- $\Omega = \{1,2,\ldots,N\}$—A set of states representing the state space. Here s_t is denoted as the state at time t. In the case of the Dow Jones Industrial average HMM, the state may indicate a bull market, a bear market, and a stable market.
- $\mathbf{A} = \{a_{ij}\}$—A transition probability matrix, where a_{ij} is the probability of taking a transition from state i to state j, i.e.,

$$a_{ij} = P(s_t = j | s_{t-1} = i) \tag{8.8}$$

- $\mathbf{B} = \{b_i(k)\}$—An output probability matrix,[2] where $b_i(k)$ is the probability of emitting symbol o_k when state i is entered. Let $\mathbf{X} = X_1, X_2, \ldots, X_t, \ldots$ be the observed output of the HMM. The state sequence $S = s_1, s_2, \ldots, s_t, \ldots$ is not observed (hidden), and $b_i(k)$ can be rewritten as follows:

$$b_i(k) = P(X_t = o_k \mid s_t = i) \tag{8.9}$$

- $\pi = \{\pi_i\}$—A initial state distribution where

$$\pi_i = P(s_0 = i) \quad 1 \le i \le N \tag{8.10}$$

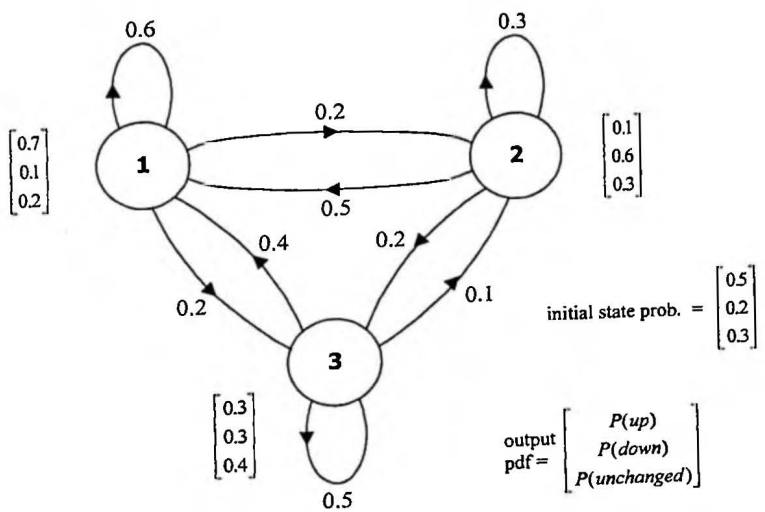


Figure 8.2 A hidden Markov model for the Dow Jones Industrial average. The three states no longer have deterministic meanings as in the Markov chain illustrated in Figure 8.1.

[2] The output distribution can also be transition-dependent. Although these two formulations look different, the state-dependent one can be reformulated as a transition-dependent one with the constraint of all the transitions entering the same state sharing the same output distribution.

Amazon/VB Assets
Exhibit 1012
Page 407

Since a_{ij}, $b_i(k)$, and π_i are all probabilities, they must satisfy the following properties:

$$a_{ij} \geq 0, \quad b_i(k) \geq 0, \quad \pi_i \geq 0 \quad \forall \text{ all } i, j, k \tag{8.11}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{8.12}$$

$$\sum_{k=1}^{M} b_i(k) = 1 \tag{8.13}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{8.14}$$

To sum up, a complete specification of an HMM includes two constant-size parameters, N and M, representing the total number of states and the size of observation alphabets, observation alphabet O, and three sets (matrices) of probability measures $\mathbf{A}$, $\mathbf{B}$, $\boldsymbol{\pi}$. For convenience, we use the following notation

$$\Phi = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}) \tag{8.15}$$

to indicate the whole parameter set of an HMM and sometimes use the parameter set Φ to represent the HMM interchangeably without ambiguity.

In the first-order hidden Markov model discussed above, there are two assumptions. The first is the *Markov assumption* for the Markov chain.

$$P(s_t | s_1^{t-1}) = P(s_t | s_{t-1}) \tag{8.16}$$

where s_1^{t-1} represents the state sequence $s_1, s_2, \ldots, s_{t-1}$. The second is the *output-independence assumption*:

$$P(X_t | X_1^{t-1}, s_1^t) = P(X_t | s_t) \tag{8.17}$$

where X_1^{t-1} represents the output sequence $X_1, X_2, \ldots, X_{t-1}$. The output-independence assumption states that the probability that a particular symbol is emitted at time t depends only on the state s_t and is conditionally independent of the past observations.

You might argue that these assumptions limit the memory of the first-order hidden Markov models and may lead to model deficiency. However, in practice, they make evaluation, decoding, and learning feasible and efficient without significantly affecting the modeling capability, since those assumptions greatly reduce the number of parameters that need to be estimated.

Amazon/VB Assets
Exhibit 1012
Page 408

Given the definition of HMMs above, three basic problems of interest must be addressed before they can be applied to real-world applications.

1. **The Evaluation Problem**—Given a model Φ and a sequence of observations $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, what is the probability $P(\mathbf{X}|\Phi)$, i.e., the probability that the model generates the observations?
2. **The Decoding Problem**—Given a model Φ and a sequence of observations $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, what is the most likely state sequence $\mathbf{S} = (s_0, s_1, s_2, \ldots, s_T)$ in the model that produces the observations?
3. **The Learning Problem**—Given a model Φ and a set of observations, how can we adjust the model parameter $\hat{\Phi}$ to maximize the joint probability (likelihood) $\prod_{\mathbf{X}} P(\mathbf{X}|\Phi)$?

If we could solve the *evaluation* problem, we would have a way of evaluating how well a given HMM matches a given observation sequence. Therefore, we could use HMM to do pattern recognition, since the likelihood $P(\mathbf{X}|\Phi)$ can be used to compute posterior probability $P(\Phi|\mathbf{X})$, and the HMM with highest posterior probability can be determined as the desired pattern for the observation sequence. If we could solve the decoding problem, we could find the best matching state sequence given an observation sequence, or, in other words, we could uncover the hidden state sequence. As discussed in Chapters 12 and 13, these are the basics for the decoding in continuous speech recognition. Last but not least, if we could solve the learning problem, we would have the means to automatically estimate the model parameter Φ from an ensemble of training data. These three problems are tightly linked under the same probabilistic framework. The efficient implementation of these algorithms shares the principle of dynamic programming that we briefly discuss next.

8.2.1. Dynamic Programming and DTW

The dynamic programming concept, also known as *dynamic time warping* (DTW) in speech recognition [40], has been widely used to derive the overall distortion between two speech templates. In these template-based systems, each speech template consists of a sequence of speech vectors. The overall distortion measure is computed from the accumulated distance between two feature vectors that are aligned between two speech templates with minimal overall distortion. The DTW method can warp two speech templates $(\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_N)$ and $(\mathbf{y}_1\mathbf{y}_2\ldots\mathbf{y}_M)$ in the time dimension to alleviate nonlinear distortion as illustrated in Figure 8.3.

This is roughly equivalent to the problem of finding the minimum distance in the trellis between these two templates. Associated with every pair (i, j) is a distance $d(i, j)$ between two speech vectors $\mathbf{x}_i$ and $\mathbf{y}_j$. To find the optimal path between starting point (1, 1) and end point (N, M) from left to right, we need to compute the optimal accumulated distance $D(N, M)$. We can enumerate all possible accumulated distance from (1,1) to (N, M) and identify the one that has the minimum distance. Since there are M possible moves for each step from left to right in Figure 8.3, all the possible paths from (1, 1) to (N, M) will be

Amazon/VB Assets
Exhibit 1012
Page 409

exponential. Dynamic programming principles can drastically reduce the amount of computation by avoiding the enumeration of sequences that cannot possibly be optimal. Since the same optimal path after each step must be based on the previous step, the minimum distance $D(i, j)$ must satisfy the following equation:

$$D(i,j) = \min_{k}\left[D(i-1,k) + d(k,j)\right] \tag{8.18}$$

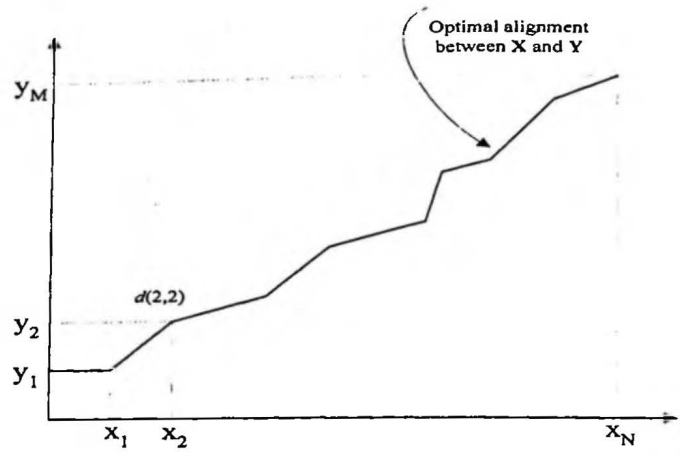


Figure 8.3 Direct comparison between two speech templates $\mathbf{X}=\mathbf{x}_1\mathbf{x}_2...\mathbf{x}_N$ and $\mathbf{Y}=\mathbf{y}_1\mathbf{y}_2...\mathbf{y}_M$.

Equation (8.18) indicates you only need to consider and keep only the best move for each pair although there are M possible moves. The recursion allows the optimal path search to be conducted incrementally from left to right. In essence, dynamic programming delegates the solution recursively to its own sub-problem. The computation proceeds from the small sub-problem ($D(i-1,k)$) to the larger sub-problem ($D(i,j)$). We can identify the optimal match $\mathbf{y}_j$ with respect to $\mathbf{x}_i$ and save the index in a back pointer table $B(i, j)$ as we move forward. The optimal path can be traced back after the optimal path is identified. The algorithm is described in Algorithm 8.1.

The advantage of the dynamic programming lies in the fact that once a sub-problem is solved, the partial result can be stored and never needs to be recalculated. This is a very important principle that you see again and again in building practical spoken language systems.

Speech recognition based on DTW is simple to implement and fairly effective for small-vocabulary speech recognition. Dynamic programming can temporally align patterns to account for differences in speaking rates across talkers as well as across repetitions of the word by the same talker. However, it does not have a principled way to derive an averaged template for each pattern from a large amount of training samples. A multiplicity of reference training tokens is typically required to characterize the variation among different utterances. As such, the HMM is a much better alternative for spoken language processing.

Amazon/VB Assets
Exhibit 1012
Page 410

ALGORITHM 8.1: *THE DYNAMIC PROGRAMMING ALGORITHM*

Step 1: Initialization

$D(1,1) = d(1,1), B(1,1) = 1$, for $j = 2,\ldots,M$ compute $D(1,j) = \infty$

Step 2: Iteration

for $i = 2,\ldots,N$ {

for $j = 1,\ldots,M$ compute {

$$D(i,j) = \min_{1 \le p \le M} [D(i-1,p) + d(p,j)]$$

$$B(i,j) = \arg\min_{1 \le p \le M} [D(i-1,p) + d(p,j)] \quad \}\}$$

Step 3: Backtracking and Termination

The optimal (minimum) distance is $D(N,M)$ and the optimal path is $(s_1, s_2, \ldots, s_N)$ where $s_N = M$ and $s_i = B(i+1, s_{i+1})$, $i = N-1, N-2, \ldots, 1$

8.2.2. How to Evaluate an HMM—The Forward Algorithm

To calculate the probability (likelihood) $P(\mathbf{X}|\Phi)$ of the observation sequence $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, given the HMM Φ, the most intuitive way is to sum up the probabilities of all possible state sequences:

$$P(\mathbf{X}|\Phi) = \sum_{\text{all } \mathbf{S}} P(\mathbf{S}|\Phi) P(\mathbf{X}|\mathbf{S},\Phi) \tag{8.19}$$

In other words, to compute $P(\mathbf{X}|\Phi)$, we first enumerate all possible state sequences $\mathbf{S}$ of length T, that generate observation sequence $\mathbf{X}$, and then sum all the probabilities. The probability of each path $\mathbf{S}$ is the product of the state sequence probability (first factor) and the joint output probability (the second factor) along the path.

For one particular state sequence $\mathbf{S} = (s_1, s_2, \ldots, s_T)$, where s_1 is the initial state, the state-sequence probability in Eq. (8.19) can be rewritten by applying Markov assumption:

$$P(\mathbf{S}|\Phi) = P(s_1|\Phi) \prod_{t=2}^{T} P(s_t|s_{t-1},\Phi) = \pi_{s_1} a_{s_1 s_2} \ldots a_{s_{T-1} s_T} = a_{s_0 s_1} a_{s_1 s_2} \ldots a_{s_{t-1} s_T} \tag{8.20}$$

where $a_{s_0 s_1}$ denotes π_{s_1} for simplicity. For the same state sequence $\mathbf{S}$, the joint output probability along the path can be rewritten by applying the output-independent assumption:

$$\begin{aligned} P(\mathbf{X}|\mathbf{S},\Phi) &= P(X_1^T | S_1^T, \Phi) = \prod_{t=1}^{T} P(X_t | s_t, \Phi) \\ &= b_{s_1}(X_1) b_{s_2}(X_2) \ldots b_{s_T}(X_T) \end{aligned} \tag{8.21}$$

Amazon/VB Assets
Exhibit 1012
Page 411

Substituting Eqs. (8.20) and (8.21) into (8.19), we get:

$$P(\mathbf{X}|\Phi) = \sum_{\text{all S}} P(\mathbf{S}|\Phi)P(\mathbf{X}|\mathbf{S},\Phi)$$
$$= \sum_{\text{all S}} a_{s_0 s_1} b_{s_1}(X_1) a_{s_1 s_2} b_{s_2}(X_2) \ldots a_{s_{T-1} s_T} b_{s_T}(X_T) \tag{8.22}$$

Equation (8.22) can be interpreted as follows. First we enumerate all possible state sequence with length T. For any given state sequence, we start from initial state s_1 with probability π_{s_1} or $a_{s_0 s_1}$. We take a transition from s_{t-1} to s_t with probability $a_{s_{t-1} s_t}$ and generate the observation X_t with probability $b_{s_t}(X_t)$ until we reach the last transition.

However, direct evaluation of Eq. (8.22) according to the interpretation above requires enumeration of $O(N^T)$ possible state sequences, which results in exponential computational complexity. Fortunately, a more efficient algorithm can be used to calculate Eq. (8.22). The trick is to store intermediate results and use them for subsequent state-sequence calculations to save computation. This algorithm is known as the *forward algorithm.*

Based on the HMM assumptions, the calculation of $P(s_t | s_{t-1}, \Phi)P(X_t | s_t, \Phi)$ involves only s_{t-1}, s_t, and X_t, so, it is possible to compute the likelihood with $P(\mathbf{X}|\Phi)$ with recursion on t. Let's define forward probability:

$$\alpha_t(i) = P(X_1^t, s_t = i|\Phi) \tag{8.23}$$

$\alpha_t(i)$ is the probability that the HMM is in state i at time t having generated partial observation X_1^t (namely $X_1 X_2 \ldots X_t$). $\alpha_t(i)$ can be calculated inductively as illustrated in Algorithm 8.2. This inductive procedure shown in Eq. (8.24) can be illustrated in a *trellis.* Figure 8.4 illustrates the computation of forward probabilities α via a trellis framework for the Dow Jones Industrial average HMM shown in Figure 8.2. Given two consecutive *up* days for the Dow Jones Industrial average, we can find the forward probability α based on the model of Figure 8.2. An arrow in Figure 8.4 indicates a transition from its origin state to its destination state. The number inside each cell indicates the forward probability α. We start the α cells from $t = 0$, where the α cells contains exactly the initial probabilities. The other cells are computed in a *time-synchronous* fashion from left to right, where each cell for time t is completely computed before proceeding to time $t+1$. When the states in the last column have been computed, the sum of all probabilities in the final column is the probability of generating the observation sequence. For most speech problems, we need to have the HMM end in some particular exit state (a.k.a final state, S_F), and we thus have $P(\mathbf{X}|\Phi) = \alpha_T(s_F)$.

It is easy to show that the complexity for the forward algorithm is $O(N^2 T)$ rather than exponential. This is because we can make full use of partially computed probabilities for the improved efficiency.

Amazon/VB Assets
Exhibit 1012
Page 412

ALGORITHM 8.2: *THE FORWARD ALGORITHM*

Step 1: Initialization

$$\alpha_1(i) = \pi_i b_i(X_1) \qquad 1 \le i \le N$$

Step 2: Induction

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] b_j(X_t) \qquad 2 \le t \le T; \quad 1 \le j \le N \tag{8.24}$$

Step 3: Termination

$$P(\mathbf{X}|\Phi) = \sum_{i=1}^{N} \alpha_T(i)$$

If it is required to end in the final state, $P(\mathbf{X}|\Phi) = \alpha_T(s_F)$

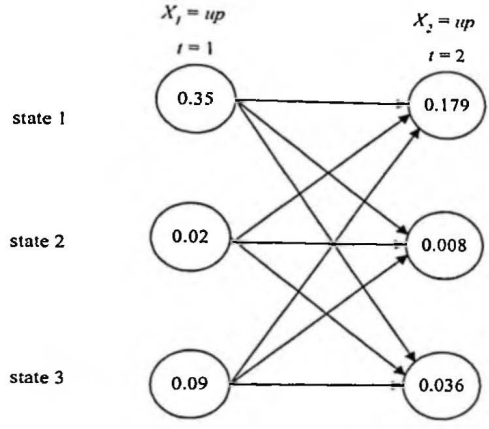


Figure 8.4 The forward trellis computation for the HMM of the Dow Jones Industrial average.

8.2.3. How to Decode an HMM—The Viterbi Algorithm

The forward algorithm, in the previous section, computes the probability that an HMM generates an observation sequence by summing up the probabilities of all possible paths, so it does not provide the best path (or state sequence). In many applications, it is desirable to find such a path. As a matter of fact, finding the best path (state sequence) is the cornerstone for searching in continuous speech recognition. Since the state sequence is hidden (unobserved) in the HMM framework, the most widely used criterion is to find the state sequence that has the highest probability of being taken while generating the observation sequence. In other words, we are looking for the state sequence $\mathbf{S} = (s_1, s_2, \ldots, s_T)$ that maximizes $P(\mathbf{S}, \mathbf{X}|\Phi)$. This problem is very similar to the optimal-path problem in dynamic programming. As a consequence, a formal technique based on dynamic programming, known

Amazon/VB Assets
Exhibit 1012
Page 413

as *Viterbi* algorithm [43], can be used to find the best state sequence for an HMM. In practice, the same method can be used to evaluate HMMs that offers an approximate solution close to the case obtained using the forward algorithm described in Section 8.2.2.

The Viterbi algorithm can be regarded as the dynamic programming algorithm applied to the HMM or as a modified forward algorithm. Instead of summing up probabilities from different paths coming to the same destination state, the Viterbi algorithm picks and remembers the best path. To define the best-path probability:

$$V_t(i) = P(X_1^t, S_1^{t-1}, s_t = i \mid \Phi)$$

$V_t(i)$ is the probability of the most likely state sequence at time t, which has generated the observation X_1^t (until time t) and ends in state i. A similar induction procedure for the Viterbi algorithm can be described in Algorithm 8.3.

ALGORITHM 8.3: *THE VITERBI ALGORITHM*

Step 1: Initialization

$V_1(i) = \pi_i b_i(X_1) \qquad 1 \le i \le N$

$B_1(i) = 0$

Step 2: Induction

$$V_t(j) = \underset{1 \le i \le N}{Max}\left[V_{t-1}(i)a_{ij}\right] b_j(X_t) \quad 2 \le t \le T; \quad 1 \le j \le N \tag{8.25}$$

$$B_t(j) = \underset{1 \le i \le N}{Arg\max}\left[V_{t-1}(i)a_{ij}\right] \quad 2 \le t \le T; \quad 1 \le j \le N \tag{8.26}$$

Step 3: Termination

The best score $= \underset{1 \le i \le N}{Max}\left[V_t(i)\right]$

$s_T^* = \underset{1 \le i \le N}{Arg\max}\left[B_T(i)\right]$

Step 4: Backtracking

$s_t^* = B_{t+1}(s_{t+1}^*) \quad t = T-1, T-2, \ldots, 1$

$S^* = (s_1^*, s_2^*, \ldots, s_T^*)$ is the best sequence

This Viterbi algorithm can also be illustrated in a trellis framework similar to the one for the forward algorithm shown in Figure 8.4. Instead of summing up all the paths, Figure 8.5 illustrates the computation of V_t by picking the best path in each cell. The number inside each cell indicates the best score V_t and the best path leading to each cell is indicated by solid lines while the rest of the paths are indicated by dashed line. Again, the computation is done in a *time-synchronous* fashion from left to right. The complexity for the Viterbi algorithm is also $O(N^2T)$.

