## NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

## AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

# Hidden Markov Models, Maximum Mutual Information Estimation, and the Speech Recognition Problem

Yves Normandin

Department of Electrical Engineering
McGill University, Montreal

March 1991

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy.

ISBN 0-315-67472-5

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ABSTRACT

Hidden Markov Models (HMMs) are one of the most powerful speech recognition tools available today. Even so, the inadequacies of HMMs as a "correct" modeling framework for speech are well known. In that context, we argue that the maximum mutual information estimation (MMIE) formulation for training is more appropriate vis-a-vis maximum likelihood estimation (MLE) for reducing the error rate. We also show how MMIE paves the way for new training possibilities.

We introduce *Corrective MMIE training*, a very efficient new training algorithm which uses a modified version of a discrete reestimation formula recently proposed by Gopalakrishnan *et al.* We propose reestimation formulas for the case of diagonal Gaussian densities, experimentally demonstrate their convergence properties, and integrate them into our training algorithm. In a connected digit recognition task, MMIE consistently improves the recognition performance of our recognizer.

# SOMMAIRE

Les modèles de Markov (MM) sont un des plus puissants outils de reconnaissance de la parole actuellement disponibles. Il n'en demeure pas moins, cependant, qu'ils sont loin d'offrir un cadre "correct" de modélisation de la parole. Dans ce contexte, nous raisonnons qu'il est plus approprié d'aborder l'apprentissage des MM de l'angle de l'information mutuelle maximale (IMM) que de celui de la vraisemblence maximale. Cela est d'autant plus vrai que IMM offre de nouvelles possibilités.

Nous introduisons un nouvel algorithme d'apprentissage très efficace pour IMM. Cet algorithme utilise une version modifiée d'une formule de réestimation pour les MM discrets récemment proposée par Gopalakrishnan *et al.* Nous proposons des nouvelles formules de réestimation pour les densités gaussiennes à covariance diagonale, nous en démontrons expérimentalement la convergence et nous les intégrons dans notre algorithme d'apprentissage. Ces techniques nous ont permis de considérablement améliorer la performance de notre système de reconnaissance de chiffres connectés.

# ACKNOWLEDGEMENTS

CRIM is a very young organization and the Speech Group within CRIM is even younger. Much of what we now have had to be built from scratch and doing this while, at the same time, working on a Ph.D. has not always been easy. Our group has, however, enjoyed constant support from CRIM and for this I am very grateful.

I would first like to thank Salvatore Morgera, my thesis supervisor, for his unwavering confidence in my capabilities, especially at times when my own confidence was getting dangerously low. I would also like to thank Renato De Mori who, as the "spiritual father" of the Speech Group at CRIM, has always been a source of inspiration for us. His inexhaustible knowledge about speech recognition and artificial intelligence has proved to be one of our group's greatest assets.

I would also like to thank the other members of our group, especially Louis Vroomen, Régis Cardin, Charles Snow and Hong Minh Cung. Louis has been there almost since the beginning, and the group can always rely on him when new software needs to be developed quickly. Régis has worked very hard with me on the connected digit task and, if we are doing well now, he certainly deserves a lot of credit for it. Charles who, unfortunately for him, knows Unix and C so well, has never hesitated to take some time off his more important tasks to help me out with some of the bugs I had. Minh has had the patience to go through several versions of some of the mathematical parts in my thesis and has helped me remove several errors in them.

I would particularly like to thank my parents for everything they have done for me. They gave me constant support, always rewarded my taste for knowledge and encouraged me to do the best I can. I understand the value of this and I realize that I am truly privileged to have been raised in such an environment. Much of what I am today, I owe it to them.

Finally, I would like to thank Sylvie for her support and understanding over the past few years. This thesis has not only meant long work hours; it has also meant many canceled or postponed projects. She has accepted all of this, patiently, without complaining. I hope that, somehow, I can make it up to her.

# 1. INTRODUCTION

Hidden Markov Models (HMMs) have been successfully used in speech recognition for close to 20 years. Considering the nature of a HMM, this success is somewhat surprising. Indeed, it is not intuitive how a model which assumes that one centisecond of speech is statistically independent of the previous one can be useful. Yet, not only do HMMs work, but they work very well. Over this period, they have been applied to a wide variety of recognition applications and their performance has steadily improved, to the point that they now often outperform all other recognition techniques.

It is now commonplace to hear about HMM-based, large-vocabulary, speaker-independent continuous speech recognition systems. Some examples are BYBLOS, from BBN [CHOW 87, KUBA 88], SPHINX, from Carnegie Mellon University (CMU) [LEEK 88], the Lincoln Labs system [PAUL 89], DECYPHER, from the Stanford Research Institute (SRI) [WEIN 89], or the AT&T system [LEEC 90a]. This was not the case a few years ago, even though the basic concepts necessary to build such systems (phoneme-based modeling, training and recognition algorithms) [JELI 76] have been known since the very beginning of HMM-based speech recognition. During this period, a large body of knowledge and practical experience has been acquired so that these systems are now slowly becoming a reality.

## 1.1. Recent developments

As a general framework for doing speech recognition, HMMs are somewhat paradoxical. On the one hand, experience shows that recognition based on their speech modeling capabilities is very effective. On the other hand, from the speech production point of view, HMMs are notoriously poor speech

models.[1] A question we might want to ask, then, is how important are the modeling deficiencies of HMMs in the context of speech recognition? Using the communication theory viewpoint of speech recognition [BAHL 83], this question can be reformulated as: given that a finite amount of training data is available, how close is it possible to get to an optimal speech decoder (recognizer) using HMMs?

It is extremely difficult to answer this question. Since the channel statistics are unknown, the optimal decoder's error rate cannot be determined. Of course, human performance can be used as a good approximation, but even this is difficult to evaluate. Within the HMM framework, how close we get to an optimal decoder depends on a number of factors such as the speech features used as input; the structure of the models; the type of output distributions; the training and recognition techniques used; and, last but not least, precisely how an optimal decoder is defined.

We will review some of the recent developments which have allowed HMMs to produce constantly better speech recognizers. Clearly, much has been done; on the other hand, comparing the best available systems with human performance, it is also clear that there is still a long way to go.

### 1.1.1. Front end

The front-end system extracts from the speech signal the features which will be used by the HMMs to model and recognize speech. As a general principle the extracted features should contain as much information as possible about the linguistic content of the acoustic signal, while being in a form that can be used by the HMMs. Also to be considered is not only how much useful information a particular feature contains, but how reliably it can be extracted from the speech signal.

The front-end system has a considerable impact on the ultimate performance of an HMM-based speech recognition system, and much effort has been spent over the years to find new and better speech features. One important source

[1] First order HMMs are assumed throughout this thesis.

of inspiration has been the knowledge about the human auditory system. Indeed, since human recognition is so good, it seems logical to imitate, up to a point, the way the ear performs its own "feature extraction". This has proved to be a fruitful area of research. Nowadays, most systems integrate some form of auditory modeling into their feature extraction, often in the form of mel-scaled parameters [DAVI 80], through the use of a bilinear transform [LEEK 88], or by using the output of a full-scale auditory model [SENE 88, COHE 89].

Recently, techniques such as principal component analysis [BROW 87], discriminant analysis [BROW 87, HUNT 89, DODD 89] and cepstral transformation,[2] by both decorrelating parameters and concentrating most of the useful information into a small number of parameters, have also allowed better recognition.

However, the most significant recent development in terms of feature extraction for speech recognition is probably the introduction of "dynamic" parameters[3] [FURU 86]. This development is important for several reasons. First, it has allowed substantial recognition improvements in most speech recognition systems.[4] This confirms the importance of information about the dynamics of speech. Now, most high-performance systems use dynamic features in one form or another.

Second, this highlights a major weakness of HMMs. Indeed, in most cases, dynamic parameters are directly computed from parameters already used by HMMs, and as such, do not contain any additional information. This shows that it is not sufficient that the extracted parameters preserve all the important linguistic information in the signal; this information has to be encoded in such a way that HMMs can take advantage of it. In other words, the front-end system and the HMM recognizer should not be considered as independent

[2]Cepstral transformation was in practice found [HUNT 89] to be very close to a principal component analysis.

[3]Dynamic parameters describe how, over a number of speech frames, other parameters are changing. They are often referred to as "differences" or "slopes", depending on how they are computed [LEEK 88]. Recently, "second derivative" parameters have been introduced in some systems [GAUV 91], and they also seem to improve recognition performance.

[4]This is especially true for speaker-independent systems.

entities.

### 1.1.2. Acoustic-phonetic modeling

The idea of acoustic-phonetic modeling is certainly not new. A language like English has a very small number of phonemes (about 40), from which every single word can be built. It is natural to think of the phoneme as *the* basic unit to model and recognize. This would, for example, allow a system to recognize words it had never heard before by simply knowing the word's phonetic pronunciation(s) (plus, possibly some phonological rules). Unfortunately, the phoneme is a very abstract linguistic unit, and its actual acoustic realization is extremely variable. It depends on a number of factors such as accent, speaking rate, intonation or phonetic context.

*Context-dependent* phonetic HMMs were introduced specifically to deal with within-word context dependencies [SCHW 85]. The idea is to use different models for the same phoneme, one for each of a number of different contexts such as either the right or left phonetic context, or both.[5]

Context-dependent models are much more specific and thus better able to make fine phonetic distinctions. The problem is that the more specific the models, the larger the total number of models, and the less data there is to train them. This results in a number of specific but poorly trained models, which is undesirable. In order to solve this problem, BBN smoothed triphone models with the corresponding left and right context-dependent models, and the corresponding phoneme models, using manually-tuned weights [SCHW 85]. The idea is to get models which are as specific as possible, while still being reliably trained, or, as Lee puts it [LEEK 88], to get the best possible compromise between "specificity" and "trainability". For SPHINX [LEEK 88], Lee used the same technique, except that he also smoothed with models having uniform distributions and, in his case, the weights were estimated automatically using a technique called *deleted interpolation* [BAHL 83].

[5]Phoneme models which depend on both the left and right phonetic contexts are usually called *triphones*.

Context-dependent models are now used in most large-vocabulary systems (e.g., BYBLOS, SPHINX, DECYPHER, the systems of SRI, Lincoln Labs, AT&T, or the INRS large-vocabulary system [DENG 90]). In all cases, their use has resulted in substantial recognition improvements.

Modeling within-word coarticulations can also be done implicitly with word models. This may actually be a very powerful technique since word-dependent context dependencies can be accurately modeled, something that cannot always be done with triphone models.[6] Word models are especially useful for small vocabulary applications. They have been successfully used for speaker-independent applications such as connected digit recognition [RABI 89b, DODD 89] or keyword spotting [WILP 90, ROHL 89].

