The Hidden Filter Model: Applications for Automatic Speech Processing

Bao Nguyen

Department of Electrical Engineering
McGill University
Montréal - Québec

June 1991

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Engineering

Abstract

This thesis examines hidden Markov filter models and their applications in speech segmentation. A method of segmenting the speech waveform is proposed. This method uses the Baum-Welch reestimation algorithm applied to the hidden filter models. Since speech signals are handled at the sample level, the amount of computations needed is very large. We will show how this issue can be dealt with effectively by using a staircase approach in the trellis calculations.

The hidden Markov filters are used to segment speech signals. Test results show very consistent locations of phone boundaries. The hidden filter model fits vocalic segments very well (with normalized prediction errors of less than 0.01), but performs less well on consonants (with normalized prediction errors of up to 0.3).

The speech segmentation by hidden filters is applied to a large vocabulary speaker dependent isolated-word recognizer at the preprocessing stage. The performances of the recognizer with and without preprocessor are compared. The results show small improvements in the recognition accuracy.

Sommaire

Cette thèse présente une étude sur les modèles de filtre de Markov cachés et leurs applications à la segmentation de parole. Il propose une méthode de segmentation des signaux de parole qui utilise l'algorithme de Baum-Welch appliqué aux modèles de filtre cachés. Comme l'analyse du signal est faite à chaque échantillon, la quantité de calcul est très grande. Nous montrons comment ce problème est résolu par une approche d'échelon dans le calcul en treillis.

Les tests de segmentation sont réalisés avec les bandes de données de parole continue et de mots isolés. Les résultats de segmentation sont très consistents au point de vue de placement des marques de frontière phonétique. Le modèle de filtre caché représente très bien les voyelles (avec les erreurs de prédiction plus petites que 0.01) mais il est moins bon pour les consonnes (aves les erreurs jusqu'à 0.3).

La segmentation des signaux acoustiques par filtre caché est appliquée au système de reconnaissance de parole sous forme d'un module de prétraitement. Les résultats de test avec et sans prétraitement sont comparés. Le prétraitement acoustique apporte une légère amélioration à la précision du système de reconnaissance.

Acknowledgements

The author wishes to express his sincere thanks to his supervisor, Professor Peter Kabal for suggesting the problem which gave rise to the present thesis. Special thanks are due to Professor Patrick Kenny at l'Institut National de la Recherche Scientifique (INRS) - Télécommunications Montréal for his wise and inspired guidance throughout the work, and for his car ful and insightful reading of drafts of this thesis.

The author is also extremely thankful to all the staffs of speech recognition group at Bell-Northern Research Montréal, especially Dr. Parthasarathy whose technical advise was very helpful.

The author would like to express his appreciation to his wife for her support and continuous encouraging during the course of this work.

This project was made possible by scholarships from the *Natural Sciences and Engineering Research Council* and by several computer facilities at INRS Télécommunications Montréal. Their contributions are greatly appreciated.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Applications of automatic speech processing based upon hidden Markov models (HMMs) have made considerable progress in the past few years. The technique of Markov modeling has been developed in a number of directions such as linear predictive HMMs, hidden filter HMMs and mixture autoregressive HMMs. These HMMs have been implemented in speech segmentation, enhancement and recognition.

This thesis is a study of an automatic segmentation processor based upon hidden filter models with application to a large vocabulary speaker dependent isolate-word recognizer.

In this chapter, we first review the theory of hidden Markov models. We focus on one class of HMMs, namely the linear predictive HMMs, and describe the fundamentals of the *hidden filter models*. Finally we will focus our attention to the problem of speech segmentation and feature extraction using the hidden filter models.

1.1 Fundamentals of Hidden Markov Models

The basic theory of hidden Markov model was first published in a classic paper by Baum [1]. A hidden Markov model is a collection of unobservable states connected by transitions. Each transition of the model is characterized by a transition probability. The evolution of these states (called the Markov chain) produces observable outputs.

Depending on the type of observation outputs, different Markov models are defined: a *discrete* HMM or a *continuous* HMM.

In a discrete model, the observations are discrete symbols emitted from a finite alphabet. An output probability distribution defines a conditional probability of emitting an output symbol given that a transition is taken (in which case we speak of a *transition-based* model) or given that a state is occupied (in which case we speak of a *state-based* model).

In a continuous model, the observations are continuous symbols, or more generally, continuous vectors. The discrete probability distribution is replaced by a probability density function (pdf). The probability density function defines the conditional probability that an observation vector lies between a certain range given that a transition is taken (a transition-based continuous model) or given that a state is occupied (a state-based continuous model).

Fig 1.1 illustrates a simple example of a state-based discrete HMM with two states s_1 and s_2 and two output symbols, A and B. When the Markov chain is in state s_1, the symbol A is observed with probability 0.8 while B is observed with probability 0.2. If the Markov chain is in state s_2, we can observe A with probability 0.3 and B with probability 0.7. While in s_1, the probability of staying is 0.6 and the probability of the transition to s_2 is 0.4. Once in s_2, it will stay there forever.

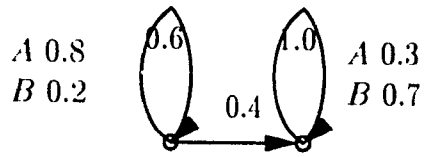

Fig. 1.1 A simple state-based discrete HMM with two states s_1, s_2 and two output symbols A and B.

Generally, a hidden Markov model is defined by:

- A finite set of states $\{s\} = (s_1, s_2, ..., s_N)$.
- A set of transition probabilities $[a_{ij}]$ where $a_{ij} = P(s_t = s_j | s_{t-1} = s_i)$ is the probability of taking a transition from state i to state j. This probability is

independent of time, i.e. $P(s_t = s_j|s_{t-1} = s_i) = P(s_j|s_i)$ for every time t. The transition probabilities define a first-order Markov chain: at each clock time t, a new state is entered based upon a transition probability distribution which depends only on the previous state, $P(s_t = s_j|s_{t-1} = s_i, s_{t-2} = s_l, \ldots) = P(s_t = s_j|s_{t-1} = s_i)$.

- A set of output probability distributions or density functions:
  - For the transition-based discrete model: the probability of emitting symbol k when taking a transition from state i to state j, $[b_{ij}(k)]$.
  - For the state-based discrete model: the output probability of emitting symbol k when state i of the model is occupied, $[b_i(k)]$.
  - For the transition-based continuous model: the probability that the observation vector lies between $\mathbf{x}$ and $\mathbf{x} + d\mathbf{x}$ when taking a transition from state i to state j, $b_{ij}(\mathbf{x})d\mathbf{x}$.
  - For the state-based continuous model: the probability that the observation vector lies between $\mathbf{x}$ and $\mathbf{x} + d\mathbf{x}$ when staying in state i, $b_i(\mathbf{x})d\mathbf{x}$

A hidden Markov model may be ergodic: every state of the model can be reached from every other state in a finite number of steps. Fig.1.2b illustrates an example of a 3-state ergodic HMM.

A special type of HMM has been developed for speech recognition [2] This model is called a *left-to-right* model, because the underlying state sequence associated with the model has the property that as time increases, the state index increases (or stays the same). Fig.1 2a illustrates a first-order 4-state left-to-right HMM with a special skip transition

In 1983, Rabiner, Levinson and Sondhi [2], [3] at Bell Laboratories presented an approach to speaker-independent isolated words recognition with the use of phoneme-based HMMs. Left-to-right HMMs with state-based discrete symbols were used in

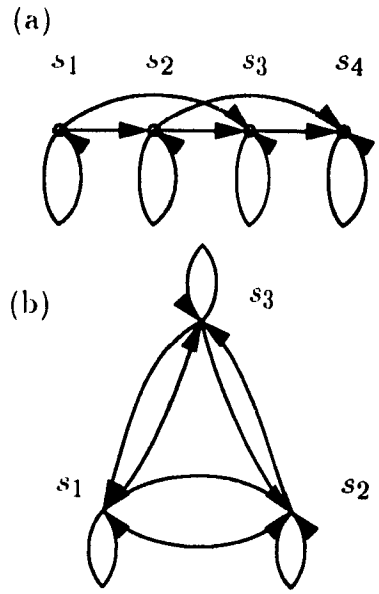

Fig. 1.2 Illustration of two types of HMMs. (a) A four-state left-to-right model.(b) A three-state ergodic model

their recognizer. The HMM's discrete symbols were obtained using the vector quantization (VQ) of linear predictive coding (LPC) analysis.

The 8-pole LPC analysis by autocorrelation technique was performed on 45ms frames, each being spaced 10ms apart. Using an iterative reestimation technique, these parameters were trained to provide the codebook entries of the vector quantizer and the model coefficients of each word HMM. Then using the Viterbi scoring algorithm [1], a probability score and a decision rule (which chooses the word whose model gives highest probability) were applied to the unknown word at the recognition stage.

Their initial experiments with this framework (in [2]) were restricted to a vocabulary of 10 digits. Recognition rates from 93% to 96% were recorded. Extended tests to a medium-size vocabulary of 129 words (in [3]) have shown that the recognition accuracy was a function of the HMM parameters: increasing the number of states in the model and/or the size of the VQ codebook improved performances of the HMM recognizer.

1.2 Linear Predictive Hidden Markov Models and Hidden Filter Models

For the applications in speech processing, especially in speech recognition, it would be advantageous to use HMMs with continuous observation densities because the observations are continuous signals. A very interesting class of continuous HMMs that is particularly applicable to speech recognition is the class of linear predictive (or autoregressive) HMMs.

In this model, a speech waveform $Y = (y_1, ..., y_{T_1})$ of length T_1 is decomposed into a sequence of T observation vectors (or T segments) of length M ($T_1 = T \times M$), i.e. $Y = (y_1, ..., y_{T_1}) = (Y_1, Y_2, ..., Y_T)$ where Y_t, $t = 1, 2, ..., T$ is an observation vector with components $(y_{tM}, y_{tM+1}, ..., y_{(t+1)M-1})$. Each segment Y_t is selected from a set of S all-pole recursive filters driven by S corresponding Gaussian noise sources $N(0, \sigma_s^2)$, $s = 1, ..., S$. The filters are defined by polynomials A_s of some degree N, i.e. $A_s = (b_{0_s}, ..., b_{N_s})$, with $s = 1, ..., S$ and $N < M$. Thus:

$$Y_t = -\sum_{i=1}^{N} b_{i_s} Y_{t-i} + e_{t_s}$$

where e_{t_s}, $t = 1, ..., T$, $s = 1, ..., S$ are Gaussian independent identically distributed random variables with zero mean and variances σ_s^2. The likelihood of the observation sequence Y is defined as:

$$L(Y) = \sum_{\omega} \prod_{t=0}^{T-1} a_{s_{t-1}s_t} D(Y_t | A_s, \sigma_s)$$

with

$$D(Y_t | A_S, \sigma_s) = \frac{2}{\sqrt{2\pi\sigma_s^2}} \exp \frac{-A_s R(Y_t) A_s^*}{2\sigma_s^2}$$

where A_s^* denotes the matrix transpose of A_s, $R(Y_t)$ is the autocorrelation of Y_t, $a_{s_{t-1}s_t}$ is the transition probability, $\omega = (s_0, ..., s_{T-1})$ is any T long sequence of states and $\sum_\omega$ means the summation over all possible paths ω.

