6.0 FACIAL EXPRESSION RECOGNITION USING HIDDEN MARKOV MODELS

If we try to build a signal model that can be used to explain and characterize the occurrence of the observable symbol sequences, then we can use this model to identify or recognize other sequences of observable symbols. A Hidden Markov Model (HMM) can be employed to represent the statistical behavior of an observable symbol sequence in terms of a network of states. For each observable symbol, the process being modeled occupies one of the states of the HMM. With each observable symbol, the HMM either stays in the same state or moves to another state based on a set of state transition probability associated with the state. The variety of the observable symbols for which the HMM uses a particular state is described in terms of the distribution of probability that each observable symbol will occur from that state. Thus, an HMM is a doubly (observable and hidden) stochastic model where the observable symbol probability distribution for each state captures the intra-state variability of the observable symbols, and the state transition probability describe the underling dynamic structure of the observable symbols.

We use HMMs to recognize subtly different facial expressions because of their simplicity and reliability. The HMM uses only three parameters: the initial state probability vector, the state-transition probability matrix, and the observable symbol probability matrix. The convergence of recognition computation may run in real time. Analysis of dynamic images naturally will yield more accurate recognition than that of a single static image, in our study, facial expressions are recognized in the context of entire image sequences of arbitrary lengths. Use of an HMM for facial expression recognition is advantageous because it is analogous to human performance which is a doubly stochastic process, involving a hidden immeasurable human mental state and measurable, observable human action. An HMM can produce satisfactory performance in the spatio-temporal domain and deal with the time warping problem. In addition, an HMM may allow for

multiple input sequences. This will result in a reliable recognition system as it will include a variety of extracted information from the facial expressions. It may also be used in combination for both expression recognition and speech recognition.

6.1 Preprocessing of Hidden Markov Models: Vector Quantization

In order to model various "expression units" of individual AUs or AU combinations for recognizing subtly different facial expressions, we train discrete HMMs (simply called HMMs in our study) to model facial expressions. We must first preprocess those training multi-dimensional vector sequences to convert them to those one-dimensional (discrete) symbol sequences. The specific preprocessing algorithm we chose is the vector quantization (VQ) [65]. VQ techniques have been used widely and successfully to solve quantization and data compression problems. In an HMM-based approach, we need to quantize each multi-dimensional feature or motion vector sequence into a finite symbol sequence before training HMMs.

The purpose of designing an M-level vector quantizer (called a codebook with size M) is to partition all k-dimensional training feature vectors into M clusters and associate each cluster C^i, whose centroid is the k-dimensional vector c^i, with a quantized value named codeword (symbol) o^i. While VQ will reduce data redundancy and get rid of small noise, it will inevitably cause a quantization error between each training feature vector x and c^i. As the size of the codebook increases, the quantization error decreases, and required storage for the codebook entries increases. It is very difficult to find a trade-off among these three factors.

In order to have a good recognition performance in using HMMs, it is critical to design a codebook for vector quantizing each k-dimensional training feature vector x into a symbol o^i with minimum quantization error. Therefore, two primary issues are considerable for the design of the codebook: (1) codebook creation (the size of codebook), and (2) distortion measurement. Defining the size of codebook is still an open

problem when we use the VQ technique. According to our experimental result, the recognition system has high performance when the size M of the codebook, which should be power of 2, is at least 1/50 less than the number of all k-dimensional training feature vectors.

For the distortion measurement, there are two main considerations for optimizing the VQ:

1. The quantizer must satisfy the nearest neighbor rule.

$$x \in C^i \qquad if \quad \left\|x - c^i\right\| < \left\|x - c^j\right\| \tag{6-1}$$

$$where \quad \left\|x - c^i\right\| = \sum_{h=1}^{k} (x_h - c_h^i)^2 \qquad and \qquad i \neq j, \quad i, j = 0,1,...,M\text{-}1 \tag{6-2}$$

and

$$q(x) = o^i \qquad\qquad where \quad 0 \le o^i \le M - 1 \tag{6-3}$$

This means that the k-dimensional feature vector $x = [x_1,x_2,...,x_k]$ is classified to cluster C^i, whose centroid is the k-dimensional vector c^i, and encoded to be the codeword o^i because the distance between x and c^i is shorter than x and c^j. $q(.)$ is the quantization operator.

2. Each cluster center c^i must minimize not only the distortion D^i in cluster C^i but also total quantization errors D.

$$D = \sum_{i=0}^{M-1} D^i \tag{6-4}$$

$$where \quad D^i = \sum_{n=1}^{N} \left\|x_n^i - c^i\right\| = \sum_{n=1}^{N} \sum_{h=1}^{k} (x_{n,h}^i - c_h^i)^2 \tag{6-5}$$

N k-dimensional feature vectors x_n^i are located at cluster C^i. Because the total distortion D is a linear combination of D^i which is the distortion in cluster C^i, the k-dimensional cluster center c^i can be independently computed after classification of x.

Using the overall distortion measurement, it is hard to guarantee global minimization. A similar technique used for cluster analysis with squared error cost functions is called the K-means algorithm. The K-means algorithm is an iterative algorithm which can guarantee a local minimum, and works well in practice.

The K-means Algorithm:

Step 1: **Initialization** - Define the codebook size to be M and choose M initial (1st iteration) k-dimensional cluster centers $c^0(1)$, $c^1(1)$,..., $c^{M-1}(1)$ corresponding to each cluster C^i where $0 \le i \le M$-1.

Step 2: **Classification** - At the lth iteration, according to the nearest neighbor rule, classify each k-dimensional sample x of training feature vectors into one of the clusters C^i.

$$x \in C^i(l) \quad \text{if} \quad \left\|x - c^i(l)\right\| < \left\|x - c^j(l)\right\| \quad \text{where} \quad i \neq j,\ i, j = 0,1,...,M\text{-}1 \tag{6-6}$$

Step 3: **Codebook Updating** - Update the codeword (symbol) o^i of each cluster C^i by computing new cluster centers $c^i(l+1)$ where $i = 0,1,...,M$-1 at the l+1th iteration.

$$c^i(l+1) = \frac{1}{N}\sum_{n=1}^{N} x_n^i \qquad \text{where } x^i \in C^i(l+1) \tag{6-7}$$

N is the number of feature vectors in cluster $C^i(l+1)$ at the l+1th iteration, and

$$q(x) = o^i \qquad \text{where} \quad 0 \le o^i \le M-1 \tag{6-8}$$

where $q(.)$ is the quantization operator.

Step 4: **Termination** - If the decrease in the overall distortion at the current iteration l+1 compared with that of the previous iteration l is below a selected threshold, then stop; otherwise goes back to Step 2.

$$\begin{cases} \text{if } |D(l+1) - D(l)| < \text{threshold, then Stop} \\ \text{if } |D(l+1) - D(l)| \ge \text{threshold, then Goes to Step } 2 \end{cases} \tag{6-9}$$

Note that the K-means algorithm can only converge to a local optimum. The behavior of the K-means algorithm is affected by the number of clusters specified and the choice of initial cluster centers. Instead of using K-means algorithm, our VQ approach is based on Linde, Buzo and Gray's algorithm [65] for vector quantizer design, which is an extended algorithm of K-means, but unlike K-means which initializes each cluster center in the beginning. This VQ algorithm uses iterative method, splits the training vectors from

assuming whole data to be one cluster to 2,4,8,...,*M* (*M*'s size is power of 2) clusters, and determines the centroid for each cluster. The centroid of each cluster is refined iteratively by K-means clustering.

The Vector Quantization Algorithm:

Step 1: **Initialization** - Assume all *N k*-dimensional training vectors to be one cluster C^0, *i.e.*, codebook size $M = 1$ and codeword $o^0 = 0$, and find its *k*-dimensional cluster centroid $c^0(1)$ where 1 is the initial iteration.

$$c^0(1) = \frac{1}{N}\sum_{n=1}^{N} x_n^0 \qquad (6\text{-}10)$$

where *x* is one sample of all *N k*-dimensional feature vectors at cluster C^0.

Step 2: **Splitting** - Double the size *M* of the codebook by splitting each cluster into two. The current codebook size *M* is split into *2M*. Set $M = 2M$ by

$$\begin{cases} c_+^i(1) = c^i(1) + \varepsilon \\ c_-^i(1) = c^i(1) - \varepsilon \end{cases} \quad where \;\; 0 \le i \le M-1 \qquad (6\text{-}11)$$

c^i is the centroid of the *i*th cluster C^i, *M* is the size of current codebook, ε is a *k*-dimensional splitting parameter vector and is value 0.0001 for each dimension in our study. 1 is the initial iteration.

Step 3: **Classification** - At the *l*th iteration, according to the nearest neighbor rule, classify each *k*-dimensional sample *x* of training feature vectors into one of the clusters C^i.

$$x \in C^i(l) \quad if \; \left\|x - c^i(l)\right\| < \left\|x - c^j(l)\right\| \quad where \;\; i \ne j,\; i, j = 0,1,...,M\text{-}1 \qquad (6\text{-}12)$$

Step 4: **Codebook Updating** - Update the codeword (symbol) o^i of each cluster C^i by computing new cluster centers $c^i(l+1)$ where $i = 0,1,...,M\text{-}1$ at the $l+1$th iteration.

$$c^i(l+1) = \frac{1}{N}\sum_{n=1}^{N} x_n^i \qquad where \;\; x^i \in C^i(l+1) \qquad (6\text{-}13)$$

N is the number of feature vectors in cluster $C^i(l+1)$ at the $l+1$th iteration. And

$$q(x) = o^i \qquad where \;\; 0 \le o^i \le M-1 \qquad (6\text{-}14)$$

where *q(.)* is the quantization operator.