The more accurate coarticulation modeling offered by word models inspired the introduction of a totally different kind of sub-word unit [BAHL 88a]. Rather than being defined in terms of linguistic principles, this new unit, called the *fenone*, is acoustically based. Thus, instead of using phonetic concepts to determine *a priori* the baseform of a word in terms of the systems basic units, the baseform is determined from acoustic realizations of that word. This results in more precise word modeling and has improved recognition rates in isolated word recognition applications [BAHL 88a].

Some of the recognition problems encountered in practice are specific to continuous speech recognition systems. One is the poor articulation of function words such as articles, prepositions or conjunctions, which results in a disproportionate number of errors arising from those words [LEEK 88]. In order to solve this problem, Lee [LEEK 88] introduced *function-word dependent phones* and observed significant improvements from their use. On the other hand, their use in the AT&T system [LEEC 90a] has only resulted in improvements for the "no grammar" case (i.e., all vocabulary words are equally probable at all times).

---

[6]Context dependencies can extend beyond the immediate phonetic neighbors. Note that it is also possible to have *word-dependent* triphones [CHOW 86], which will perform very well on the specified vocabulary with the additional advantage that they can be used to create good initial models for new words. Note also that in several systems, particularly those using the DARPA resource management corpus, triphone models are *vocabulary-dependent*, that is, they are trained and tested on the same vocabulary. These models include vocabulary-specific effects which may artificially enhance the systems performance [HON 90].

Another problem of continuous speech is between-word coarticulation. A solution to this problem, recently (and simultaneously) proposed by CMU [LEEK 89b], Lincoln Labs [PAUL 89] and SRI [WEIN 89], is simply to use context-dependent phone models for between-word coarticulation. This had not been done before because it substantially increases the number of models (from 2381 to 7057 triphones for the DARPA resource management task [LEEK 89b]) and it also substantially complicates recognition.

### 1.1.3. Dealing with speaker variability

An important cause of speech's acoustic variability is speaker variability. Speaker-independent models have a problem reminiscent of context-independent models. By averaging statistics over a number of speakers, they loose specificity and, along with it, discriminating capabilities. One obvious solution is to use speaker-dependent models. However, in order to get the desired level of performance, this may require a large amount of training data from everybody using the system. This is often undesirable, which is why *speaker adaptation* techniques have been developed.

The idea of speaker adaptation is to start from well-trained models and to adapt them to a new target speaker, using as little data as possible. Speaker adaptation techniques are usually classified as *supervised* (using labeled speech[7] ), or *unsupervised* (using unlabeled speech). Unsupervised techniques [COX 89, FURU 89] are attractive when it is desired to perform adaptation transparently during system use. However, this is not as clear an advantage as it appears: it is often possible to use supervised techniques to perform adaptation on the confirmed correctly recognized speech, thus improving the system performance as it is being used. Moreover, supervised techniques usually perform better.[8]

A different kind of speaker adaptation is used by systems which must be truly

[7]Labeled speech means that the linguistic contents (e.g., words) of the speech is known. This is different from "segmented speech", which is not only labeled, but has marks indicating where the boundaries of the linguistic units are in the signal.

[8]See Chapter 3 for more details.

speaker-independent, such as those used for telephone network applications aimed at the general public (e.g. [WILP 90]). The technique, called *HMM clustering* [RABI 88, LEEK 88, RABI 89a, DODD 89], clusters the speakers in the training set into a number of different speaker types, and creates as many versions of every model as there are speaker clusters. This makes the models more speaker-specific and generally results in a moderate improvement in recognition performance.

### 1.1.4. Increasing robustness

The performance of a speech recognition system may be quite sensitive to changes in background noise level and characteristics, microphone changes or changes in the general acoustic environment. For example, using a stereo database,[9] in an experiment with SPHINX, Acero [ACER 90] found that simply using a different microphone during recognition than the one used for training could decrease the system performance from 85.3% to 18.6% word accuracy. How sensitive a particular system is to such changes will determine its *robustness*. For most systems, the best performance is usually obtained when training and recognition are performed under identical conditions. This, however, need not necessarily be the case. Experiments have shown [GISH 90] that models trained under good conditions (and thus acoustically accurate) may be adapted to perform better in noisy conditions than models trained in the same noisy conditions. Because of this, and because it is not always possible to train and use the system in the same conditions, much work is currently being done to increase the robustness of HMM-based systems.

Increasing robustness under noisy conditions with simple spectral subtraction schemes [VANC 87] may give some improvement if noise conditions during training and testing are not too different, although differences in residual noise may cause problems [VANC 89]. Noise adaptation via probabilistic spectral mapping techniques conditioned on the instantaneous signal to noise ratio (SNR) has proven effective in a wordspotting application [GISH 90]. Performing adaptation to a new microphone using *Codeword-Dependent Cepstral*

[9] A stereo database contains the same speech simultaneously recorded with two different microphones.

*Normalization* (CDCN), Acero [ACER 90] brought word accuracy back from 18.6% to 74.9%, which is essentially the rate obtained when both training and recognition were performed with the second microphone.

### 1.1.5. Output distributions

Output distributions in HMM-based speech recognition systems are usually classified as either *discrete* or *continuous*.[10] For a long time there has been a debate as to which of discrete or continuous distributions performed best. While some researchers obtained better performance with discrete systems, others obtained the exact opposite (see [LEEK 88] for a discussion on this topic). In fact, fair comparisons were difficult to make. While discrete systems are rather straightforward to implement, continuous ones offer more degrees of freedom (in terms of the number of mixture components [JUAN 85] or the restrictions on covariances matrices[11] [BROW 87, LEEC 90a]) and are more sensitive to parameter initialization.

It now seems that semi-continuous HMMs have solved that problem. Since it is relatively trivial to convert a discrete system into a semi-continuous one, a discrete system can be evaluated in both modes. Such evaluations tend to show semi-continuous HMMs to be superior [HUAN 90], thus demonstrating the usefulness of continuous densities. Note that by combining the characteristics of both discrete and continuous distributions, SCHMMs open a number of interesting possibilities for speech recognition. For that reason, we feel their use is bound to become widespread.

### 1.1.6. Discriminative training techniques

It is probably accurate to say that HMM-based speech recognition owes its

[10] A recently introduced compromise, "semi-continuous" HMMs (SCHMMs) [HUAN 89], use both discrete distributions and continuous densities. There is a common codebook of continuous densities used by all distributions to form mixture densities. The set of mixture weight associated to one of the mixture distributions forms its discrete distribution. See Chapter 2 for more details.

[11] We have assumed that continuous distributions are made from Gaussian densities. Even though there are other possibilities (for example, Laplacian densities have been used), this thesis does not consider them.

popularity in great part to the powerful Baum-Welch algorithm [BAUM 72] for maximum likelihood estimation (MLE) training. This algorithm iteratively increases the probability that the training data was generated by the corresponding models. An approximate version of this algorithm, *segmental k-means training*, [LEEC 90a] was developed by researchers at AT&T and was experimentally found to give comparable results at a lower computational cost [RABI 89b]. Another training paradigm, *minimum discrimination information* (MDI) training, was also recently proposed by researchers at AT&T [EPHR 87]. Applied to the training of autoregressive HMMs [JUAN 85], MDI training attempts to find the HMM parameter set minimizing the discrimination information measure with respect to all sources that could have produced the set of partial covariances evaluated from the training data [EPHR 89].[12]

One problem with those training techniques is that they have no obvious relationship with the aim of minimizing the recognition error rate. Even though it can be shown that, under certain assumptions, MLE will in fact produce the best possible recognizer [NADA 83], this is not really satisfying since, in practice, the required assumptions are usually not met in speech recognition.

A few years ago, maximum mutual information estimation (MMIE) was proposed as an alternative to MLE [BAHL 86]. MMIE training[13] attempts to find the HMM parameter set maximizing the *a posteriori* probability that the training data was generated by the models corresponding to the spoken speech in the data.[14] This approach seems reasonable since recognition is usually performed by finding the model with the greatest *a posteriori* probability of generating the spoken speech. Unfortunately, the expression to optimize is quite complex and often has to be approximated [BROW 87, CHOW 90]. Moreover, since there is no equivalent of the Baum-Welch algorithm for MMIE, the standard training procedure is based on gradient descent. Nonetheless, MMIE looks like a promising technique. Several researchers have reported improvements in recognition rate through the use of MMIE [BAHL 86,

[12]There are no published results of speech recognition experiments using MDI training.

[13]Note that MMIE training can be given an MDI interpretation [EPHR 88]. However, the MDI literature doesn't develop that case.

[14]For example, if the training data contains spoken sentences, then the models corresponding to the training data will be sentence models. These are usually built from the models of the words in the sentence, which are often themselves built from sub-word models.

BROW 87, MERI 88, CARD 91].

Another training technique, designed specifically to reduce the error rate is *corrective training* [BAHL 88b]. This is a heuristic technique which attempts to improve the recognition rate on the training set by working only with sentences in the training set that were either not recognized correctly, or were correctly recognized, but with a very close second choice. In practice, corrective training is similar to MMIE and it has also shown promising results [APPL 89, LEEK 90].

Training techniques such as MMIE or corrective training are called *discriminative* techniques because they specifically aim at improving the discriminating capabilities of the models.

### 1.1.7. Search algorithms

In theory, speech recognition is done by finding the word sequence[15] which has the greatest probability of generating the given speech signal (observation sequence). This probability is computed using both the *a priori* probability of the word sequence (using a language model) and the *a posteriori* probability of the input string given the model for the word sequence. In practice, however, except in the simplest applications, there are just too many different word sequences for this to be possible. This means that theoretically suboptimal search algorithms have had to be developed. Most of these algorithms are based on the idea of a dynamic programming search through a word network [LEEC 89b]. While most are frame-synchronous trellis searches, some, like *level building* [MYER 81, RABI 85a] are word-synchronous, and others like the *stack algorithm* [JELI 76] are of the type "best-first", or A*.

Frame-synchronous algorithms are often the fastest, and they are quite adequate for simple tasks such as connected digit recognition. It is, however, difficult, if not impossible, to integrate all but the simplest language models into such search procedures.

[15]This is meant in the general sense. For isolated word recognition, the word sequence is a single word. For phoneme recognition experiments, the word sequence is actually a phoneme sequence.

However, in this era of large vocabulary and continuous speech recognition, it becomes increasingly difficult to obtain good performance based on acoustics alone and the use of sophisticated language models becomes necessary in order to make the difference between unacceptable and excellent performance. The recent development of algorithms capable of generating the N most acoustically probable word sequences [SCHW 90, SOON 90] offers the possibility of implementing the language model as a post-processor. This method, however, is both inefficient and approximate. Most language models are best implemented within an A* search, which can take both the acoustic and language models into account. An interesting possibility would be to perform such a search by taking advantage of the acoustic precomputations performed in the *tree-trellis search* algorithm recently proposed by Soong and Huang [SOON 90].

