the acoustic space in which the spectral vectors occur. Each mixture component has a spectral mean and variance that is highly dependent on the spectral characteristics of the subword unit (i.e., highly localized in the acoustic space). Hence the models for different subword units usually do not have substantial overlap in the acoustic space. Finally, a third alternative is to design a type of continuous density codebook over the entire acoustic space, as illustrated in part c of Figure 8.2. Basically the entire acoustic

space is covered by a set of independent Gaussian densities, derived in much the same way as the discrete VQ codebook, with the resulting set of means and covariances stored in a codebook. This alternative is a compromise between the previous two possibilities. It differs from the discrete density case in the way the probability of an observation vector is computed; instead of assigning a fixed probability to any observation vector that falls within an isolated region, it actually determines the probability according to the closeness of the observation vector to the codebook vector (i.e., it calculates the exponents of the Gaussian distributions). For each state of each subword unit, the density is assumed to be a mixture of the fixed codebook densities. Hence, even though each state is characterized by a continuous mixture density, one need only estimate the set of mixture gains to specify the continuous density completely. Furthermore, since the codebook set of Gaussian densities is common for all states of all subword models, one can precompute the likelihoods associated with an input spectral vector for each of the codebook vectors, and ultimately determine state likelihoods using only a simple dot product with the state mixture gains. This represents a significant computational reduction over the full mixture continuous density case. This mixed density method has been called the tied mixture approach [10, 28] as well as the semicontinuous modeling method [11] and has been applied to the entire acoustic space as well as to pieces of the acoustic space for detailed PLU modeling. This method can be further extended to the case in which a set of continuous density codebooks is designed, one for each state of each basic (context independent) speech unit. One can then estimate sets of mixture gains appropriate to context dependent versions of each basic speech unit and use them appropriately for recognition. We will return to this issue later in this chapter.

8.4 TRAINING OF SUBWORD UNITS

Implicitly it would seem that training of the models for subword units would be extremely difficult, because there is no simple way to create a bootstrap model of such short, imprecisely defined, speech sounds. Fortunately, this is not the case. The reason for this is because of the inherent tying of subword units across words and sentences—that is, every subword unit occurs a large number of times in any reasonable size training set. Hence estimation algorithms like the forward-backward procedure, or the segmental k -means algorithm, can start with a uniform segmentation (flat or random initial models) and rapidly converge to the best model estimates in just a few iterations.

To illustrate how models of subword units are estimated, assume we have a labeled training set of speech sentences, where each sentence consists of the speech waveform and its transcription into words. (We assume that waveform segmentation into words is not available.) We further assume a word lexicon is available that provides a transcription of every word in the training set strings in terms of the set of subword units being trained. We assume that silence can (but needn't) precede or follow any word within a sentence (i.e., we allow pauses in speaking), with silence at the beginning and end of each sentence the most likely situation. Based on the above assumptions, a typical sentence in the training set can be transcribed as

$$S_w : W_1 W_2 W_3 \dots W_I,$$

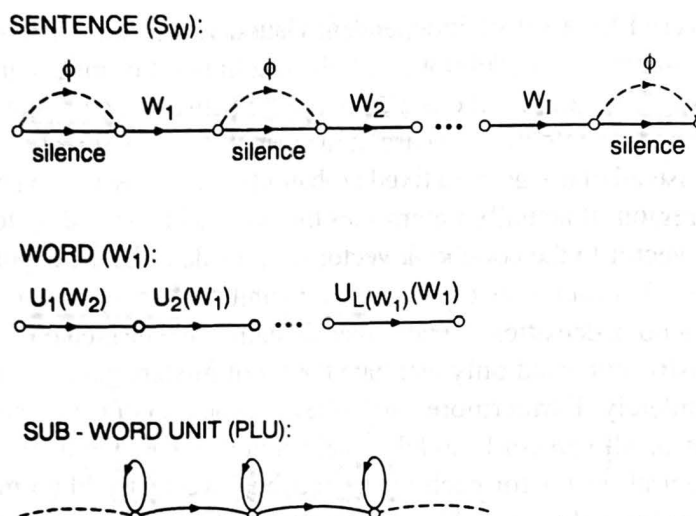


Figure 8.3 Representation of a sentence, word, and subword unit in terms of FSNs.

in which each W_i , $1 \leq i \leq I$, is a word in the lexicon. Hence the sentence “show all alerts” is a three-word sentence with $W_1 = \text{show}$, $W_2 = \text{all}$, and $W_3 = \text{alerts}$. Each word can be looked up in the lexicon to find its transcription in terms of subword units. Hence the sentence S can be written in terms of subword units as

$$S_U : U_1(W_1)U_2(W_1) \dots U_{L(W_1)}(W_1) \oplus U_1(W_2)U_2(W_2) \dots U_{L(W_2)}(W_2) \oplus U_1(W_3)U_2(W_3) \dots U_{L(W_3)}(W_3) \oplus \dots \oplus U_1(W_I)U_2(W_I) \dots U_{L(W_I)}(W_I),$$

where $L(W_1)$ is the length (in units) of word W_1 , etc. Finally we replace each subword unit by its HMM (the three-state models shown in Figure 8.1) and incorporate the assumptions about silence between words to give an extended HMM for each sentence.

The above process is illustrated (in general) in Figure 8.3. We see that a sentence is represented as a finite-state network (FSN) where the arcs are either words or silence or null arcs (where a null (ϕ) transition is required to skip the alternative silence). Each word is represented as an FSN of subword units and each subword unit is represented as a three-state HMM.

Figure 8.4 shows the process of creating the composite FSN for the sentence “Show all alerts,” based on a single-word pronunciation lexicon. One feature of this implementation is the use of a single-state HMM for the silence word. This is used (rather than the three-state HMMs used for each PLU), since silence is generally stationary and has no temporal structure to exploit.

When there are multiple representations of words in the lexicon (e.g., for two or more distinct pronunciations) it is easy to modify the FSN of Figures 8.3 and 8.4 to add parallel paths for the word arcs. (We will see that only one path is chosen in training, namely the best representation of the actual word pronunciation in the context of the spoken sentence.) Furthermore, multiple models of each subword unit can be used by introducing parallel paths in the word FSNs and then choosing the best version of each subword unit in the decoding process.