Step 5: **Termination 1** - If the difference between the current overall distortion $D(l+1)$ and that of the previous iteration $D(l)$ is below a selected threshold, proceed to Step 6; otherwise goes back to Step 3.

$$\begin{cases} if\ |D(l+1)-D(l)| < threshold,\ then\ Goes\ to\ Step\ 6 \\ if\ |D(l+1)-D(l)| \geq threshold,\ then\ Goes\ to\ Step\ 3 \end{cases} \quad (6\text{-}15)$$

(where *threshold* is 0.0001 in our study.)

Step 6: **Termination 2** -

Is the codebook size *M* equal to the VQ codebook size required ?

$$\begin{cases} if\ Yes,\ then\ Stop \\ if\ No,\ then\ Goes\ to\ Step\ 2 \end{cases} \quad (6\text{-}16)$$

Once the final codebook is obtained according to all training vectors by using this VQ algorithm, it is used to vector quantize each training and testing feature (or motion) vector into a symbol value (codeword) for the preprocessing of the HMM recognition process (Figure 43).

6.2 Beginning from Markov Models

The first order Markov chain is a stochastic process which follows the rule

$$P(q_{t+1}=j \mid q_0=k, q_1=l, \ldots, q_t=i) = P(q_{t+1}=j \mid q_t=i) \quad (6\text{-}17)$$

where q_t represents the state q at time t, and i, j, k, l represent the possible states of q at different instant of time. The first order Markov chain states that the probabilistic dependence is truncated at the preceding state. We consider only those processes in which the right-hand side of above equation is independent of time. We can then see that a time independent Markov chain is characterized by its state-transition probability a_{ij},

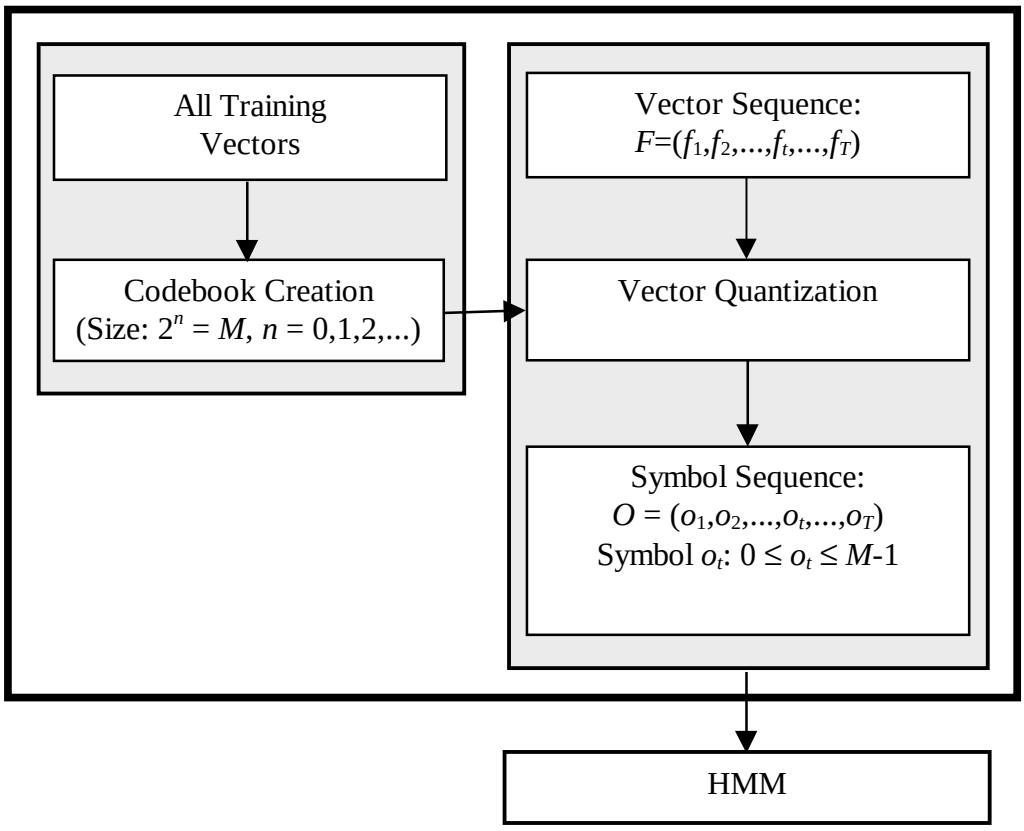


Figure 43 Vector quantization for encoding any vector sequence to a symbol sequence based on the codebook.

which is the probability of moving from one state i to another state j.

$$a_{ij} = P(q_t=j \mid q_{t-1}=i), \quad 1 \le i,j \le N \tag{6-18}$$

where N is the total number of states. The a_{ij} obeys standard stochastic constraints.

$$a_{ij} \ge 0 \qquad 1 \le i,j \le N \tag{6-19}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad 1 \le i \le N \tag{6-20}$$

The Markov model could be called an observable Markov model because the output of the stochastic process is the state sequence where each state corresponds to each instant of time with a deterministically observable event (symbol).

6.3 Extension of Markov Models: Hidden Markov Models

In the Markov model, the state sequence is observable. The output observable event in any given state is deterministic, not random. This will be too constraining when we use it to model the stochastic nature of the human performance, which is related to doubly stochastic processes, namely human mental states (hidden) and human actions (observable). It is necessary that the observable event is a probabilistic function of the state. That is why an HMM is employed. HMM is a representation of a Markov process and is a doubly embedded stochastic process with an underlying stochastic process that cannot be directly observed, but can only be observed through another set of stochastic processes that produce the sequence of observable symbols.

Before the description of HMMs, we define the elements of an HMM by specifying the following parameters:

***N*:** The number of states in the model. The state of the model at time t is given by q_t,

$$1 \le q_t \le N \qquad and \qquad 1 \le t \le T \tag{6-21}$$

where T is the length (number of frames) of the output observable symbol sequence.

***M*:** The size of the codebook or the number of distinct observable symbols per state. Assume o_t is one of all possible observable symbols for each state at time t, then

$$0 \le o_t \le M\text{-}1 \tag{6-22}$$

π_N**:** An N-element vector indicates the initial state probability.

$$\pi = \{\pi_i\}, \qquad where \quad \pi_i = P(q_1=i),\ 1 \le i \le N \tag{6-23}$$

$\boldsymbol{A_{NxN}}$**:** An N x N matrix specifies the state-transition probability that the state will transit from state i to state j.

$$A = \{a_{ij}\} \qquad where \quad a_{ij} = P(q_t=j \mid q_{t-1}=i), \quad 1 \le i,j \le N \tag{6-24}$$

and

$$a_{ij} \ge 0, \qquad \sum_{j=1}^{N} a_{ij} = 1 \qquad 1 \le i \le N \tag{6-25}$$

B_{MxN}: An *M* x *N* matrix represents the probability that the system will generate the observable symbol o_t at state *j* and at time *t*.

$$B = \{b_j(o_t)\} \quad where \quad b_j(o_t) = P(O_t=o_t \mid q_t=j), \quad 1 \le j \le N, \; 0 \le o_t \le M\text{-}1, \tag{6-26}$$

and

$$b_j(o_t) \ge 0, \quad 1 \le j \le N, \quad and \quad \sum_{o_t=0}^{M-1} b_j(o_t) = 1, \quad 1 \le j \le N \tag{6-27}$$

The complete parameter set λ of the discrete HMM is represented by one vector π and two matrices *A* and *B*

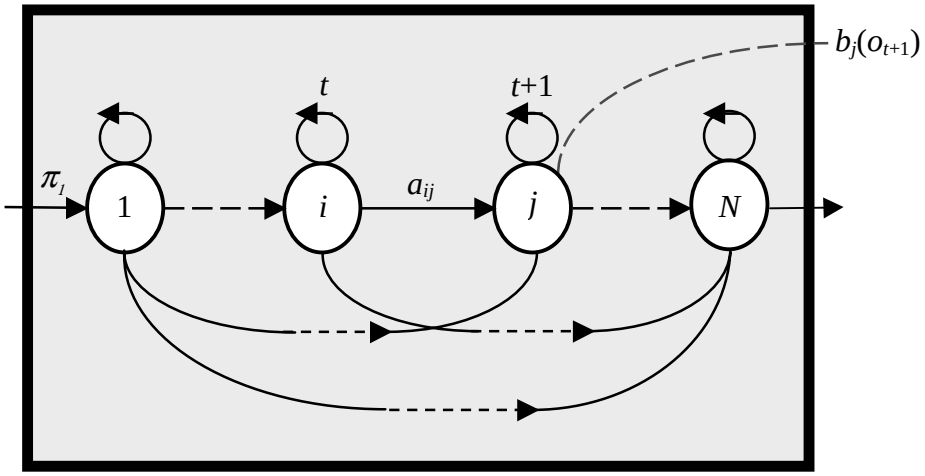
$$\lambda = (\pi, A, B) \tag{6-28}$$

In order to accurately describe a real-world process such as facial expression with an HMM, we need to appropriately select the HMM parameters. The parameter selection process is called the HMM "training."