## 1.2. Revisiting the training problem

Much has been indeed accomplished since the first HMM-based speech recognition systems were developed. With the same imperfect speech model, it has been consistently possible to improve both the performance and the capabilities of speech recognition systems. This is the result of several years of practical experience with HMMs, which have given us a much better understanding of their strengths and weaknesses.

In this thesis, we will thoroughly review the theory and practice of HMMs, as they relate to the speech recognition problem. This will provide the necessary background for our analysis of MMIE, the core of this work.

The MMIE method has been known for over five years. During that period, many researchers have explored it and experimented with it. Most have observed recognition improvements, some substantial, some negligible. It seems that, apart from one extensive study by Peter Brown [BROW 87], experiments with MMIE have mostly been sporadic. Many researchers felt that the potential improvements were probably not worth the introduction of gradient descent in their training programs and the additional computational load.

This thesis will show that not only can MMIE result in substantial recognition improvements, but that training can be done simply, efficiently and, in most cases, without gradient descent. Using the more intuitive probabilistic interpretation, we will review the theory of MMIE, including the latest developments prior to this work. We will show how a new reestimation-like formula recently proposed by Gopalakrishnan *et al.* [GOPA 89] for discrete HMMs can be modified so that convergence is dramatically improved. We will propose a similar formula for continuous HMMs and experimentally demonstrate its convergence. We will describe how the MMIE formulation offers new ways of improving the speech modeling and the discriminating capabilities of HMMs. Connected digit recognition experiments will be used to illustrate the benefits that MMIE can provide.

The outline of this thesis is as follows: Chapter 2 describes the theory of HMMs. Chapter 3 discusses practical considerations which are important when implementing an HMM-based speech recognition system. Chapter 4 describes the theory of MMIE, including the contributions of this work to that area. Chapter 5 describes the connected digit experiments and summarize the results. Chapter 6 concludes.

# 2. HIDDEN MARKOV MODELS

## 2.1. Introduction

This chapter introduces the concepts necessary to understand this thesis. It is assumed that the reader already has some knowledge of HMMs and their use in speech recognition. If not, there are excellent introductions available (see, for example [RABI 86] or [RABI 89c]), and we encourage the reader to consult one of them.

The speech recognition problem can be approached as a problem in communication theory [BAHL 83]. Using this point of view, we assume that a message $w$ (usually a word or a word sequence) is converted by an acoustic channel into an observation sequence $y$. The goal of speech recognition is to decode the message from the observation sequence.

In practice, the message is converted by a speaker into acoustic pressure waves (speech), which a microphone then transforms into an electric signal. It is the sampled (digitized) version of this signal which is fed into the computer for processing and recognition. For our purposes, this discrete-time signal will be referred to as the *speech signal* $s$.

The speech signal is blocked into frames (usually using fixed frame rate and size), each of which is then analyzed in order to extract information relevant to the recognition process. Typically, the analysis extracts information about the power spectrum of the signal, usually through FFT or LPC-based techniques. This analysis produces a small number of parameters (e.g., six cepstral coefficients) to which may be added other parameters deemed useful, such as the signal zero-crossing rate or RMS power.[1] The result of this

[1] It is now common practice to add so-called "dynamic" parameters, which describe how the mentioned parameters vary over a short time around the given frame.

analysis, applied to every signal frame, is a sequence $\boldsymbol{y}$ of observation vectors.

Observation vectors are usually categorized as *continuous* or *discrete*. Continuous parameters are often the end result of the analysis. Discrete parameters can only take a finite number of values from some symbol alphabet. Discrete parameters usually result from the vector quantization (VQ) [GRAY 84, MAKH 85] of the continuous parameter vector. In this case, the discrete alphabet contains as many symbols as there are codewords in the VQ codebook. If separate parts of the continuous parameter vectors are separately quantized using different VQ codebooks [LEEK 88], then the result is a discrete vector containing several discrete parameters. It is hoped that whatever parameters are extracted from each frame of speech contain as much information about $\boldsymbol{w}$ as possible. $\boldsymbol{y}$ is simply the length-$L_{\boldsymbol{y}}$ sequence of observation vectors $\underline{y}_1, \underline{y}_2, \cdots, \underline{y}_{L_{\boldsymbol{y}}}$.

### 2.1.1. The decoding problem

Speech decoding (recognition) is a transformation $\boldsymbol{y} \rightarrow \hat{\boldsymbol{w}}$. If $\hat{\boldsymbol{w}} \neq \boldsymbol{w}$, then there was a decoding error. The performance of a decoder is usually determined by the probability of making such an error. It is well-known that the optimum decoder in the sense of minimizing the probability of error is the maximum *a posteriori* decoder (MAP), which chooses $\hat{\boldsymbol{w}}$ such that

$$\begin{aligned} \hat{\boldsymbol{w}} &= \underset{\boldsymbol{w}}{\operatorname{argmax}}\, P(\boldsymbol{w}\,|\boldsymbol{y}) = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, \frac{P(\boldsymbol{y}\,|\boldsymbol{w})P(\boldsymbol{w})}{P(\boldsymbol{y})} \\ &= \underset{\boldsymbol{w}}{\operatorname{argmax}}\, \frac{P(\boldsymbol{y}\,|\boldsymbol{w})P(\boldsymbol{w})}{\sum_{\boldsymbol{w}'} P(\boldsymbol{y}\,|\boldsymbol{w}')P(\boldsymbol{w}')}\,, \end{aligned} \tag{2.1}$$

where $P(\boldsymbol{w}\,|\boldsymbol{y})$ is the *a posteriori* probability of $\boldsymbol{w}$, given $\boldsymbol{y}$, $P(\boldsymbol{w})$ is the *a priori* probability of $\boldsymbol{w}$ (the *language model*) and $P(\boldsymbol{y}\,|\boldsymbol{w})$ is the probability of the observation sequence, given the message. Unfortunately, these probabilities are unknown so they must be somehow estimated. This is a typical statistical pattern recognition problem [DEVI 82]. A common solution is to assume that $P(\boldsymbol{y}\,|\boldsymbol{w})$ belongs to some family of functions $P_{\Theta}(\boldsymbol{y}\,|\boldsymbol{w})$ where $\Theta$ is the family

parameter vector that needs to be estimated. This estimation is done using an amount of *training data*, which consists of a number of pairs $(w,y)$, where $y$ is the observation sequence resulting from $w$.[2] Note that a given $w$ can result in very different $y$'s. The training data usually contains only a few (if any) instances of all possible $y$'s for a given $w$.

The first problem is to find an appropriate family of functions. This should be done using as much knowledge as possible about the process being modeled. In our case, this problem is complicated by the inherent time variability of speech. Because of varying speaking rates, two observation sequences from the same word can have very different lengths (durations) and be equally probable. Moreover, certain types of sounds can be stretched much more than others. This is where HMMs come in. As probabilistic models which provide a convenient framework for dealing with the time variability, HMMs are well-suited to model the entire acoustic channel.

Another problem is to estimate the parameter vector $\Theta$ from the training data. First, we have to find a function $R(\Theta)$ that will allow us to evaluate how well our estimate approximates the "real" (and unknown) function. Second, once such a function is determined, we must find the parameter vector $\Theta$ that optimizes it. This is also a difficult problem since, even in simple applications, $\Theta$ can contain several thousands of parameters.

### 2.1.2. A simple illustration

We illustrate the above concepts involving HMMs in speech recognition with the simple case where $w$ is a single word. Figure 2.1 illustrates a typical word HMM.[3] With each transition is associated a transition probability and an output distribution. The transition probability is the probability of taking the transition, given that the process is in the departure state of the transition. When a transition is taken, an observation vector is generated. The

[2]In practice, the training data usually consists of pairs $(w,s)$, where $s$ is the speech signal resulting from a speaker saying $w$. The acoustic sequence $y$ is obtained using a transformation $s \rightarrow y$ called *feature extraction*. Within a given family of functions $P_\Theta(y|w)$, how well it is possible to approximate $P(y|w)$ will depend to a large extent on this transformation.

[3]Throughout this thesis, the terms "model" and "HMM" will be used interchangeably.

transition's output distribution gives, for every possible observation vector $\underline{y}$, the probability of generating $\underline{y}$ when the transition is taken. Note that several transitions can share the same output distribution. This is illustrated in Figure 2.1, in which the number above each transition refers to the number of the output distribution associated with it.

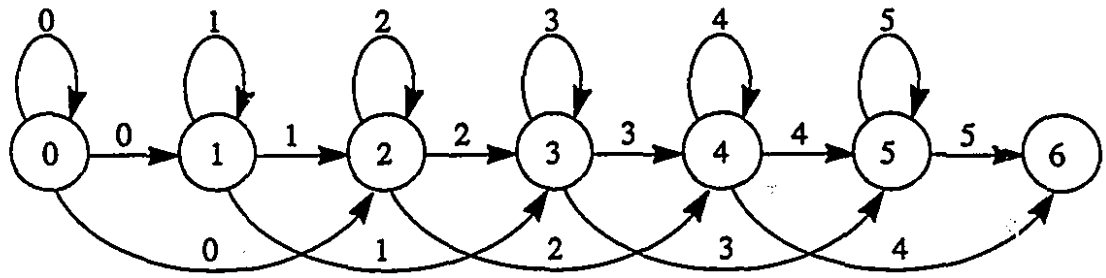


**Figure 2.1:** Word model $m_w$

We want to be able to estimate $P(y|w)$ for any $y$, using the word model in Figure 2.1. Such a model can be used to compute the estimate $P_\Theta(y|m_w)$, where $m_w$ is the model corresponding to word $w$. We assume that any $y$ resulting from $w$ arises as the result of a path in the model $m_w$. Without loss of generality, we also assume that any path in the model must start in state 0 and end in the last state.

Thus, the parameters to estimate in the word model are the transition probabilities and the output distribution parameters. Intuitively, $P_\Theta(y|m_w)$ should be high when $y$ is the result of $w$, and low otherwise. The optimization functions used are generally based on that intuition.

### 2.1.3. Unit models

In HMM-based speech recognition applications, there has to be a model $m_w$ associated with every possible $w$ in the application. Any model in this (possibly very large) collection of models is usually built from a limited set of small models. Sentence models can be created by concatenating word models. Similarly, word models are often made of the concatenation of sub-word unit

models such as phoneme or syllable models. For example, suppose an HMM-based speech recognition system uses the phoneme as its basic unit and that all phoneme models have the structure in Figure 2.2.