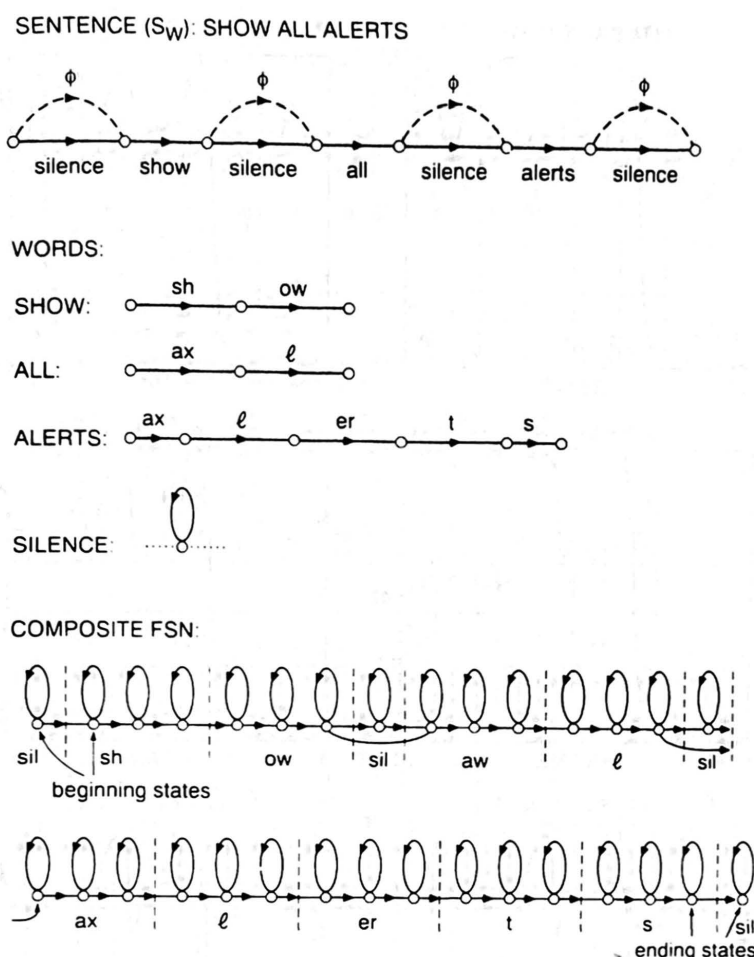


Figure 8.4 Creation of composite FSN for sentence "Show all alerts."

Once a composite sentence FSN is created for each sentence in the training set, the training problem becomes one of estimating the subword unit model parameters which maximize the likelihood of the models for all the given training data. The maximum likelihood parameters can be solved for using either the forward-backward procedure (see Ref. [2] for example) or the segmental k -means training algorithm. The way in which we use the segmental k -means training procedure to estimate the set of model parameters (based on using a mixture density with M mixtures/state) is as follows:

1. **Initialization:** Linearly segment each training utterance into units and HMM states assuming no silence between words (i.e., silence only at the beginning and end of each sentence), a single lexical pronunciation of each word, and a single model for each subword unit. Figure 8.5, iteration 0, illustrates this step for the first few units of one training sentence. Literally we assume every unit is of equal duration initially.
2. **Clustering:** All feature vectors from all segments corresponding to a given state (i) of a given subword unit are partitioned into M clusters using the k -means algorithm. (This step is iterated for all states of all subword units.)
3. **Estimation:** The mean vectors, μ_{ik} , the (diagonal) covariance matrices, U_{ik} , and the

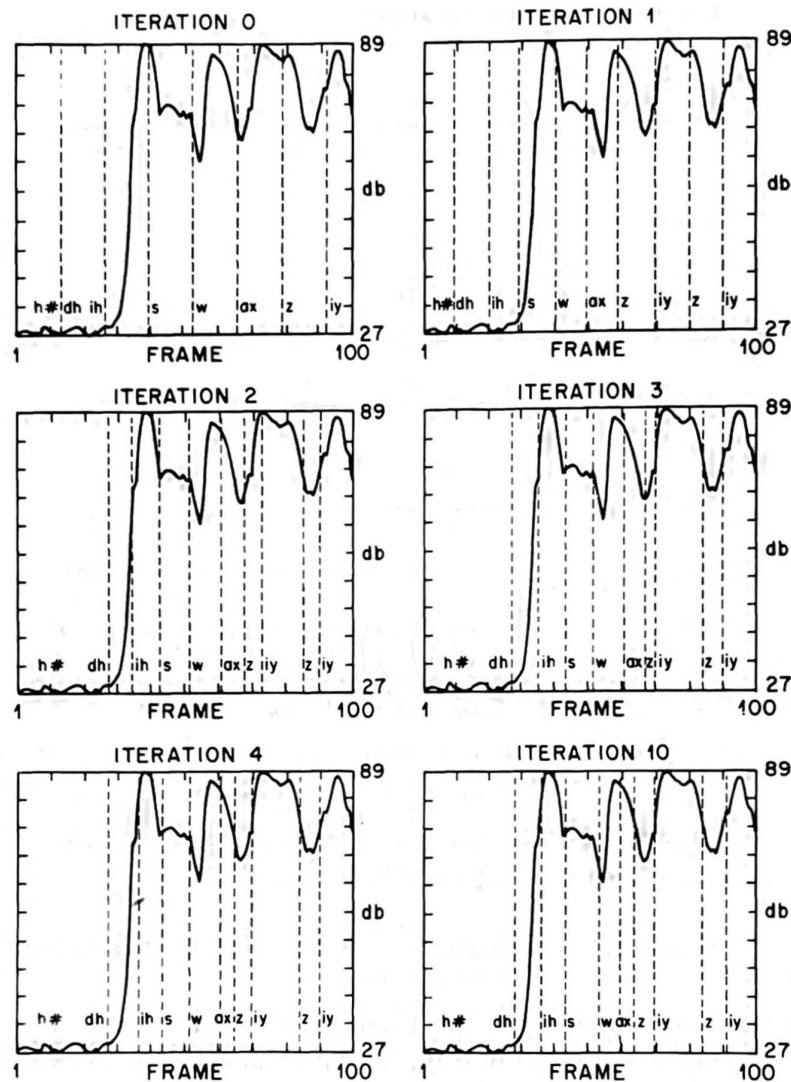


Figure 8.5 Segmentations of a training utterance resulting from the segmental k -means training for the first several iterations (after Lee et al. [7]).

mixture weights, c_{ik} , are estimated for each cluster k in state i . (This step is iterated for all states of all subword units.)

4. **Segmentation:** The updated set of subword unit models (based on the estimation of step 3) is used to resegment each training utterance into units and states (via Viterbi decoding). At this point multiple lexical entries can be used for any word in the vocabulary. Figure 8.5 shows the result of this resegmentation step for iterations 1–4 and 10 for one training utterance. It can be seen that by iteration 2 the segmentation into subword units is remarkably stable.
5. **Iteration:** Steps 2–4 are iterated until convergence (i.e., until the overall likelihoods stop increasing).

Figure 8.6 illustrates the resulting segmentation of the first few units of the utterance