The first application of linear predictive HMMs was presented by Poritz [5] in 1982. In his work, Poritz proposed a method of modeling speech signals by

a 5-state ergodic linear predictive state-based HMM. In finding a Markov model $\lambda = \{[a_{ij}], A_i, \sigma_i\}$, $i, j = 1, ..., S$ which maximizes $L(Y)$ by an iterative hill climbing technique, Poritz found a very close relationship between states of the model and traditional classes of speech events. His experiment with $T = 4000$ frames, $M = 100$ of 40 seconds of 12-bit PCM speech sampled at 10 kHz and with a model having $S = 5$ and $N = 3$ (that is, 3-order autoregressive filters) showed that the power spectra for each all-pole filter could be associated to *strong voicing, silence, nasal (liquid), stop burst* and *frication*. The result of this paper strongly suggested that linear predictive HMMs may be used to encapsulate important informations about the speech waveforms.

While Poritz only considered a single Gaussian autoregressive density per state, Juang *et al* [6] further expanded this initial work to the case of multivariate Gaussian autoregressive densities (a *mixture* autoregressive hidden Markov model). Denote K the number of mixture components in the model, the observation density $b_j(\mathbf{x})$ now has the form

$$b_j(\mathbf{x}) = \sum_{k=1}^{K} c_{jk} b_{jk}(\mathbf{x})$$

where c_{jk} is the weight of the kth mixture component and $b_{jk}(\mathbf{x})$ is the basic Gaussian pdf for the kth mixture component, all related to state j. The mixture weight c_{jk} satisfies the stochastic constraint

$$\sum_{k=1}^{K} c_{jk} = 1, \quad j = 1, 2, ..., N$$

Parameters of the model to be estimated include $\lambda = \{[a_{ij}], A_i, \sigma_i\}$, $i, j = 1, ..., S$ and $[c_{jk}]$, $j = 1, ..., N$ and $k = 1, ..., K$. Their extensive tests of speaker independent, isolated digit recognition that employed highly constrained left-to-right HMMs and mixture autoregressive densities have scored average digit error rates from 1.2% to 9.2%. Although these results were good, Juang pointed out that the model was not as good as the continuous Gaussian density models based upon the cepstral representation of the signal.

In the field of speech enhancement, Yphraim [7], [8] proposed a new approach for enhancing speech signals which have been degraded by statistically independent additive noise using HMMs. The process is basically an estimation problem in which a given function of the clean speech (e.g. speech waveform, DFT or sample spectrum) is estimated from a sample function of the noisy speech so as to minimize a distorsion measure (e.g. mean-square errors) between the clean and the estimated speech signals. Solutions to that estimation problem require an estimate of the joint probability distributions (PDs) of the speech signal and the noise process. Yphraim accomplished this task by modeling the PD of the clean speech by HMMs with mixture of Gaussian autoregressive output, and by modeling the noise process with single Gaussian autoregressive model. The parameter set of the HMMs is estimated by a minimum mean square error (MMSE) approach in [8], and by a maximum a posteriori approach (MAP) in [7]. In his experiments, the estimation stage was performed with a training sequence using 100 sentences of clean conversation speech spoken by 10 speakers using a telephone handset. The enhancement tests were performed on 8 sentences spoken by 4 speakers (recorded in a similar manner). Typical signal to noise ratio (SNR) improvements achieved by a MMSE approach were 4.5 5.5dB at 10dB input SNR [8], while SNR improvements achieved by a MAP approach were 4.0–6.0dB at 10dB input SNR [7].

Recently, Kenny *et al.* [9] developed a new type of Markov model to account for the correlations between successive frames of the speech signal. This model treats the *sequence of frames* as a non-stationary autoregressive process whose parameters are controlled by a left-to-right hidden Markov chain. Each transition in the Markov chain is associated with a set of regression coefficients together with a mean vector and a covariance matrix which serve to characterize the distribution of the prediction error. This linear predictive model has been implemented with several variants in a large-vocabulary speaker-dependent isolated-word recognizer. With a set of eight mel-based cepstral coefficients $(C_0, ..., C_7)$ calculated every 10 ms using a window of

length 25 ms, a feature vector $(C_1, ..., C_7, \Delta C_0, ..., \Delta C_7)$ was formed (ΔC_i was the difference between C_i over an interval of length 40 ms). Performances ranging from 78.9% to 83.0%, depending on the variant of model, were recorded with a test set of 399 words of text. These results showed that the model performed better than the standard multivariate Gaussian HMM when it is incorporated into a large-vocabulary isolated-word recognizer.

The models proposed by Kenny [9] are formally very similar to the *hidden filter* models defined by Poritz in [10]. However, in [9] the speech signal is handled at the frame level instead of the sample level as in [10]. In fact, the hidden filter models considered the signal waveform $Y = (y_{-N+1}, ..., y_0, ..., y_T)$ as a time series generated by a set of S states, each determined by an all-pole filter $A_s = (a_0(s), a_1(s), ..., a_N(s))$, $s = 1, ..., S$ of degree N and a positive gain factor σ_s^2 such that

$$y_t = \sum_{j=1}^{N} a_j(s) y_{t-j} + u_t$$

where $u_t \sim N(0, \sigma_s^2)$. The pair (A_s, σ_s^2) is referred as a hidden filter.

1.3 Speech Segmentation and Feature Extraction Using Hidden Filter Models

While the linear-predictive HMMs have been proved to perform well in the task of automatic speech recognition, they also presented some weaknesses. One of the weaknesses observed in automatic speech recognition using vector-valued observations is the use of uniform fixed-length windows to segment words into phones.

Generally, at the front end of a recognizer, the speech samples are blocked into sequence of fixed length segments (e.g. 10ms window). These segments are parameterized either by linear predictive coefficients or by cepstrum coefficients. The recognizer will use these acoustic segments to construct the phoneme models and to compute the likelihood scoring of the acoustic data.

The mechanics of blocking and mapping acoustic segments do not take into account the phone boundaries (e.g. boundaries between stops and sonorants, affricates and sonorants, etc). The changing statistical characteristics of speech signal at these boundaries fall often into one segment. This creates difficulties and erroneous in mapping acoustic segments into phoneme models because the recognizer must map the segment to one of the phones.

On the other hand, the hidden filter models do not assume any fixed frame size in their formulation (they work on a sample basis). Our strategy is to use the hidden filter models to eliminate the above weakness inherent in the vector-valued HMMs by windowing the speech waveform with variable frame length windows. The process is basically an automatic segmentation. It uses the hidden filter models to automatically segment the sequence of speech samples into successive frames of variable lengths, taking into account the total likelihood of the observation speech sequence.

Our filter models, parameterized by linear prediction polynomials and error variances, handle the speech signal at the sample level. The Markov chain used is a state-based continuous HMM with no skip transitions. We do not allow skip transitions because all the states of the Markov model must be visited in a monotonically increasing manner (that means left-to-right order), and the number of visits to each state will be used as segment indication of the sequence of speech samples. The number of states, in this framework, corresponds to the number of frames for a sequence of speech samples.

In order to segment the speech waveform, we proceed as follows: first a uniform-window LPC analysis is performed on the input samples of speech. These LPC coefficients are used as initial values for our filters. The Baum-Welch algorithm is used to adjust the filter coefficients so as to increase the likelihood of the speech data. The reestimation process terminates when the likelihood converges. The segmentation of the data is found using a maximal a posteriori (MAP) criterion. In the process of automatic segmentation, the LPC features of each segment are generated as a by

product.

Since our aim is to improve segment boundaries between phones during recognition, we try to implement our hidden filter models at a preprocessing stage of a large-vocabulary speaker-dependent isolated-word recognizer. Although it is a convenient idea, we also remark that the problem of real-time implementation persisted. In fact, our experimentations in automatic segmentation with an average 20 ms window of speech [†] and 12-order LPC on a DEC station 2100 computer showed that an average 5 minutes of CPU time was needed to provide acoustic segments of one word.

The organization of the thesis is as follows. In chapter 2, we will develop the basic forward-backward and Baum-Welch algorithm for our hidden filter models. In chapter 3, we will look at some implementation issues for the training of the models and illustrate how the speech segmentation is obtained as a by-product of the reestimation procedure. We will also show some segmentations of continuous speech and isolated word data in this chapter. Chapter 4 describes how the hidden filter models can be used as a feature-extractor in speech recognition. It also reports experimental results on a very large vocabulary speaker-dependent isolated-word recognition task. Finally, chapter 5 discusses the results of our work, the drawbacks of our model and the ways in which the hidden filter model preprocessor could be improved.

[†] The average time length of 20 ms is computed by dividing the total length of the sequence of speech to the number of states associated to that sequence. This is not a fixed window.

Chapter 2 The Hidden Filter Models

The hidden filter hidden Markov models were first developed by Poritz [10] to model speech waveforms. Using autoregressive polynomials and error variances to parameterize the output speech samples, Poritz derived a version of Baum-Welch reestimation formulas for his model parameters. In this chapter, we will give a mathematical description of the model and explain how its parameters can be estimated from speech data

2.1 Hidden Filter Markov Models

Let us consider a time signal $Y = (y_1, ..., y_T)$ which is a sequence of speech samples. We treat it as the output of a doubly stochastic process (S, Y) where $S = (s_1, ..., s_T)$ is a sequence of unobservable states (the *hidden* Markov chain). For our purposes, we will assume that these states are selected from a *first-order left-to-right* Markov chain characterized as follows:

1- There is a finite number, say N, of states in the model $\{s\} = (1, 2, ..., N)$. The state 1 is called the *initial state* and the state N is called the *final state*. The Markov chain must enter the model by the initial state and must leave the model by the final state. Once the Markov chain leaves a state, that state cannot be revisited at a later time.

2- At each clock time t, a new state is entered based upon a first-order Markovian property

$$P(s_t = j|s_{t-1} = i, s_{t-2} = l, ...) = P(s_t = j|s_{t-1} = i) = P(j|i) = a_{ij}$$

If $a_{ii} > 0$, the process may remain in the state occupied at time $t-1$ (the self-loop transition in the illustrated figure 2.1). The transition probabilities a_{ij} obey

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad a_{ij} \geq 0$$

Furthermore, we assume that our left-to-right model does not have any skip transitions in the chain. This assumption is required to assure that all the states of the model will be visited, and the number of visits will determine the segmentation of the sequence of speech samples as shown later in chapter 3.

3- In addition to the states $1, ..., N$ there is a state $N+1$ called the *sink state*. The observation sequence $(y_1, ..., y_T)$ terminates when the sink state is reached, i.e. $s_{T+1} = N + 1$, so there is no acoustic distribution or transition probabilities associated with the sink state.

4- Associated with each state s of the model (other than the sink state), we define a hidden filter (B_s, σ_s^2), where B_s is the set of regression coefficients of an all-pole filter of degree p, and σ_s^2 is the gain factor of this filter. When the Markov chain is in state s at time t, it generates an output sample y_t by applying the filter to the most recent samples of the sequence $Y = (y_1, y_2, ...y_T)$ and adding a sample of Gaussian noise of zero mean whose variance σ_s depends on the state s (thus, our Markov model is a state-based HMM).

Figure 2.1 illustrates the model topology we are assuming.

Now, suppose that we are at time t and at state s, the sample y_t is therefore

$$y_t = b_1(s)y_{t-1} + b_2(s)y_{t-2} + ... + b_p(s)y_{t-p} + e_t \qquad (2.1.a)$$

$$(e_t \sim N(0, \sigma_s^2)) \qquad (2.1.b)$$

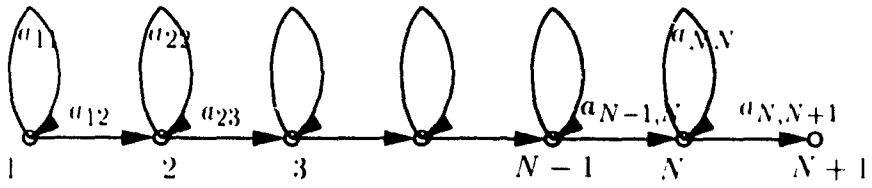

Fig. 2.1 A standard first-order left-to-right HMM.