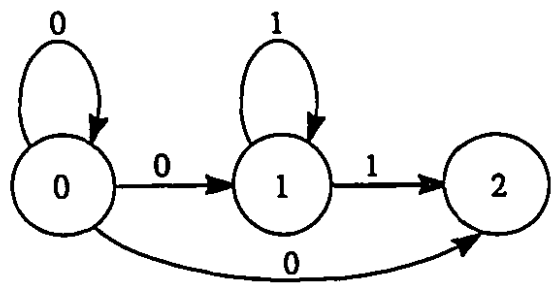


**Figure 2.2:** Basic unit

Then, as illustrated in Figure 2.3, we can create the model for the word "sauce" from the phoneme models for "s" and "ao".

There are two important observations to make about this example. First observe the dotted transitions that are used to connect the models together. They are special in one important respect: they don't have an output distribution associated with them; in other words, there is no observation vector generated when these transitions are taken. For this reason, they are called *empty* transitions. In practice, these transitions are very often needed. One example is when it is desired to allow paths to start in states different from the initial state. This can be done by having an empty transition from the initial state to the desired states (see Chapter 3). Model concatenations such as in Figure 2.3 can, however, be done more simply without empty transitions, as illustrated in Figure 2.4.

The other observation is about transitions $t$ and $t'$ in Figure 2.3. Even though they are two different transitions in the word model, they both correspond to the same transition within the model for phoneme "s" and we say that they both correspond to the same *transition component*. What this means is that not only do they have the same probability and output distribution, but, apart from their different departure and arrival states, they can in fact be regarded as the same transition. A common terminology is to say that transitions $t$ and

$t'$ have *tied* parameters.[4]

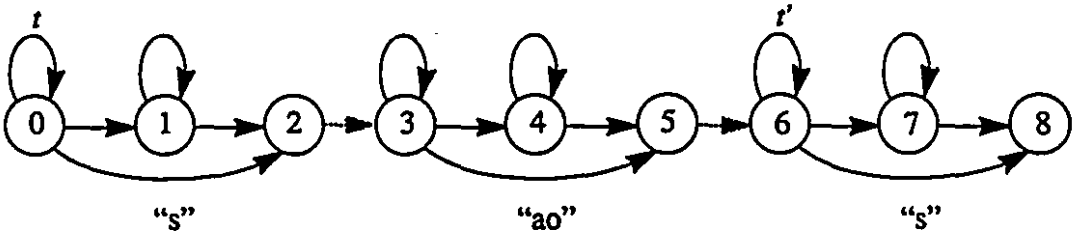


**Figure 2.3:** Model for word "sauce" from models for "s" and "ao"

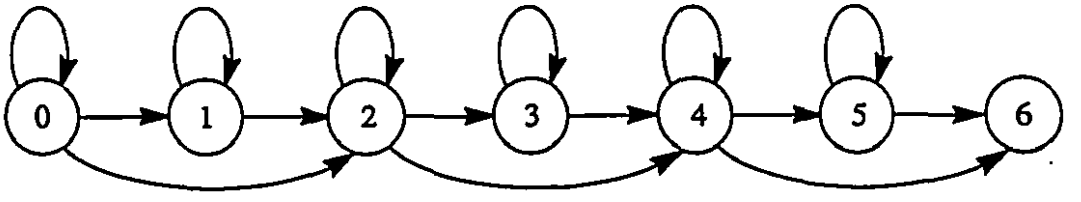


**Figure 2.4:** Model for word "sauce", without empty transitions

When designing an HMM-based speech recognition application, one usually has to first decide on a basic set of units. The models for these units will be used as the building blocks for the whole application. The set of parameters from these unit models (transition probabilities and output distribution parameters) is the complete parameter set for the whole application. Any model built from the unit models inherits its parameters from these unit models. This is more formally explained in the following section.

[4]We can also say that transitions $t$ and $t'$ are tied. Similarly, since each transition leaving state 0 in Figure 2.3 is tied to a transition leaving state 6, we can also say that states 0 and 6 are tied.

## 2.2. Definitions

In an HMM-based speech recognition application, the set of *unit models* (or *unit HMMs*) refers to a set of models which don't share any parameter with each other and from which any model for the application can be built. A unit HMM is defined by a set of transitions connecting states, one of which is defined as the initial state and one as the final state. Note that the final state must usually be a "sink state", that is, one with no transition leaving it.

A transition $\tau$ in a unit model is called a *unit transition*. Within the unit model, it is characterized by its departure (left) state $l_\tau$ and its arrival (right) state $r_\tau$, which serve to describe the model structure. It is also characterized by its probability $q_\tau$ and by the output distribution $b_\tau$ associated with it (none, if the transition is empty). States and distributions are usually designated by numbers (we use -1 for the distribution associated with empty transitions), even though these numbers are only meaningful within a given model. State numbering, however, has great practical importance and we will come back to it later. As an example, the model in Figure 2.2 can be defined by the transitions in Table 2.1

| $l_\tau$ | $r_\tau$ | $q_\tau$ | distribution |
|---|---|---|---|
| 0 | 0 | 0.33 | 0 |
| 0 | 1 | 0.33 | 0 |
| 0 | 2 | 0.33 | 0 |
| 1 | 1 | 0.50 | 1 |
| 1 | 2 | 0.50 | 1 |

**Table 2.1:** Basic unit transitions

and by the corresponding two distributions. Note that in this case, transitions from the same state have been assumed equiprobable. In all cases, however, transition probabilities must satisfy the following constraint:

$$\sum_{(\tau | l_\tau = S)} q_\tau = 1 \,, \tag{2.2}$$

where $S$ is any state in the model except sink states. As a more elaborate example, we can use the word model in Figure 2.3. This time, each transition in the model has been individually identified using a number, as shown in Table 2.2:

| $\tau$ | $l_\tau$ | $r_\tau$ | $q_\tau$ | distribution | $\tau$ | $l_\tau$ | $r_\tau$ | $q_\tau$ | distribution |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0.33 | 0 | 10 | 4 | 4 | 0.50 | 3 |
| 2 | 0 | 1 | 0.33 | 0 | 11 | 4 | 5 | 0.50 | 3 |
| 3 | 0 | 2 | 0.33 | 0 | 12 | 5 | 6 | 1.00 | -1 |
| 4 | 1 | 1 | 0.50 | 1 | 13 | 6 | 6 | 0.33 | 0 |
| 5 | 1 | 2 | 0.50 | 1 | 14 | 6 | 7 | 0.33 | 0 |
| 6 | 2 | 3 | 1.00 | -1 | 15 | 6 | 8 | 0.33 | 0 |
| 7 | 3 | 3 | 0.33 | 2 | 16 | 7 | 7 | 0.50 | 1 |
| 8 | 3 | 4 | 0.33 | 2 | 17 | 7 | 8 | 0.50 | 1 |
| 9 | 3 | 5 | 0.33 | 2 | | | | | |

**Table 2.2:** Transitions for word in Figure 2.3

The transitions of a HMM determine all possible paths through the model. We define a path $t=t_1,t_2,\ldots,t_{L_t}$ as a possible sequence of transitions through an HMM. Note that a transition sequence uniquely specifies a state sequence but, since there can be several transitions with the same departure and arrival states, the opposite is not necessarily true. We will always assume that the first transition $t_1$ of a path starts in state 0 and that the last transition $t_{L_t}$ ends in the last state. Also, the transitions in the path must satisfy

$$r_{t_i} = l_{t_{i+1}}, \qquad 1 \le i < L_t .$$

Let $t_{n_l}$ be the $l$th full transition in the transition sequence $\boldsymbol{t}$. If $\boldsymbol{t}$ generates an output sequence $\boldsymbol{y}$, then the $l$th observation vector $\underline{y}_l$ in $\boldsymbol{y}$ is the result of $t_{n_l}$. Since, in practice, the observation sequence $\boldsymbol{y}$ usually results from a time-synchronous analysis, we refer to the index $l$ in $\underline{y}_l$ as the *time* index. An empty transition, with which no observation is associated, doesn't produce any time change; it happens at the same time as the previous transition. For example, Table 2.3 uses transition numbers from Table 2.2 to illustrate the

relationship between a transition sequence and the corresponding output sequence. Time 0 corresponds to empty transitions before the first observation $\underline{y}_1$.

| time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| transition(s) | - | 1 | 2 | 4 | 4 | 5,6 | 7 | 7 | 7 | 9,12 | 14 | 17 |
| distribution | - | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 0 | 0 |
| observation | - | $\underline{y}_1$ | $\underline{y}_2$ | $\underline{y}_3$ | $\underline{y}_4$ | $\underline{y}_5$ | $\underline{y}_6$ | $\underline{y}_7$ | $\underline{y}_8$ | $\underline{y}_9$ | $\underline{y}_{10}$ | $\underline{y}_{11}$ |

**Table 2.3:** A transition sequence

We call $t[l]$ the set of all transitions in $t$ occurring at time $l$. Using the example in Table 2.3, $t[4] = \{4\}$, $t[5] = \{5,6\}$ and $t[0] = \{\}$. We use $n_l(t)$ to designate the index in $t$ of any transition $t$ in $t[l]$. For example, if $t$=6 and $l$=5, then $n_l(t)$=6. Note that there can be several empty transitions in $t[l]$, but there is exactly one full transition per $t[l]$ (except $t[0]$, in which there is none).

An output probability distribution is designated $b(\cdot)$, or simply $b$. The distribution associated with a given transition $t$ is designated $b_t$. Note that while $q_\tau$ is uniquely associated with the transition component $\tau$, it is often the case that $b_\tau = b_{\tau'}$ even though $\tau \neq \tau'$.

A distribution $b$ must satisfy

$$\begin{aligned} &\sum_{\underline{y}} b(\underline{y}) = 1, && \text{if } \underline{y} \text{ discrete;} \\ &\int_{\underline{y}} b(\underline{y}) d\underline{y} = 1, && \text{if } \underline{y} \text{ continuous.} \end{aligned} \tag{2.3}$$

The complete parameter set $\Theta$ for a given application is the set of all transition probabilities $q_\tau$ and all parameters from all output distributions $b$ from all unit models. Any model $m$ used in the application is built from those unit models and inherits their parameters. For every transition $t$ in $m$ there is a

corresponding transition $\tau$ in one of the unit models used to build $m$. We use $\tau(t)$ to designate that transition. All parameters associated to $\tau$ (except the departure and arrival states, which only make sense within a given model) are inherited by $t$. So we have $q_t \equiv q_\tau$ and $b_t \equiv b_\tau$, where we use $\equiv$ to signify that they are identically equal (and not that they happen to have the same value).