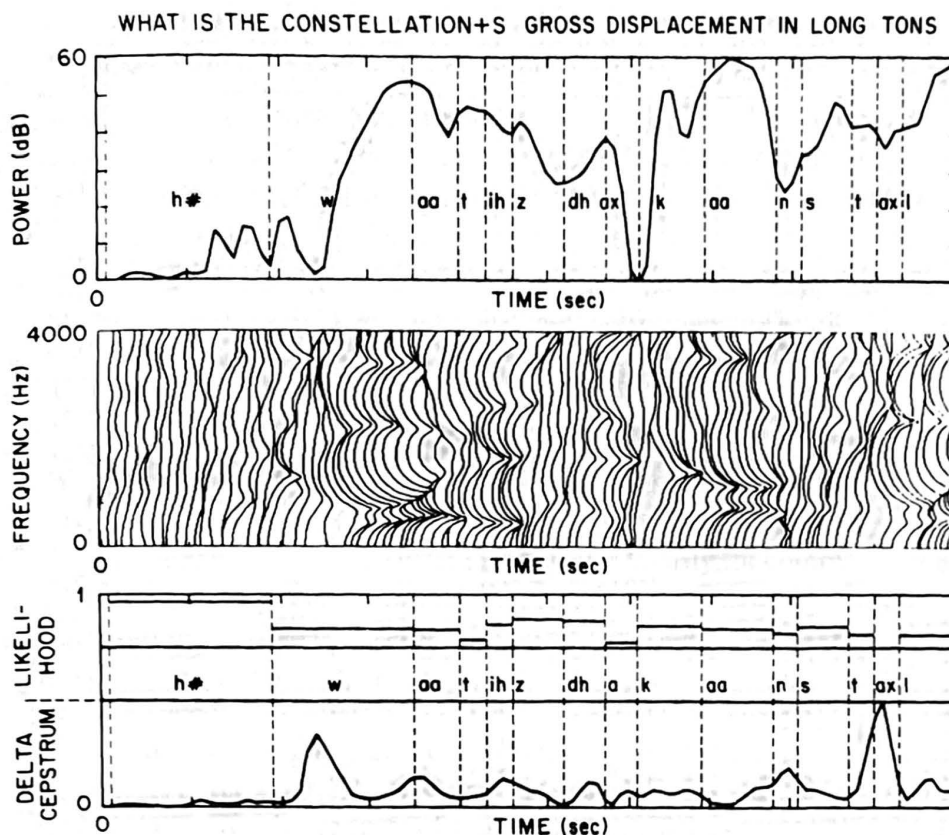


Figure 8.6 Segmentation of an utterance into PLUs (after Lee et al. [7]).

“What is the constellation. . .” Shown in this figure are the power contour in dB (upper panel), the running LPC spectral slices (the middle panel), and the likelihood scores and delta-cepstral values (lower panel) for the first second of the sentence. The resulting segmentations are generally remarkably consistent with those one might manually choose based on acoustic-phonetic criteria. Since we use an acoustic criterion for choice of segmentation points, the closeness of PLU units to true phonetic units is often remarkable, especially in light of the phonetic variability in word pronunciation discussed previously.

In summary we have shown how one can use a training set of speech sentences that have only word transcriptions associated with each sentence and optimally determine the parameters of a set of subword unit HMMs. The resulting parameter estimates are extremely robust to the training material as well as to details of word pronunciation as obtained from the word lexicon. The reason for this is that a common word lexicon (with associated word pronunciation errors) is used for both training and recognition; hence errors in associating proper subword units to words are consistent throughout the process and are less harmful than they would be in alternative methods of estimating parameters of subword models.

The results of applying the segmental k -means training procedure to a set of 3990 training sentences from 109 different talkers, in terms of PLU counts and PLU likelihood scores are shown in Table 8.3. A total of 155,000 PLUs occurred in the 3990 sentences with silence (h#) having the most occurrences (10,638 or 6.9% of the total) and nx (flapped

TABLE 8.3. PLU statistics on count and average likelihood score.

PLU	Count	%	Average likelihood	(Rank)
h#	10638	6.9	18.5	(1)
r	8997	5.8	8.4	(45)
t	8777	5.7	9.7	(37)
ax	8715	5.6	7.1	(47)
s	8625	5.6	15.4	(3)
n	8478	5.5	8.3	(46)
ih	6542	4.2	9.9	(35)
iy	5816	3.7	12.0	(17)
d	5391	3.5	8.5	(44)
ae	4873	3.1	13.3	(10)
ℓ	4857	3.1	8.9	(41)
z	4733	3.0	12.4	(14)
eh	4604	3.0	11.2	(21)
k	4286	2.8	10.6	(27)
p	3793	2.4	14.3	(6)
m	3625	2.3	8.5	(43)
ao	3489	2.2	10.4	(32)
f	3276	2.1	17.7	(2)
ey	3271	2.1	14.5	(5)
w	3188	2.1	10.2	(34)
ix	3079	2.0	8.7	(42)
dh	2984	1.9	11.8	(18)
v	2979	1.9	12.0	(16)
aa	2738	1.8	10.3	(33)
b	2138	1.4	10.7	(25)
y	2137	1.4	13.1	(11)
uw	2032	1.3	10.6	(26)
sh	1875	1.2	13.1	(12)
ow	1875	1.2	10.9	(24)
axr	1825	1.2	9.5	(38)
ah	1566	1.0	11.3	(20)
dx	1548	1.0	10.4	(31)
ay	1527	1.0	13.9	(8)
en	1478	0.9	9.1	(40)
g	1416	0.9	9.8	(36)
hh	1276	0.8	11.4	(19)
th	924	0.6	14.1	(7)
ng	903	0.6	9.1	(39)
ch	885	0.6	12.5	(13)
el	863	0.6	11.0	(23)
er	852	0.5	10.6	(29)
jh	816	0.5	10.6	(28)
aw	682	0.4	13.6	(9)
uh	242	0.2	11.0	(22)
zh	198	0.1	12.2	(15)
oy	130	0.1	15.3	(4)
nx	57	0.04	10.4	(30)

n) having the fewest occurrences (5 or 0.04% of the total). In terms of average likelihood scores, silence (h#) had the highest score (18.5) followed by f (17.7) and s (15.4), while ax had the lowest score (7.1), followed by n (8.3) and r (8.4). (Note that, in this case, a higher average likelihood implies less variation among different renditions of the particular sound.) It is interesting to note that the PLUs with the three lowest average likelihood scores (ax, n, and r) were among the most frequently occurring sounds (r was second, n sixth, and ax fourth in frequency of occurrence). Similarly, some of the sounds with the highest likelihood scores were among the least occurring sounds (e.g., oy was fourth according to likelihood score but 21st according to frequency of occurrence).

8.5 LANGUAGE MODELS FOR LARGE VOCABULARY SPEECH RECOGNITION

Small vocabulary speech-recognition systems are used primarily for command-and-control applications where the vocabulary words are essentially acoustic control signals that the system has to respond to. (See Chapter 9 for a discussion of command-and-control applications of speech recognition.) As such, these systems generally do not rely heavily on language models to accomplish their selected tasks. A large vocabulary speech-recognition system, however, is generally critically dependent on linguistic knowledge embedded in the input speech. Therefore, for large vocabulary speech recognition, incorporation of knowledge of the language, in the form of a "language" model, is essential. In this section we discuss a statistically motivated framework for language modeling.