We assume that ϵ_t at different times are independent (i.e. uncorrelated with each other). Define

$$X_t \triangleq (y_{t-1}, \ldots, y_{t-p}) \tag{2.2}$$

$$B_s^* \triangleq (b_1(s), b_2(s), \ldots, b_p(s)) \tag{2.3}$$

where B^* denotes the matrix transpose of B. Then we can write:

$$y_t \triangleq X_t B_s + \epsilon_t \tag{2.4}$$

Let $P(Y, S|X_1)$ be the joint likelihood of $Y = (y_1, \ldots, y_T)$ and $S = (s_1, \ldots, s_T)$, i.e. the event that y_1 is emitted at time $t = 1$ at state s_1, a transition occurs from s_1 to s_2, y_2 is emitted at time $t = 2$ at state s_2, etc, given an initial observation $X_1 = (y_0, y_{-1}, \ldots, y_{-p})$.

Similarly we use $P(Y|S, X_1)$ to stand for the conditional likelihood.

Since ϵ_t is a Gaussian noise $N(0, \sigma_s^2)$ we have:

$$P(\epsilon_t) = \frac{1}{\sqrt{2\pi\sigma_s^2}} \exp\left(\frac{-\epsilon_t^2}{2\sigma_s^2}\right) \tag{2.5}$$

From the definition of ϵ_t

$$P(y_t|X_t) = P(y_t - X_t B_s) \tag{2.6.a}$$

$$= (1/\sqrt{2\pi\sigma_s^2}) \exp \frac{-(y_t - X_t B_s)^2}{2\sigma_s^2} \tag{2.6.b}$$

$$\triangleq L(X_t, y_t, s) \tag{2.6.c}$$

Hence, with independent residuals ϵ_t we get:

$$P(Y|S, X_1) = \prod_{t=1}^{T} L(X_t, y_t, s_t) \tag{2.7}$$

The probability of the state sequence $S = (s_1, s_2..., s_T, s_{T+1})$ is

$$P(S|X_1) = \prod_{t=2}^{T+1} P(s_t|s_{t-1}) \tag{2.8}$$

The total likelihood of the observation sequence Y for the given state sequence S is

$$P(Y, S|X_1) = P(Y|S, X_1)P(S|X_1) \tag{2.9}$$

Thus

$$P(Y|X_1) = \sum_S P(Y, S|X_1) \tag{2.10.a}$$

$$= \sum_S [\prod_{t=2}^{T+1} P(s_t|s_{t-1}) \prod_{t=1}^{T} L(X_t, y_t, s_t)] \tag{2.10.b}$$

where $\sum_S$ means the summation over all possible state sequences.

Direct calculation of (2.10.b) involves on the order of $2TN^T(p+4)^T$ calculations (not counting the exponential evaluation) [11]. This calculation is computationally infeasible even for small values of p, N and T. For example, with $T = 100, N = 5, p = 6$ there are on the order of $2 \times 100 \times 5^{100} \times 10^{100} = 10^{172}$ computations. Clearly, a more efficient procedure is required to compute the total likelihood of the observation sequence Y. Such a procedure exists and is sometimes called the forward-backward algorithm.

The forward-backward algorithm uses two probability variables in a lattice calculation to compute the total likelihood of the observation. These variables are called forward and backward probabilities.

2.2 Forward-Backward Probabilities

Strictly speaking, only the forward probabilities are needed to compute the total likelihood of data. However, we will introduce the backward probabilities in this section since they will be used to compute the posterior probabilities in the reestimation formulas (see section §2.3).

Consider the forward probability $\alpha_t(i)$, defined for every state $i = 1, \ldots, N$ and for every time $t = 1, \ldots, T$ as

$$\alpha_t(i) = P(s_t = i, y_1, y_2, \ldots, y_t | X_1) \tag{2.11}$$

That is, $\alpha_t(i)$ is the joint probability of the partial observation sequence (until time t) and the event that state i is occupied at time t, conditioned on X_1.

In a similar manner, consider the backward probability $\beta_t(i)$, defined for every state $i = 1, \ldots, N$ and for every time $t = 1, \ldots, T-1$ as

$$\beta_t(i) = P(y_{t+1}, \ldots, y_T | s_t = i, X_t) \tag{2.12}$$

i.e. $\beta_t(i)$ is the probability of the partial observation sequence from $t+1$ to the end, given the joint event that state i is occupied at time t and X_t is observed.

The forward probabilities can be calculated recursively from this formula

$$\alpha_t(j) = [\sum_{i=1}^{N} \alpha_{t-1}(i) P(s_t = j | s_{t-1} = i)] L(X_t, y_t, j) \tag{2.13}$$

where $\sum_{i=1}^{N}$ means the summation over all possible states in the model. Recall that $L(X_t, y_t, i)$ is the likelihood of emitting the sample y_t and the event that the state i is occupied at time t, given the p previous observation samples ((2.6.c))

This is how (2.13) is computed: since $\alpha_{t-1}(i)$ is the probability of the joint event that $y_1, \ldots y_{t-1}$ are observed and the state i is occupied at time $t-1$, the product $\alpha_{t-1}(i) P(s_t = j | s_{t-1} = i)$ is then the probability of the joint event that $y_1, \ldots, y_{t-1}$ are observed and state j is reached at time t via state i at time $t-1$. Summing this product over all the N possible states i at time $t-1$ results in the probability of j at time t with all the accompanying previous partial observations. Once this is done and j is known, it is easy to see that $\alpha_t(j)$ is obtained by multiplying the summed quantity with the likelihood $L(X_t, y_t, j)$.

The total likelihood of the observation sequence Y ((2.10.b)) is the joint probability of the observation sequence $(y_1, y_2, \ldots y_T)$ and the event that state N is occupied

at time T and state $N+1$ (the sink state) occupied at time $T+1$:

$$P(Y|X_1) = \alpha_T(N)\, a_{N,N+1} \tag{2.14}$$

(again, we remind that the forward probabilities are sufficient to calculate the total likelihood of data).

On the other hand, the backward probabilities can be calculated recursively from this formula

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) P(s_{t+1} = j | s_t = i) L(X_{t+1}, y_{t+1}, j) \tag{2.15}$$

Again, the reason for (2.15) is as follows: in order to have been in state i at time t, and to account for the rest of the observation sequence, we had to make a transition to every one of the N possible states at time $t+1$, account for the observation sample y_{t+1} in that state, and then account for the rest of the observation sequence.

In term of the backward probabilities,

$$P(Y|X_1) = \beta_0(1) \tag{2.16}$$

We use (2.16) in the forward-backward algorithm as a checkpoint: the forward computation and the backward calculation must arrive to the same result which is $P(Y|X_1)$.

If we examine the computation involved in the calculation of $\alpha_t(i)$ or $\beta_t(i)$, $1 \leq t \leq T$, $1 \leq i \leq N$, we see that it requires on the order of N^2T calculation rather than $2TN^T$ as required by the direct calculation [11]. For $N = 5$, $T = 100$ the difference is about 3000 versus 10^{72}.

To complete the recursion formulas, we need to define the boundary conditions for $\alpha_1(i)$ and $\beta_T(i)$. By definition of the forward probabilities

$$\alpha_1(i) = P(s_1 = i, y_1 | X_1) \tag{2.17}$$

For our standard left-to-right HMM, since we require the first observation sample to be generated while the Markov chain is in the initial state, we have

$$\alpha_1(i) = \begin{cases} L(X_1, y_1, i) & \text{if } i = 1 \text{ (initial state)} \\ 0 & \text{otherwise;} \end{cases} \tag{2.18}$$

This serves as an initialization for the forward probability calculations. The initialization for the backward probability calculations is given by

$$\beta_1(i) = \begin{cases} a_{N,N+1} & \text{if } i = N \text{ (final state)} \\ 0 & \text{otherwise;} \end{cases} \tag{2.19}$$

This is how we can write (2.19): first, by definition (2.12) of the backward probability

$$\beta_{T-1}(i) = P(y_T | s_{T-1} = i, X_{T-1}) \tag{2.20.a}$$

$$= a_{iN} L(X_T, y_T, N) a_{N,N+1} \tag{2.20.b}$$

because, given that the state i is occupied at time $T-1$ and that the final state N must be occupied at time T, the probability to observe y_T is a_{iN} (which accounts for the transition from i to N), times $L(X_T, y_T, N)$ (which accounts for the likelihood of y_T at state N) times $a_{N,N+1}$ (which accounts for the event that the sink state must be reached to terminate the sequence).

On the other hand, bringing (2.19) and (2.15) together gives

$$\beta_{T-1}(i) = \sum_{j=1}^{N} \beta_T(j) P(s_T = j | s_{T-1} = i) L(X_T, y_T, j) \tag{2.21.a}$$

$$= \beta_T(N) P(s_T = N | s_{T-1} = i) L(X_T, y_T, N) \tag{2.21.b}$$

$$= a_{N,N+1} a_{iN} L(X_T, y_T, N) \tag{2.21.c}$$

which shows the correct values of $\beta_{T-1}(i)$.

The evaluation of $P(Y|X_1)$ can be viewed as the score of a given filter model. The filter model is specified by the parameter set $(\{a_{ij}\}, \{(B_i, \sigma_i^2)\})$. The score indicates how well that model matches the observation sequence. This viewpoint raises another question: given the sequence Y, how do we adjust the model parameters to best match the observations ? The answer to this question is the reestimation algorithm. Again, to use this algorithm effectively, we need to define two new probability variables: the posterior probabilities $\gamma_t(i,j)$ and $\gamma_t(i)$.

2.3 The Posterior Probabilities

We will define the posterior probabilities $\gamma_t(i,j)$ and $\gamma_t(i)$ that will be used later in the reestimation algorithm.

Let $\gamma_t(i,j)$ be the probability of being in state i at time $t-1$ and making a transition to state j at time t for every time $t = 1,...,T$, given the observation sequence Y. That is

$$\gamma_t(i,j) = P(s_{t-1} = i, s_t = j | Y, X_1) \tag{2.22}$$

We also define the probability of being in state i at time t for every $t = 1,...,T$, given the observation sequence Y, as

$$\gamma_t(i) = P(s_t = i | Y, X_1) \tag{2.23}$$

In terms of the forward and backward probabilities:

$$\gamma_t(i,j) = \frac{\alpha_{t-1}(i) a_{ij} \beta_t(j) L(X_t, y_t, j)}{P(Y|X_1)} \tag{2.24}$$

because the joint likelihood of Y and the event that the system is in state i at time $t-1$ and in state j at time t is $\alpha_{t-1}(i)$ (which accounts for the first $t-1$ observations ending in state i at time $t-1$), times $a_{ij}L(X_t, y_t, j)$ (which accounts for the local transition from state i to state j), times $\beta_t(j)$ (which accounts for the path being in state j at time t and then being unconstrained until the end of the observation sequence).

On the other hand, we can relate $\gamma_t(i)$ to $\gamma_t(i,j)$ by summing $\gamma_t(i,j)$ over j, giving

$$\gamma_t(i) = \sum_{j=1}^{N} \gamma_t(i,j) \tag{2.25}$$

Another way to compute $\gamma_t(i)$ is

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(Y|X_1)} \tag{2.26}$$

since $\alpha_t(i)$ accounts for $y_1, ..., y_t$ and state i at time t, and $\beta_t(i)$ accounts for $y_{t+1}, ..., y_T$ given state i at time t. The normalization factor $P(Y|X_1)$ makes $\gamma_t(i)$ a conditional probability so that

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{2.27}$$

In the next section we will show how we use the posterior probabilities to provide an algorithm for the reestimation procedure.