Amazon/VB Assets
Exhibit 1012
Page 414

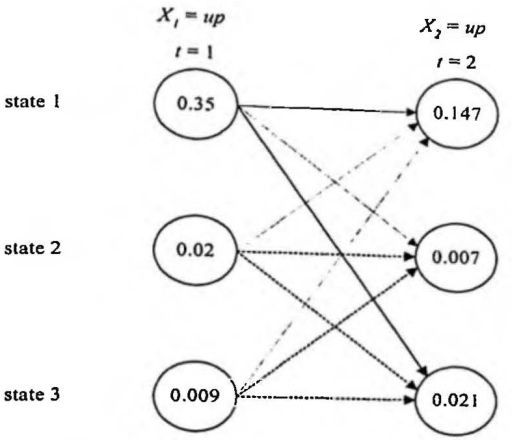


Figure 8.5 The Viterbi trellis computation for the HMM of the Dow Jones Industrial average.

8.2.4. How to Estimate HMM Parameters—Baum-Welch Algorithm

It is very important to estimate the model parameters $\Phi = (\mathbf{A}, \mathbf{B}, \pi)$ to accurately describe the observation sequences. This is by far the most difficult of the three problems, because there is no known analytical method that maximizes the joint probability of the training data in a closed form. Instead, the problem can be solved by the iterative *Baum-Welch* algorithm, also known as the *forward-backward* algorithm.

The HMM learning problem is a typical case of unsupervised learning discussed in Chapter 4, where the data is incomplete because of the hidden state sequence. The EM algorithm is perfectly suitable for this problem. As a matter of fact, Baum and colleagues used the same principle as that of the EM algorithm. Before we describe the formal Baum-Welch algorithm, we first define a few useful terms. In a manner similar to the forward probability, we define backward probability as:

$$\beta_t(i) = P(X_{t+1}^T | s_t = i, \Phi) \tag{8.27}$$

where $\beta_t(i)$ is the probability of generating partial observation X_{t+1}^T (from t+1 to the end) given that the HMM is in state i at time t, $\beta_t(i)$ can then be calculated inductively;

Initialization:

$$\beta_T(i) = 1/N \qquad 1 \le i \le N$$

Amazon/VB Assets
Exhibit 1012
Page 415

Induction:

$$\beta_t(i)=\left[\sum_{j=1}^{N} a_{ij}b_j(X_{t+1})\beta_{t+1}(j)\right] \quad t=T-1\ldots1;\quad 1\le i\le N \tag{8.28}$$

The relationship of adjacent α and β (α_{t-1} & α_t and β_t & β_{t+1}) can be best illustrated as shown in Figure 8.6. α is computed recursively from left to right, and β recursively from right to left.

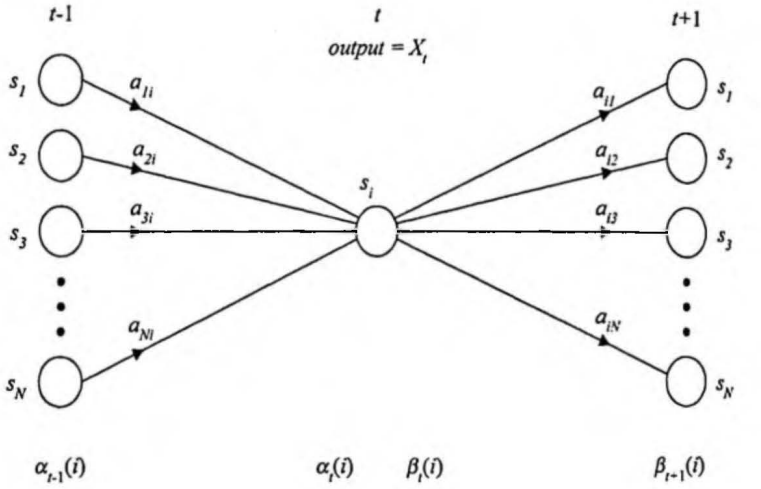


Figure 8.6 The relationship of α_{t-1} and α_t and β_t and β_{t+1} in the forward-backward algorithm.

Next, we define $\gamma_t(i,j)$, which is the probability of taking the transition from state i to state j at time t, given the model and observation sequence, i.e.,

$$\begin{aligned}\gamma_t(i,j)&=P(s_{t-1}=i,s_t=j\mid X_1^T,\Phi)\\&=\frac{P(s_{t-1}=i,s_t=j,X_1^T\mid\Phi)}{P(X_1^T\mid\Phi)}\\&=\frac{\alpha_{t-1}(i)a_{ij}b_j(X_t)\beta_t(j)}{\sum_{k=1}^{N}\alpha_T(k)}\end{aligned} \tag{8.29}$$

The equation above can be best illustrated as shown in Figure 8.7.

We can iteratively refine the HMM parameter vector $\Phi=(\mathbf{A},\mathbf{B},\boldsymbol{\pi})$ by maximizing the likelihood $P(\mathbf{X}|\Phi)$ for each iteration. We use $\hat{\Phi}$ to denote the new parameter vector derived from the parameter vector Φ in the previous iteration. According to the EM algorithm

Amazon/VB Assets
Exhibit 1012
Page 416

of Chapter 4, the maximization process is equivalent to maximizing the following Q-function:

$$Q(\Phi,\hat{\Phi}) = \sum_{\text{all S}} \frac{P(\mathbf{X},\mathbf{S}|\Phi)}{P(\mathbf{X}|\Phi)} \log P(\mathbf{X},\mathbf{S}|\hat{\Phi}) \tag{8.30}$$

where $P(\mathbf{X},\mathbf{S}|\Phi)$ and $\log P(\mathbf{X},\mathbf{S}|\hat{\Phi})$ can be expressed as:

$$P(\mathbf{X},\mathbf{S}|\Phi) = \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t}(X_t) \tag{8.31}$$

$$\log P(\mathbf{X},\mathbf{S}|\Phi) = \sum_{t=1}^{T} \log a_{s_{t-1}s_t} + \sum_{t=1}^{T} \log b_{s_t}(X_t) \tag{8.32}$$

Equation (8.30) can thus be rewritten as

$$Q(\Phi,\hat{\Phi}) = Q_{\mathbf{a}_i}(\Phi,\hat{\mathbf{a}}_i) + Q_{\mathbf{b}_j}(\Phi,\hat{\mathbf{b}}_j) \tag{8.33}$$

where

$$Q_{\mathbf{a}_i}(\Phi,\hat{\mathbf{a}}_i) = \sum_i \sum_j \sum_t \frac{P(\mathbf{X}, s_{t-1}=i, s_t=j|\Phi)}{P(\mathbf{X}|\Phi)} \log \hat{a}_{ij} \tag{8.34}$$

$$Q_{\mathbf{b}_j}(\Phi,\hat{\mathbf{b}}_j) = \sum_j \sum_k \sum_{t\in X_t=o_k} \frac{P(\mathbf{X}, s_t=j|\Phi)}{P(\mathbf{X}|\Phi)} \log \hat{b}_j(k) \tag{8.35}$$

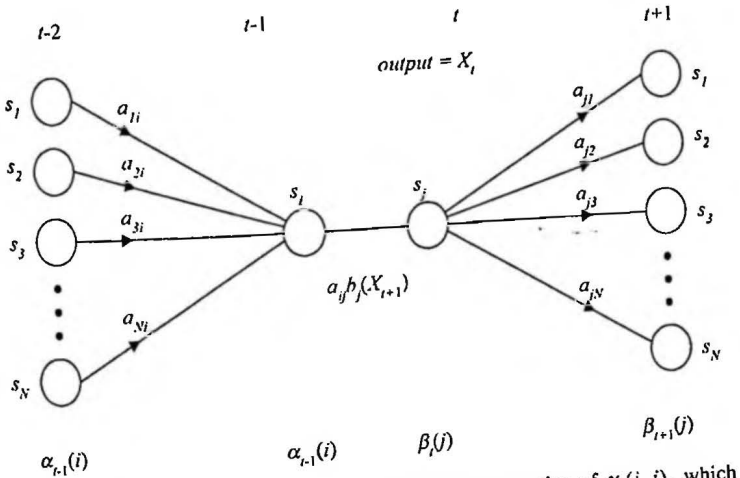


Figure 8.7 Illustration of the operations required for the computation of $\gamma_t(i,j)$, which is the probability of taking the transition from state i to state j at time t.

Amazon/VB Assets
Exhibit 1012
Page 417

Since we separate the Q-function into two independent terms, the maximization procedure on $Q(\Phi,\hat{\Phi})$ can be done by maximizing the individual terms separately, subject to probability constraints.

$$\sum_{j=1}^{N} a_{ij} = 1 \quad \forall \text{ all } i \tag{8.36}$$

$$\sum_{k=1}^{M} b_j(k) = 1 \quad \forall \text{ all } j \tag{8.37}$$

Moreover, all these terms in Eqs. (8.34) and (8.35) have the following form:

$$F(x) = \sum_i y_i \log x_i \tag{8.38}$$

where $\sum_i x_i = 1$

By using the Lagrange multipliers, the function above can be proved to achieve maximum value at

$$x_i = \frac{y_i}{\sum_i y_i} \tag{8.39}$$

Using this formation, we obtain the model estimate as[3]:

$$\hat{a}_{ij} = \frac{\frac{1}{P(\mathbf{X}|\Phi)}\sum_{t=1}^{T} P(\mathbf{X}, s_{t-1}=i, s_t=j|\Phi)}{\frac{1}{P(\mathbf{X}|\Phi)}\sum_{t=1}^{T} P(\mathbf{X}, s_{t-1}=i|\Phi)} = \frac{\sum_{t=1}^{T}\gamma_t(i,j)}{\sum_{t=1}^{T}\sum_{k=1}^{N}\gamma_t(i,k)} \tag{8.40}$$

$$\hat{b}_j(k) = \frac{\frac{1}{P(\mathbf{X}|\Phi)}\sum_{t=1}^{T} P(\mathbf{X}, s_t=j|\Phi)\delta(X_t,o_k)}{\frac{1}{P(\mathbf{X}|\Phi)}\sum_{t=1}^{T} P(\mathbf{X}, s_t=j|\Phi)} = \frac{\sum_{t\in X_t=o_k}\sum_i \gamma_t(i,j)}{\sum_{t=1}^{T}\sum_i \gamma_t(i,j)} \tag{8.41}$$

By carefully examining the HMM re-estimation Eqs. (8.40) and (8.41), you can see that Eq. (8.40) is essentially the ratio between the expected number of transitions from state i to state j and the expected number of transitions from state i. For the output probability re-estimation Eq. (8.41), the numerator is the expected number of times the observation data

[3] Notice that the initial probability $\hat{\pi}_i$ can be derived as a special case of the transition probability. $\hat{\pi}_i$ is often fixed (i.e., $\hat{\pi}_i = 1$ for the initial state) for most speech applications.

Amazon/VB Assets
Exhibit 1012
Page 418

emitted from state j is the observation symbol o_k, and the denominator is the expected number of times the observation data is emitted from state j.

According to the EM algorithm, the forward-backward (Baum-Welch) algorithm guarantees a monotonic likelihood improvement on each iteration, and eventually the likelihood converges to a local maximum. The forward-backward algorithm can be described in a way similar to the general EM algorithm as shown in Algorithm 8.4.

ALGORITHM 8.4: *THE FORWARD-BACKWARD ALGORITHM*

Step 1: Initialization: Choose an initial estimate Φ.

Step 2: E-step: Compute auxiliary function $Q(\Phi, \hat{\Phi})$ based on Φ.

Step 3: M-step: Compute $\hat{\Phi}$ according to the re-estimation Eqs. (8.40) and (8.41) to maximize the auxiliary Q-function.

Step 4: Iteration: Set $\Phi = \hat{\Phi}$, repeat from step 2 until convergence.

Although the forward-backward algorithm described above is based on one training observation sequence, it can be easily generalized to multiple training observation sequences under the independence assumption between these sequences. To train an HMM from M data sequences is equivalent to finding the HMM parameter vector Φ that maximizes the joint likelihood:

$$\prod_{i=1}^{M} P(\mathbf{X}_i \mid \Phi) \tag{8.42}$$

The training procedure performs the forward-backward algorithm on each independent observation sequence to calculate the expectations (or sometimes referred to as counts) in Eqs. (8.40) and (8.41). These counts in the denominator and numerator, respectively, can be added across M data sequences respectively. Finally, all the model parameters (probabilities) are normalized to make them sum up to one. This constitutes one iteration of Baum-Welch re-estimation; iteration continues until convergence. This procedure is practical and useful because it allows you to train a good HMM in a typical speech recognition scenario where a large amount of training data is available.

For example, if we let $\gamma_t^m(i,j)$ denote the $\gamma_t(i,j)$ from the m^{th} data sequence and T^m denote the corresponding length, Eq. (8.40) can be extended as:

$$\hat{a}_{ij} = \frac{\sum_{m=1}^{M}\sum_{t=1}^{T^m}\gamma_t^m(i,j)}{\sum_{m=1}^{M}\sum_{t=1}^{T^m}\sum_{k=1}^{N}\gamma_t^m(i,k)} \tag{8.43}$$

Amazon/VB Assets
Exhibit 1012
Page 419

8.3. CONTINUOUS AND SEMICONTINUOUS HMMS

If the observation does not come from a finite set, but from a continuous space, the discrete output distribution discussed in the previous sections needs to be modified. The difference between the discrete and the continuous HMM lies in a different form of output probability functions. For speech recognition, use of continuous HMMs implies that the quantization procedure to map observation vectors from the continuous space to the discrete space for the discrete HMM is no longer necessary. Thus, the inherent quantization error can be eliminated.

8.3.1. Continuous Mixture Density HMMs

In choosing continuous output probability density functions $b_j(\mathbf{x})$, the first candidate is multivariate Gaussian mixture density functions. This is because they can approximate any continuous probability density function, as discussed in Chapter 3. With M Gaussian mixture density functions, we have:

$$b_j(\mathbf{x}) = \sum_{k=1}^{M} c_{jk} N(\mathbf{x}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}) = \sum_{k=1}^{M} c_{jk} b_{jk}(\mathbf{x}) \tag{8.44}$$

where $N(\mathbf{x}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})$ or $b_{jk}(\mathbf{x})$ denotes a single Gaussian density function with mean vector $\boldsymbol{\mu}_{jk}$ and covariance matrix $\boldsymbol{\Sigma}_{jk}$ for state j, M denotes the number of mixture-components, and c_{jk} is the weight for the k^{th} mixture component satisfying

$$\sum_{k=1}^{M} c_{jk} = 1 \tag{8.45}$$

To take the same *divide and conquer* approach as Eq. (8.33), we need to express $b_j(\mathbf{x})$ with respect to each single mixture component as:

$$p(\mathbf{X}, \mathbf{S} \mid \Phi) = \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t}(\mathbf{x}_t) = \sum_{k_1=1}^{M} \sum_{k_2=1}^{M} \cdots \sum_{k_T=1}^{M} \{ \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t k_t}(\mathbf{x}_{s_t}) c_{s_t k_t} \} \tag{8.46}$$

Equation (8.46) can be considered as the summation of densities with all the possible state sequences S and all the possible mixture components K, defined in Ω^T as the T^{th} Cartesian product of $\Omega = \{1, 2, ..., M\}$, as follows:

$$p(\mathbf{X}, \mathbf{S}, \mathbf{K} \mid \Phi) = \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t k_t}(\mathbf{x}_t) c_{s_t k_t} \tag{8.47}$$

Therefore, the joint probability density is

Amazon/VB Assets
Exhibit 1012
Page 420

$$p(\mathbf{X}|\Phi)=\sum_{\mathbf{S}}\sum_{\mathbf{K}\in\Omega^T}p(\mathbf{X},\mathbf{S},\mathbf{K}|\Phi) \tag{8.48}$$

An auxiliary function $Q(\Phi,\hat{\Phi})$ of two model points, Φ and $\hat{\Phi}$, given an observation $\mathbf{X}$, can be written as:

$$Q(\Phi,\hat{\Phi})=\sum_{\mathbf{S}}\sum_{\mathbf{K}\in\Omega^T}\frac{p(\mathbf{X},\mathbf{S},\mathbf{K}|\Phi)}{p(\mathbf{X}|\Phi)}\log p(\mathbf{X},\mathbf{S},\mathbf{K}|\hat{\Phi}) \tag{8.49}$$

From (8.47), the following decomposition can be shown:

$$\begin{aligned}&\log p(\mathbf{X},\mathbf{S},\mathbf{K}|\hat{\Phi})\\&=\sum_{t=1}^{T}\log\hat{a}_{s_{t-1}s_t}+\sum_{t=1}^{T}\log\hat{b}_{s_t k_t}(\mathbf{x}_t)+\sum_{t=1}^{T}\log\hat{c}_{s_t k_t}\end{aligned} \tag{8.50}$$

Maximization of the likelihood by way of re-estimation can be accomplished on individual parameter sets owing to the separability shown in (8.50). The separation of (8.50) is the key to the increased versatility of a re-estimation algorithm in accommodating mixture observation densities. The auxiliary function can be rewritten in a separated form in a similar manner as Eq. (8.33):

$$\begin{aligned}Q(\Phi,\hat{\Phi})&=\sum_{\mathbf{S}}\sum_{\mathbf{K}}\frac{p(\mathbf{X},\mathbf{S},\mathbf{K}|\Phi)}{p(\mathbf{X}|\Phi)}\log p(\mathbf{X},\mathbf{S},\mathbf{K}|\hat{\Phi})\\&=\sum_{i=1}^{N}Q_{\mathbf{a}_i}(\Phi,\hat{\mathbf{a}}_i)+\sum_{j=1}^{N}\sum_{k=1}^{M}Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})+\sum_{j=1}^{N}\sum_{k=1}^{M}Q_{\mathbf{c}_{jk}}(\Phi,\hat{\mathbf{c}}_{jk})\end{aligned} \tag{8.51}$$

The only difference we have is:

$$Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})=\sum_{t=1}^{T}p(s_t=j,k_t=k|\mathbf{X},\Phi)\log\hat{b}_{jk}(\mathbf{x}_t), \tag{8.52}$$

and

$$Q_{\mathbf{c}_{jk}}(\Phi,\hat{\mathbf{c}}_{jk})=\sum_{t=1}^{T}p(s_t=j,k_t=k|\mathbf{X},\Phi)\log\hat{c}_{jk} \tag{8.53}$$

The optimization procedure is similar to what is discussed in the discrete HMM. The only major difference is $Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})$. Maximization of $Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})$ with respect to $\hat{\mathbf{b}}_{jk}$ is obtained through differentiation with respect to $\{\pi_{jk},\Sigma_{jk}\}$ that satisfies:

$$\nabla_{\hat{\mathbf{b}}_{jk}}Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})=0 \tag{8.54}$$

Amazon/VB Assets
Exhibit 1012
Page 421

The solutions are:

$$\hat{\boldsymbol{\mu}}_{jk} = \frac{\sum_{t=1}^{T} \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)} \mathbf{x}_t}{\sum_{t=1}^{T} \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)}} = \frac{\sum_{t=1}^{T} \zeta_t(j,k) \mathbf{x}_t}{\sum_{t=1}^{T} \zeta_t(j,k)} \tag{8.55}$$

$$\begin{aligned} \hat{\boldsymbol{\Sigma}}_{jk} &= \frac{\sum_{t=1}^{T} \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)} (\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jk})(\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jk})'}{\sum_{t=1}^{T} \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)}} \\ &= \frac{\sum_{t=1}^{T} \zeta_t(j,k)(\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jk})(\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jk})'}{\sum_{t=1}^{T} \zeta_t(j,k)} \end{aligned} \tag{8.56}$$

where $\zeta_t(j,k)$ is computed as:

$$\zeta_t(j,k) = \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)} = \frac{\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} c_{jk} b_{jk}(\mathbf{x}_t) \beta_t(j)}{\sum_{i=1}^{N} \alpha_T(i)} \tag{8.57}$$

In a similar manner to the discrete HMM, we can derive the reestimation equation for c_{jk} as follows:

$$\hat{c}_{jk} = \frac{\sum_{t=1}^{T} \zeta_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \zeta_t(j,k)} \tag{8.58}$$

8.3.2. Semicontinuous HMMs

Traditionally, the discrete and the continuous mixture density HMMs have been treated separately. In fact, the gap between them can be bridged under some minor assumptions with the so-called semicontinuous or tied-mixture HMM. It assumes the mixture density functions are tied together across all the models to form a set of shared kernels. In the discrete HMM, a VQ codebook is typically used to map the continuous input feature vector $\mathbf{x}$ to o_k, so we can use the discrete output probability distribution $b_j(k)$. The codebook can

Amazon/VB Assets
Exhibit 1012
Page 422

be essentially regarded as one of such shared kernels. Accordingly, Eq. (8.44) can be modified as:

$$b_j(\mathbf{x}) = \sum_{k=1}^{M} b_j(k) f(\mathbf{x} \mid o_k) = \sum_{k=1}^{M} b_j(k) N(\mathbf{x}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{8.59}$$

where o_k is the kth codeword, $b_j(k)$ is the same output probability distribution in the discrete HMM or the mixture weights for the continuous mixture density function, and $N(\mathbf{x}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ are assumed to be independent of the Markov model and they are shared across all the Markov models with a very large number of mixtures M.