This parameter set λ can be used to evaluate the probability $P(O \mid \lambda)$, that is to measure the maximum likelihood performance of an output observable symbol sequence *O*.

$$O = (o_1, o_2, ..., o_T) \tag{6-29}$$

where *T* is the number of frames for each image sequence. For evaluating each $P(O \mid \lambda)$, we need to select the number of states *N*, select the size of the codebook or the observable symbols *M*, and compute the results of probability density vector π and matrices *A* and *B* by training each HMM from a set of corresponding training data after VQ (Figure 44).



Symbol sequence:	$O = (o_1,o_2,...,o_t,...,o_T)$
State:	$q = (q_1=1,\ldots,q_t=i,q_{t+1}=j,\ldots,q_T=N)$
Codebook size:	M
HMM parameter set:	$\lambda = (\pi,A,B)$
Initial state distribution:	$\pi_1 = 1.0, \quad \pi_k = 0.0$ if $2 \le k \le N$
State-transition probability:	$A_{NxN} = \{a_{ij}\}$ from state i to j
Observable symbol probability:	$B_{MxN} = \{b_j(o_{t+1})\}$ at state j and time t+1
Output probability:	$P(O\|\lambda)$

Figure 44 The construction (topology) of the Hidden Markov Model.

6.4 Three Basic Problems of Hidden Markov Models

There are three basic problems in HMM design:

1. Problem of Probability Evaluation: How do we efficiently evaluate $P(O \mid \lambda)$, the probability (or likelihood) of an output observable symbol sequence $O = \{o_1,o_2,...,o_T\}$ given an HMM parameter set $\lambda = (\pi,A,B)$?

2. Problem of Optimal State Sequence: How do we determine an optimal state sequence $q = \{q_1,q_2,...,q_T\}$, which is associated with the given output observable symbol sequence $O = \{o_1,o_2,...,o_T\}$, by given an HMM parameter set $\lambda = (\pi,A,B)$?

3. Problem of Parameter Estimation: How do we regulate an HMM parameter set $\lambda = (\pi,A,B)$ in order to maximize the output probability $P(O \mid \lambda)$ of generating the output observable symbol sequence $O = \{o_1,o_2,...,o_T\}$?

Analyzing and solving the above three basic problems can help us to design and understand the HMM for training and recognition processes.

6.4.1 Probability Evaluation Using the Forward-Backward Procedure

In order to use an HMM for facial expression recognition, we need to compute the output probability $P(O \mid \lambda)$ with which the HMM will generate an output observable symbol sequence $O = \{o_1,o_2,...,o_T\}$ given the parameter set $\lambda = (A,B,\pi)$. The most straightforward way to compute this is by enumerating every possible state sequence of length T, so there will be N^T possible combinations of state sequence where N is the total number of states. Suppose there is one state sequence

$$q = \{q_1, q_2,..., q_T\} \tag{6-30}$$

Assume statistical independence of observable symbol o, and given the above state sequence q, the probability of the output observable symbol sequence will be

$$P(O \mid q,\lambda) = \prod_{t=1}^{T} P(o_t \mid q_t,\lambda) = b_{q_1}(o_1)\, b_{q_2}(o_2) \ldots b_{q_T}(o_T) \quad (6\text{-}31)$$

Also, we can get the probability of such a state sequence q by given an HMM parameter set λ.

$$P(q \mid \lambda) = \pi_{q_1}\, a_{q_1 q_2}\, a_{q_2 q_3} \ldots a_{q_{T-1} q_T} \quad (6\text{-}32)$$

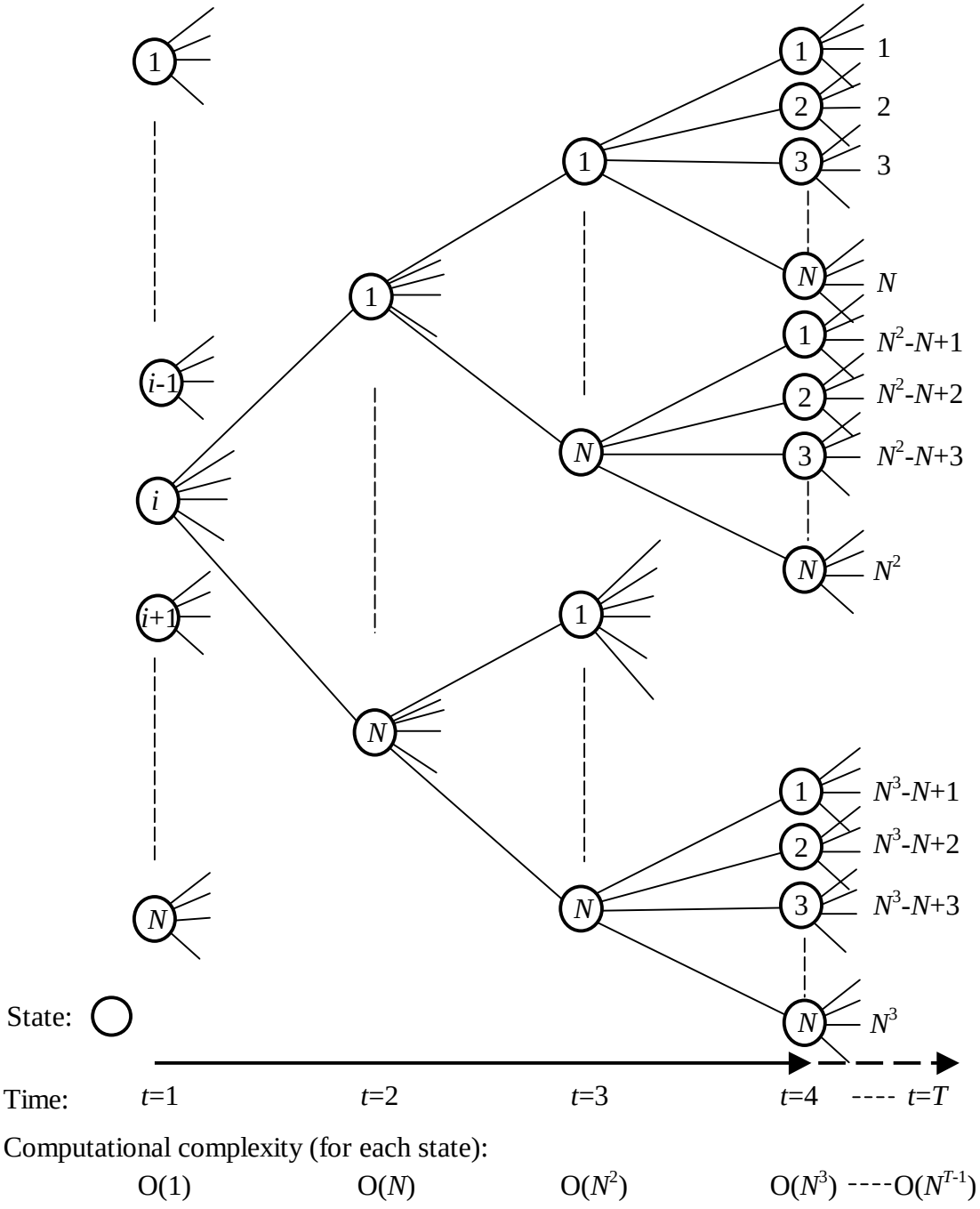
The joint probability of O and q (or the probability that O and q occur at the same time) is

$$P(O,q \mid \lambda) = P(O \mid q,\lambda)\, P(q \mid \lambda) \quad (6\text{-}33)$$

The probability of $P(O \mid \lambda)$ is the summation of this joint probability over all N^T possible state sequences q.

$$\begin{aligned} P(O \mid \lambda) &= \sum_{i=1}^{N} P(o_1 o_2 \ldots o_T,\ q_T = i \mid \lambda) \\ &= \sum_{i=1}^{N} P(O,\ q_T = i \mid \lambda) \\ &= \sum_{i=1}^{N} P(O \mid q_T = i,\lambda)\, P(q_T = i \mid \lambda) \\ &= \sum_{q_1 q_2 \ldots q_T} \pi_{q_1}\, b_{q_1}(o_1)\, a_{q_1 q_2}\, b_{q_2}(o_2) \ldots a_{q_{T-1} q_T}\, b_{q_T}(o_T) \end{aligned} \quad (6\text{-}34)$$

For the time complexity of the above computation, we can base the interpretation on Figure 45. Each state q_{t+1} at time $t+1$ has N possible paths with order O(1) calculations to be reached from the previous N state q_t at time t. That is, each state q_2 at time $t = 2$ can be reached from N possible state q_1 at time $t = 1$, and N possible state q_2 at $t = 2$. Each state q_3 at time $t = 3$ can have N^2 possible paths to be reached from N possible states q_1 at t = 1. Overall there are N^{T-1} possible paths with the order O(T) calculations to reach each final state q_T at time T for each state sequence. (According to the above equation, $2T$-1 multiplications are required for each state sequence.) The time complexity is the order O($N^{T-1}\ T$) for each state sequence. The totally N final state q_T at time T can be reached. The overall time complexity of computing the probability of $P(O \mid \lambda)$ is the order O($N^{T-1}\ T$) x O(N) = O($N^T\ T$). The value is very difficult to calculate. Even if we have a small



Computational complexity for total N states: $O(N^{T-1}) * O(N) = O(N^T)$

Figure 45 The tree structure of the computational complexity for direct evaluation of the output probability $P(O|\lambda)$ [105].

number of states N and T frames of state sequence, *e.g.*, $N = 4$ and $T = 20$, it still requires on the order of $4^{20} \times 20 \approx 2.2 \times 10^{13}$ calculations. Fortunately, we can use a more efficient procedure called the Forward-Backward procedure [79] to overcome this limitation.