The goal of the (statistical) language model is to provide an estimate of the probability of a word sequence W for the given recognition task. If we assume that W is a specified sequence of words, i.e.,

$$W = w_1 w_2 \dots w_Q, \quad (8.4)$$

then it would seem reasonable that $P(W)$ can be computed as

$$P(W) = P(w_1 w_2 \dots w_Q) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \dots P(w_Q|w_1 w_2 \dots w_{Q-1}). \quad (8.5)$$

Unfortunately, it is essentially impossible to reliably estimate the conditional word probabilities, $P(w_j|w_1 \dots w_{j-1})$ for all words and all sequence lengths in a given language. Hence, in practice, it is convenient to use an N -gram word model, where we approximate the term $P(w_j|w_1 \dots w_{j-1})$ as

$$P(w_j|w_1 w_2 \dots w_{j-1}) \approx P(w_j|w_{j-N+1} \dots w_{j-1}), \quad (8.6)$$

i.e., based only on the preceding $N - 1$ words. Even N -gram probabilities are difficult to estimate reliably for all but $N = 2$ or possibly 3. Hence, in practice, it is often convenient to use a word pair model that specifies which word pairs are valid in the language through the use of a binary indicator function, i.e.,

$$P(w_j|w_k) = \begin{cases} 1 & \text{if } w_k w_j \text{ is valid} \\ 0 & \text{otherwise} \end{cases} \quad (8.7)$$

Another simple language model, often called the no-grammar model, assumes $P(w_j|w_k) = 1$ for all j and k , so that every word is assumed capable of being followed by every other word in the language. In the next section we show how the word pair and the no-grammar models can be implemented as finite state networks so as to be integrated simply into a recognition decoding algorithm.

Alternative language models include formal grammars (e.g., context free or context dependent grammar), N -grams of word classes (rather than words) etc. These types of grammars provide more realistic models for natural language input to machines than the artificial N -grams or words, or the word pair grammars. However, they are somewhat more difficult to integrate with the acoustic decoding and hence will not be discussed here.

8.6 STATISTICAL LANGUAGE MODELING

In large vocabulary speech recognition, in which word sequences W are uttered to convey some message, the language model $P(W)$ is of critical importance to the recognition accuracy as shown in Eq. (8.3). In most cases, the language model $P(W)$ has to be estimated from a given (large) text corpus. In this section we discuss how to construct such a statistical language model from a (textual) training corpus.

For practical reasons, the word sequence probability $P(W)$ is approximated by

$$P_N(W) = \prod_{i=1}^Q P(w_i|w_{i-1}, w_{i-2}, \dots, w_{i-N+1}), \quad (8.8)$$

which is called an N -gram language model. The conditional probabilities $P(w_i|w_{i-1}, \dots, w_{i-N+1})$ can be estimated by the simple relative frequency approach,

$$\hat{P}(w_i|w_{i-1}, \dots, w_{i-N+1}) = \frac{F(w_i, w_{i-1}, \dots, w_{i-N+1})}{F(w_{i-1}, \dots, w_{i-N+1})}, \quad (8.9)$$

in which F is the number of occurrences of the string in its argument in the given training corpus. Obviously, in order for the estimate in Eq. (8.9) to be reliable, $F(w_i, w_{i-1}, \dots, w_{i-N+1})$ has to be substantial in the given corpus. The implications of this are that the size of the training corpus may be prohibitively large and that $F(w_i, w_{i-1}, \dots, w_{i-N+1}) = 0$ for many possible word strings, $w_i, w_{i-1}, \dots, w_{i-N+1}$, due to the limited size of the corpus.

One way to circumvent this problem is to smooth the N -gram frequencies as suggested by Jelinek et al. [12]. Consider $N = 3$, the trigram model. The smoothing is done by interpolating trigram, bigram and unigram relative frequencies

$$\hat{P}(w_3|w_1, w_2) = p_1 \frac{F(w_1, w_2, w_3)}{F(w_1, w_2)} + p_2 \frac{F(w_1, w_2)}{F(w_1)} + p_3 \frac{F(w_1)}{\sum F(w_i)}, \quad (8.10)$$

in which the nonnegative weights satisfy $p_1 + p_2 + p_3 = 1$ and $\sum F(w_i)$ is the size of the corpus. The weights depend on the values of $F(w_1, w_2)$ and $F(w_1)$ and can be obtained by applying the principle of cross-validation [12].

8.7 PERPLEXITY OF THE LANGUAGE MODEL

Having constructed a language model from a training corpus, one may ask how well the language model will perform in the context of speech recognition. This can be answered based on the concept of source of information in information theory. To provide such a measure of performance, we must first discuss several concepts, including entropy, estimated entropy, and perplexity.

Consider an information source that puts out sequences of words (symbols) $w_1, w_2, \dots, w_Q$, each of which is chosen from a vocabulary $\bar{V}$ with size $|\bar{V}|$, according to some stochastic law. The entropy of the source can be defined as

$$H = - \lim_{Q \rightarrow \infty} \left(\frac{1}{Q} \right) \left\{ \sum P(w_1, w_2, \dots, w_Q) \log P(w_1, w_2, \dots, w_Q) \right\}, \quad (8.11)$$

in which $P(\cdot)$ is the probability of the argument string the source will put out according to the stochastic law and the summation is over all string sequences $w_1, w_2, \dots, w_Q$. If the words in the string sequence are generated by the source in an independent manner

$$P(w_1, w_2, \dots, w_Q) = P(w_1)P(w_2) \dots P(w_Q), \quad (8.12)$$

then

$$H = - \sum_{w \in \bar{V}} P(w) \log P(w), \quad (8.13)$$

which is sometimes referred to as the first-order entropy of the source (even if Eq. (8.12) is not true).

The quantity H of Eq. (8.11) can be considered as the average information of the source when it puts out a word w . Equivalently, a source of entropy H is one that has as much information content as a source which puts out words equiprobably from a vocabulary of size 2^H .

If the source is ergodic (meaning its statistical properties can be completely characterized in a sufficiently long sequence that the source puts out), the entropy of Eq. (8.11) is equivalent to

$$H = - \lim_{Q \rightarrow \infty} \left(\frac{1}{Q} \right) \log P(w_1, w_2, \dots, w_Q). \quad (8.14)$$

In other words, we can compute the entropy from a "typical" (long) sequence of words generated by the source. The length of this typical sequence (i.e., the corpus) has to approach infinity, which is of course unattainable. We often compute H based on a finite but sufficiently large Q ; i.e.,

$$H = - \left(\frac{1}{Q} \right) \log P(w_1, w_2, \dots, w_Q). \quad (8.15)$$

An interesting interpretation of H from the perspective of speech recognition is that it is the degree of difficulty that the recognizer encounters, on average, when it is to determine a word from the same source. This difficulty, or uncertainty, is based on the actual probability $P(w_1, w_2, \dots, w_Q)$ which, for natural languages, is usually not known

beforehand and thus has to be estimated. (We do not include acoustic uncertainty in the present context of language modeling.)

One way to estimate H is to use $P(W) = P(w_1, w_2, \dots, w_Q)$ from the language model. For example, if the N -gram language model $P_N(W)$ (Eq. (8.8)) is used, an estimate of H of Eq. (8.15) is thus

$$H_p = -\frac{1}{Q} \sum_{i=1}^Q \log P(w_i | w_{i-1}, w_{i-2}, \dots, w_{i-N+1}). \quad (8.16)$$

In general,

$$H_p = -\frac{1}{Q} \log \hat{P}(w_1, w_2, \dots, w_Q), \quad (8.17)$$

where $\hat{P}(w_1, w_2, \dots, w_Q)$ is an estimate of $P(w_1, w_2, \dots, w_Q)$. The quantity H_p is an estimated entropy as calculated from a sufficiently long sequence based on a language model. If the source is ergodic and $Q \rightarrow \infty$, $H_p \geq H$. Intuitively, this can be easily verified by the fact that knowledge of the true probability $P(w_1, w_2, \dots, w_Q)$ is the best a recognizer can use and any other probability estimate or language model can never make the task easier for the recognizer. Since H_p is an indication of the recognition difficulty lower-bounded by H , a language model that achieves a lower H_p (i.e., closer to H) is therefore considered a better model than another language model which leads to a higher H_p .