From the discrete HMM point of view, the needed VQ codebook consists of M continuous probability density functions, and each *codeword* has a mean vector and a covariance matrix. Typical quantization produces a codeword index that has minimum distortion to the given continuous observation $\mathbf{x}$. In the semicontinuous HMM, the quantization operation produces values of continuous probability density functions $f(\mathbf{x} \mid o_k)$ for all the codewords o_k. The structure of the semicontinuous model can be roughly the same as that of the discrete one. However, the output probabilities are no longer used directly as in the discrete HMM. In contrast, the VQ codebook density functions, $N(\mathbf{x}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, are combined with the discrete output probability as in Eq. (8.59). This is a combination of *discrete* model-dependent weighting coefficients with the *continuous* codebook probability density functions. Such a representation can be used to re-estimate the original VQ codebook together with the HMM.

The semicontinuous model also resembles the M-mixture continuous HMM with all the continuous output probability density functions shared among all Markov states. Compared with the continuous mixture HMM, the semicontinuous HMM can maintain the modeling ability of large-mixture probability density functions. In addition, the number of free parameters and the computational complexity can be reduced, because all the probability density functions are tied together, thus providing a good compromise between detailed acoustic modeling and trainability.

In practice, because M is large, Eq. (8.59) can be simplified by using the L most significant values $f(\mathbf{x} \mid o_k)$ for each $\mathbf{x}$ without affecting the performance. Experience has shown that values of L in the range of 1-3% of M are adequate. This can be conveniently obtained during the VQ operations by sorting the VQ output and keeping the L most significant values. Let $\eta(\mathbf{x})$ denote the set of L VQ codewords that has the largest $f(\mathbf{x} \mid o_k)$ for the given $\mathbf{x}$. Then we have:

$$b_j(\mathbf{x}) \cong \sum_{k \in \eta(\mathbf{x})} f(\mathbf{x} \mid o_k) b_j(k) \tag{8.60}$$

Since the number of mixture components in $\eta(\mathbf{x})$ is of lower order than M, Eq. (8.60) can significantly reduce the amount of computation. In fact, $\eta(\mathbf{x})$ is the key to bridge the gap between the continuous and discrete HMM. If $\eta(\mathbf{x})$ contains only the most significant

Amazon/VB Assets
Exhibit 1012
Page 423

$f(\mathbf{x} \mid o_k)$ (i.e., only the closest codeword to $\mathbf{x}$), the semicontinuous HMM degenerates to the discrete HMM. On the other hand, a large VQ codebook can be used such that each Markov state could contain a number of its own codewords (a mixture of probability density functions). The discrete output probability $b_{ij}(k)$ thus becomes the mixture weights for each state. This would go to the other extreme, a standard continuous mixture density model. We can also define $\eta(\mathbf{x})$ in such a way that we can have partial tying of $f(\mathbf{x} \mid o_k)$ for different phonetic classes. For example, we can have a phone-dependent codebook.

When we have a tied VQ codebook, re-estimation of these mean vectors and covariance matrices of different models will involve interdependencies. If any observation $\mathbf{x}_t$ (no matter what model it is designated for) has a large value of posterior probability $\zeta_t(j,k)$, it will have a large contribution on re-estimation of parameters of codeword o_k. We can compute the posterior probability for each codeword from $\zeta_t(j,k)$ as defined in Eq. (8.57).

$$\zeta_t(k) = p(\mathbf{x}_t = o_k \mid \mathbf{X}, \Phi) = \sum_j \zeta_t(j,k) \tag{8.61}$$

The re-estimation formulas for the tied mixture can be written as:

$$\hat{\mu}_k = \frac{\sum_{t=1}^{T} \zeta_t(k)\mathbf{x}_t}{\sum_{t=1}^{T} \zeta_t(k)} \tag{8.62}$$

$$\hat{\Sigma}_k = \frac{\sum_{t=1}^{T} \zeta_t(k)(\mathbf{x}_t - \hat{\mu}_k)(\mathbf{x}_t - \hat{\mu}_k)^t}{\sum_{t=1}^{T} \zeta_t(k)} \tag{8.63}$$

8.4. PRACTICAL ISSUES IN USING HMMS

While the HMM provides a solid framework for speech modeling, there are a number of issues you need to understand to make effective use of spoken language processing. In this section we point out some of the key issues related to practical applications. For expedience, we mostly use the discrete HMM as our example here.

8.4.1. Initial Estimates

In theory, the re-estimation algorithm of the HMM should reach a local maximum for the likelihood function. A key question is how to choose the right initial estimates of the HMM parameters so that the local maximum becomes the global maximum.

Amazon/VB Assets
Exhibit 1012
Page 424

In the discrete HMM, if a probability is initialized to be zero, it will remain zero forever. Thus, it is important to have a reasonable set of initial estimates. Empirical study has shown that, for discrete HMMs, you can use a uniform distribution as the initial estimate. It works reasonably well for most speech applications, though good initial estimates are always helpful to compute the output probabilities.

If continuous mixture density HMMs are used, good initial estimates for the Gaussian density functions are essential. There are a number of ways to obtain such initial estimates:

- You can use the *k*-means clustering procedure, as used in vector quantization clustering. The Markov state segmentation can be derived from the discrete HMM, since it is not very sensitive to the initial parameters. Based on the segmented data, you can use the *k*-means algorithm to derive needed Gaussian mean and covariance parameters. The mixture coefficients can be based on the uniform distribution.
- You can estimate the semicontinuous HMM from the discrete HMM. You simply need to estimate an additional covariance matrix for each VQ codeword and run an additional four or five iterations to refine the semicontinuous HMM based on the discrete HMM, which typically requires four or five iterations from the uniform distribution. When the semi-continuous HMM is trained, you take the top *M codewords,* and in each Markov state use them as the initial Gaussian density functions for the continuous density mixture model.
- You can start training a single mixture Gaussian model. You can compute the parameters from previously segmented data. You can then iteratively split the Gaussian density function in a way similar to VQ codebook generation. You typically need two or three iterations to refine the continuous density after each splitting.

8.4.2. Model Topology

Speech is a time-evolving nonstationary signal. Each HMM state has the ability to capture some quasi-stationary segment in the non-stationary speech signal. A left-to-right topology, as illustrated in Figure 8.8, is a natural candidate to model the speech signal. It has a self-transition to each state that can be used to model contiguous speech features belonging to the same state. When the quasi-stationary speech segment evolves, the left-to-right transition enables a natural progression of such evolution. In such a topology, each state has a state-dependent output probability distribution that can be used to interpret the observable speech signal. This topology is, in fact, one of the most popular HMM structures used in state-of-the-art speech recognition systems. The output probability distribution can be either discrete distributions or a mixture of continuous density functions.

Amazon/VB Assets
Exhibit 1012
Page 425

For the left-to-right HMM, the most important parameter in determining the topology is the number of states. The choice of model topology depends on available training data and what the model is used for. If each HMM is used to represent a phone, you need to have at least three to five output distributions. If such a model is used to represent a word, more states are generally required, depending on the pronunciation and duration of the word. For example, the word *tetrahydrocannabino* should have a large number of states in comparison to the word *a*. You may use at least 24 states for the former and three states for the latter. If you have the number of states depending on the duration of the signal, you may want to use 15 to 25 states for each second of speech signal. One exception is that, for silence, you may want to have a simpler topology. This is because silence is stationary, and one or two states will be sufficient.

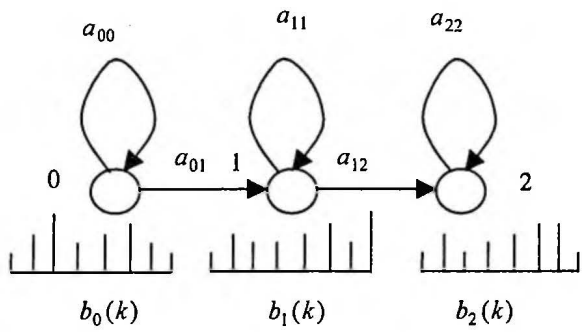


Figure 8.8 A typical hidden Markov model used to model phonemes. There are three states (0-2) and each state has an associated output probability distribution.

In practice, it is convenient to define a *null transition*. This is convenient if we want to simply traverse the HMM without consuming any observation symbol. To incorporate the null arc, you need to slightly modify the basic forward-backward or Viterbi probability equations, provided that no loops of empty transitions exist. If we denote the empty transition between state i and j as a_{ij}^{ε}, they need to satisfy the following constraints:

$$\sum_j a_{ij} + a_{ij}^{\varepsilon} = 1, \forall i \tag{8.64}$$

The forward probability can be augmented as follows:

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_i(\mathbf{x}_t) + \sum_{i=1}^{N} \alpha_t(i) a_{ij}^{\varepsilon} \right] \quad 1 \le t \le T; \quad 1 \le j \le N \tag{8.65}$$

Amazon/VB Assets
Exhibit 1012
Page 426

Equation (8.65) appears to have a recursion, but it actually uses the value of the same time column of $\alpha_t(i)$, provided that i is already computed, which is easily achievable if we have left-to-right empty transitions without loops of empty transitions.

8.4.3. Training Criteria

The argument for maximum likelihood estimation (MLE) is based on an assumption that the true distribution of speech is a member of the family of distributions used. This amounts to the assertion that the observed speech is genuinely produced by the HMM being used, and the only unknown parameters are the values. However, this can be challenged. Typical HMMs make many inaccurate assumptions about the speech production process, such as the output-independence assumption, the Markov assumption, and the continuous probability density assumption. Such inaccurate assumptions substantially weaken the rationale for maximum likelihood criteria. For instance, although maximum likelihood estimation is consistent (convergence to the true value), it is meaningless to have such a property if the wrong model is used. The true parameters in such cases will be the true parameters of the wrong models. Therefore, an estimation criterion that can work well in spite of these inaccurate assumptions should offer improved recognition accuracy compared with the maximum likelihood criterion. These alternative criteria include the MCE and MMIE, as discussed in Chapter 4. Finally, if we have prior knowledge about the model distribution, we can employ the Bayes' method such as MAP that can effectively combine both the prior and posterior distributions in a consistent way, which is particularly suitable for adaptation or dealing with insufficient training data.

Among all these criteria, MLE remains one of the most widely used, because of its simplicity and superior performance when appropriate assumptions are made about the system design. MCE and MMIE work well for small- to medium-vocabulary speech recognition [2, 26, 36]. You can train a number of other iterations based on the ML estimates. Neither MCE nor MMIE has been found extremely effective for large-vocabulary speech recognition. However, it is possible to combine the MMIE or MCE model with the MLE model for improved performance. This is because the error patterns generated from these different models are not the same. We can decode the test utterance with these different models and vote for the most consistent results [15, 25]. We discuss MAP methods in Chapter 9, since it is mostly helpful for speaker adaptive speech recognition.

8.4.4. Deleted Interpolation

For improved robustness, it is often necessary to combine well trained general models (such as speaker-independent) with those that are less well trained but more detailed (such as speaker-dependent). For example, you can typically improve speech recognition accuracy with speaker-dependent training. Nevertheless, you may not have sufficient data for a particular speaker so it is desirable to use a speaker-independent model that is more general but less accurate to smooth the speaker-dependent model. One effective way to achieve robust-

Amazon/VB Assets
Exhibit 1012
Page 427

ness is to interpolate both models with a technique called deleted interpolation, in which the interpolation weights are estimated using cross-validation data. The objective function is to maximize the probability of the model generating the held-out data.

Now, let us assume that we want to interpolate two sets of models, $P_A(\mathbf{x})$ and $P_B(\mathbf{x})$, which can be either discrete probability distributions or continuous density functions, to form an interpolated model $P_{DI}(\mathbf{x})$. The interpolation procedure can be expressed as follows:

$$P_{DI}(\mathbf{x}) = \lambda P_A(\mathbf{x}) + (1-\lambda) P_B(\mathbf{x}) \tag{8.66}$$

where the interpolation weight λ is what we need to derive from the training data.

Consider that we want to interpolate a speaker-independent model $P_A(\mathbf{x})$ with a speaker-dependent model $P_B(\mathbf{x})$. If we use speaker-independent data to estimate the interpolation weight, we may not capture needed speaker-specific information that should be reflected in the interpolation weights. What is worse is that the interpolation weight for the speaker-independent model should be equal to 1.0 if we use the same speaker-independent data from which the model was derived to estimate the interpolation weight. This is because of the MLE criterion. If we use speaker-dependent data instead, we have the weight for the speaker-dependent model equal 1.0 without achieving the desired smoothing effect. Thus the interpolation weights need to be trained using different data or *deleted* data with the so called cross-validation method.

We can have the training data normally divided into M parts, and train a set of $P_A(\mathbf{x})$ and $P_B(\mathbf{x})$ models using the standard EM algorithm from each combination of M-1 parts, with the deleted part serving as the unseen data to estimate the interpolation weights λ. These M sets of interpolation weights are then averaged to obtain the final weights.

ALGORITHM 8.5: *DELETED INTERPOLATION PROCEDURE*

Step 1: Initialize λ with a guessed estimate.

Step 2: Update λ by the following formula:

$$\hat{\lambda} = \frac{1}{M} \sum_{j=1}^{M} \sum_{t=1}^{n_j} \frac{\lambda P_{A-j}(\mathbf{x}_t^j)}{\lambda P_{A-j}(\mathbf{x}_t^j) + (1-\lambda) P_{B-j}(\mathbf{x}_t^j)} \tag{8.67}$$

where $P_{A-j}(\mathbf{x}_t^j)$ and $P_{B-j}(\mathbf{x}_t^j)$ is $P_A(\mathbf{x})$ and $P_B(\mathbf{x})$ estimated by the entire training corpus except part j, the deleted part, respectively; n_j is the total number of data points in part j that have been aligned to estimate the model; and $\mathbf{x}_t^j$ indicates the t-th data point in the j-th set of the aligned data.

Step 3: If the new value $\hat{\lambda}$ is sufficiently close to the previous value λ, stop. Otherwise, go to Step 2.

Amazon/VB Assets
Exhibit 1012
Page 428

In fact, the interpolation weights in Eq. (8.66) are similar to the Gaussian mixture weights, although $P_A(\mathbf{x})$ and $P_B(\mathbf{x})$ may not be Gaussian density functions. When we have M sets of data, we can use the same EM algorithm to estimate the interpolation weights as illustrated in Algorithm 8.5.

The deleted interpolation procedure described above can be applied after each training iteration. Then, for the following iteration of training, the learned interpolation weights can be used as illustrated in Eq. (8.66) to compute the forward-backward paths or the Viterbi maximum path. We can also have more than two distributions interpolated together. Deleted interpolation has been widely used in both acoustic and language modeling where smoothing is needed.

8.4.5. Parameter Smoothing

One simple reality for probabilistic modeling is that as many observations as possible are required to reliably estimate model parameters. However, in reality, only a finite amount of training data is available. If the training data are limited, this will result in some parameters being inadequately trained, and classification based on poorly trained models will result in higher recognition error rate. There are many possible solutions to address the problem of insufficient training data:

- You can increase the size of the training data. *There is no data like more data.*
- You can reduce the number of free parameters to be re-estimated. This has its limitations, because a number of significant parameters are always needed to model physical events.
- You can interpolate one set of parameter estimates with another set of parameter estimates, for which an adequate amount of training data exists. Deleted interpolation, discussed in Section 8.4.4, can be used effectively. In the discrete HMM, one simple approach is to set a floor to both the transition probability and the output probability in order to eliminate possible zero estimates. The same principle applies to the SCHMM as well as the mixing coefficients of the continuous density HMM. Parameter flooring can be regarded as a special case of interpolation with the uniform distribution.
- You can tie parameters together to reduce the number of free parameters. The SCHMM is a typical example of such parameter-tying techniques.

For the continuous mixture HMM, you need to pay extra attention to smoothing the covariance matrices. There are a number of techniques you can use:

- You can interpolate the covariance matrix with those that are better trained or a priori via the MAP method.

Amazon/VB Assets
Exhibit 1012
Page 429

- You can tie the Gaussian covariance matrices across different mixture components or across different Markov states. A very general shared Gaussian density model is discussed in [20].
- You can use the diagonal covariance matrices if the correlation among feature coefficients is weak, which is the case if you use uncorrelated features such as the MFCC.
- You can combine these methods together.

In practice, we can reduce the speech recognition error rate by 5-20% with various smoothing techniques, depending on the available amount of training data.

8.4.6. Probability Representations

When we compute the forward and backward probabilities in the forward-backward algorithm, they will approach zero in exponential fashion if the observation sequence length, T, becomes large enough. For sufficiently large T, the dynamic range of these probabilities will exceed the precision range of essentially any machine. Thus, in practice, it will result in underflow on the computer if probabilities are represented directly. We can resolve this implementation problem by scaling these probabilities with some scaling coefficient so that they remain within the dynamic range of the computer. All of these scaling coefficients can be removed at the end of the computation without affecting the overall precision.

For example, let $\alpha_t(i)$ be multiplied by a scaling coefficient, S_t:

$$S_t = 1/\sum_i \alpha_t(i) \tag{8.68}$$

so that $\sum_i S_t \alpha_t(i) = 1$ for $1 \le t \le T$. $\beta_t(i)$ can also be multiplied by S_t for $1 \le t \le T$. The recursion involved in computing the forward and backward variables can be scaled at each stage of time t by S_t. Notice that $\alpha_t(i)$ and $\beta_t(i)$ are computed recursively in exponential fashion; therefore, at time t, the total scale factor applied to the forward variable $\alpha_t(i)$ is

$$Scale_\alpha(t) = \prod_{k=1}^{t} S_k \tag{8.69}$$

and the total scale factor applied to the backward variable $\beta_t(i)$ is

$$Scale_\beta(t) = \prod_{k=t}^{T} S_k \tag{8.70}$$

This is because the individual scaling factors are multiplied together in the forward and backward recursion. Let $\alpha'_t(i)$, $\beta'_t(i)$, and $\gamma'_t(i,j)$ denote their corresponding scaled variables, respectively. Note that

Amazon/VB Assets
Exhibit 1012
Page 430

$$\sum_i \alpha'_T(i) = Scale_\alpha(T) \sum_i \alpha_T(i) = Scale_\alpha(T) P(\mathbf{X} \mid \Phi) \tag{8.71}$$

The scaled intermediate probability, $\gamma'_t(i,j)$, can then be written as:

$$\gamma'_t(i,j) = \frac{Scale_\alpha(t-1)\alpha_{t-1}(i)a_{ij}b_j(X_t)\beta_t(j)Scale_\beta(t)}{Scale_\alpha(T)\sum_{i=1}^{N}\alpha_T(i)} = \gamma_t(i,j) \tag{8.72}$$

Thus, the intermediate probabilities can be used in the same way as the unscaled probabilities, because the scaling factor is cancelled out in Eq. (8.72). Therefore, re-estimation formulas can be kept exactly except that $P(\mathbf{X} \mid \Phi)$ should be computed according to

$$P(\mathbf{X} \mid \Phi) = \sum_i \alpha'_T(i) / Scale_\alpha(T) \tag{8.73}$$

In practice, the scaling operation need not be performed at every observation time. It can be used at any scaling interval for which the underflow is likely to occur. In the unscaled interval, $Scale_\alpha$ can be kept as unity.

An alternative way to avoid underflow is to use a logarithmic representation for all the probabilities. This not only ensures that scaling is unnecessary, as underflow cannot happen, but also offers the benefit that integers can be used to represent the logarithmic values, thereby changing floating point operations to fixed point ones, which is particularly suitable for Viterbi-style computation, as Eq. (8.25) requires no probability addition.