Figure 46 can help us to describe the Forward procedure easily and clearly. We define the forward variable

$$\alpha_t(i) = P(o_1 o_2 \ldots o_t, q_t = i \mid \lambda) \tag{6-35}$$

as the probability of the partial observable symbol sequence $o_1 o_2 \ldots o_t$ at state i and at time t by given the HMM parameter set λ. We can solve for $\alpha_t(i)$ inductively as follows:

The Forward Procedure:

1. Initialization: The initial forward variable is the joint probability of state i, time $t = 1$ and initial observable symbol o_1 by given the HMM parameter set λ.

$$\alpha_1(i) = P(o_1, q_1 = i \mid \lambda) = \pi_i b_i(o_1) \qquad \text{where} \quad 1 \le i \le N \tag{6-36}$$

2. Induction (or Recursion): State j can be reached at time $t+1$ from the N possible states i, $1 \le i \le N$, at time t with the state-transition probability a_{ij}.

$$\begin{aligned} \alpha_{t+1}(j) &= P(o_1 o_2 \ldots o_{t+1}, q_{t+1} = j \mid \lambda) \\ &= \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(o_{t+1}), \qquad \text{where} \quad 1 \le t \le T-1,\ 1 \le i, j \le N \end{aligned} \tag{6-37}$$

3. Termination: The sum of all N final forward variables $\alpha_T(i)$, $1 \le i \le N$.

$$\begin{aligned} P(O \mid \lambda) &= \sum_{i=1}^{N} P(o_1 o_2 \ldots o_T,\ q_T = i \mid \lambda) \\ &= \sum_{i=1}^{N} P(O,\ q_T = i \mid \lambda) \\ &= \sum_{i=1}^{N} \alpha_T(i) \end{aligned} \tag{6-38}$$

The second step of the Forward procedure reduces the computational complexity since the calculation of the forward variable $\alpha_{t+1}(j)$ at time t+1, state j and observable symbol o_{t+1} depends only on the previous (at time t) N forward variables $\alpha_t(i)$, $1 \le i \le N$ (Figure 46, 47). This computation is performed for all states j, $1 \le j \le N$, and then iterated from the initial frame at $t = 1$ to $t = T$-1 for all possible state sequences. In other words, because there are only N states at each instant time, all the possible state sequences will remerge into these N states, no matter how long the observable symbol sequence will be (Figure 47). The time complexity is the order O(N T) for each observable symbol sequence. This computation obviously reduces the computational complexity of each state sequence from the order O(N^{T-1}) to O(N). There are N states for each instant time or at the end time $t = T$. The overall time complexity of computing the probability of $P(O \mid \lambda)$ is O(N T) x O(N) = O(N^2 T) whose origin is O(N^T T). Compared to the original example $N = 4$ and $T = 20$, it requires only the order of 4^2 x 20 = 3.2 x 10^2 which is much less than 4^{20} x 20 ≈ 2.2 x 10^{13} calculations.

Figure 46 describes the Backward procedure. We define a backward variable

$$\beta_t(i) = P(o_{t+1}o_{t+2}...o_T \mid q_t=i, \lambda) \qquad (6\text{-}39)$$

which means the probability of the partial observable symbol sequence from t+1 to the end time T by given state i at time t and the HMM parameter set λ. We can compute the $\beta_t(i)$ using the following steps:

The Backward Procedure:

1. **Initialization:** Arbitrarily defines the backward variable at the end time T and state i as

$$\beta_T(i) = 1 \qquad \textit{where} \quad 1 \le i \le N \qquad (6\text{-}40)$$

2. **Induction (or Recursion):** State i can reach N possible states j, $1 \le j \le N$, at time t+1 as well as the observable symbol o_{t+1} by state-transition probability a_{ij} and observation probability $b_j(o_{t+1})$.

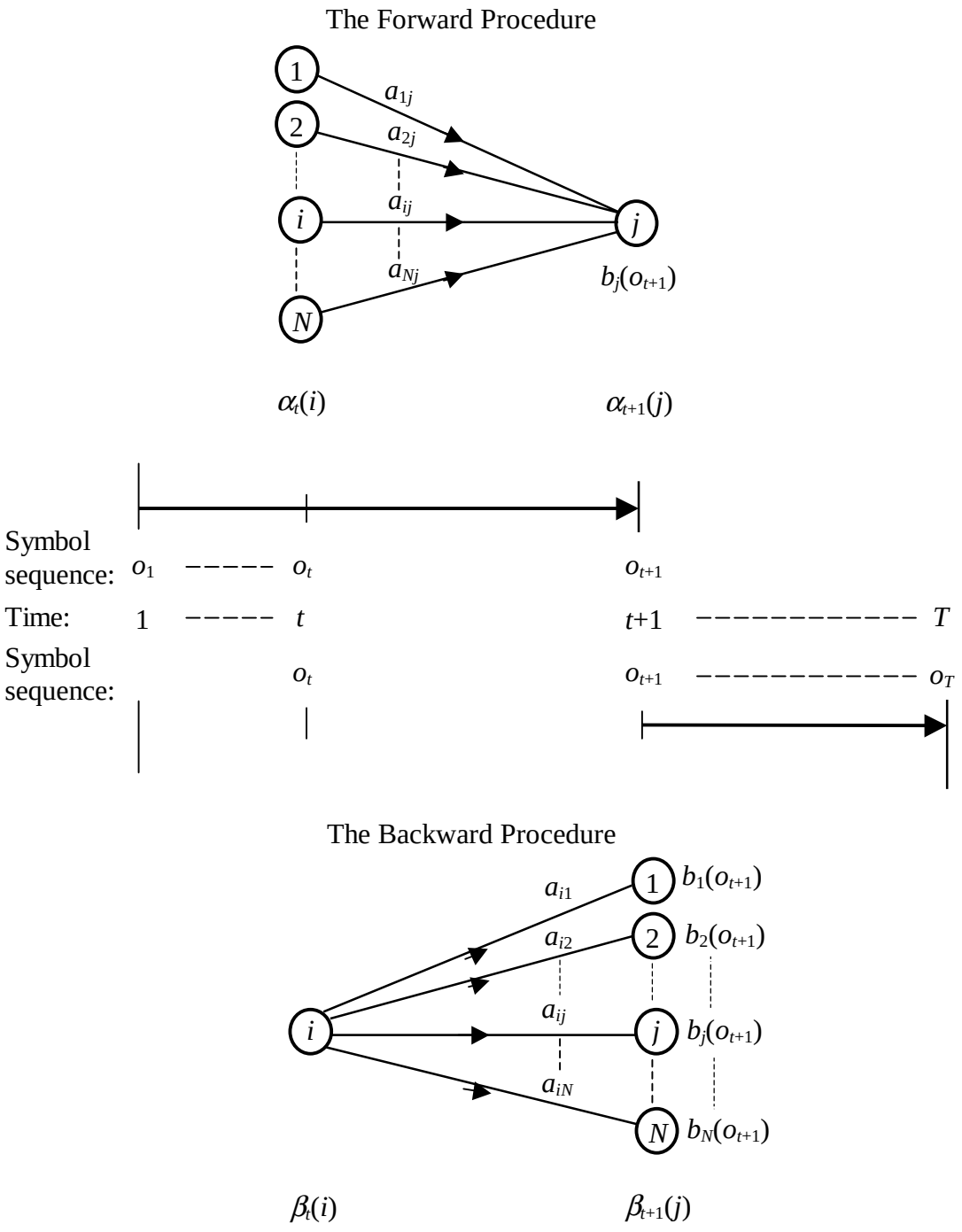


Figure 46 The Forward and Backward Procedures.