Associated with H_p is a quantity called perplexity (often called the average word branching factor of the language model) defined as

$$B = 2^{H_p} = \hat{P}(w_1, w_2, \dots, w_Q)^{-1/Q}. \quad (8.18)$$

Note that H_p is the average difficulty or uncertainty in each word based on the language model. When the recognizer uses this language model for the task, the difficulty it faces is equivalent to that of recognizing a text generated by a source that chooses words from a vocabulary size of B independently of each other and with equal probability. Another way to view perplexity is to consider it as the average number of possible words following any string of $(N - 1)$ words in a large corpus based on an N -gram language model. Perplexity is an important parameter in specifying the degree of sophistication in a recognition task, from the source uncertainty to the quality of the language model.

8.8 OVERALL RECOGNITION SYSTEM BASED ON SUBWORD UNITS

A block diagram of the overall continuous speech-recognition system based on subword speech units is shown in Figure 8.7. The first step in the processing is spectral analysis to derive the feature vector used to characterize the spectral properties of the speech input. For the most part, we will consider spectral vectors with 38 components consisting of 12 cepstral components, 12 delta cepstral components, 12 delta-delta cepstral components, delta log energy, and delta-delta log energy. (Systems with the first 12 and the first 24

CONTINUOUS SENTENCE RECOGNIZER

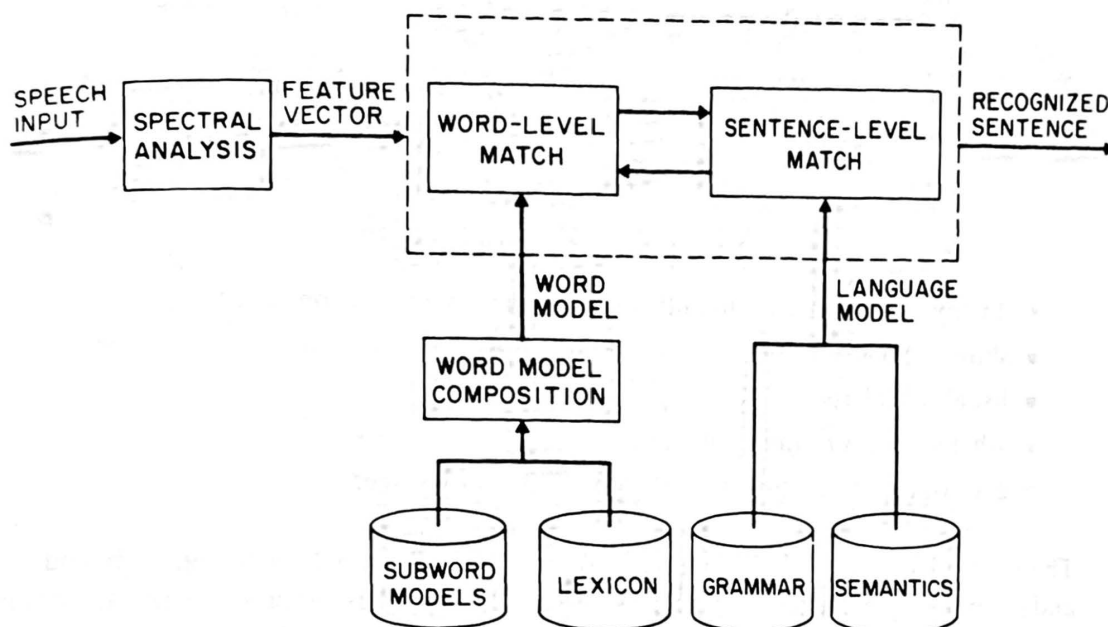


Figure 8.7 Overall block diagram of subword unit based continuous speech recognizer.

features were also studied, but results on such systems will not be presented here.)

The second step in the recognizer is a combined word-level/sentence-level match. The way this is accomplished is as follows. Using the set of subword HMMs and the word lexicon, a set of word models (HMMs) is created by concatenating each of the subword unit HMMs as specified in the word lexicon. At this point, the system is very similar to the connected word recognizers of Chapter 7. The way in which the sentence-level match is done is via an FSN realization of the word grammar (the syntax of the system) and the semantics as expressed in a composite FSN language model. The implementation of the combined word-level match/sentence-level match is via any of the structures described in Chapter 7. In particular, most systems use structures similar to the frame synchronous level-building method (usually with some type of beam search to restrict the range of paths) to solve for the “best” recognition sentence.

Consider using the recognizer of Figure 8.7 for a database management task called the Naval Resource (Battleship) Management Task—as popularly defined within the DARPA community [13]. This task, which has a 991-word vocabulary (plus a separate silence word), can be used to query a database as to locations, attributes, constraints, history, and other information about ships within the database. Typical examples of sentences used to query the database include

- what is mishawaka’s percent fuel
- total the ships that will arrive in diego-garcia by next month

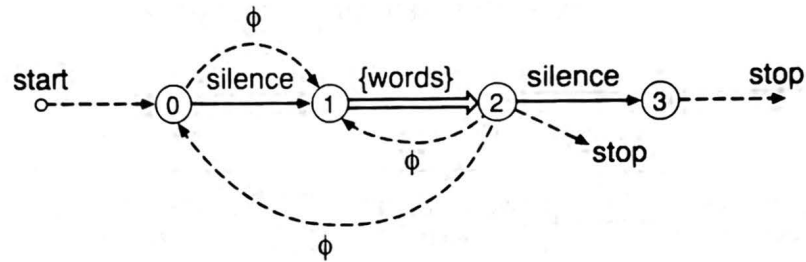


Figure 8.8 FSN for the NG syntax.

- do any vessels that are in gulf of tonkin have asw mission area of m4
- show the names of any submarines in yellow sea on twenty eight october
- list all the alerts
- what's jason's m-rating on mob
- give t-lam vessels that weren't deployed in november.

The vocabulary thus includes many jargon words, such as m4, m-rating, mob, and t-lam, and several long-content words, such as mishawaka's, diego-garcia, submarines, november, etc., and many short-function words, such as is, the, by, do, in, of, and on.

A wide range of sentences can be constructed from the 991-word vocabulary to query this database. It is possible to construct a finite-state network representation of the full grammar associated with all such sentences. The perplexity (average word branching factor) (see Section 8.7) of the full grammar network is computed to be about 9. However, such a network is rather large (because of the high degree of constraint among words within the vocabulary which form syntactically valid and semantically meaningful sentences) with upward of 50,000 arcs and 20,000 nodes, and cannot easily be implemented as a practical system. Instead, several types of FSN approximations to the full grammar have been constructed.

Perhaps the least constraining grammar (and the simplest to implement) is the no grammar (NG) case, in which any word in the vocabulary is allowed to follow any word in the vocabulary. Such an FSN has the property that, although its coverage of valid sentences is perfect, its overcoverage of the language (i.e., the ratio of sentences generated by the grammar to valid sentences within the task language) is extremely large. The perplexity of the FSN for the NG case is 991, since each word can follow every word in the grammar (assuming all words are essentially equiprobable). The FSN for the NG case is shown in Figure 8.8. (Note that the FSN of Figure 8.8 allows arbitrary phrasing, i.e., groups of words spoken together followed by a pause, because of the silence model and the null arcs.)