## 2.3. Probability computations with HMMS

### 2.3.1 Basic concepts

Given a HMM parameter set $\Theta$, a model $m$ and a length-$L_y$ observed output sequence $y = \underline{y}_1, \underline{y}_2, \cdots, \underline{y}_{L_y}$, several useful probabilities can be computed using the HMM's parameters. The probability that is most often computed is $P_\Theta(y|m)$, the probability that the sequence $y$ was generated by the model $m$. $P_\Theta(y|m)$ can be computed by summing over all possible paths $t$ in the model $m$ the probability $P_\Theta(y|t)$ of observing $y$ given the path, times the path *a priori* probability $P_\Theta(t|m)$. Note here that, in order for $P_\Theta(y|t)$ to be different from zero, the number of full transitions in the path must equal $L_y$, the length of the output string. This means that $L_t \geq L_y$. Thus, $P_\Theta(y|m)$ can be expressed as

$$P_\Theta(y|m) = \sum_{t \in m} P_\Theta(y,t|m) = \sum_{t \in m} P_\Theta(t|m) P_\Theta(y|t)$$

$$= \sum_{t \in m} \left( \prod_{i=1}^{L_t} q_{t_i} \right) \left( \prod_{j=1}^{L_y} b_{t_{n_j}}(\underline{y}_j) \right), \tag{2.4}$$

where $t_{n_j}$ is the $j$th full transition in $t$, and where the notation $\sum_{t \in m}$ is used to mean a summation over all paths in the model $m$. In practice, however, the above expression cannot be used because the number of possible paths increases exponentially with $L_y$. Fortunately, the first-order Markov nature of HMMs makes it possible to devise a simple and efficient procedure that allows easy computation of all necessary probabilities [BAUM 72]. Before

showing this, we must first make the following definitions. Let

$$\alpha_l(i) = P(\underline{y}_1, \cdots, \underline{y}_l, S_l = i \,|\, \boldsymbol{m}) , \tag{2.5}$$

$$\beta_l(i) = P(\underline{y}_{l+1}, \cdots, \underline{y}_{L_y} | S_l = i, \boldsymbol{m}) . \tag{2.6}$$

The quantity $\alpha_l(i)$ is the probability that the model generates the output subsequence $\underline{y}_1, \cdots, \underline{y}_l$ using a transition sequence ending in state $i$. The quantity $\beta_l(i)$ is the probability that the model $\boldsymbol{m}$ generates the output subsequence $\underline{y}_{l+1}, \cdots, \underline{y}_{L_y}$, given that first transition in the generating transition sequence starts from state $i$. It will always be clear from the context to what model $\alpha_l(i)$ and $\beta_l(i)$ refer, so the dependence on $\boldsymbol{m}$ is not made explicit. Using these, we can now compute $P_\Theta(\boldsymbol{y} \,|\, \boldsymbol{m})$ as $P_\Theta(\boldsymbol{y} \,|\, \boldsymbol{m}) = \alpha_L(F) = \beta_0(0)$, where $F$ is the final state.

### 2.3.2 Recursive computation of $\alpha_l(i)$ and $\beta_l(i)$

The computation of $\alpha_l(i)$ and $\beta_l(i)$ is done by creating a *trellis* in which the $l$th column corresponds to time $l$ (and the $l$th observation, except for the 0th column) and the $i$th line corresponds to the $i$th state in the HMM. They are computed recursively, column by column, $\alpha_l(i)$ starting with column 0 and $\beta_l(i)$ starting with column $L_y$.

Empty transitions make the computations somewhat more difficult and we have to make the assumption that there is no path from a state to itself that only uses empty transitions. If this were not the case, there would be situations where the values of some $\alpha_l(i)$ and $\beta_l(i)$ would be needed in order to compute their own value. When the above assumption is true, it is always possible to order the HMMs' states in such a way that empty transitions always reach a state with a number larger than the one they leave. This state numbering is used by the algorithms to determine in what order computations have to be performed. It will always be assumed that such a numbering is carried out.

For both $\alpha_l(i)$ and $\beta_l(i)$, the recursive computation is divided into a sum over empty transitions and a sum over full transitions. For $\alpha_l(i)$, the computation is done in a forward manner. Each trellis element is computed, in increasing order of column and state numbers, using the following algorithm:

a) Initialization:

$$\alpha_0(i) = \begin{cases} 1, & \text{if } i=0 \\ 0, & \text{if } i\neq 0 \end{cases}$$

b) Recursion:

$$\alpha_l(i) = \sum_{(t\,|\,t \text{ full},\ r_t=i)} \alpha_{l-1}(l_t) q_t\, b_t(y_l) + \sum_{(t\,|\,t \text{ empty},\ l_t<i, r_t=i)} \alpha_l(l_t) q_t \ , \qquad (2.7)$$

where the notation $\sum_{(t\,|\,t \text{ full},\ r_t=i)}$ means a sum over all full transitions reaching state $i$. Note that, for empty transitions, we have used $\sum_{(t\,|\,t \text{ empty},\ l_t<i, r_t=i)}$ to make explicit our assumption that the numbering, as described above, is correct. Because computations are carried out in a forward manner, $\alpha_l(i)$ is usually referred to as the *forward trellis*.

For $\beta_l(i)$, the computation is done in a backward manner. Each trellis element is computed, in decreasing order of column and state numbers, using the following algorithm:

a) Initialization:

$$\beta_{L,}(i) = \begin{cases} 1, & \text{if } i=F \\ 0, & \text{if } i\neq F \end{cases}$$

b) Recursion:

$$\beta_l(i) = \sum_{(t \mid t \text{ full}, l_t = i)} \beta_{l+1}(r_t) q_t b_t(\underline{y}_{l+1}) + \sum_{(t \mid t \text{ empty}, r_t > i, l_t = i)} \beta_l(r_t) q_t \ , \quad (2.8)$$

where $F$ is again the final state. Because computations are carried out in a backward manner, $\beta_l(i)$ is usually referred to as the *backward trellis*. Using these algorithms, the forward and backward trellises can be computed in a time which increases linearly with the sequence length $L_y$.

## 2.3.3 Probability computations using $\alpha_l(i)$ and $\beta_l(i)$

Let us define $P_{\Theta,l}(t,\underline{y} \mid m)$ as the probability that the observation sequence $\underline{y}$ was generated by model $m$, using a transition sequence in which $t$ was taken at time $l$. Remember that, for a given path $\underline{t}$, $t$ taken at time $l$ means that $t \in \underline{t}[l]$. In practice, a transition at time $l$ is a transition that reaches a node in the $l$th trellis column.

In theory, $P_{\Theta,l}(t,\underline{y} \mid m)$ can be computed as follows:

$$P_{\Theta,l}(t,\underline{y} \mid m) = \sum_{(\underline{t} \in m \mid t \in \underline{t}[l])} P_{\Theta}(\underline{y},\underline{t} \mid m)$$

$$= \sum_{(\underline{t} \in m \mid t \in \underline{t}[l])} \prod_{i=1}^{L_t} q_{t_i} \prod_{j=1}^{L_y} b_{t_{n_j}}(\underline{y}_j) \ , \quad (2.9)$$

where the notation $\sum_{(\underline{t} \in m \mid t \in \underline{t}[l])}$ means a summation over all paths in $m$ in which transition $t$ is taken at time $l$. But again, this is prohibitively expensive. From the definitions (2.5) and (2.6), it is not difficult to see, however, that $P_{\Theta,l}(t,\underline{y} \mid m)$ can be computed much more simply as

$$P_{\Theta,l}(t,\underline{y}\,|m) = \begin{cases} \alpha_l(l_t)q_t\beta_l(r_t), & \text{if } t \text{ empty} \\ \alpha_{l-1}(l_t)q_t b_t(\underline{y}_l)\beta_l(r_t), & \text{if } t \text{ full.} \end{cases} \tag{2.10}$$

As we shall realize, $P_{\Theta,l}(t,\underline{y}\,|m)$ is a very important quantity in the theory (and the practice) of HMMs. It is central to most computations involved in the estimation of the HMM parameter set $\Theta$. The fact that (2.10) allows $P_{\Theta,l}(t,\underline{y}\,|m)$ to be efficiently computed from the forward and backward trellises is thus extremely important.

## 2.4. A survey of output distributions

So far, little has been said about the different forms that $b(\cdot)$ can take. Different forms arise as a result of the basic distinction between discrete and continuous parameters. Even within these two categories, several different forms arise. This section describes the forms most often encountered in practice.

1) Discrete (basic):

$$b(\underline{y}) = p(\underline{y}\,|b), \quad \underline{y} = y \in \{0,1,\cdots,K-1\}. \tag{2.11}$$

This is the simplest case. $\underline{y}$ is a discrete scalar which can only take one of $K$ different values. It is the form most often encountered in earlier systems [JELI 76], [RABI 83b], and, for a long time, it was the form usually producing the best results.

2) Discrete with multiple codebooks [GUPT 87]:

$$b(\underline{y}) = \prod_{c=1}^{NC} p_c(y_c\,|b), \tag{2.12}$$

where $NC$ is the number of codebooks, $y_c$ is the $c$th component of $\underline{y}$ and $y_c \in \{0,1,\cdots,K_c-1\}$, where $K_c$ is the size of the $c$th codebook. Multiple codebooks are useful when the number of continuous parameters is

large and it is desired to keep quantization distortion as small as possible [LEEK 88]. The continuous parameters are separated into groups which are independently vector quantized using different codebooks. All discrete components are assumed independent and their probabilities are simply multiplied. Note that the basic discrete distribution is a special case of this distribution.

3) Continuous (Gaussian) [BROW 87]:

$$b(\underline{y}) = \frac{1}{|\Sigma_b|^{1/2}(2\pi)^{N/2}} e^{-\frac{1}{2}(\underline{y}-\underline{\mu}_b)^T \Sigma_b^{-1}(\underline{y}-\underline{\mu}_b)}, \qquad (2.13)$$

where $N$ is the number of elements in $\underline{y}$, and $\mu_b$ and $\Sigma_b$ are, respectively, the mean vector and covariance matrix of the distribution $b$. This is an $N$-dimensional multivariate Gaussian distribution. We have the following special case:

4) Continuous (diagonal Gaussian):

$$b(\underline{y}) = \prod_{i=1}^{N} \frac{1}{\sigma_{b,i}\sqrt{2\pi}} e^{\frac{-(y_i-\mu_{b,i})^2}{2\sigma_{b,i}^2}}, \qquad (2.14)$$

where $y_i$ is the $i$th element of $\underline{y}$, and $\mu_{b,i}$ and $\sigma_{b,i}^2$ are, respectively, the corresponding mean and variance. This assumes that all parameters are uncorrelated, which, in some circumstances, can be close to reality [HUNT 89].