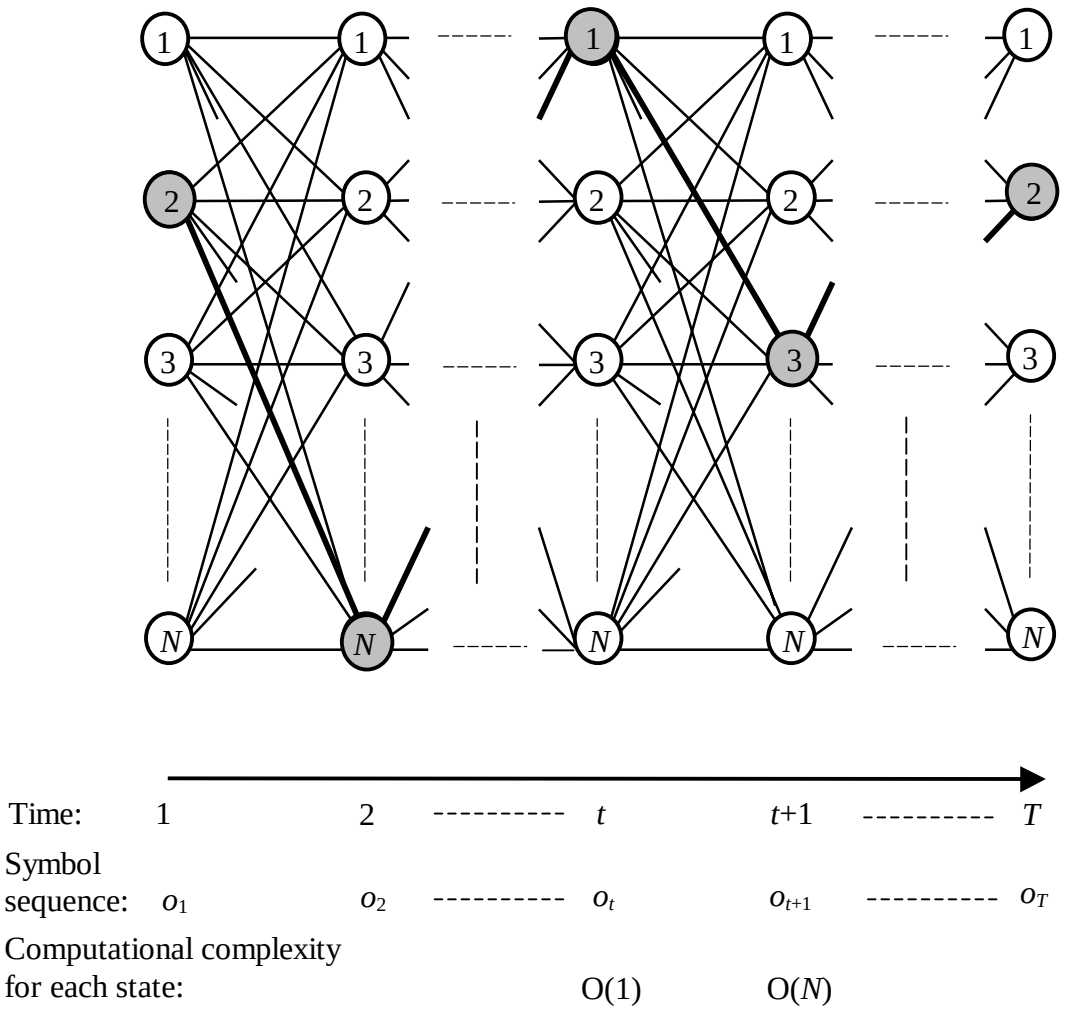


Figure 47 The tree structure of the computational complexity for the forward and backward procedures [79].

$$\beta_t(i) = P(o_{t+1}o_{t+2}...o_T \mid q_t=i,\lambda)$$

$$= \sum_{j=1}^{N} a_{ij} b_j(o_{t+1})\beta_{t+1}(j) \quad \textit{where} \quad t = T\text{-}1,\ T\text{-}2,...,1, \textit{ and } 1 \le i \le N \qquad (6\text{-}41)$$

The backward procedure's computational complexity is the same as the Forward procedure, $O(N^2 T)$, using the similar but opposite approach direction of Figure 47.

6.4.2 Optimal State Sequence Using the Dynamic Programming Approach

We use a dynamic programming method called the Viterbi algorithm [37,79,98] to find the single best state sequence $q = (q_1q_2...q_T)$ (or the most likely path) given the observable symbol sequence $O = (o_1o_2...o_T)$ and the HMM parameter set λ in order to maximize $P(q \mid O,\lambda)$. Since

$$P(q \mid O,\lambda) = \frac{P(q,O \mid \lambda)}{P(O \mid \lambda)} \qquad (6\text{-}42)$$

Maximizing $P(q \mid O,\lambda)$ is equivalent to maximizing $P(q,O \mid \lambda)$ using the Viterbi algorithm. The basic idea of the Viterbi algorithm (a dynamic programming method) is similar to the Forward procedure (Figure 46) whose calculation at each time is considered only between two consecutive times t and $t+1$, and starts at the initial time $t = 1$ and proceeds forward to the end time $t = T$. The major difference is during this calculation between two instant times. The control, which produces the maximum value corresponding to the single shortest or best path (state sequence), is "saved" instead of the summation of overall calculations (Figure 47). At the end of the state sequence for the calculation, the "remembered" best controls can be used to recover the state space trajectory based on path backtracking.

We define the maximum probability along a single best path at time t, which accounts for the first t observable symbols and ends in state i given the HMM parameter set λ, as

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1 q_2 \ldots q_{t-1}, q_t = i, o_1 o_2 \ldots o_t \mid \lambda) \tag{6-43}$$

We also define the "remembered" array $\psi_t(j)$ for each state j at time t, which keeps track of the argument that maximizes the value $\delta_{t-1}(i)$ x a_{ij}, in order to retrieve the optimal state sequence during the path backtracking. $\bar{q}_t$ is the single most likely state at time t.

The Viterbi Algorithm:

1. Initialization: The initial probability $\delta_1(i)$ is at state i, time $t = 1$ and initial observable symbol o_1 by given the HMM parameter set λ.

$$\delta_1(i) = P(o_1, q_1 = i \mid \lambda) = \pi_i b_i(o_1) \qquad \textit{where} \quad 1 \le i \le N \tag{6-44}$$

$$\psi_1(i) = 0 \qquad \textit{where} \quad 1 \le i \le N \tag{6-45}$$

$$\bar{q}_1 = \arg \max_{1 \le i \le N} \left(\delta_1(i)\right) \tag{6-46}$$

2. Recursion: The single best path (state sequence) among N possible paths from N possible states i at time t to state j at time t+1 with the state-transition probability a_{ij} is

$$\delta_{t+1}(j+1) = \max_{1 \le i \le N} P(o_1 o_2 \ldots o_{t+1}, q_{t+1} = j \mid \lambda)$$

$$= \left[\max_{1 \le i \le N} \left(\delta_t(i) a_{ij}\right)\right] b_j(o_{t+1}) \quad \textit{where} \quad 1 \le t \le T\text{-}1,\ 1 \le j \le N \tag{6-47}$$

$$\psi_{t+1}(j) = \arg \left[\max_{1 \le i \le N} \left(\delta_t(i) a_{ij}\right)\right] \qquad \textit{where} \quad 1 \le t \le T\text{-}1,\ 1 \le j \le N \tag{6-48}$$

$$\bar{q}_{t+1} = \arg \max_{1 \le i \le N} \left(\delta_{t+1}(i)\right) \tag{6-49}$$

3. Termination: The single best path reaches the end time T for each state sequence.

$$\bar{P} = \max_{1 \le i \le N} P(o_1 o_2 \ldots o_T,\ q_T = i \mid \lambda)$$

$$= \max_{1 \le i \le N} P(O,\ q_T = i \mid \lambda)$$

$$= \max_{1 \le i \le N} \left(\delta_T(i)\right) \tag{6-50}$$

$$\bar{q}_T = \arg \max_{1 \le i \le N} \left(\delta_T(i)\right) \tag{6-51}$$

4. Path (State Sequence) Backtracking: Backtracking is retrieving the path which have been saved as the most likely states.

$$\bar{q}_t = \psi_{t+1}(\bar{q}_{t+1}) \qquad \textit{where} \quad t = T\text{-}1, T\text{-}2, \ldots, 1 \tag{6-52}$$

For computation simplicity, the Viterbi algorithm can be implemented by additional preprocessing which takes the logarithms of the HMM parameters in order to convert multiplication to addition.

0. Preprocessing:

$$\pi_i^{\#} = \log(\pi_i) \qquad \textit{where} \quad 1 \le i \le N \tag{6-53}$$

$$a_{ij}^{\#} = \log(a_{ij}) \qquad \textit{where} \quad 1 \le i \le N \textit{ and } 1 \le t \le T \tag{6-54}$$

$$b_i^{\#}(o_t) = \log(b_i(o_t)) \qquad \textit{where} \quad 1 \le i \le N \textit{ and } 1 \le t \le T \tag{6-55}$$

1. Initialization:

$$\delta_1^{\#}(i) = \log(\delta_1(i)) = \pi_i^{\#} + b_i^{\#}(o_1) \qquad \textit{where} \quad 1 \le i \le N \tag{6-56}$$

$$\psi_1^{\#}(i) = 0 \qquad \textit{where} \quad 1 \le i \le N \tag{6-57}$$

$$\bar{q}_1^{\#} = \arg \max_{1 \le i \le N} \left(\delta_1^{\#}(i)\right) \tag{6-58}$$

2. Recursion:

$$\delta_{t+1}^{\#}(j) = \log(\delta_{t+1}(j))$$

$$= \left[\max_{1 \le i \le N} \left(\delta_t^{\#}(i) + a_{ij}^{\#}\right)\right] + b_j^{\#}(o_{t+1}) \qquad \textit{where} \quad 1 \le t \le T\text{-}1,\ 1 \le j \le N \tag{6-59}$$

$$\psi_{t+1}^{\#}(j) = \arg \left[\max_{1 \le i \le N} \left(\delta_t^{\#}(i) + a_{ij}^{\#}\right)\right] \qquad \textit{where} \quad 1 \le t \le T\text{-}1,\ 1 \le j \le N \tag{6-60}$$

$$\bar{q}_{t+1}^{\#} = \arg \max_{1 \le i \le N} \left(\delta_{t+1}^{\#}(i)\right) \tag{6-61}$$

3. Termination:

$$\bar{P}^{\#} = \max_{1 \le i \le N} \left(\delta_T^{\#}(i)\right) \tag{6-62}$$

$$\bar{q}_T^{\#} = \arg \max_{1 \le i \le N} \left(\delta_T^{\#}(i) \right) \tag{6-63}$$

4. Path (State Sequence) Backtracking:

$$\bar{q}_t^{\#} = \psi_{t+1}^{\#}(\bar{q}_{t+1}^{\#}) \qquad \textit{where} \quad t = T\text{-}1, T\text{-}2, ..., 1 \tag{6-64}$$

6.4.3 Parameter Estimation Using the Baum-Welch Method

We can use a set of training observable symbol sequences to adjust the model parameters in order to build a signal model that can be used to identify or recognize other sequences of observable symbols. There is, however, no efficient way to optimize the model parameter set that globally maximizes the probability of the symbol sequence. Therefore, the Baum-Welch method [6] is used for choosing the maximum likelihood model parameter set $\lambda = (\pi, A, B)$ such that its likelihood function $P(O \mid \lambda)$ is locally maximized using an iterative procedure.