A second FSN form of the task syntax is to create a word pair (WP) grammar that specifies explicitly which words can follow each of the 991 words in the vocabulary. The perplexity of this grammar is about 60, and the overcoverage, while significantly below that of the NG case, is still very high. Although the network of Figure 8.8 could be used for the WP grammar (by explicitly including the word pair information at node 2), a somewhat more efficient structure exploits the fact that only a subset of the vocabulary occurs as the first word in a sentence (*B* or beginning words), and only a subset of the vocabulary occurs

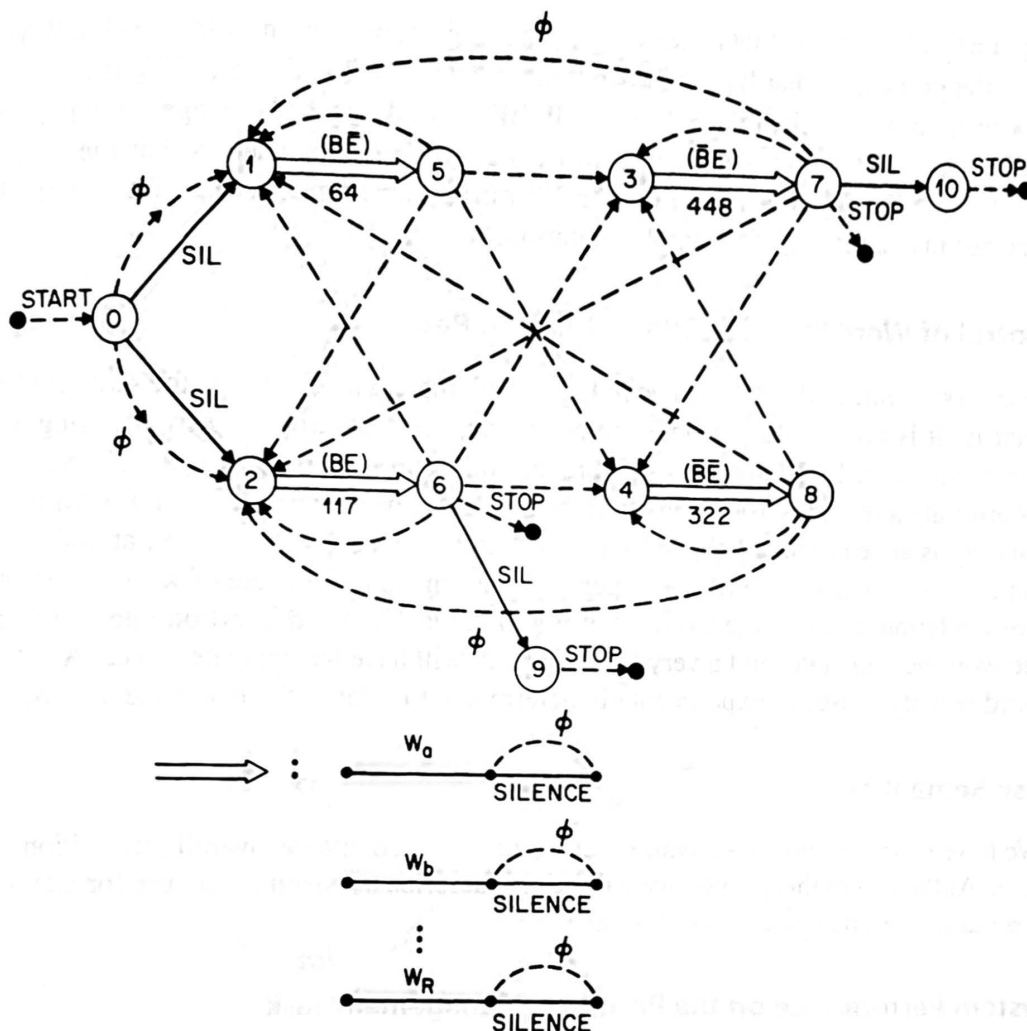


Figure 8.9 FSN of the WP syntax.

as the last word in a sentence (E or ending words); hence we can partition the vocabulary into four nonoverlapping sets of words, namely

$\{BE\}$ = set of words that can either begin or end a sentence, $|BE| = 117$

$\{B\bar{E}\}$ = set of words that can begin a sentence but cannot end a sentence, $|B\bar{E}| = 64$

$\{\bar{B}E\}$ = set of words that cannot begin a sentence but can end a sentence, $|\bar{B}E| = 488$

$\{\bar{B}\bar{E}\}$ = set of words that cannot begin or end a sentence, $|\bar{B}\bar{E}| = 322$.

The resulting FSN, based on this partitioning scheme, is shown in Figure 8.9. This network has 995 real arcs and 18 null arcs. To account for silence between words (which is optional), each word arc bundle (e.g., nodes 1 to 4) is expanded to individual words followed by optional silence, as shown at the bottom of Figure 8.9. Hence the overall FSN allows recognition of sentences of the form

$S : (\text{silence}) - \{B\bar{E}, BE\} - (\text{silence}) - (\{W\}) \dots (\{W\}) - (\text{silence}) - \{\bar{B}E, BE\} - (\text{silence}).$

Finally, one could construct a task syntax based on statistical word bigram (or even

trigram) probabilities—that is, we assign a probability, p_{ij} , to each word pair (W_i, W_j) where p_{ij} is the probability that W_i is followed immediately by W_j . That is, if W_n is the n^{th} word in a string of words, then $p_{ij} = P(W_n = W_j | W_{n-1} = W_i)$ is the language model according to Eq. (8.6). The advantage of the word bigram (WB) approach is that the perplexity is reduced considerably (to 20) for the Resource Management task, with essentially no increase in complexity of the implementation.

8.8.1 Control of Word Insertion/Word Deletion Rate

Using a structure of the type shown in Figure 8.9, there is no control on the sentence length. That is, it is possible to generate sentences that are arbitrarily long by inserting a large number of short-function words. To prevent this from occurring, it is a simple matter to incorporate a word insertion penalty into the Viterbi decoding, such that a fixed negative quantity is added to the likelihood score at the end of *each* word arc (i.e., at nodes 5–8 in Figure 8.9). By adjusting the word penalty, we can control the rate of word insertion and word deletion; a very large word penalty will reduce the word insertion rate and increase the word deletion rate, and a very small penalty will have the opposite effect. A value for word penalty is usually experimentally determined to balance these adverse effects.

8.8.2 Task Semantics

We have discussed how task syntax can be incorporated into the overall recognition structure. At the end of this chapter we will briefly describe a general procedure for integrating a semantic component into the recognizer.

8.8.3 System Performance on the Resource Management Task

Using the segmental k -means training algorithm, the set of 47 PLUs of Table 8.1 were trained using a set of 4360 sentences from 109 talkers. The likelihood scores were essentially unchanged after two iterations of the k -means loop. The number of mixtures per state was varied from 1 to 256 in multiples of 2 to investigate the effects of higher acoustic resolution on performance.