HMMs with continuous distributions are usually referred to as *Continuous Densities Hidden Markov Models* (CDHMMs). Note that since the "true" distributions often have complex shapes, a good approximation to such distributions will usually not be possible with a single Gaussian density. This is especially true when the "true" distribution is not unimodal. In order to obtain better approximations, it is usually necessary to use *mixtures* of Gaussian densities [RABI 85b].

A mixture distribution is a weighted sum of distributions. Suppose, for example, that we have $K$ distributions $P_k(\underline{y})$, $k = 1, \cdots, K$, from which we want to build a mixture distribution $P_{mix}(\underline{y})$. Then, the expression for the mixture distribution would be

$$P_{mix}(\underline{y}) = \sum_{k=1}^{K} w_k P_k(\underline{y}) , \tag{2.15}$$

where $\sum_{k=1}^{K} w_k = 1$. In (2.15), the $w_k$'s are called *mixture weights* and the individual distributions $P_k(\underline{y})$ are called *mixture components*. The mixture components can be any type of distribution; however, in practice, Gaussian distributions are used most of the time. Nowadays, it is not uncommon to see systems using a large number of components per mixture, sometimes as many as 256 [LEEC 90a]. Note that mixture distributions can be implemented by having several parallel transitions between the same two states, each having one of the mixture components as its output distribution. This is illustrated in Figure 2.5.

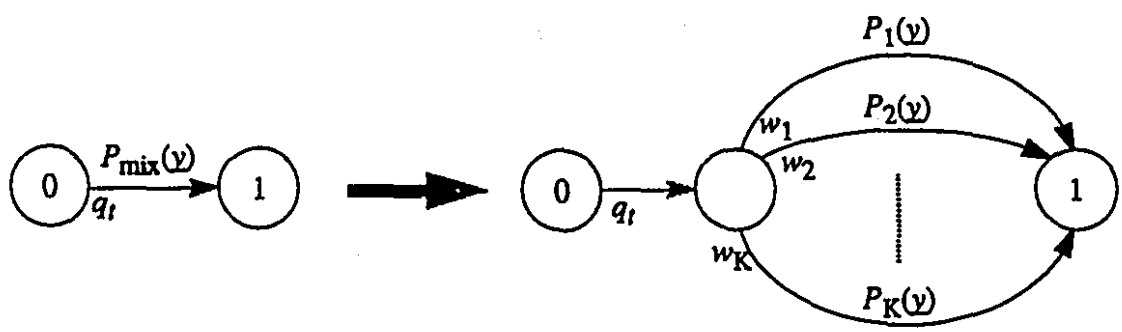


**Figure 2.5:** Implementation of a mixture distribution using one mixture component per transition

A special type of mixture distribution is:

5) Semi-continuous [HUAN 89]:

$$b(\underline{y}) = \sum_{k=0}^{K-1} P(\underline{y}|k)p(k|b) , \tag{2.16}$$

where $P(\underline{y}|k)$ is any continuous distribution (usually Gaussian). HMMs using distributions of this type are usually called *Semi-Continuous Hidden Markov Models* (SCHMMs). Note that the $K$ different continuous densities are shared by all output distributions in the application. This is why HMMs of this type are sometimes referred to as *Tied Mixtures* HMMs [BELL 89, BELL 90]. Note that the set of $K$ continuous densities is often called the "codebook". SCHMMs can be seen as an extension of discrete HMMs in the sense that, instead of only considering the closest codeword in the codebook, every codeword (density) is taken into account, each contributing with a weight proportional to its *a posteriori* probability.

A generalization of semi-continuous distributions is:

6) Multiple codebooks semi-continuous:

$$b(\underline{y}) = \prod_{c=1}^{NC} \sum_{k=0}^{K_c-1} P_c(\underline{y}_c|k)p_c(k|b) , \tag{2.17}$$

where $K_c$ is the size of the $c$th codebook and $\underline{y}_c$ is the part of $\underline{y}$ corresponding to the $c$th codebook. Again, codebooks are assumed independent. Note that it is straightforward to convert from multiple-codebook discrete HMMs to multiple-codebook semi-continuous HMMs. Indeed, the codebook probabilities $p_c(k|b)$ can be left unchanged, while the parameters of the tied mixtures can be computed directly from the codebook and the training data.

## 2.5. Maximum likelihood estimation of HMM parameters

We have seen, in the previous sections, how to compute probabilities using a model $\boldsymbol{m}$ and the HMM parameter vector $\Theta$. Nothing, however, was said about how $\Theta$ was obtained in the first place. Maximum likelihood estimation (MLE), the most commonly used approach to estimate $\Theta$, is the topic of this section. This estimation procedure is usually referred to as the *Baum-Welch* algorithm [BAUM 72], also known as the *forward-backward* algorithm. Since, in the literature, *MLE training* is usually synonymous with the Baum-Welch algorithm, we will be using these three names interchangeably.

In MLE training, we try to find the HMM parameter set $\Theta$ which maximizes the probability of the observed data, given the model corresponding to the data. Assuming that the observed data is made of several *independent* observation sequences $\boldsymbol{y}^r$, $r$=1,2, $\cdots$ , we try to find

$$\Theta = \arg\max_{\Theta'} \prod_r P_{\Theta'}(\boldsymbol{y}^r | \boldsymbol{m}_r) ,$$

where $\boldsymbol{m}_r \equiv \boldsymbol{m}_{w_r}$, $w_r$ being the word sequence corresponding to the $r$th observation $\boldsymbol{y}^r$ in the training set, and where $\prod_r$ means a product over all observation sequences in the training set. In general, it is not feasible to find a globally optimal $\Theta$. Instead, the optimization technique used starts from an initial value of $\Theta$ and converges to a local optimum in the parameter space. It does this iteratively, each iteration starting with a parameter vector $\Theta$ and finding $\hat{\Theta}$ such that

$$\prod_r P_{\hat{\Theta}}(\boldsymbol{y}^r | \boldsymbol{m}_r) \geq \prod_r P_{\Theta}(\boldsymbol{y}^r | \boldsymbol{m}_r) . \tag{2.18}$$

In order to see how this is done, first note that (2.18) is equivalent to

$$\log \frac{\prod_r P_{\hat{\theta}}(y^r|m_r)}{\prod_r P_{\theta}(y^r|m_r)} = \sum_r \log \frac{P_{\hat{\theta}}(y^r|m_r)}{P_{\theta}(y^r|m_r)} \geq 0 . \tag{2.19}$$

This, in turn, is the same as

$$\sum_r \log \sum_{t \in m_r} \frac{P_{\hat{\theta}}(y^r,t|m_r)}{P_{\theta}(y^r|m_r)} =$$

$$\sum_r \log \sum_{t \in m_r} \frac{P_{\theta}(y^r,t|m_r)}{P_{\theta}(y^r|m_r)} \frac{P_{\hat{\theta}}(y^r,t|m_r)}{P_{\theta}(y^r,t|m_r)} \geq 0 , \tag{2.20}$$

where as before we assume that the sum is taken over all transition sequences with $L_r = L_{y^r}$ full transitions (all other sequences have zero probability), so that we have $L_t \geq L_r$. But, since

$$\sum_{t \in m_r} \frac{P_{\theta}(y^r,t|m_r)}{P_{\theta}(y^r|m_r)} = \sum_{t \in m_r} P_{\theta}(t|y^r,m_r) = 1$$

and because log is a concave function, we have

$$\sum_r \log \sum_{t \in m_r} \frac{P_{\theta}(y^r,t|m_r)}{P_{\theta}(y^r|m_r)} \frac{P_{\hat{\theta}}(y^r,t|m_r)}{P_{\theta}(y^r,t|m_r)} \geq$$

$$\sum_r \sum_{t \in m_r} \frac{P_{\theta}(y^r,t|m_r)}{P_{\theta}(y^r|m_r)} \log \frac{P_{\hat{\theta}}(y^r,t|m_r)}{P_{\theta}(y^r,t|m_r)} , \tag{2.21}$$

which means that making the right-hand side of (2.21) positive automatically makes the left-hand side also positive. The advantage of the right-hand side, however, is that it is much easier to work with. Indeed, if $t \in m_r$, then

$$\log P_\Theta(y^r,t) = \log \prod_{i=1}^{L_t} q_{t_i} \prod_{l=1}^{L_r} b_{t_{n_l}}(y_l) = \sum_{i=1}^{L_t} \log q_{t_i} + \sum_{l=1}^{L_r} \log b_{t_{n_l}}(y_l)\,,$$

which means that (2.21) can be rewritten as

$$\sum_r \sum_{t\in m_r} \sum_{i=1}^{L_t} \frac{P_\Theta(y^r,t\,|m_r)}{P_\Theta(y^r|m_r)} \log \frac{\hat{q}_{t_i}}{q_{t_i}} +$$

$$\sum_r \sum_{t\in m_r} \sum_{l=1}^{L_r} \frac{P_\Theta(y^r,t\,|m_r)}{P_\Theta(y^r|m_r)} \log \frac{\hat{b}_{t_{n_l}}(y_l)}{b_{t_{n_l}}(y_l)} \geq 0\,. \qquad (2.22)$$

This, in turn, can be written as

$$\sum_r \sum_\tau \sum_{(t\,|\tau(t)=\tau)} \sum_{l=1}^{L_r} \sum_{(t\,|\,t\in t[l])} \frac{P_\Theta(y^r,t\,|m_r)}{P_\Theta(y^r|m_r)} \log \frac{\hat{q}_\tau}{q_\tau} +$$

$$\sum_r \sum_b \sum_{(t\,|b_t=b)} \sum_{l=1}^{L_r} \sum_{(t\,|t_{n_l}=t)} \frac{P_\Theta(y^r,t\,|m_r)}{P_\Theta(y^r|m_r)} \log \frac{\hat{b}(y_l)}{b(y_l)} \geq 0\,, \qquad (2.23)$$

where the notation $\sum_{(t\,|\tau(t)=\tau)}$ means a sum over all transitions whose corresponding unit transition is $\tau$ and $\sum_{(t\,|b_t=b)}$ means the summation over all transition whose corresponding output distribution (if any) is $b$. From (2.23) and (2.9), we obtain

$$\sum_r \sum_\tau \sum_{(t\,|\tau(t)=\tau)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} \log \frac{\hat{q}_\tau}{q_\tau} +$$

$$\sum_r \sum_b \sum_{(t\,|b_t=b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} \log \frac{\hat{b}(y_l)}{b(y_l)} \geq 0 \qquad (2.24)$$

In (2.24), all transition probabilities $q_\tau$, as well as all output distributions $b$ are separated into different elements of a summation, which means that they can

be individually optimized. We now describe how this is done.