To easily describe the procedure for reestimation (iterative computation) of the HMM parameter set $\lambda = (\pi, A, B)$, we define a posterior probability variable $\gamma_t(i)$, shown in Figure 48, as the probability of being in state i at time t by given the HMM parameter set λ and the entire observable symbol sequence O.

$$\gamma_t(i) = P(q_t = i \mid O, \lambda) = \frac{P(O, q_t = i \mid \lambda)}{P(O \mid \lambda)} = \frac{P(O, q_t = i \mid \lambda)}{\sum_{i=1}^{N} P(O, q_t = i \mid \lambda)}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{6-65}$$

where

$$P(O, q_t = i \mid \lambda) = \alpha_t(i)\, \beta_t(i) \tag{6-66}$$

$$\alpha_t(i) = P(o_1 o_2 ... o_t, q_t = i \mid \lambda) \tag{6-67}$$

$$\beta_t(i) = P(o_{t+1} o_{t+2} ... o_T \mid q_t = i, \lambda) \tag{6-68}$$

We define the other probability variable $\xi_t(i,j)$ (illustrated in Figure 49), which represents the probability of being in state i at time t, and state j at time t+1 given the observable symbol sequence O and the HMM parameter set λ.

$$\begin{aligned}\xi_t(i,j) &= P(q_t = i, q_{t+1} = j \mid O, \lambda) \\ &= \frac{P(q_t = i, q_{t+1} = j, O \mid \lambda)}{P(O \mid \lambda)} \\ &= \frac{P(q_t = i, q_{t+1} = j, O \mid \lambda)}{\sum_{i=1}^{N} P(O, q_t = i \mid \lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}\end{aligned} \tag{6-69}$$

Then the relationship between $\gamma_t(i)$ and $\xi_t(i,j)$ is

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{6-70}$$

If we sum $\gamma_t(i)$ and $\xi_t(i,j)$ from the initial time t = 1 to the time $t = T$-1, we can find

$$\sum_{t=1}^{T-1} \gamma_t(i) \text{ = expected number of transitions or times, } i.e.\text{, frequency, from state } i \text{ given observable symbol sequence } O \tag{6-71}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) \text{ = expected number of transitions from state } i \text{ to state } j \text{ given } O \tag{6-72}$$

A set of reasonable reestimation formulas for HMM parameters π, A, and B is given

$$\begin{aligned}\pi_i &= \text{expected number of transitions in state } i \text{ at time } t\ (=1) \\ &= \gamma_1(i) = \frac{P(O, q_1 = i \mid \lambda)}{P(O \mid \lambda)} \\ &= \frac{\alpha_1(i)\beta_1(i)}{\sum_{i=1}^{N} \alpha_T(i)}\end{aligned} \tag{6-73}$$

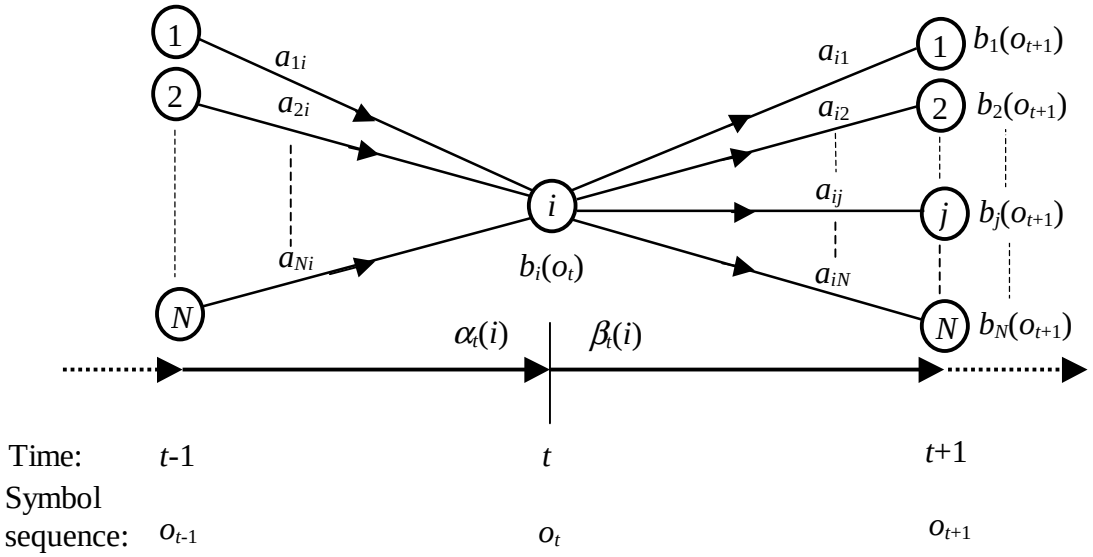


Figure 48 A posterior probability variable $\gamma_t(i)$ which is the probability of being in state i at time t by given the HMM parameter set λ and the entire observable symbol sequence O.

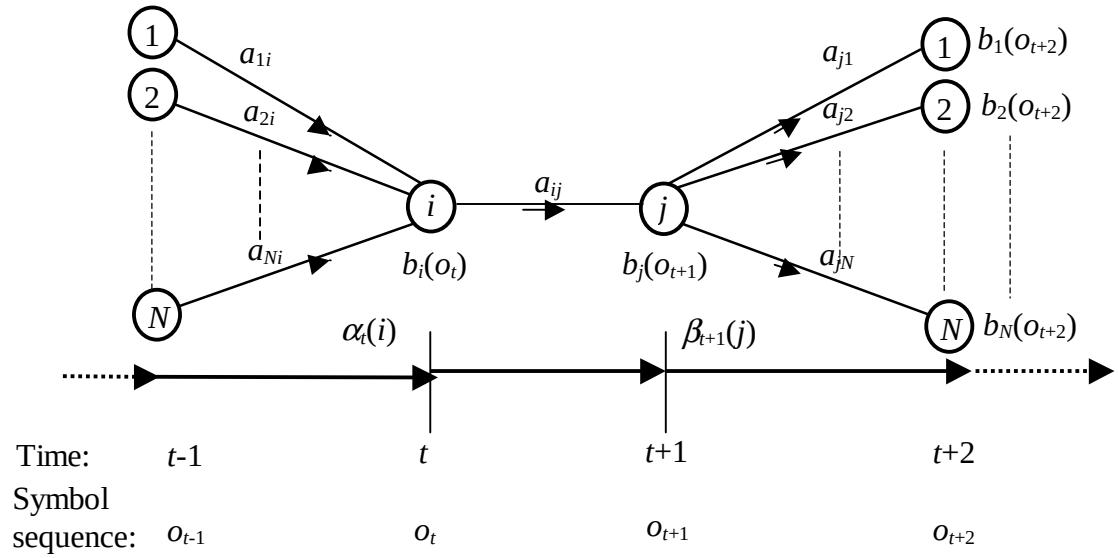


Figure 49 The probability variable $\xi_t(i,j)$ which represents the probability of being in state i at time t, and state j at time t+1 given the observable symbol sequence O and the HMM parameter set λ.

$$a_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{exptected number of transitions in state } i}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} = \frac{\sum_{t=1}^{T-1} P(q_t = i, q_{t+1} = j, O \mid \lambda)}{\sum_{t=1}^{T-1} P(q_t = i, O \mid \lambda)}$$

$$= \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i) \beta_t(i)} \qquad (6\text{-}74)$$

$$b_j(o_t) = \frac{\text{expected number of transitions in state } j \text{ and observable symbol } o_t \text{ at time } t}{\text{expected number of transitions in state } j}$$

$$= \frac{\sum_{t=1}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \overset{s.t.\ O_t = o_t}{=} \frac{\sum_{t=1}^{T} P(q_t = j, O \mid \lambda) \delta(O_t, q)}{\sum_{t=1}^{T} P(q_t = j, O \mid \lambda)}$$

$$= \frac{\sum_{t=1}^{T} \alpha_t(j) \beta_t(j) \delta(O_t, o_t)}{\sum_{t=1}^{T} \alpha_t(j) \beta_t(j)} \qquad \text{where } \delta(O_t, o_t) = \begin{cases} 1 & \text{if } O_t = o_t \\ 0 & \text{otherwise} \end{cases} \qquad (6\text{-}75)$$

where

$$P(q_t = i, O \mid \lambda) = \alpha_t(i)\, \beta_t(i) \qquad (6\text{-}76)$$

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_t(i) \beta_t(i) = \sum_{i=1}^{N} \alpha_T(i) \qquad (6\text{-}77)$$

$$P(q_t = i, q_{t+1} = j, O \mid \lambda) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \qquad (6\text{-}78)$$

Note that these updated parameters should satisfy stochastic constraints for computation normalization.

$$\sum_{i=1}^{N} \pi_i = 1 \qquad (6\text{-}79)$$

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad where \quad 1 \le i \le N \tag{6-80}$$

$$\sum_{o_t=0}^{M-1} b_j(o_t) = 1 \quad where \quad 1 \le j \le N \qquad and \qquad 1 \le t \le T \tag{6-81}$$

6.5 Computation Considerations

To be able to enhance the effectiveness of HMM performance in the practical applications, such as facial expression recognition, it is necessary to have accurate computation and guarantee the local maximum of the likelihood function using an iterative procedure (reestimation procedure), *i.e.* the Forward-Backward procedure, convergence.