2.4 The Reestimation Formulas

Ideally the hidden Markov filter parameters $(\{a_{ij}\}, \{(B_i, \sigma_i^2)\})$ would be chosen so as to maximize the probability of the observation sequence $P(Y|X_1)$ given the model S. There is no closed form solution to this problem. An iterative solution, which leads to a local maximum of the likelihood function, is obtained by maximizing the following auxiliary function:

$$Q(M_0, M) = \sum_{S} P_0(S) \ln P(Y, S|X_1, M) \tag{2.28}$$

Here M, M_0 are two models corresponding to different choices for the parameter values, and $P_0(S)$ is the probability of S conditioned on the observation sequence:

$$P_0(S) = P(S|Y, X_1, M_0) \tag{2.29}$$

The following lemma is a simple consequence of the convexity of the logarithmic function [12]:

Lemma 1:

$$\ln \frac{P(Y|X_1, M)}{P(Y|X_1, M_0)} \geq Q(M_0, M) - Q(M_0, M_0)$$

The point of this inequality is that if M_0 is the model corresponding to an initial estimate of the parameters, the likelihood of the observation sequence can be increased by choosing the new parameters of the new model M so as to maximize $Q(M_0, M)$.

From (2.9) we have,

$$P(Y,S|X_1,M) = \prod_{t=2}^{T+1} P(s_t|s_{t-1}) \prod_{t=1}^{T} L(X_t,y_t,s_t) \tag{2.30}$$

Thus,

$$\ln P(Y,S|X_1,M) = \sum_{t=2}^{T+1} \ln P(s_t|s_{t-1}) + \sum_{t=1}^{T} \ln L(X_t,y_t,s_t) \tag{2.31}$$

and

$$Q(M_0,M) = \sum_{S} P_0(S) \left[\sum_{t=2}^{T+1} \ln P(s_t|s_{t-1}) + \sum_{t=1}^{T} \ln L(X_t,y_t,s_t)\right] \tag{2.32}$$

Breaking the brackets and manipulating the first term **I** of the right-hand-side of (2.32) gives the following:

$$\mathbf{I} = \sum_{S} P_0(S) \sum_{t=2}^{T+1} \ln P(s_t|s_{t-1}) = \sum_{i,j} \sum_{t=2}^{T+1} \left(\sum_{S} P_0(S)\delta_{s_t,j}\delta_{s_{t-1},i}\right) \ln a_{ij} \tag{2.33}$$

where

$$\delta_{s_t,i} \triangleq \begin{cases} 1 & \text{if } s_t = i \\ 0 & \text{otherwise;} \end{cases} \tag{2.34}$$

Using the definition of $\gamma_t(i,j)$ we can identify the expression between the parentheses as:

$$\sum_{S} P_0(S)\delta_{s_t,j}\delta_{s_{t-1},i} = \gamma_t(i,j) \tag{2.35}$$

Therefore:

$$\mathbf{I} = \sum_{i,j} \sum_{t=2}^{T+1} \gamma_t(i,j) \ln a_{ij} \tag{2.36}$$

In a similar manner, manipulating the second term **II** of the right-hand-side of (2.32) gives

$$\mathbf{II} = \sum_{S} P_0(S) \sum_{t=1}^{T} \ln L(X_t,y_t,s_t) = \sum_{i,j} \sum_{t=1}^{T} \gamma_t(i,j) \ln L(X_t,y_t,j) \tag{2.37}$$

Hence,

$$Q(M_0,M) = \sum_{i,j} \sum_{t=2}^{T+1} \gamma_t(i,j) \ln a_{ij} + \sum_{i,j} \sum_{t=1}^{T} \gamma_t(i,j) \ln L(X_t,y_t,j) \tag{2.38}$$

The two terms **I** and **II** can be maximized independently of each other since they depend on disjoint subsets of the model parameters.

We state here a useful lemma to help solving the maximization problem (the proof of this lemma can be found in [13]).

Lemma 2:

If $c_i > 0$ $i=1,\ldots,N$, *then subject to the constraint* $\sum_i x_i = 1$, *the function*

$$F(x) = \sum_i c_i \ln x_i$$

attains its unique global maximum when

$$x_i = \frac{c_i}{\sum_i c_i}$$

Now in the lemma 2, let c_i be the sum of $\gamma_t(i,j)$, i.e.

$$c_i = \sum_{t=2}^{T+1} \gamma_t(i,j) \tag{2.39}$$

and let $x_j = a_{ij}$. The first term **I** is maximized if

$$\overline{a_{ij}} = \frac{\sum_{t=2}^{T+1} \gamma_t(i,j|M_0)}{\sum_{p=1}^{N} \sum_{t=2}^{T+1} \gamma_t(i,p|M_0)} \tag{2.40}$$

where $\gamma_t(i,j|M_0)$ denotes the posterior probability $\gamma_t(i,j)$ of the model M_0.

Let us now consider the second term

$$M(i) = \sum_{i,j} \sum_{t=1}^{T} \gamma_t(i,j|M_0) \ln L(X_t, y_t, i) \tag{2.41}$$

Since $\ln L(X_t, y_t, j)$ is independent of i, (2.41) reduces to

$$M(i) = \sum_{t=1}^{T} \gamma_t(i) \ln L(X_t, y_t, i) \tag{2.42.a}$$

$$= \sum_{t=1}^{T} \gamma_t(i)[-\frac{1}{2}\ln \sigma_i^2 - \frac{(y_t - X_t B_i)^2}{2\sigma_i^2}] \tag{2.42.b}$$

To maximize $M(i)$ we set $\partial M/\partial B_i = 0$, which gives

$$\sum_{t=1}^{T} \gamma_t(i) X_t^*(y_t - X_t B_i) = 0 \tag{2.43}$$

Therefore

$$\overline{B_i} = [\sum_{t=1}^{T} \gamma_t(i) X_t^* X_t]^{-1} [\sum_{t=1}^{T} \gamma_t(i) X_t^* y_t] \tag{2.44}$$

Likewise, setting $\partial M/\partial \sigma_i = 0$ gives

$$\sum_{t=1}^{T} \gamma_t(i)[-\frac{1}{\sigma_i} + \frac{(y_t - X_t B_i)^2}{\sigma_i^3}] = 0 \tag{2.45}$$

Hence

$$\overline{\sigma_i^2} = \frac{[\sum_{t=1}^{T} \gamma_t(i)(y_t - X_t B_i)^2]}{\sum_{t=1}^{T} \gamma_t(i)} \tag{2.46}$$

An interesting detail is that the reestimation formula of the filter coefficients B_i in equation (2.13) has the form of the LPC solutions by least-square covariance method ([14] page 103). In fact, the equation of the LPC solutions by covariance method is:

$$\sum_{k=1}^{p} b_k \phi(i,k) = \phi(i,0) \tag{2.47}$$

for every $i = 1, 2, ..., p$, where

$$\phi(i,k) = \sum_{m=-i}^{K-i-1} y_m y_{m+i-k} \tag{2.48}$$

(K is the time interval inside which the signal is considered). Thus

$$\sum_{k=1}^{p} b_k \left(\sum_{m=-i}^{K-i-1} y_m y_{m+i-k} \right) = \sum_{m=-i}^{K-i-1} y_m y_{m+i} \tag{2.49}$$

or

$$\sum_{m=-i}^{K-i-1} y_m \left(y_{m+i} - \sum_{k=1}^{p} b_k y_{m+i-k} \right) = 0 \tag{2.50}$$

Let $t = m + i$ and $K = p$, then we can write (2.50) as

$$X_t^*(y_t - X_t B_i) = 0 \tag{2.51}$$

The only difference between (2.50) and (2.43) is the factor $\sum_{t=1}^{T} \gamma_t(i)$. This factor is a weighting factor. It takes into account the probability that the state i is occupied at time t. Since there is a similarity between (2.43) and (2.50), we will see (in the next chapter) that the value of the filter gain computed by covariance method is similar to the value of our filter variance σ^2 given by (2.46).

The equation (2.10) gives the reestimation formula for the transition probabilities. Equations (2.11) and (2.16) are the reestimation formulas for the filter coefficients and the error variances. Together they produce a set of new model parameters $M = (\{a_{ij}\}, \{(B_i, \sigma_i^2)\})$ based upon the previous model parameters M_0. Therefore, if we iteratively use M in place of M_0 and repeat the above reestimation calculation, we can improve the likelihood of the data Y on each iteration. The result is the estimated model. This leads to the following iterative algorithm (referred as forward-backward or Baum-Welch training procedure):

1. Guess an initial set of parameters $(\{a_{ij}\}, B_i, \sigma_i^2)$.
2. Compute $\overline{a_{ij}}$, $\overline{B_i}$, and $\overline{\sigma_i^2}$ according to the reestimation formulas in Eq.(2.10), (2.11) and (2.16).
3. Set a_{ij} to $\overline{a_{ij}}$, B_i to $\overline{B_i}$ and σ_i^2 to $\overline{\sigma_i^2}$.
4. If some convergence criteria are not met, go to step 2.

Step 1 requires a good initialization technique in order to speed up the convergence of steps 2,3 and 4 and to avoid the problem of short segments. Furthermore, the calculation of the posteriori probabilities $\gamma_t(i,j)$ and $\gamma_t(i)$ requires some mathematical manipulations that reduces the computation load. All these implementation issues of the algorithm will be discussed in the next chapter.

Chapter 3 Speech Segmentation

Segmenting a sequence of speech samples consists of finding a sequence of segments (or states) that has the highest likelihood of generating the observation speech samples. This chapter will show how we use the Baum-Welch algorithm in speech segmentation. It also discuss some problems that arise in implementing the Baum-Welch algorithm. The results of segmentation tests on continuous speech and isolated word speech data will be presented.

3.1 Segmentation Approaches

The segmentation problem consists of finding the optimal state sequence S associated with the given observation sequence Y. The most widely used approach is to find the the state sequence (path) S for which the posterior probability $P(S|Y,M)$ is maximal. This is equivalent to maximizing $P(S,Y|M)$ (the total likelihood of the observation sequence Y of the model M).

A formal technique for finding this best state sequence exists, based on dynamic programming methods, and is called the Viterbi algorithm [4]. We will briefly present the steps of this algorithm here (without any proofs). Interested readers are directed to [4] for a more fundamental development of the Viterbi procedure.

Define the quantity $\delta_t(i)$ for every $i = 1,...,N$ and $t = 1,...,T$ as

$$\delta_t(i) = \max_{s_1,s_2..s_{t-1}} P(s_1 s_2..s_t = i, y_1 y_2..y_t|M) \tag{3.1}$$

i.e. $\delta_t(i)$ is the score on the state sequence ending in state i at time t which best accounts for the first t observations. An array $\phi_t(j)$ (also defined for every $i = 1, ..., N$ and $t = 1, ..., T$) is used to keep track of the argument which maximizes $\delta_t(i)$ for each t and j. The complete Viterbi procedure can now be stated as follows:

1-Initialization:

$$\delta_1(i) = \begin{cases} L(X_1, y_1, i) & \text{if } i = 1 \text{ (initial state)} \\ 0 & \text{otherwise;} \end{cases}$$

$$\phi_1(i) = \begin{cases} 1 & \text{if } i = 1 \text{ (initial state)} \\ 0 & \text{otherwise;} \end{cases}$$

2-Recursion for $2 \leq t \leq T$ and $1 \leq j \leq N$:

$$\delta_t(j) = [\max_{1 \leq i \leq N} \delta_{t-1}(i) a_{ij}] L(X_t, y_t, j)$$

$$\phi_t(j) = [\arg \max_{1 \leq j \leq N} \delta_{t-1}(i) a_{ij}] L(X_t, y_t, j)$$

3-State sequence backtracking: the optimal state sequence is $q_1^*, ..., q_T^*$, where

$$q_T^* = N$$

and for $1 \leq t \leq T - 1$

$$q_t^* = \phi_{t+1}(q_{t+1}^*)$$

It should be noted that the Viterbi algorithm is similar (except for the backtracking step) in implementation to the forward-backward training procedure of chapter 2. However, a maximum over previous states is used in place of the summing procedure used previously.