In the forward-backward algorithm we need to have probability addition. We can keep a table on $\log_b P_2 - \log_b P_1$. If we represent probability P by $\log_b P$, more precision can be obtained by setting b closer to unity. Let us assume that we want to add P_1 and P_2 and that $P_1 \geq P_2$. We have:

$$\log_b(P_1 + P_2) = \log_b P_1 + \log_b(1 + b^{\log_b P_2 - \log_b P_1}) \tag{8.74}$$

If P_2 is many orders of magnitude smaller than P_1, adding the two numbers will just result in P_1. We could store all possible values of $\log_b(1+b^x)$ in a table. Using logarithms introduces errors for addition operation. In practice, double-precision float representation can be used to minimize the impact of the precision problems.

8.5. HMM LIMITATIONS

There are a number of limitations in the conventional HMMs. For example, HMMs assume the duration follows an exponential distribution, the transition probability depends only on the origin and destination, and all observation frames are dependent only on the state that generated them, not on neighboring observation frames. Researchers have proposed a number of techniques to address these limitations, albeit these solutions have not significantly improved speech recognition accuracy for practical applications.

Amazon/VB Assets
Exhibit 1012
Page 431

8.5.1. Duration Modeling

One major weakness of conventional HMMs is that they do not provide an adequate representation of the temporal structure of speech. This is because the probability of state occupancy decreases exponentially with time as shown in Eq. (8.75). The probability of t consecutive observations in state i is the probability of taking the self-loop at state i for t times, which can be written as

$$d_i(t) = a_{ii}^t(1-a_{ii}) \tag{8.75}$$

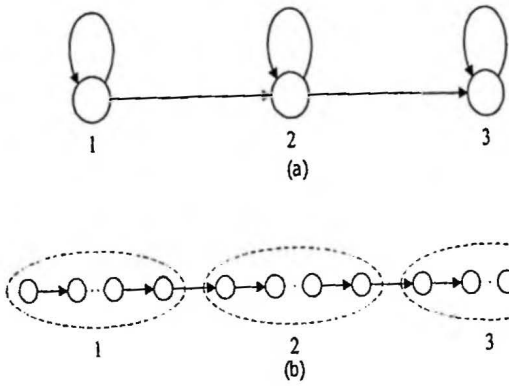


Figure 8.9 A standard HMM (a) and its corresponding explicit duration HMM (b) where the self transitions are replaced with the explicit duration probability distribution for each state.

An improvement to the standard HMM results from the use of HMMs with an explicit time duration distribution for each state [30, 39]. To explain the principle of time duration modeling, a conventional HMM with exponential state duration density and a time duration HMM with specified state duration densities (which can be either a discrete distribution or a continuous density) are illustrated in Figure 8.9. In (a), the state duration probability has an exponential form as in Eq. (8.75). In (b), the self-transition probabilities are replaced with an explicit duration probability distribution. At time t, the process enters state i for duration τ with probability density $d_i(\tau)$, during which the observations $X_{t+1}, X_{t+2} \ldots X_{t+\tau}$ are generated. It then transfers to state j with transition probability a_{ij} only after the appropriate τ observations have occurred in state i. Thus, by setting the time duration probability density to be the exponential density of Eq. (8.75) the time duration HMM can be made equivalent to the standard HMM. The parameters $d_i(\tau)$ can be estimated from observations along with the other parameters of the HMM. For expedience, the duration density is usually truncated at a maximum duration value T_d. To re-estimate the parameters of the HMM with time duration modeling, the forward recursion must be modified as follows:

Amazon/VB Assets
Exhibit 1012
Page 432

$$\alpha_t(j) = \sum_{\tau} \sum_{i,i\neq j} \alpha_{t-\tau}(i) a_{ij} d_j(\tau) \prod_{l=1}^{\tau} b_j(X_{t-\tau+l}) \tag{8.76}$$

where the transition from state i to state j depends not only upon the transition probability a_{ij} but also upon all the possible time periods τ that may occur in state j. Intuitively, Eq. (8.76) illustrates that when state j is reached from previous states i, the observations may stay in state j for a period of τ with duration density $d_j(\tau)$, and each observation emits its own output probability. All possible durations must be considered, which leads to summation with respect to τ. The independence assumption of observations results in the Π term of the output probabilities. Similarly, the backward recursion can be written as:

$$\beta_t(i) = \sum_{\tau} \sum_{j,j\neq i} a_{ij} d_j(\tau) \prod_{l=1}^{\tau} b_j(X_{t+l}) \beta_{t+\tau}(j) \tag{8.77}$$

The modified Baum-Welch algorithm can then be used based on Eq. (8.76) and (8.77). The proof of the re-estimation algorithm can be based on the modified Q-function except that $P(\mathbf{X},\mathbf{S}|\Phi)$ should be replaced with $P(\mathbf{X},\mathbf{S},\mathbf{T}|\Phi)$, which denotes the joint probability of observation, $\mathbf{X}$, state sequence, $\mathbf{S} = \{s_1, s_2 \ldots, s_k \ldots s_{N_s}\}$ in terms of state s_k with time duration τ_k, and the corresponding duration sequence, $\mathbf{T} = \{\tau_1, \tau_2, \ldots \tau_k \ldots \tau_{N_s}\}$.

$$Q(\Phi, \hat{\Phi}) = \frac{1}{P(\mathbf{X}|\Phi)} \sum_{\mathbf{T}} \sum_{\mathbf{S}} P(\mathbf{X},\mathbf{S},\mathbf{T}|\Phi) \log P(\mathbf{X},\mathbf{S},\mathbf{T}|\hat{\Phi}) \tag{8.78}$$

In a manner similar to the standard HMM, $\gamma_{t,\tau}(i,j)$ can be defined as the transition probability from state i at time t to state j with time duration τ in state j. $\gamma_{t,\tau}(i,j)$ can be written as:

$$\gamma_{t,\tau}(i,j) = \alpha_t(i) a_{ij} d_j(\tau) \prod_{l=1}^{\tau} b_j(X_{t+l}) \beta_{t+\tau}(j) / \sum_{k=1}^{N} \alpha_T(k) \tag{8.79}$$

Similarly, the probability of being in state j at time t with duration τ can be computed as:

$$\gamma_{t,\tau}(j) = \sum_{i} \gamma_{t,\tau}(i,j) \tag{8.80}$$

The re-estimation algorithm can be derived from Eq. (8.80), the Viterbi decoding algorithm can be used for the time duration model, and the optimal path can be determined according to:

$$V_t(j) = \underset{i}{Max}\, \underset{\tau}{Max} [V_{t-\tau}(i) a_{ij} d_j(\tau) \prod_{l=1}^{\tau} b_j(X_{t-\tau+l})] \tag{8.81}$$

There are drawbacks to the use of the time duration modeling discussed here. One is the great increase in computational complexity by a factor of $O(D^2)$, where D is the time

Amazon/VB Assets
Exhibit 1012
Page 433

duration distribution length. Another problem is the large number of additional parameters D that must be estimated. One proposed remedy is to use a continuous density function instead of the discrete distribution $d_j(\tau)$.

In practice, duration models offer only modest improvement for speaker-independent continuous speech recognition. Many systems even eliminate the transition probability completely because the output probabilities are so dominant. Nevertheless, duration information is very effective for pruning unlikely candidates during the large-vocabulary speech recognition decoding process.

8.5.2. First-Order Assumption

As you can see from the previous section, the duration of each stationary segment captured by a single state is inadequately modeled. Another way to alleviate the duration problem is to eliminate the first-order transition assumption and to make the underlying state sequence a second-order Markov chain [32]. As a result, the transition probability between two states at time t depends on the states in which the process was at time t-1 and t-2. For a given state sequence $\mathbf{S} = \{s_1, s_2, \ldots s_T\}$, the probability of the state should be computed as:

$$P(\mathbf{S}) = \prod_t a_{s_{t-2}s_{t-1}s_t} \tag{8.82}$$

where $a_{s_{t-2}s_{t-1}s_t} = P(s_t | s_{t-2}s_{t-1})$ is the transition probability at time t, given the two-order state history. The re-estimation procedure can be readily extended based on Eq. (8.82). For example, the new forward probability can be re-defined as:

$$\alpha_t(j,k) = P(X_1^t, s_{t-1} = j, s_t = k|\lambda) = \sum_i \alpha_{t-1}(i,j) a_{ijk} b_k(X_t) \tag{8.83}$$

where $a_{ijk} = P(s_t = k | s_{t-2} = i, s_{t-1} = j)$. Similarly, we can define the backward probability as:

$$\beta_t(i,j) = P(X_{t+1}^T | s_{t-1} = i, s_t = j, \lambda) = \sum_k a_{ijk} b_k(X_{t+1}) \beta_{t+1}(j,k) \tag{8.84}$$

With Eq. (8.83) and (8.84), the MLE estimates can be derived easily based on the modified $\gamma_t(i,j,k)$:

$$\begin{aligned} \gamma_t(i,j,k) &= P(s_{t-1} = i, s_t = j, s_{t+1} = k, \mathbf{X}|\Phi) \\ &= \alpha_t(i,j) a_{ijk} b_k(X_{t+1}) \beta_{t+1}(j,k) / P(\mathbf{X}|\Phi) \end{aligned} \tag{8.85}$$

In practice, the second-order model is computationally very expensive as we have to consider the increased state space, which can often be realized with an equivalent first-order hidden Markov model on the two-fold product state space. It has not offered significantly improved accuracy to justify its increase in computational complexity for most applications.

Amazon/VB Assets
Exhibit 1012
Page 434

8.5.3. Conditional Independence Assumption

The third major weakness in HMMs is that all observation frames are dependent only on the state that generated them, not on neighboring observation frames. The conditional independence assumption makes it hard to effectively handle nonstationary frames that are strongly correlated. There are a number of ways to alleviate the conditional independence assumption [34]. For example, we can assume the output probability distribution depends not only on the state but also on the previous frame. Thus, the probability of a given state sequence can be rewritten as:

$$P(\mathbf{X}|\mathbf{S},\Phi) = \prod_{t=1}^{T} P(X_t|X_{t-1},s_t,\Phi) \tag{8.86}$$

As the parameter space becomes huge, we often need to quantize X_{t-1} into a smaller set of codewords so that we can keep the number of free parameters under control. Thus, Eq. (8.86) can be simplified as:

$$P(\mathbf{X}|\mathbf{S},\Phi) = \prod_{t=1}^{T} P(X_t|\Re(X_{t-1}),s_t,\Phi) \tag{8.87}$$

where $\Re(\,)$ denotes the quantized vector that has a small codebook size, L. Although this can dramatically reduce the space of the free conditional output probability distributions, the total number of free parameters will still increase by L times.

The re-estimation for conditional dependent HMMs can be derived with the modified Q-function, as discussed in the previous sections. In practice, it has not demonstrated convincing accuracy improvement for large-vocabulary speech recognition.

8.6. Historical Perspective and Further Reading

The Markov chain was named after Russian scientist A. Markov for his pioneering work in analyzing the letter sequence in the text of a literary work in 1913 [33]. In the 1960s, Baum and others further developed efficient methods for training the model parameters [4, 5]. When the observation is real valued, the use of continuous or semi-continuous HMMs can improve the overall performance. Baum et al. also developed the method to use continuous density functions that are strictly log concave [5], which was relaxed by Liporace [31] and expanded by Juang to include mixture density functions [27].

The Viterbi algorithm shares the same concept that was independently discovered by researchers in many separate fields [28], including Vintsyuk [42], Needleman and Wunsch [35], Sankoff [41], Sakoe and Chiba [40], and Wagner and Fischer [44].

Jim Baker did his Ph.D. thesis under Raj Reddy at Carnegie Mellon using HMMs for speech recognition [3]. At the same time Fred Jelinek and his colleagues at IBM Research pioneered widespread applications [23]. Since the 1980s, partly because of the DARPA-

Amazon/VB Assets
Exhibit 1012
Page 435

funded speech projects, HMMs have become a mainstream technique for modeling speech, as exemplified by advanced systems developed at BBN, Bell Labs, Carnegie Mellon, IBM, Microsoft, SRI, and others [9, 17, 29, 46]. The Ph.D. theses from Kai-Fu Lee [29], Hsiao-Wuen Hon [16], and Mei-Yuh Hwang [22] at Carnegie Mellon addressed many important practical issues in using HMMs for speech recognition. There are also a number of good books on the practical use of HMMs [18, 24, 38, 45].

The choice of different output probabilities depends on a number of factors such as the availability of training data, the feature characteristics, the computational complexity, and the number of free parameters [19] [34]. The semi-continuous model, also known as the tied-mixture model, was independently proposed by Huang and Jack [21] and Bellegarda and Nahamoo [6]. Other improvements include explicit duration modeling [1, 11, 13, 14, 30, 39], high-order and conditional models [7, 32, 34], which have yet to be shown effective for practical speech recognition.

Both Carnegie Mellon University's open speech software[4] and Cambridge University's HTK[5] are a good starting point for those interested in using the existing tools for running experiments.

HMMs have become the most prominent techniques for speech recognition today. Most of the state-of-the-art speech recognition systems on the market are based on HMMs described in this chapter.

REFERENCES

[1] Anastasakos, A., R. Schwartz, and H. Sun, "Duration Modeling in Large Vocabulary Speech Recognition" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 628-631.

[2] Bahl, L.R., *et al.*, "Speech Recognition with Continuous-Parameter Hidden Markov Models," *Computer Speech and Language*, 1987, **2**, pp. 219-234.

[3] Baker, J.K., "The DRAGON System—An Overview," *Trans. on Acoustics, Speech and Signal Processing*, 1975, **23**(1), pp. 24-29.

[4] Baum, L.E. and J.A. Eagon, "An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology," *Bulletin of American Mathematical Society*, 1967, **73**, pp. 360-363.

[5] Baum, L.E., *et al.*, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *Annals of Mathematical Statistics*. 1970, **41**, pp. 164-171.

[6] Bellegarda, J.R. and D. Nahamoo, "Tied Mixture Continuous Parameter Models for Large Vocabulary Isolated Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, pp. 13-16.

[7] Brown, P., *The Acoustic-Modeling Problem in Automatic Speech Recognition*, Ph.D. Thesis in *Computer Science*, 1987, Carnegie Mellon University, Pittsburgh.

[4] http://www.speech.cs.cmu.edu/sphinx/
[5] http://htk.eng.cam.ac.uk/

Amazon/VB Assets
Exhibit 1012
Page 436

[8] Brown, P.F., *et al.*, "The Mathematics of Statistical Machine Translation: Parameter Estimation," *Computational Linguistics*, 1995, **19**(2), pp. 263-312.

[9] Chou, W., C.H. Lee, and B.H. Juang, "Minimum Error Rate Training of Inter-Word Context Dependent Acoustic Model Units in Speech Recognition" in *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 439-442.

[10] Church, K., "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," *Proc. of the Second Conf. on Applied Natural Language Processing*, 1988, Austin, Texas, pp. 136-143.

[11] Deng, L., M. Lennig, and P. Mermelstein, "Use of Vowel Duration Information in a Large Vocabulary Word Recognizer," *Journal of the Acoustical Society of America*, 1989, **86**(2 August), pp. 540-548.

[12] DeRose, S.J., "Grammatical Category Disambiguation by Statistical Optimization," *Computational Linguistics*, 1988(1), pp. 31-39.

[13] Dumouchel, P. and D. Shaughnessy, "Segmental Duration and HMM Modeling," *Proc. of the European Conf. on Speech Communication and Technology*, 1995, Madrid, Spain, pp. 803-806.

[14] Ferguson, J.D., "Variable Duration Models for Speech" in *Proc. of the Symposium on the Application of Hidden Markov Models to Text and Speech*, J.D. Ferguson, ed., 1980, Princeton, New Jersey, pp. 143-179.

[15] Fiscus, J., "A Post-Processing System to Yield Reduced Word Error Rates: Recognizer Output Voting Error Reduction (ROVER)," *IEEE Workshop on Automatic Speech Recognition and Understanding*, 1997, Santa Barbara, CA, pp. 347-352.

[16] Hon, H.W., *Vocabulary-Independent Speech Recognition: The VOCIND System*, Ph.D Thesis in *Department of Computer Science* 1992, Carnegie Mellon University, Pittsburgh.

[17] Huang, X.D., *et al.*, "The SPHINX-II Speech Recognition System: An Overview," *Computer Speech and Language*, 1993, pp. 137-148.

[18] Huang, X.D., Y. Ariki, and M.A. Jack, *Hidden Markov Models for Speech Recognition*, 1990, Edinburgh, U.K., Edinburgh University Press.

[19] Huang, X.D., *et al.*, "A Comparative Study of Discrete, Semicontinuous, and Continuous Hidden Markov Models," *Computer Speech and Language*, 1993, **7**(4), pp. 359-368.

[20] Huang, X.D., *et al.*, "Deleted Interpolation and Density Sharing for Continuous Hidden Markov Models," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1996.

[21] Huang, X.D. and M.A. Jack, "Semi-Continuous Hidden Markov Models with Maximum Likelihood Vector Quantization" in *IEEE Workshop on Speech Recognition*, 1988.

[22] Hwang, M., *Subphonetic Modeling in HMM-based Speech Recognition Systems*, Ph.D. Thesis in *Computer Science*, 1994, Carnegie Mellon University, Pittsburgh.

[23] Jelinek, F., "Continuous Speech Recognition by Statistical Methods," *Proc. of the IEEE*, 1976, **64**(4), pp. 532-556.

Amazon/VB Assets
Exhibit 1012
Page 437

[24] Jelinek, F., *Statistical Methods for Speech Recognition*, 1998, Cambridge, MA, MIT Press.

[25] Jiang, L. and X. Huang, "Unified Decoding and Feature Representation for Improved Speech Recognition," *Proc. of the 6th European Conf. on Speech Communication and Technology*, 1999, Budapest, Hungary, pp. 1331-1334.

[26] Juang, B.H., W. Chou, and C.H. Lee, "Statistical and Discriminative Methods for Speech Recognition" in *Automatic Speech and Speaker Recognition—Advanced Topics*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Boston, pp. 109-132, Kluwer Academic Publishers.

[27] Juang, B.H., S.E. Levinson, and M.M. Sondhi, "Maximum Likelihood Estimation for Multivariate Mixture Observations of Markov Chains," *IEEE Trans. on Information Theory*, 1986, **IT-32**(2), pp. 307-309.

[28] Kruskal, J., "An Overview of Sequence Comparison" in *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, D. Sankoff and J. Kruskal, eds. 1983, Reading, MA., Addison-Wesley, pp. 1-44.

[29] Lee, K.F., *Large-Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System*, Ph.D. Thesis in Computer Science Dept., 1988, Carnegie Mellon University, Pittsburgh.

[30] Levinson, S.E., "Continuously Variable Duration Hidden Markov Models for Automatic Speech Recognition," *Computer Speech and Language*, 1986, pp. 29-45.

[31] Liporace, L.R., "Maximum Likelihood Estimation for Multivariate Observations of Markov Sources," *IEEE Trans. on Information Theory*, 1982, **28**, pp. 729-734.

[32] Mari, J., J. Haton, and A. Kriouile, "Automatic Word Recognition Based on Second-Order Hidden Markov Models," *IEEE Trans. on Speech and Audio Processing*, 1977, **5**(1), pp. 22-25.

[33] Markov, A.A., "An Example of Statistical Investigation in the Text of 'Eugene Onyegin', Illustrating Coupling of Tests in Chains," *Proc. of the Academy of Sciences of St. Petersburg*, 1913, Russia, pp. 153-162.

[34] Ming, J. and F. Smith, "Modelling of the Interframe Dependence in an HMM Using Conditional Gaussian Mixtures," *Computer Speech and Language*, 1996, **10**(4), pp. 229-247.

[35] Needleman, S. and C. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino-acid Sequence of Two Proteins," *Journal of Molecular Biology*, 1970, **48**, pp. 443-453.

[36] Normandin, Y., "Maximum Mutual Information Estimation of Hidden Markov Models" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Norwell, MA, Kluwer Academic Publishers.

[37] Rabiner, L.R., "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. of IEEE*, 1989, **77**(2), pp. 257-286.

[38] Rabiner, L.R. and B.H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall Signal Processing Series, ed. A.V. Oppenheim, 1993, Englewood Cliffs, NJ, Prentice-Hall.

Amazon/VB Assets
Exhibit 1012
Page 438

[39] Russell, M.J. and R.K. Moore, "Explicit Modeling of State Occupancy in Hidden Markov Models for Automatic Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1985, pp. 5-8.

[40] Sakoe, H. and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1978, **26**(1), pp. 43-49.