6.5.1 Choice of Hidden Markov Model

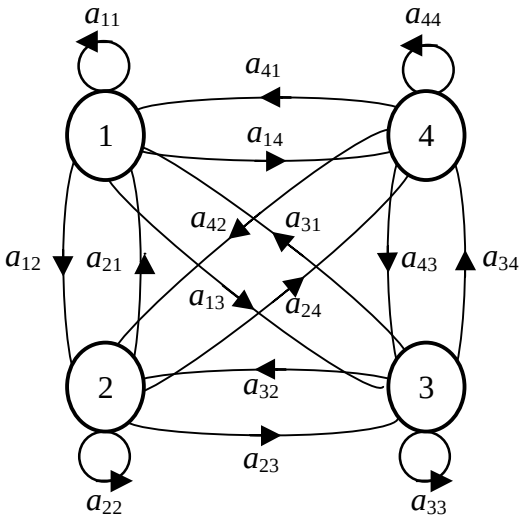
There are several types of HMMs [79] such as the ergodic model (Figure 50.a) in which every state of the model can be reached in a single step from any state of the model. Its state-transition probability matrix A is a full matrix.

$$A = \{a_{ij}\} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix} \tag{6-82}$$

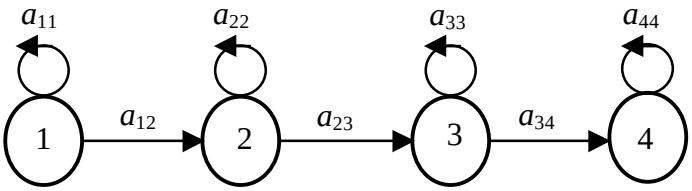
where

$$a_{ij} \ge 0, \qquad \sum_{j=1}^{N} a_{ij} = 1 \qquad and \qquad 1 \le i \le N \tag{6-83}$$

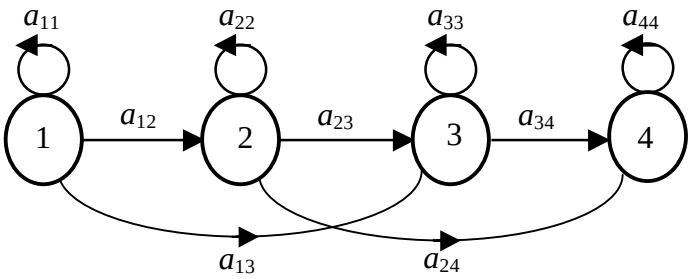
The left-right model (the Bakis model) which is used for facial expression recognition in various lengths of image sequences because they perform well in the spatio-temporal domain and are able to deal with the time warping problem. The left-right type of HMM



(a)



(b)



(c)

Figuer 50 (a) 4 state ergodic HMM (b) 1st-order 4-state left-right (Bakis) HMM (c) 2nd-order 4-state left-right HMM.

has the desirable properties that the initial state probability has the characteristic

$$\pi_i = \begin{cases} 0 & \text{if } i \neq 1 \\ 1 & \text{if } i = 1 \end{cases} \qquad \text{where } 1 \leq i \leq N \tag{6-84}$$

The state sequence must begin at the first state 1 with left to right order and end at the final state *N*. As the time increases, the observable symbols in each sequence either stay at the same state or increase in a successive manner. The state-transition coefficients of the left-right model have the property

$$a_{ij} = 0 \qquad \text{if } j < i \tag{6-85}$$

No transitions can occur from the current state to a state with a lower index. An additional constraint for the state-transition coefficients of the left-right HMM is

$$a_{ij} = 0 \qquad \text{if } j > i + \Delta i \tag{6-86}$$

for some value Δi which means the order of the left-right HMM. No jumps of more than Δi number of states are allowed. For example, $\Delta i = 1$ and $N = 4$ means the 1st-order 4-state left-right HMM (Figure 50.b) whose state-transition probability matrix *A* is

$$A=\{a_{ij}\}=\begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \tag{6-87}$$

where

$$a_{ij} \geq 0, \qquad \sum_{j=1}^{4} a_{ij} = 1 \qquad \textit{and} \qquad 1 \leq i \leq 4 \tag{6-88}$$

$\Delta i = 2$ and $N = 4$ indicates the 2nd-order 4-state left-right HMM (Figure 50.c) whose state-transition probability matrix *A* is

$$A=\{a_{ij}\}=\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \tag{6-89}$$

where

$$a_{ij} \geq 0, \qquad \sum_{j=1}^{4} a_{ij} = 1 \qquad and \qquad 1 \leq i \leq 4 \tag{6-90}$$

6.5.2 Initialization of Hidden Markov Model Parameter Estimation

For the training process, accurate initial estimations of the HMM parameters π, A and B will help the local maximum approach using an iteration procedure as close as possible to the global maximum of the likelihood function. If the elements of the parameters are set to zero initially, they will remain at zero during the entire process. In practice, it is not important to reach the global maximum of the likelihood function. Instead, finding a set of HMM parameters which promote a highly accurate recognition result is more important. Our experiment shows that very small random initial values of the HMM parameters (smaller than 10^{-6}) are adequate and useful and recognition accuracy is high. Remember that for the left-right HMM, the initial state probability for the first state is 1 ($\pi_1 = 1$) and other states are 0.

6.5.3 Computation of Scaling

The forward and backward variables $\alpha_t(i)$ and $\beta_t(i)$ are computed recursively, so they are composed of a large number of accumulated $\{a_{ij}\}$ and $\{b_j(o_t)\}$ multiplications. Since each a_{ij} and $b_j(o_t)$ is less (or extensively less) than 1, each term of $\alpha_t(i)$ or $\beta_t(i)$ will start to head exponentially to zero when the length (or number of frames) T in each image sequence increases. The dynamic range of the $\alpha_t(i)$ or $\beta_t(i)$ computation will then be beyond the precision range of any computer capability. To keep $\alpha_t(i)$ and $\beta_t(i)$ within the dynamic range of the computer, the most straightforward method is to multiply $\alpha_t(i)$ and $\beta_t(i)$ by scaling coefficients which can be canceled out completely at the end of the computation. The scaling coefficient for $\alpha_t(i)$ is defined as c_t [79]

$$\sum_{i=1}^{N} c_t \alpha_t(i) = c_t \sum_{i=1}^{N} \alpha_t(i) = 1 \quad where \quad 1 \le t \le T \tag{6-91}$$

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{6-92}$$

The scaling coefficient c_t is the inverse of the sum overall states N of $\alpha_t(i)$ at time t, is dependent only on time t and independent of state i, and effectively rebuilds the magnitude of the $\alpha_t(i)$ to 1. To keep the $\beta_t(i)$ computation within reasonable bounds as $\alpha_t(i)$, we can apply the same scaling coefficient c_t to $\beta_t(i)$ because the magnitudes of the $\alpha_t(i)$ and $\beta_t(i)$ are comparable. For efficient computation within rational bounds, the $\alpha_t(i)$ and $\beta_t(i)$ should be replaced by

$$\tilde{\alpha}_t(i) = c_t \alpha_t(i) = \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} \tag{6-93}$$

$$\tilde{\beta}_{t+1}(j) = c_{t+1} \beta_{t+1}(j) = \frac{\beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_{t+1}(i)} \tag{6-94}$$

where $\tilde{\alpha}_t(i)$ and $\tilde{\beta}_{t+1}(j)$ are the scaling results of $\alpha_t(i)$ and $\beta_{t+1}(j)$, respectively.

In addition, when $\tilde{\alpha}_t(i)$ and $\tilde{\beta}_{t+1}(j)$ are applied at the scaling intermediate probability $\tilde{\gamma}_t(i)$ and $\tilde{\xi}_t(i,j)$.

$$\tilde{\gamma}_t(i) = \frac{\tilde{\alpha}_t(i)\tilde{\beta}_t(i)}{\sum_{i=1}^{N} \tilde{\alpha}_t(i)\tilde{\beta}_t(i)} = \frac{(c_t\alpha_t(i))(c_t\beta_t(i))}{\sum_{i=1}^{N} (c_t\alpha_t(i))(c_t\beta_t(i))} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} = \gamma_t(i) \tag{6-95}$$

$$\tilde{\xi}_t(i,j) = \frac{\tilde{\alpha}_t(i)a_{ij}b_j(o_{t+1})\tilde{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \tilde{\alpha}_t(i)a_{ij}b_j(o_{t+1})\tilde{\beta}_{t+1}(j)} = \frac{(c_t\alpha_t(i))a_{ij}b_j(o_{t+1})(c_t\beta_{t+1}(j))}{\sum_{i=1}^{N}\sum_{j=1}^{N} (c_t\alpha_t(i))a_{ij}b_j(o_{t+1})(c_t\beta_{t+1}(j))}$$

$$= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)} = \xi_t(i,j) \tag{6-96}$$

The numerator and denominator terms are both deleted because the scaling coefficient c_t is independent of states *i* and *j*. The scaling intermediate probability $\tilde{\gamma}_t(i)$ and $\tilde{\xi}_t(i,j)$ has the same values as the intermediate probability without scaling $\gamma_t(i)$ and $\xi_t(i,j)$. Furthermore, the HMM parameters π, *A*, and *B* will also keep the same probability values when both $\alpha_t(i)$ and $\beta_{t+1}(j)$ are scaled, because those parameters are constructed from either or both intermediate probability $\gamma_t(i)$ and $\xi_t(i,j)$.