An alternative segmentation approach – which we use in this thesis – is to employ the posterior probability $\gamma_t(i)$ (which is the probability of being in state i at time t): we will choose the state sequence S that maximizes $\gamma_t(i)$ at every time $t = 1, 2, ..., T$. More precisely, at every time $t = 1, 2, ..., T$ we select the state s_t whose posteriori probability $\gamma_t(s_t)$ is highest, i.e.

$$s_t = \arg \max_{1 \leq i \leq N} \gamma_t(i) \tag{3.2}$$

Since $\gamma_i(t)$ is computed at each time t for every state $i = 1, ..., N$, the segmentation using $\gamma_i(t)$ is automatically a by-product of our forward-backward training algorithm.

There is a problem that might occur with the segmentation by (3.2): the state sequence S is chosen without regard to the neighboring (in time) states. The result is then S might be a non-valid left-to-right Markov chain, because there is no constraint on the event that the state i must appear in S before state j (under the condition $i < j$, of course). In other word, while it seems appropriate to have a result segmentation as

$$S = (s_1, s_1, s_2, s_2, s_2, s_3, s_3, s_4, ..., s_N)$$

mathematically we might end up having a sequence

$$S = (s_1, s_1, s_1, s_3, s_3, s_3, s_2, s_2, s_5, ..., s_N)$$

that does not fit to our model. However, several of our experiments have shown that the *unsupervised* result of segmentation of the forward-backward algorithm always yield a valid left-to-right Markov chain that satisfies all the constraints stated earlier in chapter 2.

A more serious problem common to both the segmentation using posterior probabilities and the Viterbi segmentation is that artificially short segments may be produced as a result of the Baum-Welch training procedure. The point is that the likelihood of the data may be made very large by using a very small number of samples to estimate the variances associated with some of the state (see [15], pp. 198–202 for a discussion of this problem in connection with the variances of the components of mixture distributions).

One of the various techniques that can be used to solve the problem of short segments is to start the Baum-Welch training with a good initial estimate for the parameter set $(\{a_{ij}\}, B_i, \sigma_i^2)$ ([15] page 201). In fact, our experiences indicated that when we initiated the filter parameter with an ordinary LPC analysis performed on the input speech samples, the problem of short segments has not occured.

To be more specific, the speech waveform is first windowed to N uniform segments (N is the number of states in the model). Linear predictive coefficients and the noise variance of each segment are then computed. These values are used as initial values for the hidden filters. Assuming a reasonable amount of training data exist, this initialization method shows fairly conclusive that it is sufficient.

Another fact in our experiences is that the average ratio samples-per-state (T/N) has a direct effect on the problem of short segments. The fewer number of states allocated for a sequence of speech samples results in longer segments of speech. However, fewer states might lead to long segments that overview the characteristics of some short phones of speech (e.g stops, fricatives). Several of our experiments show that with the average ratio of 320 samples/state (at 16kHz sampling frequency), no short segments are observed.

3.2 Implementation Issues

In this section, we present some problems that arise in the implementation of the Baum-Welch algorithm. They include initialization method, approximative computation, staircase approach and segment duration.

3.2.1 Initialization

We have not addressed in previous section the issue of how the statistics and the filter parameters are initialized for the forward-backward training. While the forward-backward algorithm guarantees an improvement every iteration, it requires a good initial estimate for the parameter set $(\{a_{ij}\}, B_i, \sigma_i^2)$ to avoid the problem of short segments (also called the problem of singular maximum likelihood) and to speed up the convergence of the training procedure.

The transition probabilities $[a_{ij}]$ are initialized by equal distribution such that

$$a_{ij} = \begin{cases} 0.5 & \text{if } j = i \text{ or } j = i+1 \\ 0 & \text{otherwise;} \end{cases} \tag{3.3}$$

The filter coefficients and error variance are initialized as follows: first, the speech sequence $Y = (y_1, \ldots, y_T)$ is uniformly segmented to N equal segments (the number of states N is chosen according to the length of the sequence and the speech sampling frequency as discussed later in this chapter). The linear-predictive coefficients and the variance of each segment are then computed. This set of parameters are used as initial values of the filter parameters.

Our tests have shown that this simple initialization technique was sufficient to avoid the problem of short segments. It also helped the training algorithm to converge after 3–5 iterations with a threshold of the total likelihood $P(Y|X_1)$ fixed at 30 dB. The threshold δ is defined by a difference in log domain of the total likelihood of the observation sequence Y recorded in two successive iterations, i.e.

$$\delta = \ln P_{k+1}(Y|X_1) - \ln P_k(Y|X_1) \tag{3.4}$$

where $P_k(Y|X_1)$ is the total likelihood of observation recorded at the kth iteration.

3.2.2 Log. Compression and Approximative Computing

The evaluation of $\alpha_t(i)$ and $\beta_t(i)$ for $1 \leq t \leq T$ and $1 \leq i \leq N$ in the recursive formulas shows that as $T \rightarrow \infty, \alpha_T(i) \rightarrow 0$ and $\beta_1(i) \rightarrow 0$ in exponential fashion. In practice the number of observations necessary to adequately train a model or compute its probability will result in underflow of any real computer if $\alpha_t(i)$ and $\beta_t(i)$ are evaluated directly. Therefore, a scaling procedure is necessary to avoid this underflow.

An alternative approach which we use is to represent probabilities by their logarithms to deal with underflowing probabilities. If we represent probability P with its log, $\log_b P$, we could get more precision in computing. To multiply two numbers, we simply add their logarithms. To add two numbers, we proceed as follows:

$$\log_b(P_1 + P_2) = \log_b[b^{\log_b P_1} + b^{\log_b P_2}]$$
$$= \log_b[b^{\log_b P_1}(1 + b^{\log_b P_2 - \log_b P_1})]$$
$$= \log_b P_1 + \log_b(1 + b^{\log_b P_2 - \log_b P_1})$$

If P_2 is many orders of magnitude smaller than P_1, adding two numbers will just result in P_1.

3.2.3 Staircase Approaches

The forward and backward probabilities $\alpha_t(i), \beta_t(i)$ and the posterior probabilities $\gamma_t(i,j), \gamma_t(i)$ are computed for every time t and every state i of the model. With a typical speech sentence of 1 sec and a sampling rate of 8 kHz, the time T is 32000. Moreover, ordinary speech contains on average 15-20 phones per second, each phone could be accurately modeled by 3 segments. Therefore, for the 4 sec speech sentence, one iteration of our algorithm has to handle up to $3 \times 20 \times 4 = 200$ segments ($1 \leq i \leq 200$) and 32000 units of time ($1 \leq t \leq 32000$). These dimensions would apply to all vectors $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i,j)$, and $\gamma_t(i)$. It clearly overflows the capacity of computer.

However, in the left-to-right HMM, at time t close to T, the initial state 1 (and perhaps states 2,3,...) is certainly not visited, so there is no need to evaluate $\alpha_t(1), \beta_t(1)$... at that time. To optimally use our reestimation algorithm in that situation, we introduce the staircase approach.

We assume that each state i of the Markov model can only occupy a limit number of samples, say Δt_i. This occupation can only start at some time T_i. Therefore, the probabilities $\alpha_t(i)$ and $\beta_t(i)$ of the state i are zero outside the interval $[T_i, T_i + \Delta t_i]$. We will only evaluate these probabilities inside that time interval, thus reduce the dimension of the vectors.

The crucial point is how we choose the starting time T_i and the length Δt_i. Our tests indicated that overlapped time intervals are needed to assure the same result as the original model. More precisely, we use the intervals defined as follows: for every

$i = 1, \ldots, N$, we define

$$T_i = \frac{i \times T}{N} \tag{3.5}$$

and

$$\Delta t_i = \Delta t = \frac{T}{N} \tag{3.6}$$

and the time intervals are $[1, T_1 + \Delta t], [T_1 - \Delta t, T_2 + \Delta t], \ldots, [T_{N-2} - \Delta t, T_{N-1} + \Delta t], [T_{N-1} - \Delta t, T]$. Figure 3.1 illustrates the state occupation we are assuming.

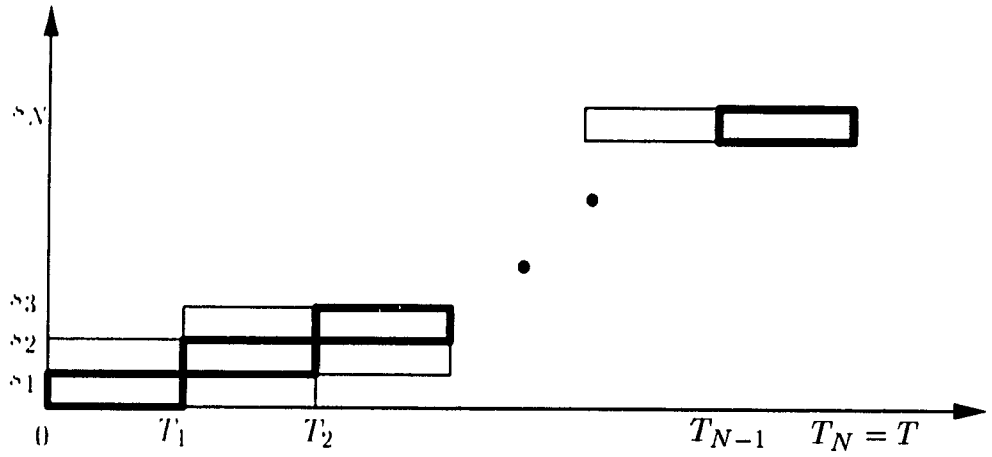

Fig. 3.1 The staircase approach

3.2.4 Segment Duration

The segment duration problem consists of constraining the minimum number of samples per segment. If the number of samples allocated to a segment is close to the order p of the hidden filter, we could have a local maximum because the prediction errors are very small. For example, a 10 ms stop burst of speech recorded at 16000 Hz is represented by 160 samples. With a minimum required 3 segments (or 3 states) per phone (one source, one stationary and one end), that stop burst has on average 50 60 samples/segment. If we pick a female speaker with a pitch frequency 320 Hz (hence one pitch period has 16000/320=50 samples), the 60-sample segment will cover a length of 60/50=1.2 pitch, barely enough to perform an LPC analysis with $p = 12$.

In our first implementation of the forward-backward algorithm, we did not use any complex duration constraint methods. We just simply isolated a sequence of speech samples, allocated on average 20 phones per second of speech and a fixed 3 states/phone (that gives on average 60 states/second), then ran the training program. At the output segmentation, we checked all the segment lengths. If any segment having fewer than 50 samples was observed (the 50-sample level was used in both continuous-speech and isolated-word database), we reduced the number of states (usually by 1 or 2 states) of the model and reran the training.

Our segmentation tests on the isolated-word data with an average 20ms per segment shows that about 5% of the words requires the reruns of the forward-backward algorithm.

In the next section, we will show the test results of speech segmentation using the posterior probabilities.