To evaluate the recognizer performance, five different sets of test data were used, including:

train 109	A randomly selected set of 2 sentences from each of the 109 training talkers; this set was used to evaluate the ability of the algorithm to recognize the training material
feb 89	A set of 30 sentences from each of 10 talkers, none of whom was in the training set; this set was distributed by DARPA in February of 1989 to evaluate performance
oct 89	A second set of 30 sentences from each of 10 additional talkers, none of whom was in the training set; this set was distributed by DARPA in October of 1989
jun 90	A set of 120 sentences from each of 4 new talkers, none of whom was in the training set (distributed by DARPA in June of 1990)
feb 91	A set of 30 sentences from each of 10 new talkers, none of whom was in the training set (distributed by DARPA in February of 1991).

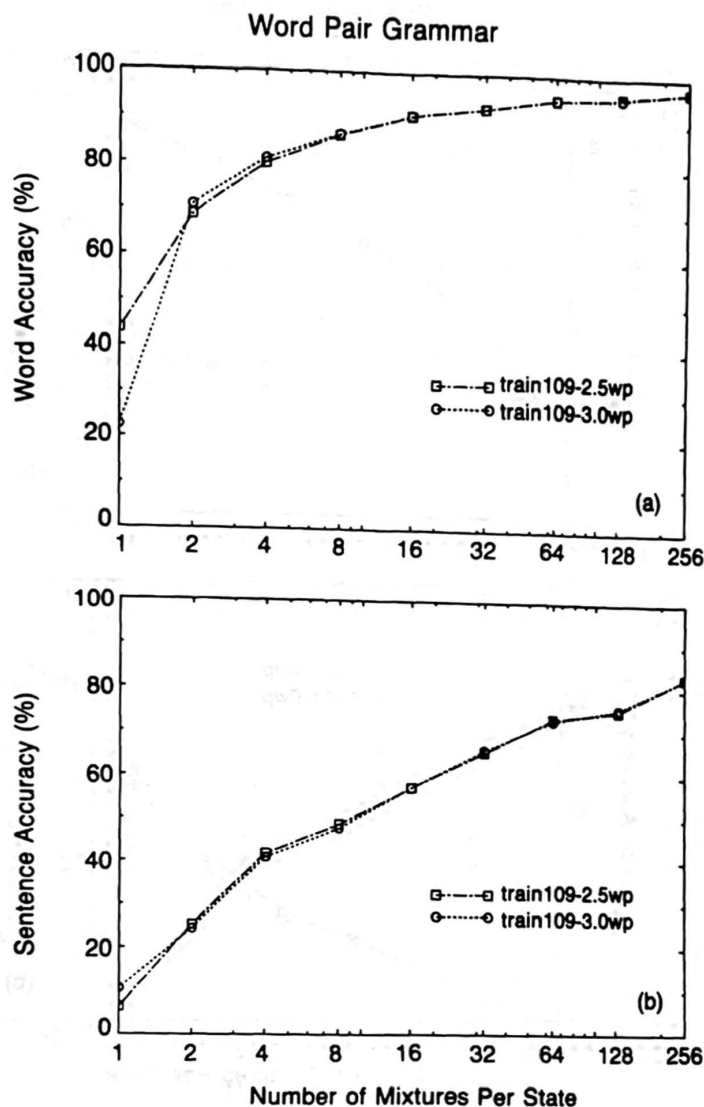


Figure 8.10 Word and sentence accuracies versus number of mixtures per state for the training subset using the WP syntax.

The recognizer performance was evaluated for each of the test sets, using both WP and NG syntax, and with different word penalties. For all cases, evaluations were made using models with from 1 to 256 mixtures per state for each PLU.

The recognition results are presented in terms of word accuracy (percentage words correct minus percentage word insertions) and sentence accuracy as a function of the number of mixtures per state for each PLU model. The alignment of the text of the recognized string with the text of the spoken string was performed using a dynamic programming alignment method as specified by DARPA.

The recognition results on the training subset (train 109) are given in Figures 8.10 (for the WP syntax) and 8.11 (for the NG syntax). The upper curves show word accuracy (in percentage) versus number of mixtures per state (on a logarithmic scale) for two different values of the word penalty, and the lower curves show sentence accuracy for the same

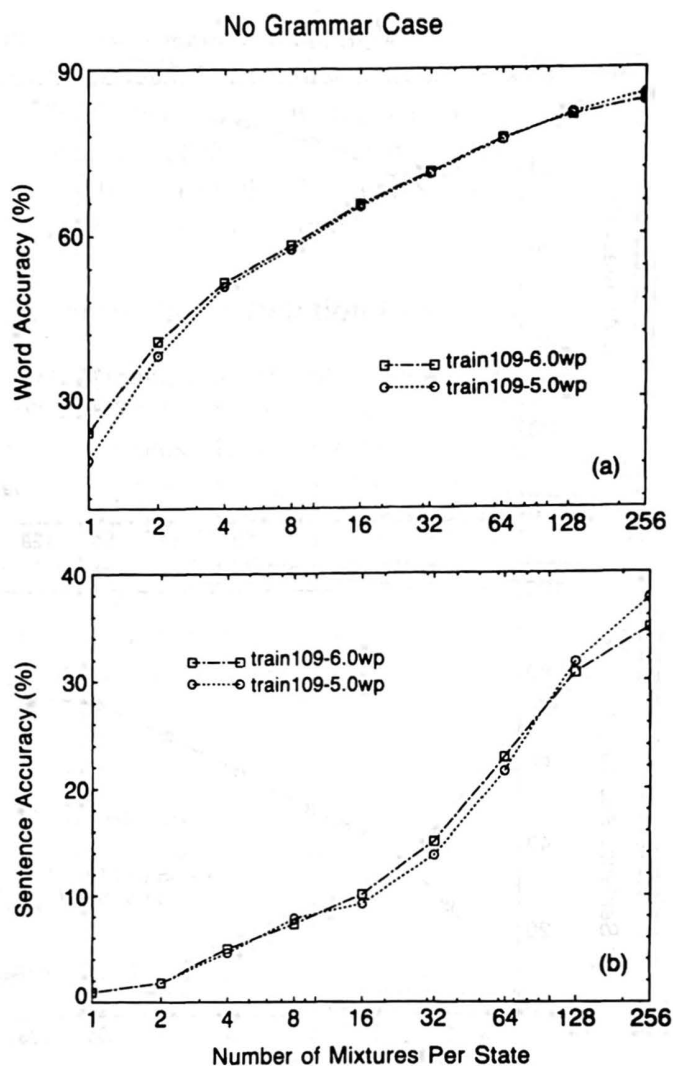


Figure 8.11 Word and sentence accuracies versus number of mixtures per state for the training subset using the NG syntax.

parameters. A sharp and steady increase in accuracy is obtained as the number of mixtures per state increases, going from about 43.6% word accuracy (10.6% sentence accuracy) for 1 mixture per state to 97.3% word accuracy (83% sentence accuracy) for 256 mixtures per state for the WP syntax using a word penalty of 2.5. For the NG syntax (using a word penalty of 6.0), the comparable results were 24% word accuracy (0.9% sentence accuracy) for 1 mixture per state and 84.2% word accuracy (34.9% sentence accuracy) for 256 mixtures per state.