$$\tilde{\pi}_i = \pi_i, \qquad \tilde{a}_{ij} = a_{ij}, \qquad \text{and} \qquad \tilde{b}_j(o_t) = b_j(o_t) \tag{6-97}$$

The only real affected event of the HMM procedure by scaling coefficient is computing the maximum likelihood function $P(O \mid \lambda)$. Since [79]

$$\prod_{t=1}^{T} c_t \sum_{i=1}^{N} \alpha_T(i) = 1 \tag{6-98}$$

where scaling coefficient c_t is independent of state *i*. So

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) = \frac{1}{\prod_{t=1}^{T} c_t} = \frac{1}{\prod_{t=1}^{T} \frac{1}{\sum_{i=1}^{N} \alpha_t(i)}} \tag{6-99}$$

The computation for the denominator term of the maximum likelihood function $P(O \mid \lambda)$ will be extremely small, which is out of the dynamic range of the computer's ability. It can be solved by taking logarithms.

$$\log P(O \mid \lambda) = -\sum_{t=1}^{T} \log c_t = \sum_{t=1}^{T} \log \sum_{i=1}^{N} \alpha_t(i) \tag{6-100}$$

This can be used for evaluating the most likely performance of any input data corresponding to a set of HMMs. The Viterbi algorithm for finding the maximum likelihood state sequence also uses the logarithms, which will be within the dynamic bounds of the computer, so no scaling process is needed.

6.5.4 Computation of Smoothing for Insufficient Training Data

The amount of data (observable symbol sequences) used to train an HMM is always limited because of considerations of the computational cost of HMM training or the availability of training data. Thus, there is always an inadequate number of occurrences of low-probability state transitions between states and observable symbols within states to give reasonable estimates of the model parameters. If the training number of the observable symbol sequences are so small that they do not have any occurrences simultaneously to satisfy the conditional probability of HMM parameters, then $\tilde{\pi}_i = 0$, $\tilde{a}_{ij} = 0$, and $\tilde{b}_j(o_t) = 0$ will occur and will stay at 0 after each reestimation. When this resultant model is employed to evaluate any other observable symbol sequence which possibly contains the state transitions or the observable symbol that do not occur in the training sequences, this model will produce a zero probability result for this evaluated observable symbol sequence. Such a singular outcome is certainly a consequence of the unreliable estimation that $\tilde{\pi}_i = 0$, $\tilde{a}_{ij} = 0$, and $\tilde{b}_j(o_t) = 0$ due to the insufficiency of the training data to cover all possible varieties.

There are many possible solutions for handling the effects of insufficient training data [59,79], such as production of the codebook size (reduction of the number of observable symbols at each state) or the number of states. The simplest and most practical way for combating the insufficient training data problem is to add the numeric floor ε for smoothing the probability distributions of HMM parameters in order to ensure that no model parameter estimation falls below a specified threshold ε for each iterative estimation.

$$\tilde{\pi}_i = \begin{cases} \tilde{\pi}_i & \text{if } \tilde{\pi}_i \geq \varepsilon_{\tilde{\pi}} \\ \varepsilon_{\tilde{\pi}} & \text{if } \tilde{\pi}_i < \varepsilon_{\tilde{\pi}} \end{cases} \quad \text{where } \varepsilon_{\tilde{\pi}} > 0.0 \tag{6-101}$$

$$\tilde{a}_{ij} = \begin{cases} \tilde{a}_{ij} & \text{if } \tilde{a}_{ij} \geq \varepsilon_{\tilde{a}} \\ \varepsilon_{\tilde{a}} & \text{if } \tilde{a}_{ij} < \varepsilon_{\tilde{a}} \end{cases} \quad \text{where } \varepsilon_{\tilde{a}} > 0.0 \tag{6-102}$$

$$\tilde{b}_i(o_t) = \begin{cases} \tilde{b}_i(o_t) & \text{if } \tilde{b}_i(o_t) \geq \varepsilon_{\tilde{b}} \\ \varepsilon_{\tilde{b}} & \text{if } \tilde{b}_i(o_t) < \varepsilon_{\tilde{b}} \end{cases} \qquad \text{where } \varepsilon_{\tilde{b}} > 0.0 \tag{6-103}$$

In our study, $\varepsilon_{\tilde{\pi}} = \varepsilon_{\tilde{a}} = \varepsilon_{\tilde{b}} = 0.0001$.

6.5.5 Computation of Normalization

Each probability distribution of HMM parameters, which consist of conditional probability, should satisfy the stochastic constraints at each iteration estimation.

$$\sum_{i=1}^{N} \tilde{\pi}_i = 1 \tag{6-104}$$

$$\sum_{j=1}^{N} \tilde{a}_{ij} = 1 \qquad \text{where } 1 \leq i \leq N \tag{6-105}$$

$$\sum_{o_t=0}^{M-1} \tilde{b}_j(o_t) = 1 \quad \text{where } 1 \leq j \leq N \tag{6-106}$$

Since the probability of each parameter is reestimated at each iteration, the conflict with the stochastic constraints always occurs. The sum of the above equation is not equal to 1, particularly if the probability distribution of each parameter are smoothed by a numeric floor at each iterative estimation. Therefore, it is necessary to normalize the probability distributions of the HMM parameters so that the densities obey the required stochastic constraints after each iteration of parameter reestimation and smoothing.

$$\tilde{\pi}_i = \frac{\tilde{\pi}_i}{\sum_{k=1}^{N} \tilde{\pi}_k} \qquad \text{where } 1 \leq i \leq N \tag{6-107}$$

$$\tilde{a}_{ij} = \frac{\tilde{a}_{ij}}{\sum_{k=1}^{N} \tilde{a}_{ik}} \qquad \text{where } 1 \leq i,j \leq N \tag{6-108}$$

$$\tilde{b}_j(o_t) = \frac{\tilde{b}_j(o_t)}{\sum_{k_t=0}^{M-1} \tilde{b}_j(k_t)} \qquad where \;\; 1 \le j \le N \;\; and \;\; 0 \le o_t \le M\text{-}1 \tag{6-109}$$

6.5.6 Computation of Convergence

Since the Forward-Backward procedure is based on local maxima estimation by iterative computation to achieve global maximum, it is important to guarantee that the reestimated parameter set $\tilde{\lambda} = (\tilde{\pi}, \tilde{A}, \tilde{B})$ is convergent. It is necessary to prove that model $\tilde{\lambda}^{i+1} = (\tilde{\pi}^{i+1}, \tilde{A}^{i+1}, \tilde{B}^{i+1})$ following the i+1th iterative reestimation is either equal or more likely than model $\tilde{\lambda}^{i} = (\tilde{\pi}^{i}, \tilde{A}^{i}, \tilde{B}^{i})$ at current ith reestimation in the sense that $P(O \mid \tilde{\lambda}^{i+1}) \ge P(O \mid \tilde{\lambda}^{i})$. In other words, if the model $\tilde{\lambda}^{i}$ is replaced by $\tilde{\lambda}^{i+1}$ and this reestimation is repeated, then the probability of symbol sequence O being observed from the given update model $\tilde{\lambda}^{i+1}$ is improved until some limiting point is reached. The final result of this reestimation procedure is a maximum likelihood estimation of the HMM.

Because

$$P(O \mid \tilde{\lambda}) = \sum_{q_1 \ldots q_T} P(O, q \mid \tilde{\lambda}) \tag{6-110}$$

Then an auxiliary function [6,79] is defined as

$$Q(\tilde{\lambda}^{i}, \tilde{\lambda}^{i}) = \sum_{q_1 \ldots q_T} P(O, q \mid \tilde{\lambda}^{i}) \;\log\; P(O, q \mid \tilde{\lambda}^{i}) \tag{6-111}$$

$$Q(\tilde{\lambda}^{i}, \tilde{\lambda}^{i+1}) = \sum_{q_1 \ldots q_T} P(O, q \mid \tilde{\lambda}^{i}) \;\log\; P(O, q \mid \tilde{\lambda}^{i+1}) \tag{6-112}$$

over $\tilde{\lambda}^{i+1}$. Since

$$Q(\tilde{\lambda}^{i}, \tilde{\lambda}^{i+1}) \ge Q(\tilde{\lambda}^{i}, \tilde{\lambda}^{i}) \Rightarrow P(O \mid \tilde{\lambda}^{i+1}) \ge P(O \mid \tilde{\lambda}^{i}) \tag{6-113}$$

we can maximize the auxiliary function $Q(\tilde{\lambda}^i, \tilde{\lambda}^{i+1})$ over $\tilde{\lambda}^{i+1}$ to the better $\tilde{\lambda}^i$ in order to optimize the likelihood function $P(O \mid \tilde{\lambda})$. By iterating the procedure, the likelihood function eventually converges to a critical point.

6.5.7 Computation of Confidence

Unlike the artificial neural networks, even though there is no learnable mapping between input and output from the training process, HMM still can generate a satisfactory input-output confidence (mapping) for the recognition process (because of the computation consideration in section 6.5.4). The output of the HMM, $P(O \mid \tilde{\lambda})$, is a probability instead of one taking all. If the output probability is close to 1, it indicates that the input symbol sequence has high confidence (similarity) with the training model. If the output probability is close to 0, it implies the input symbol sequence has low confidence with the training model. The highest output probability among all training models is always chosen to be the recognition result and the recognition confidence is evaluated as well.