3.3 Experimental Results

The segmentation approach using the posterior probabilities $\gamma_l(t)$ was examined in segmenting speech waveforms of two categories of speech: continuous-speech and isolated-word.

The speech material used for the continuous-speech category includes the following sentences:

1- Oak is strong and also gives shade.

2- Add the sum to the product of these three.

The first sentence was spoken by a male and the second sentence by a female. The sampling rate of these sentences is 8 kHz.

The data used in the isolated-word category include 427 words spoken by a native English female speaker with a pause of at least 150 ms between words. The words

correspond to paragraphs selected from magazines, books and newspaper articles. The sampling frequency is 16 kHz.

We apply the forward-backward training procedure to the speech data with an order of hidden filter $p = 12$ (i.e 12 poles). This choice of p is based on previous LPC analysis ([11] pp 119 120): generally 2 poles per kiloHertz due to the vocal tract contribution plus 3 4 poles to represent the source excitation and the radiation load.

Figures 3.2 and 3.3 show the result of our segmentation approach when it is applied to the two sentences of continuous speech. The graphics display

(a)- The signal time waveforms along with the segment boundaries (vertical bars) chosen by maximizing $\gamma_t(i)$.

(b)- The *normalized prediction error* E of each segment (variable lenght).

(c)- The normalized prediction error E of the uniform segments.

We present the normalized prediction error E because it is a very useful parameter for the determination of the optimal number of poles p and for the measure of the spread of the data [16]. The normalized prediction error E of an LPC filter is defined as the estimated variance of the filter scaled by the average of the square of the segment amplitude.

To be more specific, let $(y_1, ..., y_M)$ be M samples of a segment having a Gaussian noise source $N(0, \sigma_s^2)$, then the normalized prediction error E of that segment is

$$E = \frac{\sigma_s^2}{\frac{1}{M}\sum_{t=1}^{M} y_t^2} \tag{3.7}$$

Since the mean of the segment amplitude of speech is very small (≈ 0), this normalized error also corresponds to the variance of the filter output scaled by the variance of the segment speech signal σ_y^2:

$$E = \frac{\sigma_s^2}{\sigma_y^2} \tag{3.8}$$

It is easy to see that the prediction gain P of the filter (in dB) is

$$P = 10\log_{10}(\frac{1}{E}) \tag{3.9}$$

Recall that there is a similarity between our filter coefficients and the filter coefficients computed by covariance method. The comparisons of our filter's normalized prediction errors and the normalized prediction errors computed by the covariance method should give similar results.

In Figure 3.2.(a) and 3.2.(b), the speech waveform of *Oak is...* with segment boundaries and their corresponding normalized prediction errors are shown. The speech sequence has 3021 samples in 23 segments (average 16.4ms/segment). The /k/ burst, located in the interval 0.22-0.25sec, is characterized by three segments of 100,78 and 181 samples (at 8 kHz sampling frequency) respectively, with a peak error recorded for about 10ms. The vowel /O/, started at 0.04sec and ended at 0.2sec, occupies 12 segments with normalized prediction errors $E < 0.01$. The period of silence at the beginning of the sequence has small error E while the silence between words has larger error E. With an average 131 samples/segment (16.4ms/segment), the shortest segment (2nd from left) has 53 samples and the longest segment (belong to the vowel /O/) has 239 samples. Figure 3.2.(c) shows the normalized prediction errors of the 16.4ms uniform segments. It is easy to see that the uniform window produces higher normalized prediction errors for the stop /k/.

Figure 3.3.(a) and 3.3.(b) show the waveform of *Add the...* with segments and the corresponding normalized prediction errors. There are 3360 samples in 28 segments (average 20ms/segment). Moderate amplitudes of normalized prediction errors E's are observed at the transition from silence to vowel /A/. High values of normalized prediction errors are seen at the fricative /*θ*/ (in the word "the") and again at the silence at the end of sequence. We can see in Figure 3.3.(c) that the normalized prediction errors of the 20ms uniform segments are generally higher than the errors of the variable frame length segments.

It is interesting to note from these figures that the normalized prediction errors E's for unvoiced speech (e.g. stops, fricatives) is significantly higher than for voiced speech (vowels, diphthongs). In fact, the normalized prediction error curves show

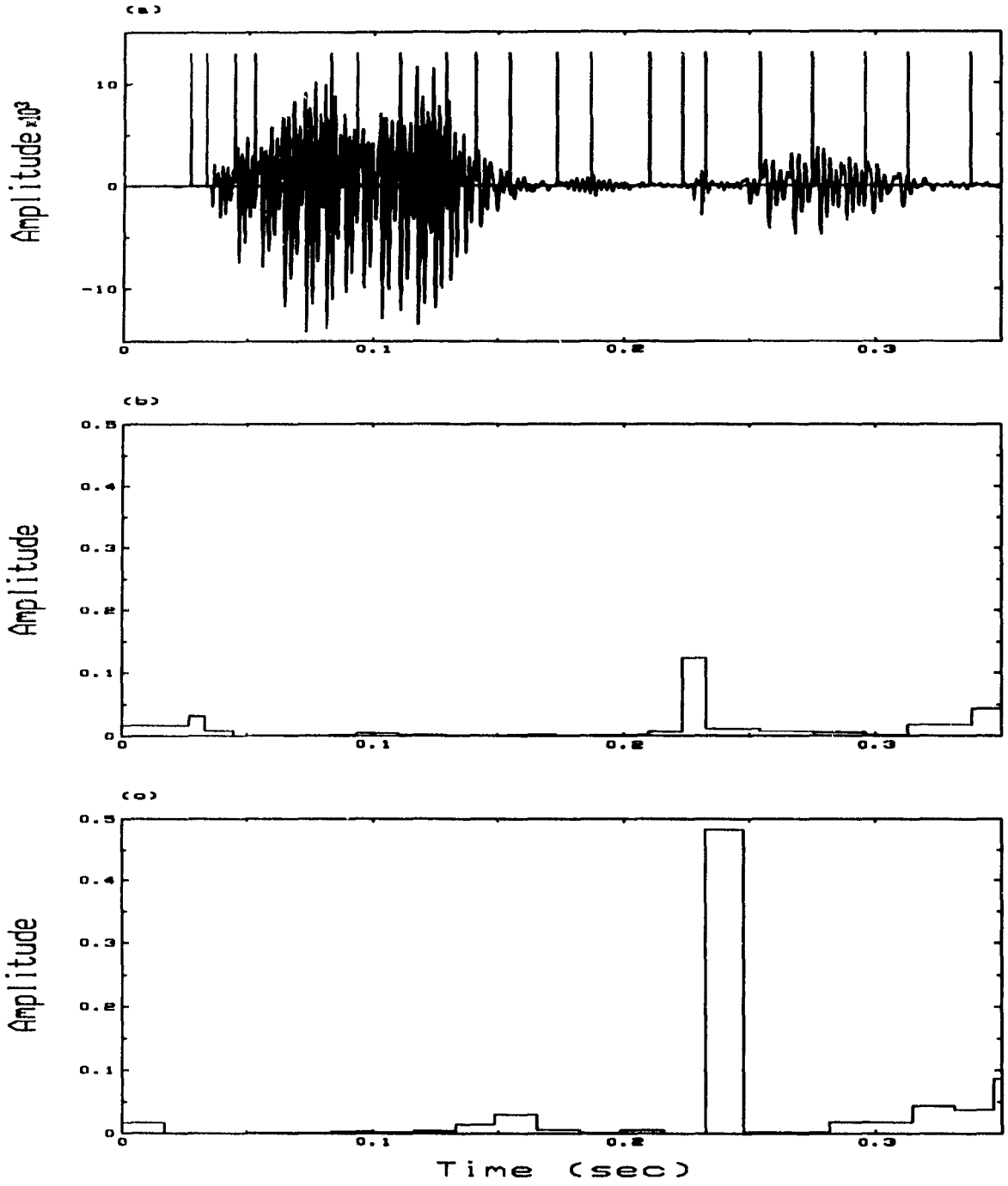

Fig. 3.2 (a)-Speech waveform of *Oak is..* with segment boundaries (variable length), (b)-The normalized prediction errors of the segments of (a) and (c)-The normalized prediction errors of the 16 4ms uniform segments The speech sequence has 3024 samples, distributed into 23 segments (average 16 4ms/segment). In (a) and (b), the shortest segment has 53 samples and the longest segment (belong to the vowel /O/) has 239 samples The burst /k/ is located in the interval 0 2-0 25sec

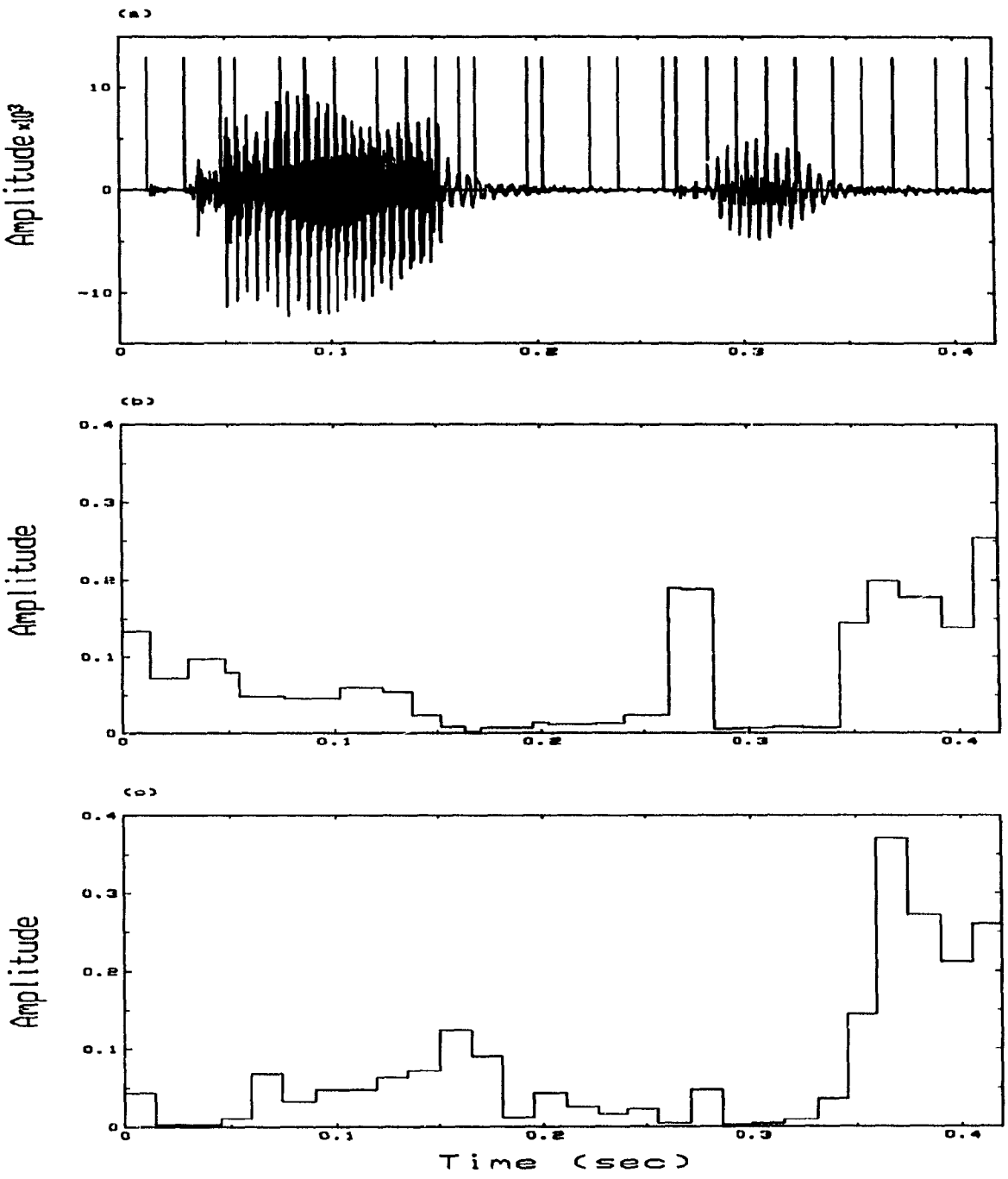

Fig. 3.3 (a)-Speech waveform of *Add the. .* with segments boundaries (variable length), (b)-The normalized prediction errors of the segments of (a) and (c)-The normalized prediction errors of the 20ms uniform segments. There are 3360 samples in 28 segments (average 20ms/segment) In (b), moderate amplitudes of error E are observed at transition from silence to vowel /A/ High values of E are seen at the fricative /θ/ (in the word "the") and at the end of sequence

a very small range of error E (0.005–0.01) for voiced speech (such as vowels /O/ in "Oak", /I/ in "gives"or voiced fricative /v/ in "of") and a high level of error E (0.12–0.25) for unvoiced speech (e.g. stop /k/ in "Oak", fricative /s/ in "sum"). Also the lengths of segments associated to voiced speech are generally longer than the lengths of segments associated to unvoiced speech.