### 2.5.1. Transition probabilities

In solving for transition probabilities, we are faced with a constrained optimization problem. The constraint is that the probabilities for all transitions leaving the same state should sum to 1. Following the development in [LEVI 83], if we have a function $F(Z)$ of the form

$$F(Z) = \sum_i a_i \log z_i \ , \tag{2.25}$$

where the $a_i$'s are all positive and the $z_i$'s satisfy $\sum_i z_i = 1$, then using Lagrange multipliers to find the optimal values for the $z_i$'s, we find that

$$z_i = \frac{a_i}{\sum_{i'} a_{i'}} \ . \tag{2.26}$$

Going back to the problem of transition probabilities estimation, we see from (2.24) that in order to make the first term positive for all $q$'s, we need to maximize

$$\sum_{(\tau'|l_{\tau'}=l_\tau)} \sum_{\tau} \sum_{(t|\tau(t)=\tau')} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} \log \hat{q}_{\tau'} \ , \tag{2.27}$$

in which we used the notation $\sum_{(\tau'|l_{\tau'}=l_\tau)}$ to mean a summation over all transitions with the same left state as $\tau$ (which means they belong to the same transition probability distribution). Using (2.26), this leads to

$$\hat{q}_r = \frac{\sum_r \sum_{(t|\tau(t)=r)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)}}{\sum_{(r'|l_{r'}=l_r)} \sum_r \sum_{(t|\tau(t)=r')} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)}} . \tag{2.28}$$

Note that if we define

$$c_r = \sum_r \sum_{(t|\tau(t)=r)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} , \tag{2.29}$$

then (2.28) can be expressed as

$$\hat{q}_r = \frac{c_r}{\sum_{(r'|l_{r'}=l_r)} c_{r'}} . \tag{2.30}$$

As we will see, quantities similar to $c_r$ appear in the MLE reestimation formulas for all HMM parameters. Since such quantities are computed by summing a contribution from all time instants of all sequences in the training set, they are usually referred to as *counts*.

### 2.5.2. Output distributions

Output distributions can be formed by the product of other distributions (multiple-codebook discrete). They can also be formed by a weighted sum of distributions (semi-continuous, mixture Gaussian), in which case we talk about a *mixture* of distributions. And, of course, they can be both (multiple-codebook semi-continuous). Individual distributions in a product are additively separated by the logarithm in (2.24), so they can be considered separately. Mixtures, on the other hand, need special handling. This will be considered later.

### 2.5.2.1. Discrete distributions

This section covers both basic discrete and multiple-codebook distributions. From (2.24), we must for each $b$ find the codebook probabilities $\hat{p}(k\,|\,b)$ that maximize

$$\sum_r \sum_{(t\,|\,b_t=b)} \sum_k \sum_{(l\,|\,y_l^r=k)} \frac{P_{\Theta,l}(t,\underline{y}^r\,|\,m_r)}{P_\Theta(\underline{y}^r\,|\,m_r)} \log \hat{p}(k\,|\,b)\,. \tag{2.31}$$

Using (2.26), this leads to

$$\hat{p}(k\,|\,b) = \frac{\sum_r \sum_{(t\,|\,b_t=b)} \sum_{(l\,|\,y_l^r=k)} \frac{P_{\Theta,l}(t,\underline{y}^r\,|\,m_r)}{P_\Theta(\underline{y}^r\,|\,m_r)}}{\sum_{k'} \sum_r \sum_{(t\,|\,b_t=b)} \sum_{(l\,|\,y_l^r=k')} \frac{P_{\Theta,l}(t,\underline{y}^r\,|\,m_r)}{P_\Theta(\underline{y}^r\,|\,m_r)}}\,. \tag{2.32}$$

### 2.5.2.2. Gaussian distributions

From (2.24), we must maximize

$$F(\hat{\underline{\mu}}_b, \hat{\Sigma}_b) = \sum_r \sum_{(t\,|\,b_t=b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,\underline{y}^r\,|\,m_r)}{P_\Theta(\underline{y}^r\,|\,m_r)} \left( -\frac{1}{2}\log\,|\hat{\Sigma}_b\,| - \frac{1}{2}(\underline{y}_l^r - \hat{\underline{\mu}}_b)^T \hat{\Sigma}_b^{-1} (\underline{y}_l^r - \hat{\underline{\mu}}_b) \right). \tag{2.33}$$

Now, using the following matrix identities:

$$\frac{\partial |A|}{\partial A^{-1}} = -|A|\,A\,, \quad \frac{\partial \underline{\nu}^T A^{-1} \underline{\nu}}{\partial \underline{\nu}} = 2A^{-1}\underline{\nu}, \quad \frac{\partial \underline{\nu}^T A^{-1} \underline{\nu}}{\partial A^{-1}} = \underline{\nu}\,\underline{\nu}^T\,, \tag{2.34}$$

where $\underline{\nu}$ is a column vector of dimension $N$ and $A$ is an $N \times N$ symmetric

matrix, we can take the derivative of $F(\hat{\underline{\mu}}_b, \hat{\Sigma}_b)$ with respect to $\hat{\underline{\mu}}_b$ and $\hat{\Sigma}_b$:

$$\frac{\partial F(\hat{\underline{\mu}}_b, \hat{\Sigma}_b)}{\partial \hat{\underline{\mu}}_b} = \sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t, \underline{y}^r | m_r)}{P_\Theta(\underline{y}^r | m_r)} \hat{\Sigma}_b^{-1} (\underline{y}_t^r - \hat{\underline{\mu}}_b) \tag{2.35}$$

$$\frac{\partial F(\hat{\underline{\mu}}_b, \hat{\Sigma}_b)}{\partial \hat{\Sigma}_b^{-1}} =$$

$$\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t, \underline{y}^r | m_r)}{P_\Theta(\underline{y}^r | m_r)} \left( \frac{1}{2} \hat{\Sigma}_b - \frac{1}{2} (\underline{y}_t^r - \hat{\underline{\mu}}_b)(\underline{y}_t^r - \hat{\underline{\mu}}_b)^T \right). \tag{2.36}$$

Setting the derivatives to zero, we obtain the following reestimation formulas:

$$\hat{\underline{\mu}}_b = \frac{\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t, \underline{y}^r | m_r)}{P_\Theta(\underline{y}^r | m_r)} \underline{y}_t^r}{\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t, \underline{y}^r | m_r)}{P_\Theta(\underline{y}^r | m_r)}} \tag{2.37}$$

$$\hat{\Sigma}_b = \frac{\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t, \underline{y}^r | m_r)}{P_\Theta(\underline{y}^r | m_r)} \underline{y}_t^r \underline{y}_t^{rT}}{\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t, \underline{y}^r | m_r)}{P_\Theta(\underline{y}^r | m_r)}} - \hat{\underline{\mu}}_b \, \hat{\underline{\mu}}_b^T . \tag{2.38}$$

### 2.5.2.3. Diagonal Gaussian distributions

Since diagonal Gaussian distributions are products of one-dimensional Gaussian distributions, each of these can be considered separately. Let us consider the parameters $\mu_i$ and $\hat{\sigma}_i$ of a diagonal Gaussian distribution $b$. From (2.24), we must maximize

$$F_b(\hat{\mu}_i,\hat{\sigma}_i) = \sum_r \sum_{(t|b_t\equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} \left[ -\log \hat{\sigma}_i - \frac{(y^r_{l,i}-\hat{\mu}_i)^2}{2\hat{\sigma}_i^{\,2}} \right]. \qquad (2.39)$$

Setting the derivatives to zero, we obtain

$$\frac{\partial F_b(\hat{\mu}_i,\hat{\sigma}_i)}{\partial \hat{\mu}_i} = \sum_r \sum_{(t|b_t\equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} \frac{(y^r_{l,i}-\hat{\mu}_i)}{\hat{\sigma}_i^{\,2}} = 0$$

$$\frac{\partial F_b(\hat{\mu}_i,\hat{\sigma}_i)}{\partial \hat{\sigma}_i} = \sum_r \sum_{(t|b_t\equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} \left[ \frac{1}{\hat{\sigma}_i} + \frac{(y^r_{l,i}-\hat{\mu}_i)^2}{\hat{\sigma}_i^{\,3}} \right] = 0\,, \qquad (2.40)$$

which leads to

$$\hat{\mu}_i = \frac{\sum_r \sum_{(t|b_t\equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} y^r_{l,i}}{\sum_r \sum_{(t|b_t\equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)}} \qquad (2.41)$$

and

$$\hat{\sigma}_i^{\,2} = \frac{\sum_r \sum_{(t|b_t\equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} y^{r\,2}_{l,i}}{\sum_r \sum_{(t|b_t\equiv b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)}} - \hat{\mu}_i^{\,2}\,. \qquad (2.42)$$

### 2.5.2.4. Mixture distributions

As mentioned earlier, mixture distributions can be implemented by having several parallel transitions between the same two states, each with one of the mixture components as its output distribution. In that case, both the mixture weights (which, as illustrated in Figure 2.5, are the transition probabilities) and

the parameters of the mixture components can be estimated in the usual way.

Such an implementation, however, may add unnecessary overhead through the processing of a large number of transitions, which, in turn, may slow down the program. Moreover, in some cases it may be preferable that output distributions be full mixtures rather than simply one of their components. One example is when it is desired to find the most probable path (transition sequence) in a given model. If mixtures are broken into their individual components, then each component will be associated to a different path, which may be undesirable.