[41] Sankoff, D., "Matching Sequences under Deletion-Insertion Constraints," *Proc. of the National Academy of Sciences*, 1972, **69**, pp. 4-6.

[42] Vintsyuk, T.K., "Speech Discrimination by Dynamic Programming," *Cybernetics*, 1968, **4**(1), pp. 52-57.

[43] Viterbi, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. on Information Theory*, 1967, **13**(2), pp. 260-269.

[44] Wagner, R. and M. Fischer, "The String-to-String Correction Problem," *Journal of the ACM*, 1974, **21**, pp. 168-173.

[45] Waibel, A.H. and K.F. Lee, *Readings in Speech Recognition*, 1990, San Mateo, CA, Morgan Kaufman Publishers.

[46] Young, S.J. and P.C. Woodland, "The Use of State Tying in Continuous Speech Recognition," *Proc. of Eurospeech*, 1993, Berlin, pp. 2203-2206.

Amazon/VB Assets
Exhibit 1012
Page 439

CHAPTER 9

Acoustic Modeling

After years of research and development, accuracy of automatic speech recognition remains one of the most important research challenges. A number of well-known factors determines accuracy; those most noticeable are variations in context, in speaker, and in environment. Acoustic modeling plays a critical role in improving accuracy and is arguably the central part of any speech recognition system.

For the given acoustic observation $\mathbf{X} = X_1 X_2 ... X_n$, the goal of speech recognition is to find out the corresponding word sequence $\hat{\mathbf{W}} = w_1 w_2 ... w_m$ that has the maximum posterior probability $P(\mathbf{W}|\mathbf{X})$ as expressed by Eq. (9.1).

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) = \arg\max_{\mathbf{W}} \frac{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}{P(\mathbf{X})} \tag{9.1}$$

Since the maximization of Eq. (9.1) is carried out with the observation $\mathbf{X}$ fixed, the above maximization is equivalent to maximization of the following equation:

Amazon/VB Assets
Exhibit 1012
Page 441

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W}) \tag{9.2}$$

The practical challenge is how to build accurate acoustic models, $P(\mathbf{X} \mid \mathbf{W})$, and language models, $P(\mathbf{W})$, that can truly reflect the spoken language to be recognized. For large-vocabulary speech recognition, since there are a large number of words, we need to decompose a word into a subword sequence. Thus $P(\mathbf{X} \mid \mathbf{W})$ is closely related to phonetic modeling. $P(\mathbf{X} \mid \mathbf{W})$ should take into account speaker variations, pronunciation variations, environmental variations, and context-dependent phonetic coarticulation variations. Last, but not least, any static acoustic or language model will not meet the needs of real applications. So it is vital to dynamically adapt both $P(\mathbf{W})$ and $P(\mathbf{X} \mid \mathbf{W})$ to maximize $P(\mathbf{W} \mid \mathbf{X})$ while using spoken language systems. The decoding process of finding the best word sequence $\mathbf{W}$ to match the input speech signal $\mathbf{X}$ in speech recognition systems is more than a simple pattern recognition problem, since in continuous speech recognition you have an infinite number of word patterns to search, as discussed in detail in Chapters 12 and 13.

In this chapter we focus on discussing solutions that work well in practice. To highlight solutions that are effective, we use the Whisper speech recognition system [49] developed at Microsoft Research as a concrete example to illustrate how to build a working system and how various techniques can help to reduce speech recognition errors.[1] We hope that by studying what worked well in the past we can illuminate the possibilities for further improvement of the state of the art.

The hidden Markov model we discussed in Chapter 8 is the underpinning for acoustic phonetic modeling. It provides a powerful way to integrate segmentation, time warping, pattern matching, and context knowledge in a unified manner. The underlying technologies are undoubtedly evolving, and the research community is aggressively searching for more powerful solutions. Most of the techniques discussed in this chapter can be readily derived from the fundamentals discussed in earlier chapters.

9.1. VARIABILITY IN THE SPEECH SIGNAL

The research community has produced technologies that, with some constraints, can accurately recognize spoken input. Admittedly, today's state-of-the-art systems still cannot match humans' performance. Although we can build a very accurate speech recognizer for a particular speaker, in a particular language and speaking style, in a particular environment, and limited to a particular task, it remains a research challenge to build a recognizer that can essentially understand anyone's speech, in any language, on any topic, in any free-flowing style, and in almost any speaking environment.

[1] Most of the experimental results used here are based on a development test set for the 60,000-word speaker-independent continuous dictation task. The training set consists of 35,000 utterances from about 300 speakers. The test set consists of 410 utterances from 10 speakers that were not used in the training data. The language model is derived from 2 billion words of English text corpora.

Amazon/VB Assets
Exhibit 1012
Page 442

Accuracy and robustness are the ultimate measures for the success of speech recognition algorithms. There are many reasons why existing algorithms or systems did not deliver what people want. In the sections that follow we summarize the major factors involved.

9.1.1. Context Variability

Spoken language interaction between people requires knowledge of word meanings, communication context, and common sense. Words with widely different meanings and usage patterns may have the same phonetic realization. Consider the challenge represented by the following utterance:

> *Mr. Wright should write to Ms. Wright right away about his Ford or four door Honda.*

For a given word with the same pronunciation, the meaning could be dramatically different, as indicated by *Wright*, *write*, and *right*. What makes it even more difficult is that *Ford or* and *Four Door* are not only phonetically identical, but also semantically relevant. The interpretation is made within a given word boundary. Even with smart linguistic and semantic information, it is still impossible to decipher the correct word sequence, unless the speaker pauses between words or uses intonation to set apart these semantically confusable phrases.

In addition to the context variability at word and sentence level, you can find dramatic context variability at the phonetic level. As illustrated in Figure 9.1, the acoustic realization of phoneme */eel* for word *peat* and *wheel* depends on its left and right context. The dependency becomes more important in fast speech or spontaneous speech conversations, since many phonemes are not fully realized.

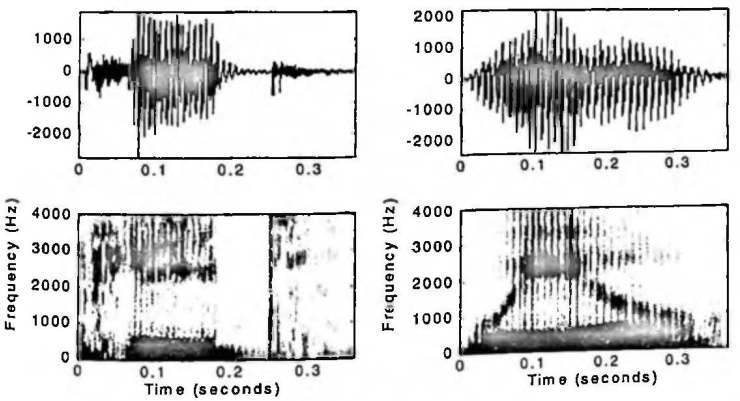


Figure 9.1 Waveforms and spectrograms for words *peat* (left) and *wheel* (right). The phoneme */ee/* is illustrated with two different left and right contexts. This illustrates that different contexts may have different effects on a phone.

Amazon/VB Assets
Exhibit 1012
Page 443

9.1.2. Style Variability

To deal with acoustic realization variability, a number of constraints can be imposed on the use of the speech recognizer. For example, we can have an *isolated* speech recognition system, in which users have to pause between each word. Because the pause provides a clear boundary for the word, we can easily eliminate errors such as *Ford or* and *four door.* In addition, isolated speech provides a correct silence context to each word so that it is easier to model and decode the speech, leading to a significant reduction in computational complexity and error rate. In practice, the word-recognition error rate of an isolated speech recognizer can typically be reduced by more than a factor of three (from 7% to 2%) as compared with to a comparable continuous speech recognition system [5]. The disadvantage is that such an isolated speech recognizer is unnatural to most people. The throughput is also significantly lower than that for continuous speech.

In continuous speech recognition, the error rate for casual, spontaneous speech, as occurs in our daily conversation, is much higher than for carefully articulated read-aloud speech. The rate of speech also affects the word recognition rate. It is typical that the higher the *speaking rate* (words/minute), the higher the error rate. If a person whispers, or shouts, to reflect his or her emotional changes, the variation increases even more significantly.

9.1.3. Speaker Variability

Every individual speaker is different. The speech he or she produces reflects the physical vocal tract size, length and width of the neck, a range of physical characteristics, age, sex, dialect, health, education, and personal style. As such, one person's speech patterns can be entirely different from those of another person. Even if we exclude these interspeaker differences, the same speaker is often unable to precisely produce the same utterance. Thus, the shape of the vocal tract movement and rate of delivery may vary from utterance to utterance, even with dedicated effort to minimize the variability.

For *speaker-independent* speech recognition, we typically use more than 500 speakers to build a combined model. Such an approach exhibits large performance fluctuations among new speakers because of possible mismatches in the training data between exiting speakers and new ones [50]. In particular, speakers with accents have a tangible error-rate increase of 2 to 3 times.

To improve the performance of a speaker-independent speech recognizer, a number of constraints can be imposed on its use. For example, we can have a user enrollment that requires the user to speak for about 30 minutes. With the *speaker-dependent* data and training, we may be able to capture various speaker-dependent acoustic characteristics that can significantly improve the speech recognizer's performance. In practice, speaker-dependent speech recognition offers not only improved accuracy but also improved speed, since decoding can be more efficient with an accurate acoustic and phonetic model. A typical speaker-dependent speech recognition system can reduce the word recognition error by more than 30% as compared with a comparable speaker-independent speech recognition system.

Amazon/VB Assets
Exhibit 1012
Page 444

The disadvantage of speaker-dependent speech recognition is that it takes time to collect speaker-dependent data, which may be impractical for some applications such as an automatic telephone operator. Many applications have to support walk-in speakers, so speaker-independent speech recognition remains an important feature. When the amount of speaker-dependent data is limited, it is important to make use of both speaker-dependent and speaker-independent data using *speaker-adaptive* training techniques, as discussed in Section 9.6. Even for speaker-independent speech recognition, you can still use speaker-adaptive training based on recognition results to quickly adapt to each individual speaker during usage.

9.1.4. Environment Variability

The world we live in is full of sounds of varying loudness from different sources. When we interact with computers, we may have people speaking in the background. Someone may slam the door, or the air conditioning may start humming without notice. If speech recognition is embedded in mobile devices, such as PDAs (personal digital assistants) or cellular phones, the spectrum of noises varies significantly because the owner moves around. These external parameters, such as the characteristics of the environmental noise and the type and placement of the microphone, can greatly affect speech recognition system performance. In addition to the background noises, we have to deal with noises made by speakers, such as lip smacks and noncommunication words. Noise may also be present from the input device itself, such as microphone and A/D interference noises.

In a similar manner to speaker-independent training, we can build a system by using a large amount of data collected from a number of environments; this is referred to as *multistyle training* [70]. We can use adaptive techniques to normalize the mismatch across different environment conditions in a manner similar to speaker-adaptive training, as discussed in Chapter 10. Despite the progress being made in the field, environment variability remains as one of the most severe challenges facing today's state-of-the-art speech systems.

9.2. How to Measure Speech Recognition Errors

It is critical to evaluate the performance of speech recognition systems. The word recognition error rate is widely used as one of the most important measures. When you compare different acoustic modeling algorithms, it is important to compare their relative error reduction. Empirically, you need to have a test data set that contains more than 500 sentences (with 6 to 10 words for each sentence) from 5 to 10 different speakers to reliably estimate the recognition error rate. Typically, you need to have more than 10% relative error reduction to consider adopting a new algorithm.

As a sanity check, you may want to use a small sample from the training data to measure the performance of the *training set*, which is often much better than what you can get from testing new data. Training-set performance is useful in the development stage to identify potential implementation bugs. Eventually, you need to use a *development set* that typically consists of data never used in training. Since you may tune a number of parameters

Amazon/VB Assets
Exhibit 1012
Page 445

with your development set, it is important to evaluate performance of a *test set* after you decide the optimal parameter setting. The test set should be completely new with respect to both training and parameter tuning.

There are typically three types of word recognition errors in speech recognition:

- *Substitution*: an incorrect word was substituted for the correct word
- *Deletion*: a correct word was omitted in the recognized sentence
- *Insertion*: an extra word was added in the recognized sentence[2]

For instance, a speech recognition system may produce an incorrect result as follows, where substitutions are ***bold***, insertions are underlined, and deletions are denoted as **. There are four errors in this example.

> **Correct:** *Did mob mission area of the Copeland ever go to m4 in nineteen eighty one*
> **Recognized:** *Did mob mission area ** **the copy** land ever go to m4 in nineteen **east** one*

To determine the minimum error rate, you can't simply compare two word sequences one by one. For example, suppose you have utterance *The effect is clear* recognized as *Effect is not clear*. If you compare word to word, the error rate is 75% (*The* vs. *Effect*, *effect* vs. *is*, *is* vs. *not*). In fact, the error rate is only 50% with one deletion (*The*) and one insertion (*not*). In general, you need to align a recognized word string against the correct word string and compute the number of substitutions (*Subs*), deletions (*Dels*), and insertions (*Ins*). The *Word Error Rate* is defined as:

$$\text{Word Error Rate} = 100\% \times \frac{Subs + Dels + Ins}{\text{No. of words in the correct sentenc}} \tag{9.3}$$

This alignment is also known as the *maximum substring matching* problem, which can be easily handled by the dynamic programming algorithm discussed in Chapter 8.

Let the correct word string be $w_1 w_2 \cdots w_n$, where w_i denotes the ith word in the correct word string, and the recognized word string be $\hat{w}_1 \hat{w}_2 \cdots \hat{w}_m$, where $\hat{w}_i$ denotes the i^{th} word in the recognized word string. We denote $R[i, j]$ as the minimum error of aligning substring $w_1 w_2 \cdots w_n$ against substring $\hat{w}_1 \hat{w}_2 \cdots \hat{w}_m$. The optimal alignment and the associated word error rate $R[n, m]$ for correct word string $w_1 w_2 \cdots w_n$ and the recognized word string $\hat{w}_1 \hat{w}_2 \cdots \hat{w}_m$ are obtained via the dynamic programming algorithm illustrated in Algorithm 9.1. The accumulated cost function $R[i, j]$ progresses from $R[1, 1]$ to $R[n, m]$ corresponding to the minimum distance from (1, 1) to (n, m). We store the back pointer information $B[i, j]$ as we move along. When we reach the final grid (n, m), we back trace along the optimal path to find out if there are substitutions, deletions, or insertions on the matched path, as stored in $B[i, j]$.

[2] Even for isolated speech recognition, you may still have the insertion error, since the word boundary needs to be detected in most applications. It is possible that one isolated utterance is recognized as two words.

Amazon/VB Assets
Exhibit 1012
Page 446

ALGORITHM 9.1: *ALGORITHM TO MEASURE THE WORD ERROR RATE*

Step 1: *Initialization* $R[0,0] = 0$ $R[i,j] = \infty$ if $(i < 0)$ or $(j < 0)$ $B[0,0] = 0$

Step 2: *Iteration*

for $i = 1,\ldots,n$ {

for $j = 1,\ldots,m$ {

$$R[i,j] = \min \begin{bmatrix} R[i-1,j]+1 \text{ (deletion)} \\ R[i-1,j-1] \text{ (match)} \\ R[i-1,j-1]+1 \text{ (substitution)} \\ R[i,j-1]+1 \text{ (insertion)} \end{bmatrix}$$

$$B[i,j] = \begin{cases} 1 & \text{if deletion} \\ 2 & \text{if insertion} \\ 3 & \text{if match} \\ 4 & \text{if substitution} \end{cases} \quad \} \}$$

Step 3: *Backtracking and termination*

$$\text{word error rate} = 100\% \times \frac{R(n,m)}{n}$$

optimal backward path = $(s_1, s_2, \ldots, 0)$

$$\text{where } s_1 = B[n,m],\ s_t = \begin{bmatrix} B[i-1,j] \text{ if } s_{t-1} = 1 \\ B[i,j-1] \text{ if } s_{t-1} = 2 \\ B[i-1,j-1] \text{ if } s_{t-1} = 3 \text{ or } 4 \end{bmatrix} \text{ for } t = 2,\ldots \text{ until } s_t = 0$$

For applications involved with rejection, such as word confidence measures as discussed in Section 9.7, you need to measure both false rejection rate and false acceptance rate. In speaker or command verification, the false acceptance of a valid user/command is also referred to as Type I error, as opposed to the false rejection of a valid user/command (Type II) [17]. A higher false rejection rate generally leads to a lower false acceptance rate. A plot of the false rejection rate versus the false acceptance rate, widely used in communication theory, is called the *receiver operating characteristic* (ROC) curve.

9.3. SIGNAL PROCESSING—EXTRACTING FEATURES

The role of a signal processing module, as illustrated in Figure 1.2, is to reduce the data rate, to remove noises, and to extract salient features that are useful for subsequent acoustic matching. Using as building blocks the topics we discussed in earlier chapters, we briefly illustrate here what is important in modeling speech to deal with variations we must address. More advanced environment normalization techniques are discussed in Chapter 10.

Amazon/VB Assets
Exhibit 1012
Page 447

9.3.1. Signal Acquisition

Today's computers can handle most of the necessary speech signal acquisition tasks in software. For example, most PC sound cards have direct memory access, and the speech can be digitized to memory without burdening the CPU with input/output interrupts. The operating system can correctly handle most of the necessary AD/DA functions in real time.

To perform speech recognition, a number of components—such as digitizing speech, feature extraction and transformation, acoustic matching, and language model-based search—can be pipelined time-synchronously from left to right. Most operating systems can supply mechanisms for organizing pipelined programs in a multitasking environment. Buffers must be appropriately allocated so that you can ensure time-synchronous processing of each component. Large buffers are generally required on slow machines because of potential delays in processing an individual component. The right buffer size can be easily determined by experimentally tuning the system with different machine load situations to find a balance between resource use and relative delay.

For speech signal acquisition, the needed buffer typically ranges from 4 to 64 kB with 16-kHz sampling rate and 16-bit A/D precision. In practice, 16-kHz sampling rate is sufficient for the speech bandwidth (8 kHz). Reduced bandwidth, such as telephone channel, generally increases speech recognition error rate. Table 9.1 shows some empirical relative word recognition error increase using a number of different sampling rates. If we take the 8-kHz sampling as our baseline, we can reduce the word recognition error with a comparable recognizer by about 10% if we increase the sampling rate to 11 kHz. If we further increase the sampling rate to 16 kHz, the word recognition error rate can be further reduced by an additional 10%. Further increasing the sampling rate to 22 kHz does not have any additional impact on the word recognition errors, because most of the salient speech features are within an 8-kHz bandwidth.

Table 9.1 Relative error rate reduction with different sampling rates. The reduction is relative to that of the preceding row.

Sampling Rate	Relative Error-Rate Reduction
8 kHz	Baseline
11 kHz	+10%
16 kHz	+10%
22 kHz	+0%

9.3.2. End-Point Detection

To activate speech signal capture, you can use a number of modes including either *push to talk* or *continuously listening*. The push-to-talk mode uses a special push event to activate or

Amazon/VB Assets
Exhibit 1012
Page 448

deactivate speech capture, which is immune to the potential background noise and can eliminate unnecessary use of processing resources to detect speech events. This mode sometimes also requires you to *push and hold while talking*. You push to indicate speech's beginning and then release to indicate the end of speech capture. The disadvantage is the necessity to activate the application each time the person speaks.

The continuously listening model listens all the time and automatically detects whether there is a speech signal or not. It needs a so-called *speech end-point detector*, which is typically based on an extremely efficient two-class pattern classifier. Such a classifier is used to filter out obvious silence, but the ultimate decision on the utterance boundary is left to the speech recognizer. In comparison to the push-to-talk mode, the continuously listening mode requires more processing resources, also with potential classification errors.

The endpoint detector is often based on an energy threshold that is a function of time. The logarithm of the energy threshold can be dynamically generated based on energy profiles across a certain period of time. Constraints on word duration can also be imposed to better classify a sequence of frames so that extremely short spikes can be eliminated.

It is not critical for the automatic end-point detector to offer exact end-point accuracy. The key feature required of it is a low rejection rate (i.e., the automatic end-point detector should not interpret speech segments as silence/noise segments). Any false rejection leads to an error in the speech recognizer. On the other hand, a possible false acceptance (i.e., the automatic end-point detector interprets noise segments as speech segments) may be rescued by the speech recognizer later if the recognizer has appropriate noise models, such as specific models for clicks, lip smacks, and background noise.