The reason for the above observations is that voiced speech is usually longer (in time) than unvoiced speech (e.g. several pitch periods in vowels) and it has a predictably waveform pattern which helps the linear predictive filters to perform better. On the other hand, the silence may have large error E because it has very small signal amplitudes which are in the same order of the Gaussian noise source of the filter.

A comparison of these values of normalized errors Es with previous studies on speech analysis by linear prediction [14], pp. 426–429, shows that our errors are within the order of the ordinary prediction errors by covariance method.

In a second test, we apply the segmentation procedure to the isolated word database. These words have been segmented into phones (of a set of 44 phones) by an experimental large-vocabulary-speaker-dependent isolated-word recognizer (described in detail in [17]). The normalized prediction errors E's of each phone are computed for 20 phones of the set and are presented in Table 3-1 (the number of occurrences of each phone used to compute the average values is given in parentheses beside the phone in the table). Again, by comparison with data in [14] pp. 426–429, the normalized prediction errors E's show very accurate measurements of the phone's characteristic.

The segmentation using hidden filter model shows that it performs exceedingly well on the difficult task of locating short phones of speech. Since our aim is to improve segment boundaries between phones during recognition, we try to implement the segmentation at a preprocessing stage of a large-vocabulary speaker-dependent isolated-word recognizer. The next chapter will describe how the hidden filter models can be used in the recognizer. It also reports experimental results of speech recogni-

Phone	Normalized Error E ($\times 10^{-3}$)
/A/ (35)	3.03
/I/ (104)	9.09
/U/ (6)	3.59
/E/ (48)	8.07
/O/ (14)	0.978
/aj/ (23)	4.52
/ɪ/ (70)	4.79
/u/ (40)	1.44
/ɛ/ (30)	8.07
/o/ (18)	1.51
/ˆ/ (68)	4.01
/&/ (9)	6.81
/f/ (31)	72.81
/t/ (129)	38.38
/p/ (33)	42.43
/s/ (100)	136.69
/d/ (78)	20.04
/k/ (53)	21.93
/r/ (128)	2.16
/g/ (21)	6.47

Table 3.1 Results of segmentation of the isolate-word database

tion.

Chapter 4

Speech Recognition with the Hidden Filter Preprocessor

Several large-vocabulary recognizers have been developed in the past few years [18], [17]. Speaker independence and continuous speech pose the greatest challenges for these recognizers. Speaker independence was the most difficult constraint to overcome because most parametric representations of speech are highly speaker-dependent, and a set of reference pattern suitable for one speaker may perform poorly for another speaker. On the other hand, continuous speech recognition is significantly more difficult than isolated word recognition, resulting from problems of word boundary, coarticulatory effects and word emphasizing.

In our first recognition tests, we choose to work only with a speaker-dependent isolated-word recognizer. This chapter will show how we apply the forward-backward training and segmentation procedure to a large vocabulary speaker dependent isolated word recognizer. The segmentation is used as a feature-extractor at the preprocessing stage of the recognizer.

4.1 Overview of the 60000-Word Recognizer

The recognizer we are using is a 60000-word vocabulary speaker trained isolated-

word recognizer which uses a phonemic Markov model approach to speech recognition. The goal of the recognizer is to transcribe text spoken as a sequence of isolated word.

For each spoken word, the recognizer uses acoustic information and rough likelihoods in a fast search algorithm to narrow the possible word hypotheses from the 60000 words in the total vocabulary to a sequence of lexically valid, most likely phoneme strings, together with their likelihood. One Markov model per phone for the 44 phones is used in recognition.

The fast search algorithm of the recognizer is an A^* admissible heuristic (developed by Kenny [19]) for rapid lexical access. It is capable, on demand, of generating multiple recognition hypotheses from a lexicon and a dictionary of 60000 words. Phone duration constraints are also incorporated in the recognizer to improve the accuracy and the speed of the search.

The acoustic information (or acoustic features) is extracted from the speech waveform by the parameter estimation module of the recognizer. These features are sets of 15-dimensional feature vector computed every 10 ms from the speech waveform using 25.6 ms overlapped window. The 15-dimensional vector consists of seven mel-based cepstrum coefficients ($C_1, ..., C_7$) and eight dynamic parameters ($\Delta C_0, ..., \Delta C_7$).

The static cepstral coefficients ($C_1, ..., C_7$) are computed by first dividing the spectrum between 0 and 8 kHz into 24 channels spaced according to the mel scale of frequency. The center frequencies for the first ten channels are spaced 100 Hz apart, while the remaining 14 channels are spaced logarithmically. The energy in each channel is computed by summing a triangularly weighted spectrum located at the center of the channel. Taking the log of the channel energies yields the log channel energies. The cosine transform of the vector of 24 log channel energies given by

$$C_i = \sum_{j=1}^{24} E_j \cos(i(j-0.5)\frac{\pi}{24}), \quad i = 1, 2, ...7 \tag{4.1}$$

where E_j is the log channel energies of the jth channel, gives us the cepstrum coefficients.

C_0 is computed as the weighted sum of the log channel energies

$$C_0 = \sum_{j=1}^{24} W_j E_j \tag{4.2}$$

where the weights W_j, $j = 1, ..., 24$ are (0.0016, 0.0256, 0.1296, 0.4096, 1.0,..., 1.0).

The eight dynamic parameters $(\Delta C_0, ..., \Delta C_7)$ are obtained by taking signed differences between the corresponding static cepstral values 40ms apart. The resulting 15-dimensional feature vector $(C_1, ..., C_7, \Delta C_0, ..., \Delta C_7)$ is computed every 10ms.

4.2 Hidden Filter Preprocessor

The construction of acoustic segments every 10ms does not take into account the phone boundaries (e.g. boundaries between stops and sonorants, affricates and sonorants etc). The changing statistical characteristics of speech signal at these boundaries fall often into one segment. This creates difficulties and erroneous in mapping acoustic segments into phoneme models because the recognizer must map the segment to one of the phones.

Our strategy is to replace the parameter estimation module by a preprocessor which generates sets of 15-dimensional feature vectors based upon hidden filter models. More precisely, for each spoken word in the training and test sets of the recognizer, the preprocessor performs the forward-backward reestimation to provide segments of speech characterized by variable time lengths, 12th order linear predictive coefficients (B_s) and normalized errors E's.

The linear predictive coefficients are translated to cepstrum coefficients $(C_k, k = 1, ..., 7)$ using the formula (see [20] pp. 229-231 for a development of this formula):

$$\ln A(z) = -\sum_{k=1}^{\infty} C_k z^{-k} \tag{4.3}$$

where $A(z) = 1 - b_1 z^{-1} - b_2 z^{-2} - ... - b_{12} z^{-12}$ is the impulse response of the segment filter. We choose to work with cepstrum coefficients as acoustic features because they

have been shown to give improved recognition performance compared to a number of other feature parameters used in speech recognition [21].

Only the first seven cepstrum coefficients are retained from (4.3) to form the static cepstrum coefficients $(C_1, ..., C_7)$. C_0 is defined as log magnitude of the filter variance ([20] page 230), i.e.

$$C_0 = \ln \sigma^2 \tag{4.4}$$

These coefficients are called *LPC-based* cepstrum coefficients since they are computed from the linear predictive coefficients.

The eight dynamic parameters $(\Delta C_0, ..., \Delta C_7)$ are obtained by taking signed differences between the corresponding static cepstral values.

The seven LPC-based cepstrum coefficients $(C_1, ..., C_7)$ and the eight dynamic parameters $(\Delta C_0, ..., \Delta C_7)$ are put together to form a 15-dimensional feature vector $(C_1, ..., C_7, \Delta C_0, ..., \Delta C_7)$.

Although there is no fixed-time window when we compute our 15-dimensional feature vectors $(C_1, ..., C_7, \Delta C_0, ..., \Delta C_7)$, the number of segments allocated for each word in the training and test sets of the recognizer result on average 20ms per segment. The 20ms length is chosen to compromise the calculation time used by the preprocessor to complete the segmentation, the computer load to sustain all arrays declared in the forward-backward training program and the minimum duration of 50 samples per segment as stated in §3.2.4.

The preprocessor runs with either a maximum of 8 reestimation iterations or a scoring threshold δ of 30dB, which ever comes first. Under these conditions, we observe that the CPU time needed to provide acoustic segments of one word is 5 minutes on average on a DEC station 2100 computer.

4.3 Experimental Results

The experimental setup consists of different sentences read in a quiet room by a female native English speaker. The sentences are read from texts with pauses of at

least 150ms between the words. The texts are selected from magazines, books and newspaper articles. Part of the texts is used for training the phone models while the remaining is used for estimating the recognition accuracy of the system. Of the approximately 2200 words involved in the experiment, 1299 words are used for training the model.

Table 4.1 shows the recognition results with the 15-dimensional variable length LPC-based cepstrum coefficient feature vectors. With 808 words of the test set, the recognizer correctly identifies only 441 words (54.6% of accuracy).

Several possible causes could lead to this poor performance:

- The average 20ms per segment could result in some misrepresentations of phone characterizations, especially if the segment of one phone covered samples belong to other phones.

- The number of segments allocated for each word may, in some cases, come to fewer segments associated to a phone than the minimum number of states of that phone in the recognition phone models.

  For example, in the recognizer, the Markov model for the phone /k/ had 6 states. A left-to-right Markov chain with skip transitions for that 6-state model is illustrated in figure 4.1. As one can see, the minimum number of states (or segments) required to go through this Markov chain is 4: either a path through $s_1 - s_2 - s_4 - s_6$ or $s_1 - s_3 - s_4 - s_6$ or $s_1 - s_2 - s_5 - s_6$. A survey of the segmentation at the preprocessor showed that, over 53 segmented phones /k/, 12 phones had only 3 segments per phone, less than the required minimum number of states. Thus the phone model for /k/ was not accurate.

- The recognizer does not take into account the length of each segment calculated at the preprocessor. It treats the sequence of 15-dimensional feature vectors as a sequence of uniform-time vector-valued observations.

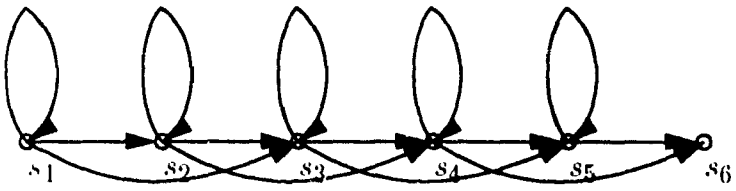

Fig. 4.1 The Markov model for a 6-state phone.