The recognition results on the independent test sets are given in Figures 8.12 (for the WP syntax) and 8.13 (for the NG syntax). Although there are detailed differences in performance among the different test sets (especially for small numbers of mixtures per state), the performance trends are essentially the same for all the test sets. In particular we see that for the WP syntax, the range of word accuracies for 1 mixture per state is 42.9%

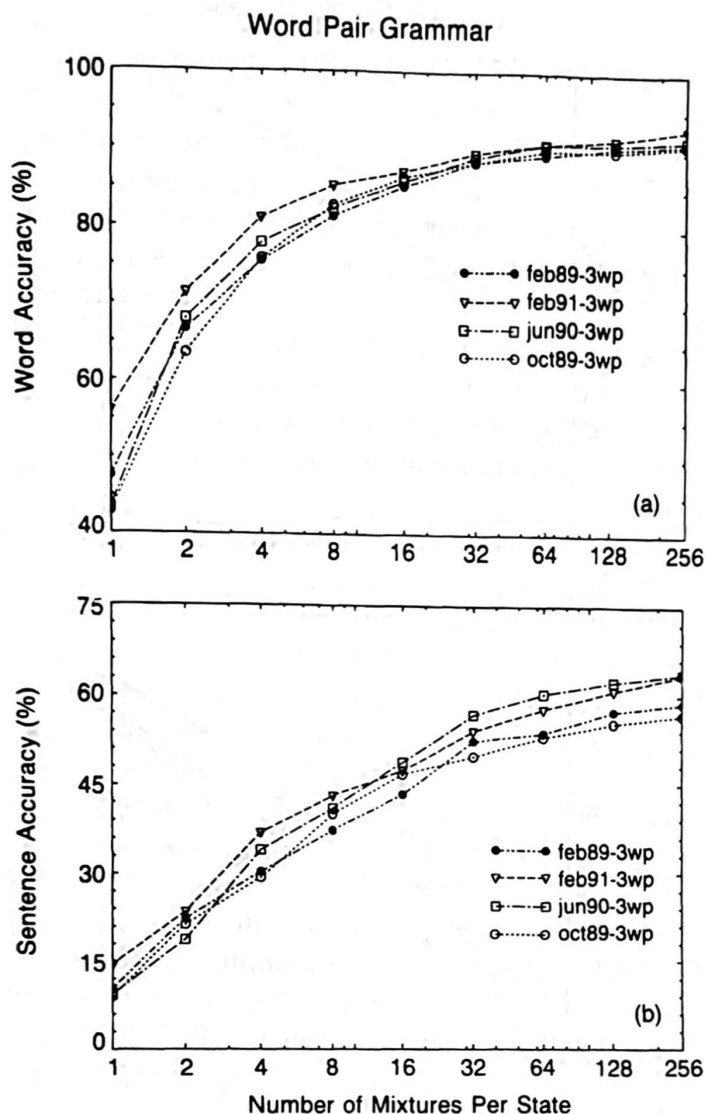


Figure 8.12 Word and sentence accuracies versus number of mixtures per state for the four test sets using the WP syntax.

(for feb 89) to 56.0% (for jun 90), whereas for 256 mixtures per state the range is 90.9% (for feb 89) to 93.0% (for jun 90). For the NG syntax, the range of word accuracies for 1 mixture per state is 20.1% (for feb 91) to 28.5% (for jun 90) and for 256 mixtures per state it is 68.5% (for oct 89) to 70.0% (for feb 91).

Perhaps the most significant aspect of the performance is the difference in accuracies between the test sets and the training subset. Thus there is a gap of 4–7% in word accuracy for the WP syntax at 256 mixtures per state, and a gap of 14.2–15.7% for the NG syntax at 256 mixtures per state. Such gaps are indicative of the ability of the training procedure to overtrain (learn details) on the training set, thereby achieving significantly higher recognition accuracy on this set than on any other representative test set.

The results presented in this section show that a simple set of context-independent PLUs can be trained for a continuous speech large vocabulary recognition task, using

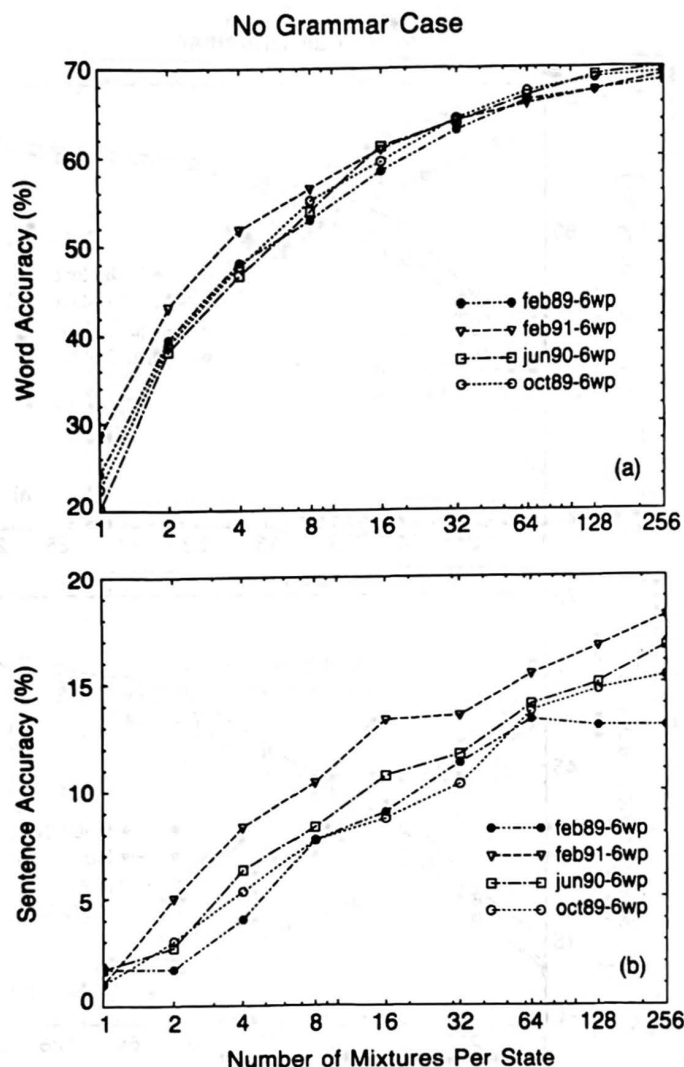


Figure 8.13 Word and sentence accuracies versus number of mixtures per state for the four test sets using the NG syntax.

standard Viterbi training procedures, and be used to provide reasonably good recognition accuracy for a moderately complex task. The key issue now is what can be done, in meaningful ways to improve recognizer performance. To answer this question, we will examine several possible extensions of the basic recognition system in the next few sections.

8.9 CONTEXT-DEPENDENT SUBWORD UNITS

There are several advantages to using a small basic set of context-independent subword units for large vocabulary speech recognition. First of all we have shown that the models of these subword units are easily trained from fluent speech, with essentially no human decisions as to segmentation and labeling of individual sections of speech. Second, the