Explicit end-point detectors work reasonably well with recordings exhibiting a signal-to-noise ratio of 30 dB or greater, but they fail considerably on noisier speech. As discussed, speech recognizers can be used to determine the end points by aligning the vocabulary words preceded and followed by a silence/noise model. This scheme is generally much more reliable than any threshold-based explicit end-point detection, because recognition can jointly detect both the end points and words or other explicit noise classes, but requires more computational resources. A compromise is to use a simple adaptive two-class (speech vs. silence/noise) classifier to locate speech activities (with enough buffers at both ends) and notify the speech recognizer for subsequent processing. For the two-class classifier, we can use both the log-energy and delta log-energy as the feature. Two Gaussian density functions, $\{\Phi_1, \Phi_2\} = \Phi$, can be used to model the background stationary noise and speech, respectively. The parameters of the Gaussian density can be estimated using the labeled speech and noise data or estimated in an unsupervised manner.

When enough frames are classified as speech segments by the efficient two-class classifier, the speech recognizer is notified to start recognizing the signal. As shown in Figure 9.2, we should include enough frames before the beginning frame, t_b, for the speech recognizer to minimize the possible detection error. In the same manner, when enough noise/silence frames are detected at t_e, we should keep providing the speech recognizer with enough frames for processing before declaring that the end of the utterance has been reached.

Amazon/VB Assets
Exhibit 1012
Page 449



Figure 9.2 End-point detection boundary t_b and t_e may need extra buffering for subsequent speech recognition.

Since there are only two classes, these parameters can be dynamically adapted using the EM algorithm during runtime. As discussed in Chapter 4, the EM algorithm can iteratively estimate the Gaussian parameters without having a precise segmentation between speech and noise segments. This is very important, because we need to keep the parameters dynamic for robust end-point detection in constantly changing environments.

To track the varying background noises, we use an exponential window to give weight to the most recent signal:

$$w_k = \exp(-\alpha k) \tag{9.4}$$

where α is a constant that controls the adaptation rate, and k is the index of the time. In fact, you could use different rates for noise and speech when you use the EM algorithm to estimate the two-class Gaussian parameters. It is advantageous to use a smaller time constant for noise than for speech. With such a weighting window, the means of the Gaussian density, as discussed in Chapter 4, can be rewritten as:

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{i=-\infty}^{t} w_i \frac{c_k P(\mathbf{x}_i \mid \Phi_k)\mathbf{x}_i}{\sum_{k=1}^{2} P(\mathbf{x}_i \mid \Phi_k)}}{\sum_{i=-\infty}^{t} w_i \frac{c_k P(\mathbf{x}_i \mid \Phi_k)}{\sum_{k=1}^{2} P(\mathbf{x}_i \mid \Phi_k)}}, k \in \{0,1\} \tag{9.5}$$

9.3.3. MFCC and Its Dynamic Features

The extraction of reliable features is one of the most important issues in speech recognition. There are a large number of features we can use. However, as discussed in Chapter 4, the

Amazon/VB Assets
Exhibit 1012
Page 450

curse-of-dimensionality problem reminds us that the amount of training data is always limited. Therefore, incorporation of additional features may not lead to any measurable error reduction. This does not necessarily mean that the additional features are poor ones, but rather that we may have insufficient data to reliably model those features.

The first feature we use is the speech waveform itself. In general, time-domain features are much less accurate than frequency-domain features such as the mel-frequency cepstral coefficients (MFCC) discussed in Chapter 6 [23]. This is because many features such as formants, useful in discriminating vowels, are better characterized in the frequency domain with a low-dimension feature vector.

As discussed in Chapter 2, temporal changes in the spectra play an important role in human perception. One way to capture this information is to use *delta coefficients* that measure the change in coefficients over time. Temporal information is particularly complementary to HMMs, since HMMs assume each frame is independent of the past, and these dynamic features broaden the scope of a frame. It is also easy to incorporate new features by augmenting the static feature.

When 16-kHz sampling rate is used, a typical state-of-the-art speech system can be build based on the following features.

- 13th-order MFCC $\mathbf{c}_k$
- 13th-order 40-msec 1st-order delta MFCC computed from $\Delta\mathbf{c}_k = \mathbf{c}_{k+2} - \mathbf{c}_{k-2}$
- 13th-order 2nd-order delta MFCC computed from $\Delta\Delta\mathbf{c}_k = \Delta\mathbf{c}_{k+1} - \Delta\mathbf{c}_{k-1}$

A short-time analysis Hamming window of 25 ms is typically used to compute the MFCC $\mathbf{c}_k$. The window shift is typically 10 ms. Please note that $c_k[0]$ is included in the feature vector, which has a role similar to that of the log power. The feature vector used for speech recognition is typically a combination of these features

$$\mathbf{x}_k = \begin{pmatrix} \mathbf{c}_k \\ \Delta\mathbf{c}_k \\ \Delta\Delta\mathbf{c}_k \end{pmatrix} \tag{9.6}$$

The relative error reduction with a typical speech recognition system is illustrated in Table 9.2. As you can see from the table, the 13th-order MFCC outperforms 13th-order LPC cepstrum coefficients, which indicates that perceptually motivated mel-scale representation indeed helps recognition. In a similar manner, perceptually based LPC features such as PLP can achieve similar improvement. The MFCC order has also been studied experimentally for speech recognition. The higher-order MFCC does not further reduce the error rate in comparison with the 13th-order MFCC, which indicates that the first 13 coefficients already contain most salient information needed for speech recognition. In addition to mel-scale representation, another perceptually motivated feature such as the first- and second-order delta features can significantly reduce the word recognition error, while the higher-order delta features provide no further information.

Amazon/VB Assets
Exhibit 1012
Page 451

Feature extraction in these experiments is typically optimized together with the classifier, since there are a number of modeling assumptions, such as the diagonal covariance in the Gaussian density function, that are closely related to what features to use. It is possible that these relative error reductions would vary if a different speech recognizer were used.

Table 9.2 Relative error reduction with different features. The reduction is relative to that of the preceding row.

Feature Set	Relative Error Reduction
13th-order LPC cepstrum coefficients	Baseline
13th-order MFCC	+10%
16th-order MFCC	+0%
+1st- and 2nd-order dynamic features	+20%
+3rd-order dynamic features	+0%

9.3.4. Feature Transformation

Before you use feature vectors such as MFCC for recognition, you can preprocess or transform them into a new space that alleviates environment noise, channel distortion, and speaker variations. You can also transform the features that are most effective for preserving class separability so that you can further reduce the recognition error rate. Since we devote Chapter 10 completely to environment and channel normalization, we briefly discuss here how we can transform the feature vectors to improve class separability.

To further reduce the dimension of the feature vector, you can use a number of dimension reduction techniques to map the feature vector into more effective representations. If the mapping is linear, the mapping function is well defined and you can find the coefficients of the linear function so as to optimize your objective functions. For example, when you combine the first- and second-order dynamic features with the static MFCC vector, you can use *principal-component analysis* (PCA) (also known as *Karhunen-Loeve* transform) [32] to map the combined feature vector into a smaller dimensional vector. The optimum basis vectors of the principal-component analysis are the eigenvectors of the covariance matrix of a given distribution. In practice, you can compute the eigenvectors of the autocorrelation matrix as the basis vectors. The effectiveness of the transformed vector, in terms of representing the original feature vector, is determined by the corresponding eigenvalue of each value in the vector. You can discard the feature with the smallest eigenvalue, since the mean-square error between the transformed vector and the original vector is determined by the eigenvalue of each feature in the vector. In addition, the transformed feature vector is uncorrelated. This is particularly suitable for the Gaussian probability density function with a diagonal covariance matrix.

The recognition error is the best criterion for deciding what feature sets to use. However, it is hard to obtain such an estimate to evaluate feature sets systematically. A simpler

Amazon/VB Assets
Exhibit 1012
Page 452

approach is to use within-class and between-class scatter matrices to formulate criteria of class separability, which is also called *Linear Discriminant Analysis* (LDA) transformation. We can compute the within-class scatter matrix as:

$$S_w = \sum_{\mathbf{x}\in\omega_i} P(\omega_i)E\{(\mathbf{x}-\boldsymbol{\mu}_i)(\mathbf{x}-\boldsymbol{\mu}_i)^t \mid \omega_i\} = \sum_{\mathbf{x}\in\omega_i} P(\omega_i)\Sigma_i \tag{9.7}$$

where the sum is for all the data $\mathbf{x}$ within the class ω_i. This is the scatter of samples around their respective class mean. On the other hand, the between-class scatter matrix is the scatter of the expected vectors around the mixture mean:

$$S_B = \sum_{\boldsymbol{\mu}_i\in\omega_i} P(\omega_i)(\boldsymbol{\mu}_i - \mathbf{m}_0)(\boldsymbol{\mu}_i - \mathbf{m}_0)^t \tag{9.8}$$

where $\mathbf{m}_0$ represents the expected mean vector of the mixture distribution:

$$\mathbf{m}_0 = E\{\mathbf{x}\} = \sum_{\omega_i} P(\omega_i)\mathbf{m}_i \tag{9.9}$$

To formulate criteria to transform feature vector $\mathbf{x}$, we need to derive the linear transformation matrix $\mathbf{A}$. One of the measures can be the trace of $S_w^{-1}S_B$:

$$J = tr(S_w^{-1}S_B) \tag{9.10}$$

The trace is the sum of the eigenvalues of $S_w^{-1}S_B$ and hence the sum of the variances in the principal directions. The number is larger when the between-class scatter is large or the within-class scatter is small. You can derive the transformation matrix based on the eigenvectors of $S_w^{-1}S_B$. In a manner similar to PCA, you can reduce the dimension of the original input feature vector by discarding the smallest eigenvalues [16, 54].

Researchers have used the LDA method to measure the effectiveness of several feature vectors for speaker normalization [41]. Other feature processing techniques designed for speaker normalization include neural-network-based speaker mapping [51], frequency warping for vocal tract normalization (VTN) via mel-frequency scaling [67, 100], and bilinear transformation [2].

To reduce interspeaker variability by a speaker-specific frequency warping, you can simply shift the center frequencies of the mel-spaced filter bank. Let $k\Delta f_{mel}$, $k = 1, \ldots, K$, denote the center frequencies in mel-scale. Then the center frequencies in hertz for a warping factor of α are computed by Eq. (9.11) before the cosine transformation of the MFCC feature vector.

$$f_{Hz}^{\alpha}(k\Delta f_{mel}) = 700(10^{k\Delta f_{mel}/2595} - 1)/\alpha \tag{9.11}$$

Amazon/VB Assets
Exhibit 1012
Page 453

The warping factor is estimated for each speaker by computing the likelihood of the training data for feature sets obtained with different warping factors using the HMM. The relative error reduction based on the feature transformation method has been limited, typically under 10%.

9.4. PHONETIC MODELING—SELECTING APPROPRIATE UNITS

As discussed in Chapter 2, the phonetic system is related to a particular language. We focus our discussion on language-independent technologies but use English in our examples to illustrate how we can use the language-independent technologies to model the salient phonetic information in the language. For general-purpose large-vocabulary speech recognition, it is difficult to build whole-word models because:

- Every new task contains novel words without any available training data, such as proper nouns and newly invented jargons.
- There are simply too many words, and these different words may have different acoustic realizations, as illustrated in Chapter 2. It is unlikely that we have sufficient repetitions of these words to build context-dependent word models.

How to select the most basic units to represent salient acoustic and phonetic information for the language is an important issue in designing a workable system. At a high level, there are a number of issues we must consider in choosing appropriate modeling units.

- The unit should be *accurate,* to represent the acoustic realization that appears in different contexts.
- The unit should be *trainable.* We should have enough data to estimate the parameters of the unit. Although words are accurate and representative, they are the least trainable choice in building a working system, since it is nearly impossible to get several hundred repetitions for all the words, unless we are using a speech recognizer that is domain specific, such as a recognizer designed for digits only.
- The unit should be *generalizable,* so that any new word can be derived from a predefined unit inventory for task-independent speech recognition. If we have a fixed set of word models, there is no obvious way for us to derive the new word model.

A practical challenge is how to balance these selection criteria for speech recognition. In this section we compare a number of units and point out their strengths and weaknesses in practical applications.

Amazon/VB Assets
Exhibit 1012
Page 454

9.4.1. Comparison of Different Units

What is a unit of language? In English, words are typically considered as a principal carrier of meaning and are seen as the smallest unit that is capable of independent use. As the most natural unit of speech. whole-word models have been widely used for many speech recognition systems. A distinctive advantage of using word models is that we can capture the phonetic coarticulation inherent within these words. When the vocabulary is small, we can create word models that are context dependent.

For example, if the vocabulary is English digits, we can have different word models for the word *one* to represent the word in different contexts. Thus each word model is dependent on its left and right context. If someone says *three one two*, the recognizer uses the word model *one* that specifically depends on the left context *three* and right context *two*. Since the vocabulary is small (10), we need to have only 10*10*10=1000 word models, which is achievable when you collect enough training data. With context-dependent, or even context-independent, word models, a wide range of phonological variations can be automatically accommodated. When these word models are adequately trained, they usually yield the best recognition performance in comparison to other modeling units. Therefore, for small vocabulary recognition, whole-word models are widely used, since they are both *accurate* and *trainable*, and there is no need to be *generalizable*.

While words are suitable units for small-vocabulary speech recognition, they are not a practical choice for large-vocabulary continuous speech recognition. First, each word has to be treated individually, and data cannot be shared across word models; this implies a prohibitively large amount of training data and storage. Second, for some task configurations, the recognition vocabulary may consist of words that never appeared in the training data. As a result, some form of word-model composition technique is required to generate word models. Third, it is very expensive to model interword coarticulation effects or adapt a word-based system for a new speaker, a new channel, or new context usage.

To summarize, word models are *accurate* if enough data are available. Thus, they are *trainable* only for small tasks. They are typically not *generalizable*.

Alternatively, there are only about 50 phones in English, and they can be sufficiently trained with just a few hundred sentences. Unlike word models, phonetic models provide no training problem. Moreover, they are also vocabulary independent by nature and can be trained on one task and tested on another. Thus, phones are more *trainable* and *generalizable*. However, the phonetic model is inadequate because it assumes that a phoneme in any context is identical. Although we may try to say each word as a concatenated sequence of independent phonemes, these phonemes are not produced independently, because our articulators cannot move instantaneously from one position to another. Thus, the realization of a phoneme is strongly affected by its immediately neighboring phonemes. For example, if context-independent phonetic models are used, the same model for *t* must capture various events, such as flapping, unreleased stops, and realizations in */t s/* and */t r/*. Then, if */t s/* is the only context in which *t* occurs in the training, while */t r/* is the only context in the testing,

Amazon/VB Assets
Exhibit 1012
Page 455

the model used is highly inappropriate. While word models are not generalizable, phonetic models overgeneralize and, thus, lead to less accurate models.

A compromise between the word and phonetic model is to use larger units such as *syllables*. These units encompass phone clusters that contain the most variable contextual effects. However, while the central portions of these units have no contextual dependencies, the beginning and ending portions are still susceptible to some contextual effects. There are only about 1200 tone-dependent syllables in Chinese and approximately 50 syllables in Japanese, which makes syllable a suitable unit for these languages. Unfortunately, the large number of syllables (over 30,000) in English presents a challenge in terms of trainability.

9.4.2. Context Dependency

If we make units context dependent, we can significantly improve the recognition accuracy, provided there are enough training data to estimate these context-dependent parameters. Context-dependent phonemes have been widely used for large-vocabulary speech recognition, thanks to its significantly improved accuracy and trainability. A context usually refers to the immediate left and/or right neighboring phones.

A *triphone* model is a phonetic model that takes into consideration both the left and the right neighboring phones. If two phones have the same identity but different left or right contexts, they are considered different triphones. We call different realizations of a phoneme *allophones*. Triphones are an example of allophones.

The left and right contexts used in triphones, while important, are only two of many important contributing factors that affect the realization of a phone. Triphone models are powerful because they capture the most important coarticulatory effects. They are generally much more consistent than context-independent phone models. However, as context-dependent models generally have increased parameters, trainability becomes a challenging issue. We need to balance trainability and accuracy with a number of parameter-sharing techniques.

Modeling interword context-dependent phones is complicated. For example, in the word *speech*, pronounced */s p iy ch/*, both left and right contexts for */p/* and */iy/* are known, while the left context for */s/* and the right context for */ch/* are dependent on the preceding and following words in actual sentences. The juncture effect on word boundaries is one of the most serious coarticulation phenomena in continuous speech, especially with short function words like *the* or *a*. Even with the same left and right context identities, there may be significantly different realizations for a phone at different word positions (the *beginning, middle,* or *end* of a word). For example, the phone */t/* in *that rock* is almost extinct, while the phone */t/* in the middle of *theatrical* sounds like */ch/*. This implies that different word positions have an effect on the realization of the same triphone.

In addition to the context, stress also plays an important role in the realization of a particular phone. Stressed vowels tend to have longer duration, higher pitch, and more intensity, while unstressed vowels appear to move toward a neutral, central *schwa*-like phoneme. Agreement about the phonetic identity of a syllable has been reported to be greater in

Amazon/VB Assets
Exhibit 1012
Page 456

stressed syllables for both humans and automatic phone recognizers. In English, *word-level* stress is referred to as *free stress,* because the stressed syllable can take on any position within a word, in contrast to *bound stress* found in languages such as French and Polish, where the position of the stressed syllable is fixed within a word. Therefore, stress in English can be used as a constraint for lexical access. In fact, stress can be used as a unique feature to distinguish a set of word pairs, such as ***import*** vs. ***import***, and ***export*** vs. ***export***. For example, the phone set used for Whisper, such as */er/-/axr/* and */ah/-/ix/-/ax/*, describes these stressed and unstressed vowels. One example illustrating how stress can significantly affect the realization of phone is demonstrated in Figure 9.3, where phone */t/* in word *Italy* vs. *Italian* is pronounced differently in American English due the location of the stress, albeit the triphone context is identical for both words.

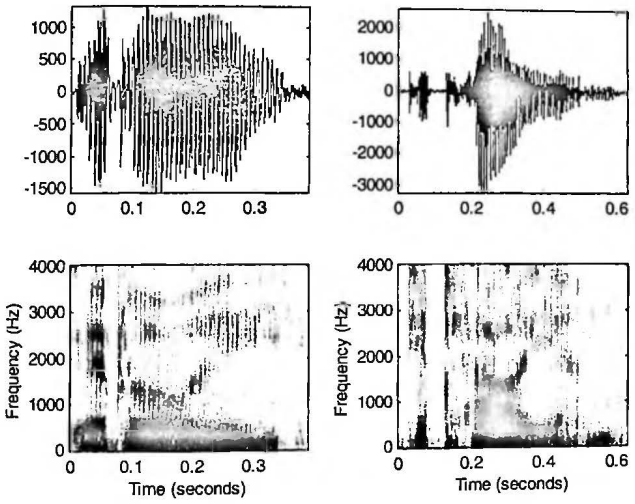


Figure 9.3 The importance of stress is illustrated in *Italy* vs. *Italian* for phone */t/*. The realizations are quite different, even though they share the same left and right context.

Sentence-level stress, on the other hand, represents the overall stress pattern of continuous speech. While sentence-level stress does not change the meaning of any particular lexicon item, it usually increases the relative prominence of portions of the utterance for the purpose of contrast or emphasis. *Contrastive stress* is normally used to coordinate constructions such as *there are **import** records and there are **domestic** ones*, as well as for the purpose of correction, as in *I said **import**, not **export**. Emphatic* stress is commonly used to distinguish a sentence from its negation, e.g., *I did have dinner*. Sentence-level stress is very hard to model without incorporating high-level semantic and pragmatic knowledge. In most state-of- the-art speech recognition systems, only word-level stress is used for creating allophones.

Amazon/VB Assets
Exhibit 1012
Page 457

9.4.3. Clustered Acoustic-Phonetic Units

Triphone modeling assumes that every triphone context is different. Actually, many phones have similar effects on the neighboring phones. The position of our articulators has an important effect on how we pronounce neighboring vowels. For example, /b/ and /p/ are both labial stops and have similar effects on the following vowel, while /r/ and /w/ are both liquids and have similar effects on the following vowel. Contrary to what we illustrate in Figure 9.1, Figure 9.4 illustrates this phenomenon. It is desirable to find instances of similar contexts and merge them. This would lead to a much more manageable number of models that can be better trained.