Feature vectors input to recognizer	Total words		Acoustic recognition accuracy
	Training	Test	
variable-length LPC-based cepstrum	1299	808	441 (54.6%)

Table 4.1 Recognition results with LPC-based cepstrum preprocessor.

To correctly interpret the results of the recognizer with variable-length LPC-based coefficients in Table 4.1, we performed another recognition test, where 20ms-uniform LPC-based cepstrum coefficients are used to evaluate the recognition accuracy. To obtain the 20ms-uniform LPC-based cepstrum coefficients, we window the input speech waveform to several segments of 20ms (uniform) and compute their linear predictive coefficients by covariance method. We then translate these coefficients to cepstrum coefficients and form the 15-dimensional vectors as we did with the variable-length LPC-based cepstrum coefficients.

Again, we run the recognizer with the 20ms-uniform LPC-based cepstrum coefficients and compare the results with Table 4.1. Table 4.2 shows the output of this test. Final results proves that the preprocessor actually increases the recognition accuracy from 429 to 441 (53.1% to 54.6%). Broken down to each text, the variable-length model outperforms the 20ms-uniform model (in term of recognition accuracy) by a score of 5:2, i.e. in five of the seven text files used in the test set, the variable-length model scores higher accuracy rates than the 20ms-uniform model.

Text	Total words	Acoustic recog accuracy	
		20ms-uniform	variable-length
Hitman	105	50 (47.6%)	58 (55.2%)
Ira	102	59 (57.8%)	60 (58.8%)
Reptiles	110	48 (43.6%)	49 (44.5%)
Riot	135	74 (54.8%)	76 (56.3%)
Soap	142	81 (57.0%)	84 (59.2%)
Spy	111	61 (54.9%)	60 (54.1%)
Women	103	56 (54.4%)	54 (52.4%)
Average	808	429 (53.1%)	441 (54.6%)

Table 4.2 Recognition results with the 20ms-uniform LPC-based model and with the variable-length LPC-based cepstrum model.

A close analysis of the results in Table 4.2 shows that when the recognizer with 20ms-uniform coefficients correctly identifies the word, most of the time, the recognizer with variable-length coefficients also correctly identifies the word. In particular, for the data of the texts "Hitman"and "Ira", Table 4.3 shows some differences between the top word choices of the recognizer with two set of acoustic features.

Among the words correctly identified by the recognizer with variable-length coefficients (the preprocessor) and incorrectly identified by the recognizer with 20ms-uniform coefficients, we find some long words such as "privileges", "police", "informer", and some simple words such as "since", "other", "jail". The errors made by the recognizer without preprocessor are usually stops (confusion with nasals or fricatives) and fricatives (confusion with stops).

The results of this test show fairly conclusively that, if placed in the same context (i.e. 20ms fixed window versus average 20ms segment length), the preprocessor lightly improves acoustic accuracy of the speaker-dependent isolated-word recognizer. The improvement comes generally from stops, nasals and fricatives, i.e. short phones.

In summary, the comparison of the results of recognition with and without pre-

Words (with true transcriptions)	Recognizer with variable-length LPC-based coeff.	Recognizer with 20ms-uniform LPC-based coeff.
murder /mrdr/	murder (C) /mrdr/	burder (W) /brdr/
since /sIns/	since (C) /sIns/	sits (W) /sIts/
jail /dzel/	jail (C) /dzel/	tell (W) /tel/
machine /m'Sin/	machine (C) /m*Sin/	ercier (W) /ErSir/
crime /krajm/	crime (C) /krajm/	crine (W) /krajn*/
police /p'lis/	police (C) /p*lis/	felice (W) /flis/
other /^Dr/	other (C) /^Dr/	under (W) /^ndr/
informer /'nfOmr/	informer (C) /*nfOmr/	enter (W) /Entr/
privileges /prIvl'dZ'z/	privileges (C) /prIvl*dZ*z/	pledges (W) /plEdZ*z/
has /h@z/	hanes (W) /h@ns/	has (C) /h@z/
became /b'kem/	akeen (W) /*kIn/	became (C) /b*kem/
said /sEd/	set (W) /sEt/	said (C) /sEd/

Table 4.3 Comparisons of some differences of the top word choices in the texts "Hitman"and "Ira".

processor indicates that the preprocessor achieves the objective of improving the recognition rate, but only at a modest margin. There is a discouraging finding on the accuracy of recognition with the LPC-based cepstrum coefficients. This points out that the phone models built from the 20ms updating analysis window (either fixed or

variable) is not a reliable model.

An obvious solution to this problem is to increase the number of states allocated for the phones while segmenting the speech signal. Since this could lead to the problem of short segments, a dynamic duration constraint technique should be employed in the segmentation algorithm. Nonetheless, more extensive experimentations with various window time lengths should be performed to achieve an optimal procedure which assures the improvement of recognition accuracy without too much computation at the preprocessor.

Another detail overlooked at the preprocessing stage of the recognizer is the threshold level used to stop the forward-backward reestimation algorithm. Although the chosen threshold (30 dB) is enough to guarantee the convergence of the likelihood of data ($P(\cdot\,|X_T)$), a few more iterations should be made to lower the difference δ to 3 5 dB before determining the segment boundaries and performing the recognition procedure. What also remains to be determined is whether a more complex method (in the recognizer) that accounts for the variable time length of each segment could give greatly improved performance.

Chapter 5 Conclusions

In this thesis we have shown that the technique of linear predictive modeling and hidden filter models can be combined in a simple straightforward manner to implement an automatic speech segmentation. We have also shown the application of this automatic segmentation to a large-vocabulary isolated-word speaker-dependent recognizer as a preprocessor unit.

In speech segmentation, the normalized prediction errors of the hidden filters are compared with the normalized prediction errors of the covariance linear predictive filters. The comparisons show that the hidden filters provides smaller normalized prediction errors (which means higher prediction gain), especially when they are applied to segment of stop burst and aveolar fricative of speech.

The segmentation stage performs exceedingly well on the difficult task of locating short phones of speech. The fact that some segments have very few samples appears to be primarily because of no dynamic duration constraint method had been implemented in the algorithm. Moreover, the time interval $\Delta t = T/N$ used by the staircase approach might contribute to the problem of short segments, presuming that the interval boundaries do not match the phone boundaries.

The overall performance of the recognizer running with LPC-based cepstrum coefficients and the preprocessor is somewhat poorer than the performance of the recognizer running with original mel-based cepstrum coefficients. The fact that the 15-dimensional feature vectors of our model are computed differently with the feature

vectors of the mel-based cepstrum model might contribute to the overall performance. The dynamic coefficients in the mel-based cepstrum model are computed by 40ms intervals while our model does not have any fixed-time intervals. Furthermore, the loudness C_0 in the mel-based cepstrum model is computed by weighting the sum of the log channel energies while the variable C_0 of our model is simply the log filter variance.

The recognition performance of our tests also suggests that the chosen 20ms (average time) window is inadequate to obtain a good Markov model for the phone. This suggestion is made plausible by considering the results of the recognition test with 20ms-uniform LPC-based cepstrum coefficients.

A second suggestion that can be drawn from the results is that a lower level of threshold should be used to increase the number of iterations in the preprocessor. This requires a substantial increase in CPU time: an average 1 minute per iteration per word is calculated at the preprocessor. With 2200 words of the experiment, increasing one iteration of computation will require almost 40 hours of CPU. However, since the training procedure needs to be done only once, the expense of CPU time is worthwhile.

An important fact in implementing the recognizer with preprocessor is that we have not exploited the variable window length of each segment. In our experimentations, the recognizer only treats the sequence of 15-dimensional feature vectors (variable length) as a sequence of uniform (in time) vector-valued observations. To determine whether considering variable time length leads to improving recognition accuracy, we have to rewrite all the structure of the recognizer. Such a task exceeds the scope of this thesis; however we could answer this question by further experimentations.

References

[1] L. E. Baum and J. A. Eagon, "An inequality with application to statistical prediction for functions of Markov processes and to a model for ecology", *Bull. Amer. Math. Soc.*, vol. 73, pp. 360-363, 1963.

[2] L. R. Rabiner, S. E. Levinson and M. Sondhi, "On the application of vector quantization and hidden Markov models to speaker-independent, isolated-word recognition", *Bell Sys. Tech. Journal*, pp. 1075-1105, April 1983.

[3] L. R. Rabiner, S. E. Levinson and M. M. Sondhi, "On the use of hidden Markov models for speaker-independent recognition of isolated-words from a medium-size vocabulary", *Bell Sys. Tech. Journal*, pp. 627 641, April 1984.

[4] G. D Forney, "The Viterbi algorithm", *Proc. IEEE*, vol. 61, pp. 268 278, 1973.

[5] A. B. Poritz, "Linear predictive hidden Markov models and the speech signal", *Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing*, pp. 1291 1294, April 1982.

[6] B. H. Juang and L. R. Rabiner, "Mixture Autoregressive hidden Markov models for speech signals", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 33, pp. 1404 1413, December 1985.

[7] Y. Ephraim, "A minimum mean square error approach for speech enhancement", *Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing*, pp. 829 832, April 1990.

[8] Y. Ephraim, D. Malah and B. H. Juang, "On the application of hidden Markov models for enhancing noisy speech", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1846-1856, December 1989.

[9] P. Kenny, M. Lennig and P. Mermelstein, "A linear predictive HMM for vector valued observations with applications to speech recognition", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 220 225, February 1990.

[10] A. B Poritz, "Hidden Markov models: A guided tour", *Proc. IEEE Int. Conf Acoust. Speech, Signal Processing*, pp. 7 13, April 1988.

[11] L. R. Rabiner and B. H. Juang, "An introduction to hidden Markov model", *IEEE Acoust. Speech, Signal Processing Magazine*, pp. 4–16, January 1983.

[12] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm", *Journal Roy. Stat. Soc.*, Serie 39, pp. 1–38, 1979.

[13] L. R. Rabiner, S. E. Levinson and M. M. Sondhi, "An introduction to the application of the theory of probabilistic function of a Markov process to automatic speech recognition", *Bell Sys. Tech. Journal*, pp. 1035–1074, April 1983.

[14] L. R. Rabiner and R. W. Scheiner (1978) "Digital processing of speech signal", *Prentice Hall. Englewood Cliffs, NJ.*

[15] R. O. Duda and P. E. Hart (1979) "Pattern classification and scene analysis", *Addison Willey: New York.*

[16] J. Makhoul, "Linear prediction: a tutorial review", *Proc. IEEE*, vol. 63, pp. 561–580, 1975.

[17] V. Gupta, M. Lennig and P. Mermelstein, "Fast search strategy in a large vocabulary word recognizer", *Journal Acoust. Soc. Amer.*, vol. 84, pp. 2007–2017, December 1988.

[18] K. F. Lee, H. W. Hon and R. Reddy, "An overview of the SPHINX speech recognition system", *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 35–44, January 1990.

[19] P. Kenny, R. Hollan, V. Gupta, M. Lennig, D. O'Shaughnessy and P. Mermelstein, "A* admissible heuristics for rapid lexical access", *Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing*, vol. 1, pp. 689–692, May 1991.

[20] J. D. Markel and A. H. Gray (1976) "Linear prediction of speech", *Springer-Verlag New York.*

[21] S. B. Davis and P. Mermelstein "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences",*IEEE Trans. Acoust., Speech, Signal Processing*, vol. 28, pp. 357-365, 1980.