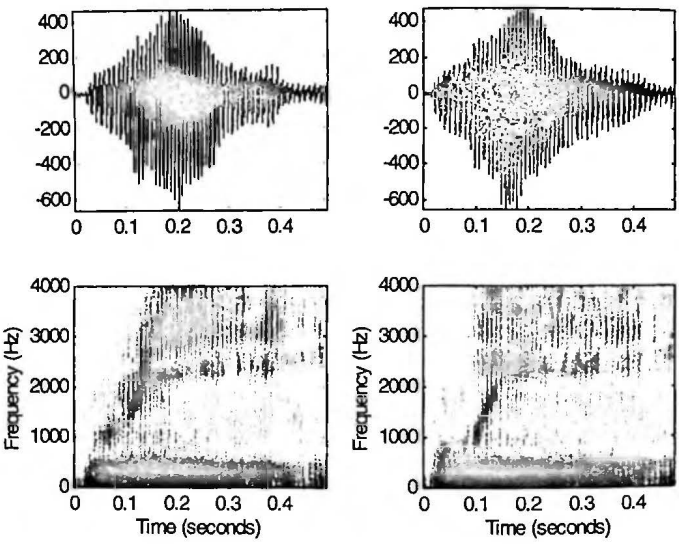


Figure 9.4 The spectrograms for the phoneme /iy/ with two different *left-contexts* are illustrated. Note that /r/ and /w/ have similar effects on /iy/. This illustrates that different left-contexts may have similar effects on a phone.

The trainability and accuracy balance between phonetic and word models can be generalized further to model subphonetic events. In fact, both phonetic and subphonetic units have the same benefits, as they share parameters at the unit level. This is the key benefit in comparison to the word units. Papers by [11, 45, 57, 66, 111] provide examples of the application of this concept to cluster hidden Markov models. For subphonetic modeling, we can treat the state in phonetic HMMs as the basic subphonetic unit. Hwang and Huang further generalized clustering to the state-dependent output distributions across different phonetic models [57]. Each cluster thus represents a set of similar Markov states and is called a

Amazon/VB Assets
Exhibit 1012
Page 458

senone [56]. A subword model is thus composed of a sequence of senones after the clustering is finished. The optimal number of senones for a system is mainly determined by the available training corpus and can be tuned on a development set.

Each allophone model is an HMM made of states, transitions, and probability distributions. To improve the reliability of the statistical parameters of these models, some distributions can be tied. For example, distributions for the central portion of an allophone may be tied together to reflect the fact that they represent the stable (context-independent) physical realization of the central part of the phoneme, uttered with a stationary configuration of the vocal tract. Clustering at the granularity of the state rather than the entire model can keep the dissimilar states of two models apart while the other corresponding states are merged, thus leading to better parameter sharing.

Figure 9.5 illustrates how state-based clustering can lead to improved representations. These two HMMs come from the same phone class with a different right context, leading to very different output distributions in the last state. As the left contexts are identical, the first and second output distributions are almost identical. If we measure the overall model similarity based on the accumulative overall output distribution similarities of all states, these two models may be clustered, leading to a very inaccurate distribution for the last state. Instead, we cluster the first two output distributions while leaving the last one intact.

There are two key issues in creating trainable context-dependent phonetic or subphonetic units:

- We need to enable better parameter sharing and smoothing. As Figure 9.4 illustrates, many phones have similar effects on neighboring phones. If the acoustic realization is indeed identical, we tie them together to improve trainability and efficiency.
- Since the number of triphones in English is very large (over 100,000), there are many new or unseen triphones that are in the test set but not in the training set. It is important to map these unseen triphones into appropriately trained triphones.

As discussed in Chapter 4, a decision tree is a binary tree to classify target objects by asking binary questions in a hierarchical manner. Modeling unseen triphones is particularly important for *vocabulary independence*, since it is difficult to collect a training corpus which covers enough occurrences of every possible subword unit. We need to find models that are accurate, trainable, and especially generalizable. The senonic decision tree classifies Markov states of triphones represented in the training corpus by asking linguistic questions composed of conjunctions, disjunctions, and/or negations of a set of predetermined simple categorical linguistic questions. Examples of these simple categorical questions are: *Is the left-context phone a fricative? Is the right-context phone a front vowel?* The typical question set used in Whisper to generate the senone tree is shown in Table 9.3. So, for each node in the tree, we check whether its left or right phone belongs to one of the categories. As discussed in Chapter 4, we measure the corresponding entropy reduction or likelihood increase for each question and select the question that has the largest entropy decrease to split the node.

Amazon/VB Assets
Exhibit 1012
Page 459

Thus, the tree can be automatically constructed by searching, for each node, the question that renders the maximum entropy decrease. Alternatively, complex questions can be formed for each node for improved splitting.

When we grow the tree, it needs to be pruned using cross-validation as discussed in Chapter 4. When the algorithm terminates, the leaf nodes of the tree represent the senones to be used. Figure 9.6 shows an example tree we built to classify the second state of all */k/* triphones seen in a training corpus. After the tree is built, it can be applied to the second state of *any /k/* triphone, thanks to the generalizability of the binary tree and the general linguistic questions. Figure 9.6 indicates that the second state of the */k/* triphone in *welcome* is mapped to the second senone, no matter whether this triphone occurs in the training corpus or not.

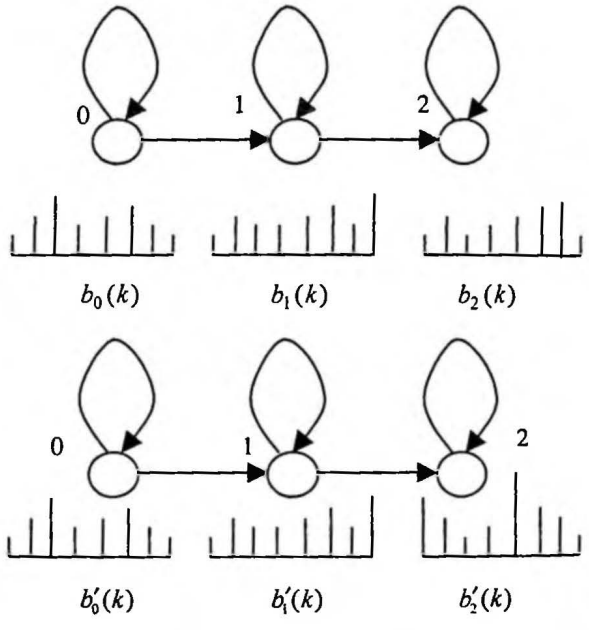


Figure 9.5 State-based vs. model-based clustering. These two models are very similar, as both the first and the second output distributions are almost identical. The key difference is the output distribution of the third state. If we measure the overall model similarity, which is often based on the accumulative output distribution similarities of all states, these two models may be clustered, leading to a very inaccurate distribution for the last state. If we cluster output distributions at state level, we can cluster the first two output distributions while leaving the last ones intact, leading to more accurate representations.

Amazon/VB Assets
Exhibit 1012
Page 460

Table 9.3 Some example questions used in building senone trees.

Questions	Phones in Each Question Category
Aspseg	*hh*
Sil	*sil*
Alvstp	*d t*
Dental	*dh th*
Labstp	*b p*
Liquid	*l r*
Lw	*l w*
S/Sh	*s sh*
Sylbic	*er axr*
Velstp	*g k*
Affric	*ch jh*
Lqgl-B	*l r w*
Nasal	*m n ng*
Retro	*r er axr*
Schwa	*ax ix axr*
Velar	*ng g k*
Fric2	*th s sh f*
Fric3	*dh z zh v*
Lqgl	*l r w y*
S/Z/Sh/Zh	*s z sh zh*
Wglide	*uw aw ow w*
Labial	*w m b p v*
Palatl	*y ch jh sh zh*
Yglide	*iy ay ey oy y*
High	*ih ix iy uh uw y*
Lax	*eh ih ix uh ah ax*
Low	*ae aa ao aw ay oy*
Orstp2	*p t k*
Orstp3	*b d g*
Alvelr	*n d t s z*
Diph	*uw aw ay ey iy ow oy*
Fric1	*dh th s sh z zh v f*
Round	*uh ao uw ow oy w axr er*
Frnt-R	*ae eh ih ix iy ey ah ax y aw*
Tense	*iy ey ae uw ow aa ao ay oy aw*
Back-L	*uh ao uw ow aa er axr l r w aw*
Frnt-L	*ae eh ih ix iy ey ah ax y oy ay*
Back-R	*uh ao uw ow aa er axr oy l r w ay*
Orstp1	*b d g p t k ch jh*
Vowel	*ae eh ih ix iy uh ah ax aa ao uw aw ay ey ow oy er axr*
Son	*ae eh ih ix iy ey ah ax oy ay uh ao uw ow aa er axr aw l r w y*
Voiced	*ae eh ih ix iy uh ah ax aa ao uw aw ay ey ow oy l r w y er axr m n ng jh b d dh g v z zh*

Amazon/VB Assets
Exhibit 1012
Page 461

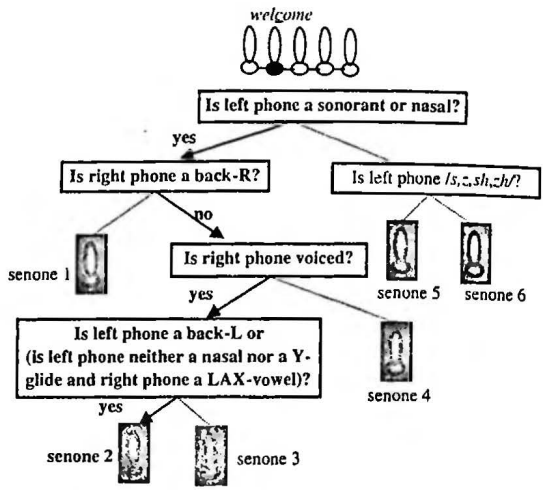


Figure 9.6 A decision tree for classifying the second state of K-triphone HMMs [48].

In practice, senone models significantly reduce the word recognition error rate in comparison with model-based clustered triphone models, as illustrated in Table 9.4. It is the senonic model's significant reduction of the overall system parameters that enables the continuous mixture HMMs to perform well for large-vocabulary speech recognition [56].

Table 9.4 Relative error reductions for different modeling units. The reduction is relative to that of the preceding row.

Units	Relative Error Reductions
Context-independent phone	Baseline
Context-dependent phone	+25%
Clustered triphone	+15%
Senone	+24%

9.4.4. Lexical Baseforms

When appropriate subword units are used, we must have the correct pronunciation for each word so that concatenation of subword units can accurately represent the word to be recognized. The dictionary represents the standard pronunciation used as a starting point for building a workable speech recognition system. We also need to provide alternative pronunciations for words such as *tomato* that may have very different pronunciations. For example, the COMLEX dictionary from LDC has about 90,000 baseforms that cover most words used in many years of *The Wall Street Journal*. The CMU Pronunciation Dictionary, which was optimized for continuous speech recognition, has about 100,000 baseforms.

Amazon/VB Assets
Exhibit 1012
Page 462

In continuous speech recognition, we must also use phonologic rules to modify inter-word pronunciations or to have reduced sounds. Assimilation is a typical coarticulation phenomenon—a change in a segment to make it more like a neighboring segment. Typical examples include phrases such as *did you /d ih jh y ah/, set you /s eh ch er/, last year / l ae s ch iy r/, because you've /b iy k ah zh uw v/,* etc. Deletion is also common in continuous speech. For example, /t/ and /d/ are often deleted before a consonant. Thus, in conversational speech, you may find examples like *find him /f ay n ix m/, around this /ix r aw n ih s/,* and *Let me in /l eh m eh n/.*

Dictionaries often don't include proper names. For example, the 20,000 names included in the COMLEX dictionary are a small fraction of 1–2 million names in the USA. To deal with these new words, we often have to derive their pronunciation automatically. These new words have to be added on the fly, either by the user or through an interface from speech-aware applications. Unlike Spanish or Italian, rule-based letter-to-sound (LTS) conversion for English is often impractical, since so many words in English don't follow phonological rules. A trainable LTS converter is attractive, since its performance can be improved by constantly learning from examples so that it can generalize rules for the specific task. Trainable LTS converters can be based on neural networks, HMMs, or the CART described in Chapter 4. In practice, CART-based LTS has a very accurate performance [10, 61, 71, 89].

When CART is used, the basic YES-NO question for LTS conversion looks like: *Is the second right letter 'p'?* or: *Is the first left output phone /ay/?* The question for letters and phones can be on either the left or the right side. The range of question positions should be long enough to cover the most important phonological variations. Empirically, a 10-letter window (5 for left letter context and 5 for right letter context) and 3-phone window context is generally sufficient. A primitive set of questions can include all the singleton questions about each letter or phone identity. If we allow the node to have a complex question—that is, a combination of primitive questions—the depth of the tree can be greatly reduced and performance improved. For example, a complex question: *Is the second left letter 't' and the first left letter 'i' and the first right letter 'n'?* can capture *o* in the common suffix *tion* and convert it to the correct phone. Complex questions can also alleviate possible data-fragmentation problems caused by the greedy nature of the CART algorithm.

Categorical questions can be formed in both the letter and phone domains with our common linguistic knowledge. For example, the most often used set includes the letter or phone clusters for vowels, consonants, nasals, liquids, fricatives, and so on. In growing the decision tree, the context distance also plays a major role in the overall quality. It is very important to weight the entropy reduction according to the distance (either letter or phoneme) to avoid overgeneralization, which forces the tree to look more carefully at the *nearby* context than at the *far-away* context. Each leaf of the tree has a probability distribution for letter-to-phoneme mapping.

There are a number of ways to improve the effectiveness of the decision tree. First, pruning controls the tree's depth. For example, certain criteria have to be met for a node to be split. Typically splitting requires a minimum number of counts and a minimum entropy reduction. Second, the distribution at the leaves can be smoothed. For example, a leaf distri-

Amazon/VB Assets
Exhibit 1012
Page 463

bution can be interpolated with the distributions of its ancestor nodes using deleted-interpolation. Finally, we can partition the training data and build multiple trees with different prediction capabilities. These trees accommodate different phonological rules with different language origins.

When the decision tree is used to derive the phonetic pronunciation, the phonetic conversion error is about 8% for the *Wall Street Journal* newspaper text corpora [61]. These errors can be broadly classified into two categories. The first includes errors of proper nouns and foreign words. For example, *Pacino* can be mistakenly converted to */p ax s iy n ow/* instead of */p ax ch iy n ow/*. The second category includes generalization errors. For example, *shier* may be converted to */sh ih r/* instead of the correct pronunciation */sh ay r/* if the word *cashier /k ae sh ih r/* appears in the training data. The top three phone confusion pairs are */ix/ax/*, */dx/t/*, and */ae/ax/*. The most confusing pair is */ix/ax/*. This is not surprising, because */ix/ax/* is among the most inconsistent transcriptions in most of the published dictionaries. There is no consensus for */ix/ax/* transcription among phoneticians.

Although automatic LTS conversion has a reasonable accuracy, it is hardly practical if you don't use an exception dictionary. This is especially true for proper nouns. In practice, you can often ask the person who knows how to pronounce the word to either speak or write down the correct phonetic pronunciation, updating the exception dictionary if the correct one disagrees with what the LTS generates. When acoustic examples are available, you can use the decision tree to generate multiple results and use these results as a language model to perform phone recognition on the acoustic examples. The best overall acoustic and LTS probability can be used as the most likely candidate in the exception dictionary. Since there may be many ways to pronounce a word, you can keep multiple pronunciations in the dictionary with a probability for each possible one. If the pronunciation probability is inaccurate, an increase in multiple pronunciations essentially increases the size and confusion of the vocabulary, leading to increased speech recognition error rate.

Even if you have accurate phonetic baseforms, pronunciations in spontaneous speech differ significantly from the standard baseform. Analysis of manual phonetic transcription of conversational speech reveals a large number (> 20%) of cases of genuine ambiguity: instances where human labelers disagree on the identity of the surface form [95]. For example, the word *because* has more than 15 different pronunciation variations, such as */b iy k ah z/, /b ix k ah z/, /k ah z/, /k ax z/, /b ix k ax z/, /b ax k ah z/, /b ih k ah z/, /k s/, /k ix z/, /k ih z/,/b iy k ah s/, /b iy k ah/, /b iy k ah zh/, /ax z/*, etc., in the context of conversational speech [39]. To characterize the acoustic evidence in the context of this ambiguity, you can partly resolve the ambiguity by deriving a suitable phonetic baseform from speech data [29, 95, 97]. This is because the widespread variation can be due either to a lexical fact (such as that the word *because* can be *'cause* in informal speech) or to the dialect differences. African American vernacular English has many vowels different from general American English.

To incorporate widespread pronunciations, we can use a probabilistic finite state machine to model each word's pronunciation variations, as shown in Figure 9.7. The probability with each arc indicates how likely that path is to be taken, with all the arcs that leave a node summing to 1. As with HMMs, these weights can be estimated from a real corpus for

Amazon/VB Assets
Exhibit 1012
Page 464

improved speech recognition [20, 85, 102, 103, 110]. In practice, the relative error reduction of using probabilistic finite state machines is very modest (5–10%).

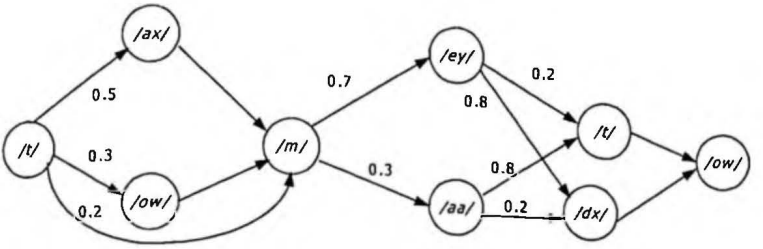


Figure 9.7 A possible pronunciation network for word *tomato*. The vowel */ey/* is more likely to flap, thereby having a higher transition probability into */dx/*.

9.5. ACOUSTIC MODELING—SCORING ACOUSTIC FEATURES

After feature extraction, we have a sequence of feature vectors, **X**, such as the MFCC vector, as our input data. We need to estimate the probability of these acoustic features, given the word or phonetic model, **W**, so that we can recognize the input data for the correct word. This probability is referred to as acoustic probability, $P(\mathbf{X} \mid \mathbf{W})$. In this section we focus our discussion on the HMM. As discussed in Chapter 8, it is the most successful method for acoustic modeling. Other emerging techniques are discussed in Section 9.8.

9.5.1. Choice of HMM Output Distributions

As discussed in Chapter 8, you can use discrete, continuous, or semicontinuous HMMs. When the amount of training data is sufficient, parameter tying becomes unnecessary. A continuous model with a large number of mixtures offers the best recognition accuracy, although its computational complexity also increases linearly with the number of mixtures. On the other hand, the discrete model is computationally efficient, but has the worst performance among the three models. The semicontinuous model provides a viable alternative between system robustness and trainability.

When either the discrete or the semicontinuous HMM is employed, it is helpful to use multiple codebooks for a number of features for significantly improved performance. Each codebook then represents a set of different speech parameters. One way to combine these multiple output observations is to assume that they are independent, computing the output probability as the product of the output probabilities of each codebook. For example, the semicontinuous HMM output probability of multiple codebooks can be computed as the product of each codebook:

Amazon/VB Assets
Exhibit 1012
Page 465

$$b_i(\mathbf{x}) = \prod_m \sum_{k=1}^{L^m} f^m(\mathbf{x}^m \mid o_k^m) b_i^m(o_k^m) \tag{9.12}$$

where superscript m denotes the codebook-m related parameters. Each codebook consists of L^m-mixture continuous density functions.

Following our discussion in Chapter 8, the re-estimation algorithm for the multiple-codebook-based HMM could be extended. Since multiplication of the output probability density of each codebook leads to several independent terms in the Q-function, for codebook m, $\zeta_t(j,k^m)$ can be modified as follows:

$$\zeta_t(j,k^m) = \frac{\sum_i \alpha_{t-1}(i) a_{ij} b_j^m(k^m) f^m(\mathbf{x}_t \mid v_k^m) \prod_{m \neq n} \sum_k b_j^n(k^n) f^n(\mathbf{x}_t \mid v_k^n) \beta_t(j)}{\sum_k \alpha_T^m(k)} \tag{9.13}$$

Other intermediate probabilities can also be computed in a manner similar to what we discussed in Chapter 8.

Multiple codebooks can dramatically increase the representation power of the VQ codebook and can substantially improve speech recognition accuracy. You can typically build a codebook for $\mathbf{c}_k$, $\Delta\mathbf{c}_k$, and $\Delta\Delta\mathbf{c}_k$, respectively. As energy has a very different dynamic range, you can further improve the performance by building a separate codebook for $c_k[0]$, $\Delta c_k[0]$, and $\Delta\Delta c_k[0]$. In comparison to building a single codebook for $\mathbf{x}_k$ as illustrated in Eq. (9.6), the multiple-codebook system can reduce the error rate by more than 10%.

In practice, the most important parameter for the output probability distribution is the number of mixtures or the size of the codebooks. When there are sufficient training data, relative error reductions with respect to the discrete HMM are those shown in Figure 9.8.

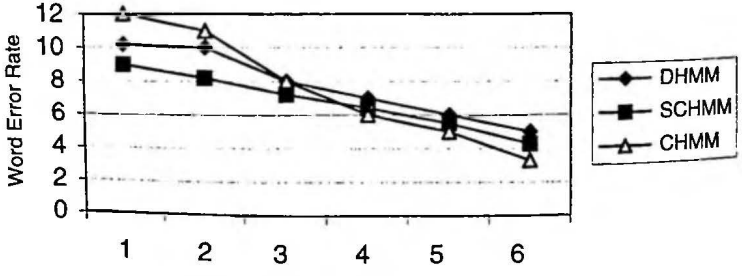


Figure 9.8 Continuous speaker-independent word recognition error rates of the discrete HMM (DHMM), SCHMM, and the continuous HMM (CHMM) with respect to the training set sizes (thousands of training sentences). Both the DHMM and SCHMM have multiple codebooks. The CHMM has 20 mixture diagonal Gaussian density functions.

Amazon/VB Assets
Exhibit 1012
Page 466

As you can see from Figure 9.8, the SCHMM offers improved accuracy in comparison with the discrete HMM or the continuous HMM when the amount of training data is limited. When we increase the training data size, the continuous mixture density HMM starts to outperform both the discrete and the semicontinuous HMM, since the need to share model parameters becomes less critical.

Performance is also a function of the number of mixtures. With a small number of mixtures, the continuous HMM lacks the modeling power and it actually performs worse than the discrete HMM across the board. Only after we dramatically increase the number of mixtures does the continuous HMM start to offer improved recognition accuracy. The SCHMM can typically reduce the discrete HMM error rate by 10–15% across the board. The continuous HMM with 20 diagonal Gaussian density functions performed worse than either the discrete or the SCHMM when the size of training data was small. It outperformed either the discrete HMM or the SCHMM when sufficient amounts of training data became available. When the amount of training data is sufficiently large, it can reduce the error rate of the semicontinuous HMM by 15–20%.

9.5.2. Isolated vs. Continuous Speech Training

If we build a word HMM for each word in the vocabulary for isolated speech recognition, the training or recognition can be implemented directly, using the basic algorithms introduced in Chapter 8. To estimate model parameters, examples of each word in the vocabulary are collected. The model parameters are estimated from all these examples using the forward-backward algorithm and the reestimation formula. It is not necessary to have precise end-point detection, because the silence model automatically determines the boundary if we concatenate silence models with the word model in both ends.

If subword units,[3] such as phonetic models, are used, we need to share them across different words for large-vocabulary speech recognition. These subword units are concatenated to form a word model, possibly adding silence models at the beginning and end, as illustrated in Figure 9.9.

To concatenate subword units to form a word model, you can have a null transition from the final state of the previous subword HMM to the initial state of the next subword HMM, as indicated by the dotted line in Figure 9.9. As described in Chapter 8, you can estimate the parameters of the concatenated HMM accordingly. Please notice that the added null transition arc should satisfy the probability constraint with the transition probability of each phonetic HMM. The self-loop transition probability of the last state in each individual HMM has the topology illustrated in Figure 9.9. If we estimate these parameters with the concatenated model, the null arc transition probability, a_{ij}^{ε}, should satisfy the constraint $\sum_j (a_{ij} + a_{ij}^{\varepsilon}) = 1$ such that the self-loop transition probability of the last state is no longer equal to 1. For interword concatenation or concatenation involving multiple pronunciations, you can use multiple null arcs to concatenate individual models together.

[3] We have a detailed discussion on word models vs. subword models in Section 9.4.1.

Amazon/VB Assets
Exhibit 1012
Page 467

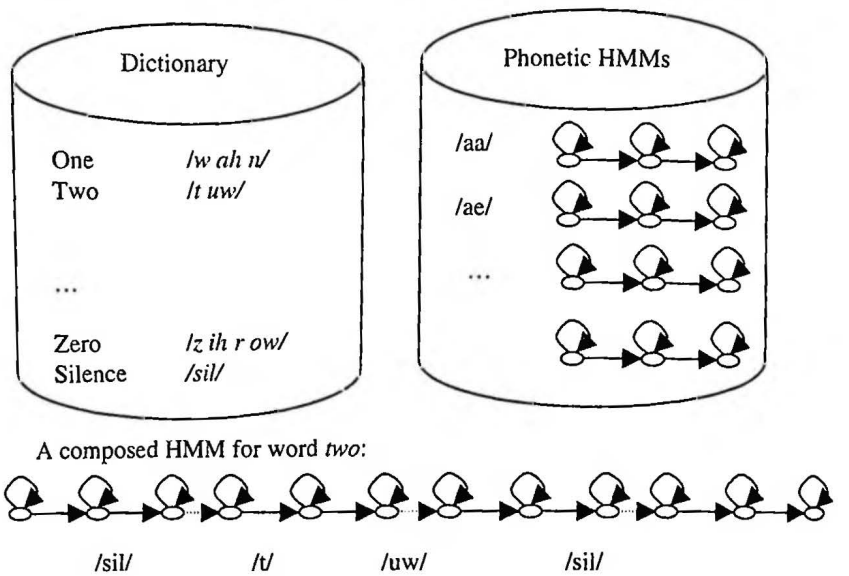


Figure 9.9 The construction of an isolated word model by concatenating multiple phonetic models based on the pronunciation dictionary.

In the example given in Figure 9.9, we have ten English digits in the vocabulary. We build an HMM for each English phone. The dictionary provides information on each word's pronunciation. We have a special word, *Silence*, that maps to a */sil/* HMM that has the same topology as the standard phonetic HMM. For each word in the vocabulary we first derive the phonetic sequence for each word from the dictionary. We link these phonetic models together to form a word HMM for each word in the vocabulary. The link between two phonetic models is shown in the figure as the dotted arrow.

For example, for word *two*, we create a word model based on the beginning silence */sil/*, phone */t/*, phone */uw/*, and ending silence */sil/*. The concatenated word model is then treated in the same manner as a standard large composite HMM. We use the standard forward-backward algorithm to estimate the parameters of the composite HMM from multiple sample utterances of the word *two*. After several iterations, we automatically get the HMM parameters for */sil/*, */t/*, and */uw/*. Since a phone can be shared across different words, the phonetic parameters may be estimated from acoustic data in different words.

The ability to automatically align each individual HMM to the corresponding unsegmented speech observation sequence is one of the most powerful features in the forward-backward algorithm. When the HMM concatenation method is used for continuous speech, you need to compose multiple words to form a sentence HMM based on the transcription of the utterance. In the same manner, the forward-backward algorithm absorbs a range of pos-

Amazon/VB Assets
Exhibit 1012
Page 468

sible word boundary information of models automatically. There is no need to have a precise segmentation of the continuous speech.

In general, to estimate the parameters of the HMM, each word is instantiated with its concatenated word model (which may be a concatenation of subword models). The words in the sentence are concatenated with optional silence models between them. If there is a need to modify interword pronunciations due to interword pronunciation change, such as *want you*, you can add a different optional phonetic sequence for *t-y* in the concatenated sentence HMM.

In the digit recognition example, if we have a continuous training utterance *one three*, we compose a sentence HMM, as shown in Figure 9.10, where we have an optional silence HMM between the words *one* and *three*, linked with a null transition from the last state of the word model *one* to the first state of the word model *three*. There is also a direct null arc connection between the models *one* and *three* because a silence may not exist in the training example. These optional connections ensure that all the possible acoustic realizations of the natural continuous speech are considered, so that the forward-backward algorithm can automatically discover the correct path and accurately estimate the corresponding HMM from the given speech observation.

In general, the concatenated sentence HMM can be trained using the forward-backward algorithm with the corresponding observation sequence. Since the entire sentence HMM is trained on the entire observation sequence for the corresponding sentence, most possible word boundaries are inherently considered. Parameters of each model are based on those state-to-speech alignments. It does not matter where the word boundaries are. Such a training method allows complete freedom to align the sentence model against the observation, and no explicit effort is needed to find word boundaries.

In speech decoding, a word may begin and end anywhere within a given speech signal. As word boundaries cannot be detected accurately, all possible beginning and end points have to be accounted for. This converts a linear search (as for isolated word recognition) to a tree search, and a polynomial recognition algorithm to an exponential one. How to design an efficient decoder is discussed in Chapters 12 and 13.

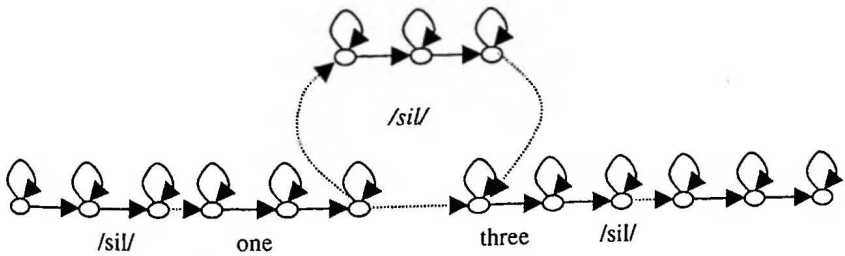


Figure 9.10 A composite sentence HMM. Each word can be a word HMM or a composite phonetic word HMM, as illustrated in Figure 9.9.

Amazon/VB Assets
Exhibit 1012
Page 469

9.6. ADAPTIVE TECHNIQUES—MINIMIZING MISMATCHES

As Figure 1.2 illustrated, it is important to adapt both acoustic models and language models for new situations. A decent model can accommodate a wide range of variabilities. However, the mismatch between the model and operating conditions always exists. One of the most important factors in making a speech system usable is to minimize the possible mismatch dynamically with a small amount of *calibration data*. Adaptive techniques can be used to modify system parameters to better match variations in microphone, transmission channel, environment noise, speaker, style, and application contexts. As a concrete example, speaker-dependent systems can provide a significant word error-rate reduction in comparison to speaker-independent systems if a large amount of speaker-dependent training data exists [50]. Speaker-adaptive techniques can bridge the gap between these two configurations with a small fraction of the speaker-specific training data needed to build a full speaker-dependent system. These techniques can also be used incrementally as more speech is available from a particular speaker. When speaker-adaptive models are built, you can have not only improved accuracy but also improved speed and potentially reduced model parameter sizes because of accurate representations, which is particularly appealing for practical speech recognition.

There are a number of ways to use adaptive techniques to minimize mismatches. You can have a nonintrusive adaptation process that works in the background all the time. This is typically unsupervised, using only the outcome of the recognizer (with a high confidence score, as discussed in Section 9.7) to guide the model adaptation. This approach can continuously modify the model parameters so that any nonstationary mismatches can be eliminated. As discussed in Chapter 13, systems that are required to transcribe speech in a non-real-time fashion may use multiple recognition passes. You can use unsupervised adaptation on the test data to improve the models after each pass to improve performance for a subsequent recognition pass.

Since the use of recognition results may be imperfect, there is a possibility of divergence if the recognition error rate is high. If the error rate is low, the adaptation results may still not be as good as supervised adaptation in which the correct transcription is provided for the user to read, a process referred to as the *enrollment process*. In this process you can check a wide range of parameters as follows:

- Check the background noise by asking the user not to speak.
- Adjust the microphone gain by asking the user to speak normally.
- Adapt the acoustic parameters by asking the user to read several sentences.
- Change the decoder parameters for the best speed with no loss of accuracy.
- Compose dynamically new enrollment sentences based on the user-specific error patterns.

The challenge for model adaptation is that we can use only a small amount of observable data to modify model parameters. This constraint requires different modeling strategies

Amazon/VB Assets
Exhibit 1012
Page 470

from the ones we discussed in building the baseline system, as the amount of training data is generally sufficient for offline training. In this section we focus on a number of adaptive techniques that can be applied to compensate either speaker or environment variations. Most of these techniques are model-based, since the acoustic model parameters rather than the acoustic feature vectors are adapted. We use speaker-adaptation examples to illustrate how these techniques can be used to improve system performance. We can generalize to environment adaptation by using environment-specific adaptation data and a noise-compensation model, which we discuss in Chapter 10. In a similar manner, we can modify the language model as discussed in Chapter 11.

9.6.1. Maximum a Posteriori (MAP)

Maximum a posteriori (MAP) estimation, as discussed in Chapter 4, can effectively deal with data-sparse problems, as we can take advantage of prior information about existing models. We can adjust the parameters of pretrained models in such a way that limited new training data would modify the model parameters guided by the prior knowledge to compensate for the adverse effect of a mismatch [35]. The prior density prevents large deviations of the parameters unless the new training data provide strong evidence.

More specifically, we assume that an HMM is characterized by a parameter vector Φ that is a random vector, and that prior knowledge about the random vector is available and characterized by a prior probability density function $p(\Phi)$, whose parameters are to be determined experimentally.

With the observation data $\mathbf{X}$, the MAP estimate is expressed as follows:

$$\hat{\Phi} = \arg\max_{\Phi}[p(\Phi \mid \mathbf{X})] = \arg\max_{\Phi}[p(\mathbf{X} \mid \Phi)p(\Phi)] \tag{9.14}$$

If we have no prior information, $p(\Phi)$ is the uniform distribution, and the MAP estimate becomes identical to the ML estimate. We can use the EM algorithm as the ML to estimate the parameters of HMMs. The corresponding Q-function can be defined as:

$$Q_{MAP}(\Phi, \hat{\Phi}) = \log p(\hat{\Phi}) + Q(\Phi, \hat{\Phi}) \tag{9.15}$$

The EM algorithm for the ML criterion can be applied here directly. The actual expression depends on the assumptions made about the prior density. For the widely used continuous Gaussian mixture HMM, there is no joint conjugate prior density. We can assume different components of the HMM to be mutually independent, so that the optimization can be split into different subproblems involving only a single component of the parameter set. For example, the prior density function for the mixture Gaussian can be as follows:

$$p_{h_i}(\mathbf{c}_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = p_{c_i}(\mathbf{c}_i)\prod_k p_{b_{ik}}(\boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik}) \tag{9.16}$$

where $p_{c_i}(\mathbf{c}_i)$ is a Dirichlet prior density for the mixing coefficient vector of all mixture components in the Markov state i, and $p_{b_{ik}}(\boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik})$ denotes the prior density for parameters

Amazon/VB Assets
Exhibit 1012
Page 471

of the kth Gaussian component in the state i. The Dirichlet prior density $p_{c_i}(\mathbf{c}_i)$ is characterized by a vector v_i of positive hyperparameters such that:

$$p_{c_i}(\mathbf{c}_i) \propto \prod_k c_{ik}^{v_{ik}-1} \tag{9.17}$$

For full covariance D-dimensional Gaussian densities, the prior density can be a normal-Wishart density parameterized by two values $\eta > D-1, \tau > 0$, the vector $\boldsymbol{\mu}_{m_{ik}}$, and the symmetric positive definite matrix S as follows:

$$p_{b_{ik}}(\boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik}) \propto$$
$$\sqrt{\det(\boldsymbol{\Sigma}_{ik})^{D-\eta}} \exp\left(-\frac{\tau}{2}(\boldsymbol{\mu}_{ik}-\boldsymbol{\mu}_{m_{ik}})\boldsymbol{\Sigma}_{ik}^{-1}(\boldsymbol{\mu}_{ik}-\boldsymbol{\mu}_{m_{ik}})' - \frac{1}{2}tr(\mathbf{S}\boldsymbol{\Sigma}_{ik}^{-1})\right) \tag{9.18}$$

We can apply the same procedure as the MLE Baum-Welch reestimation algorithm. For example, with the Q-function defined in Eq. (9.15), we can apply the Lagrange method to derive the mixture coefficients as follows:

$$\begin{cases} \dfrac{\partial}{\partial \hat{c}_{ik}}(\log p_{c_i}(\hat{\mathbf{c}}_i) + \sum_k \sum_t \xi_t(i,k)\log \hat{c}_{ik}) + \lambda = 0, \forall k \\ \sum_k \hat{c}_{ik} = 1 \end{cases} \tag{9.19}$$

Based on Eqs. (9.17) and (9.19), the solution is:

$$\hat{c}_{ik} = \frac{v_{ik} - 1 + \sum_t \xi_t(i,k)}{\sum_l (v_{il} - 1 + \sum_t \xi_t(i,l))} \tag{9.20}$$

A comparison between Eq. (9.20) and the ML estimate Eq. (8.58) shows that the MAP estimate is a weighted average between the mode of the prior density and the ML estimate with proportions given by $v_{ijk} - 1$ and $\sum \xi_t(i,k)$, respectively.

We can optimize Eq. (9.15) with respect to mean and covariance parameters in a similar fashion. For example, the solution of these estimates is:

$$\hat{\boldsymbol{\mu}}_{ik} = \frac{\tau_{ik}\boldsymbol{\mu}_{m_{ik}} + \sum_{t=1}^{T} \zeta_t(i,k)\mathbf{x}_t}{\tau_{ik} + \sum_{t=1}^{T} \zeta_t(i,k)} \tag{9.21}$$

$$\hat{\boldsymbol{\Sigma}}_{ik} = \frac{\mathbf{S}_{ik} + \tau_{ik}(\hat{\boldsymbol{\mu}}_{ik} - \boldsymbol{\mu}_{m_{ik}})(\hat{\boldsymbol{\mu}}_{ik} - \boldsymbol{\mu}_{m_{ik}})' + \sum_{t=1}^{T} \zeta_t(i,k)(\mathbf{x} - \hat{\boldsymbol{\mu}}_{ik})(\mathbf{x} - \hat{\boldsymbol{\mu}}_{ik})'}{\eta_{ik} - D + \sum_{t=1}^{T} \zeta_t(i,k)} \tag{9.22}$$

Amazon/VB Assets
Exhibit 1012
Page 472

where τ_{ik} is the parameter in the normal-gamma density for the corresponding state i.

Thus, the reestimation formula for the Gaussian mean is a weighted sum of the prior mean with the ML mean estimate $\sum_{t=1}^{T}\zeta_t(i,k)\mathbf{x}_t / \sum_{t=1}^{T}\zeta_t(i,k)$. τ_{ik} is a balancing factor between prior mean and the ML mean estimate. When τ_{ik} is large, the value of the prior knowledge is small and the value of the mean $\mu_{mw_{ik}}$ is assumed to have high certainty, leading to the dominance of the final estimate. When the amount of adaptation data increases, the MAP estimate approaches the ML estimate, as the adaptation data overwrite any important prior that may influence the final estimate. Similarly, the covariance estimation formula has the same interpretation of the balance between the prior and new data.

One major limitation of the MAP-based approach is that it requires an accurate initial guess for the prior $p(\Phi)$, which is often difficult to obtain. We can use the already trained initial models that embody some characteristics of the original training conditions. A typical way to generate an initial Gaussian prior is to cluster the initial training data based on speaker or environment similarity measures. We can derive a set of models based on the partition, which can be seen as a set of observations drawn from a distribution having $p(\Phi)$. We can, thus, estimate the prior based on the sample moments to derive the corresponding prior parameters.

Another major limitation is that the MAP-based approach is a local approach to updating the model parameters. Namely, only model parameters that are observed in the adaptation data can be modified from the prior value. When the system has a large number of free parameters, the adaptation can be very slow. Thus in practice we need to find correlations between the model parameters, so that the unobserved or poorly adapted parameters can be altered [3, 22]. Another possibility is to impose structural information so the model parameters can be shared for improved adaptation speed [96].

The MAP training can be iterative, too, which requires an initial estimate of model parameters. A careful initialization for the Gaussian densities is also very important. Unlike the discrete distributions, there is no such a thing as a uniform density for a total lack of information about the value of the parameters. We need to use the same initialization procedure as discussed in Chapter 8.

For speaker-adaptive speech recognition, it has been experimentally found that τ_{ik} can be a fixed constant value for all the Gaussian components across all the dimensions. Thus the MAP HMM can be regarded as an interpolated model between the speaker-independent and speaker-dependent HMM. Both are derived from the standard ML forward-backward algorithm. Experimental performance of MAP training is discussed in Section 9.6.3.

9.6.2. Maximum Likelihood Linear Regression (MLLR)

When the continuous HMM is used for acoustic modeling, the most important parameter set to adapt is the output Gaussian density parameters, i.e., the mean vector and the covariance matrix. We can use a set of linear regression transformation functions to map both means

Amazon/VB Assets
Exhibit 1012
Page 473