We will thus consider the implementation of mixture distributions associated to a single output distribution. This will be illustrated using the semi-continuous output distribution (2.16). From (2.24), we must have

$$\sum_r \sum_{(t|b_t=b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,\underline{y}^r|m_r)}{P_\Theta(\underline{y}^r|m_r)} \log \frac{\sum_{k=0}^{K-1} \hat{P}(\underline{y}_j^r|k)\hat{p}(k|b)}{\sum_{k=0}^{K-1} P(\underline{y}_j^r|k)p(k|b)} \geq 0 , \qquad (2.43)$$

which is the same as

$$\sum_r \sum_{(t|b_t=b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,\underline{y}^r|m_r)}{P_\Theta(\underline{y}^r|m_r)} \log \sum_{k=0}^{K-1} \frac{P(\underline{y}_j^r|k)p(k|b)}{\sum_{k'=0}^{K-1} P(\underline{y}_j^r|k')p(k'|b)} \frac{\hat{P}(\underline{y}_j^r|k)\hat{p}(k|b)}{P(\underline{y}_j^r|k)p(k|b)} =$$

$$\sum_r \sum_{(t|b_t=b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,\underline{y}^r|m_r)}{P_\Theta(\underline{y}^r|m_r)} \log \sum_{k=0}^{K-1} \gamma_b(k|\underline{y}_j^r) \frac{\hat{P}(\underline{y}_j^r|k)\hat{p}(k|b)}{P(\underline{y}_j^r|k)p(k|b)} \geq 0 , \qquad (2.44)$$

where we used

$$\gamma_b(k|\underline{y}) = \frac{P(\underline{y}|k)p(k|b)}{\sum_{k'} P(\underline{y}|k')p(k'|b)} . \qquad (2.45)$$

From $\sum_k \gamma_b(k|\underline{y}_j^r) = 1$ and the concavity of the log function, we have that:

$$\sum_r \sum_{(t|b_t=b)} \sum_{l=1}^{L_r} \frac{P_{\Theta,l}(t,\mathbf{y}^r|m_r)}{P_\Theta(\mathbf{y}^r|m_r)} \log \sum_{k=0}^{K-1} \gamma_b(k|\underline{y}_j^r) \frac{\hat{P}(\underline{y}_j^r|k)\hat{p}(k|b)}{P(\underline{y}_j^r|k)p(k|b)} \geq$$

$$\sum_r \sum_{(t|b_t=b)} \sum_{l=1}^{L_r} \sum_{k=0}^{K-1} \gamma_b(k|\underline{y}_j^r) \frac{P_{\Theta,l}(t,\mathbf{y}^r|m_r)}{P_\Theta(\mathbf{y}^r|m_r)} \log \frac{\hat{P}(\underline{y}_j^r|k)\hat{p}(k|b)}{P(\underline{y}_j^r|k)p(k|b)}. \quad (2.46)$$

Since the parameters in the second term in (2.46) are now separated by the logarithm, we know how to maximize it; this will automatically maximize the first.

## 2.6. Derivatives of probabilities

In several circumstances, it will be necessary to take the derivative of some HMM-related probabilities with respect to the individual components of the HMM parameter set $\Theta$. One example is when it desired to use gradient descent to optimize some function of those probabilities. In order to compute the gradient, it is necessary to be able to compute the derivative of certain probabilities with respect to the HMM parameters. Fortunately, such derivatives are usually fairly easy to compute. The present section shows how.

We want to compute the partial derivative of $P_\Theta(\mathbf{y}|m)$ with respect to the components of $\Theta$. We will use $P_\Theta(\mathbf{y}|m)$ as expressed in (2.4). First, let us find the derivative with respect to $q_r$

$$\frac{\partial P_\Theta(\mathbf{y}|m)}{\partial q_r} = \sum_{l=1}^{L_y} \sum_{(t|\tau(t)=r)} \sum_{(t\in m|t\in t[l])} \prod_{\substack{i=1 \\ i\neq n_l(t)}}^{L_t} q_{t_i} \prod_{j=1}^{L_y} b_{t_{n_j}}(\underline{y}_j)$$

$$= \frac{1}{q_r} \sum_{(t|\tau(t)=r)} \sum_{l=1}^{L_y} \sum_{(t\in m|t\in t[l])} \prod_{i=1}^{L_t} q_{t_i} \prod_{j=1}^{L_y} b_{t_{n_j}}(\underline{y}_j)$$

$$= \frac{1}{q_\tau} \sum_{(t \mid \tau(t) \equiv \tau)} \sum_{l=1}^{L_y} P_{\Theta,l}(t, \underline{y} \mid m) \,. \tag{2.47}$$

Now, let us consider the partial derivative of $P_\Theta(\underline{y} \mid m)$ with respect to the output probability $b(\underline{y}_l)$.

$$\frac{\partial P_\Theta(\underline{y} \mid m)}{\partial b(\underline{y}_l)} = \sum_{(t \mid b_t \equiv b)} \sum_{(t \in m \mid t_{n_l} \equiv t)} \prod_{i=1}^{L_t} q_{t_i} \prod_{\substack{j=1 \\ j \neq l}}^{L_y} b_{t_{n_j}}(\underline{y}_j)$$

$$= \frac{1}{b(\underline{y}_l)} \sum_{(t \mid b_t \equiv b)} \sum_{(t \in m \mid t_{n_l} \equiv t)} \prod_{i=1}^{L_t} q_{t_i} \prod_{j=1}^{L_y} b_{t_{n_j}}(\underline{y}_j)$$

$$= \frac{1}{b(\underline{y}_l)} \sum_{(t \mid b_t \equiv b)} P_{\Theta,l}(t, \underline{y} \mid m) \,. \tag{2.48}$$

Using this result, we can now compute the derivative with respect to any parameter $\theta$ of $b(\cdot)$, using, in effect, the chain rule, that is,

$$\frac{\partial P_\Theta(\underline{y} \mid m)}{\partial \theta} = \sum_{l=1}^{L_y} \frac{\partial P_\Theta(\underline{y} \mid m)}{\partial b(\underline{y}_l)} \frac{\partial b(\underline{y}_l)}{\partial \theta}$$

$$= \sum_{l=1}^{L_y} \frac{1}{b(\underline{y}_l)} \frac{\partial b(\underline{y}_l)}{\partial \theta} \sum_{(t \mid b_t \equiv b)} P_{\Theta,l}(t, \underline{y} \mid m) \,. \tag{2.49}$$

If $\theta$ is a parameter that applies to all distributions in a certain group $\Gamma$ (which may include all distribution), like the parameters of tied mixtures in semi-continuous HMMs or global exponents (see Chapter 4), then the derivative with respect to $\theta$ becomes

$$\frac{\partial P_\Theta(\underline{y} \mid m)}{\partial \theta} = \sum_{b \in \Gamma} \sum_{l=1}^{L_y} \frac{\partial P_\Theta(\underline{y} \mid m)}{\partial b(\underline{y}_l)} \frac{\partial b(\underline{y}_l)}{\partial \theta}$$

$$= \sum_{l=1}^{L_y} \sum_{(t\,|\,b_t \in \Gamma)} \frac{1}{b_t(\underline{y}_l)} P_{\Theta,l}(t,\underline{y}\,|\,m) \frac{\partial b_t(\underline{y}_l)}{\partial \theta} . \qquad (2.50)$$

Now, using (2.49) with the output distributions described previously, we obtain:

1) Discrete (basic):

$$\frac{\partial P_\Theta(\underline{y}\,|\,m)}{\partial p(k\,|\,b)} = \frac{1}{p(k\,|\,b)} \sum_{(l\,|\,\underline{y}_l = k)} \sum_{(t\,|\,b_t \equiv b)} P_{\Theta,l}(t,\underline{y}\,|\,m) . \qquad (2.51)$$

2) Discrete with multiple codebooks:

$$\frac{\partial P_\Theta(\underline{y}\,|\,m)}{\partial p_c(k\,|\,b)} = \frac{1}{p_c(k\,|\,b)} \sum_{(l\,|\,y_{l,c} = k)} \sum_{(t\,|\,b_t \equiv b)} P_{\Theta,l}(t,\underline{y}\,|\,m) . \qquad (2.52)$$

3) Continuous (Gaussian):

$$\frac{\partial P_\Theta(\underline{y}\,|\,m)}{\partial \underline{\mu}_b} = \sum_{l=1}^{L_y} \sum_{(t\,|\,b_t \equiv b)} P_{\Theta,l}(t,\underline{y}\,|\,m)\, \Sigma_b{}^{-1}(\underline{y}_l - \underline{\mu}_b) , \qquad (2.53)$$

$$\frac{\partial P_\Theta(\underline{y}\,|\,m)}{\partial \Sigma_b{}^{-1}} =$$

$$\sum_{l=1}^{L_y} \sum_{(t\,|\,b_t \equiv b)} P_{\Theta,l}(t,\underline{y}\,|\,m) \left( \frac{1}{2}\Sigma_b - \frac{1}{2}(\underline{y}_l - \underline{\mu}_b)(\underline{y}_l - \underline{\mu}_b)^T \right) . \qquad (2.54)$$

4) Continuous (diagonal Gaussian):

$$\frac{\partial P_\Theta(\underline{y}\,|\,m)}{\partial \mu_i} = \sum_{l=1}^{L_y} \sum_{(t\,|\,b_t \equiv b)} P_{\Theta,l}(t,\underline{y}\,|\,m) \frac{(y_{l,i} - \mu_i)}{\sigma_i{}^2} , \qquad (2.55)$$

$$\frac{\partial P_\Theta(\underline{y}\,|m)}{\partial \sigma_i} = \sum_{l=1}^{L_y} \sum_{(t\,|b_t \equiv b)} P_{\Theta,l}(t,\underline{y}\,|m) \left[ -\frac{1}{\sigma_i} + \frac{(y_{l,i} - \mu_i)^2}{\sigma_i^{\,3}} \right]. \qquad (2.56)$$

5) Semi-continuous:

$$\frac{\partial P_\Theta(\underline{y}\,|m)}{\partial p(k\,|b)} = \frac{1}{p(k\,|b)} \sum_{l=1}^{L_y} \sum_{(t\,|b_t \equiv b)} \frac{P(\underline{y}_l\,|k)p(k\,|b)}{\sum_{k'} P(\underline{y}_l\,|k')p(k'\,|b)} P_{\Theta,l}(t,\underline{y}\,|m)$$

$$= \frac{1}{p(k\,|b)} \sum_{l=1}^{L_y} \sum_{(t\,|b_t \equiv b)} \gamma_b(k\,|\underline{y}_l)\, P_{\Theta,l}(t,\underline{y}\,|m)\,, \qquad (2.57)$$

where we used $\gamma_b(k\,|\underline{y})$ as expressed in (2.45). Now let $\alpha$ be a parameter of the $P(\underline{y}\,|k)$, one of the tied continuous distributions. Then, since all distributions are functions of $P(\underline{y}\,|k)$, we obtain

$$\frac{\partial P_\Theta(\underline{y}\,|m)}{\partial \alpha} = \sum_{l=1}^{L_y} \sum_{t\text{full}} \frac{1}{P(\underline{y}_l\,|k)}\, \gamma_b(k\,|\underline{y}_l) P_{\Theta,l}(t,\underline{y}\,|m)\, \frac{\partial P(\underline{y}_l\,|k)}{\partial \alpha}\,. \qquad (2.58)$$

6) Multiple codebooks semi-continuous:

$$\frac{\partial P_\Theta(\underline{y}\,|m)}{\partial p_c(k\,|b)} = \frac{1}{p_c(k\,|b)} \sum_{l=1}^{L_y} \sum_{(t\,|b_t \equiv b)} \frac{P_c(\underline{y}_{l,c}\,|k)p_c(k\,|b)}{\sum_{k'} P_c(\underline{y}_{l,c}\,|k')p_c(k'\,|b)} P_{\Theta,l}(t,\underline{y}\,|m)$$

$$= \frac{1}{p_c(k\,|b)} \sum_{l=1}^{L_y} \sum_{(t\,|b_t \equiv b)} \gamma_{b,c}(k\,|\underline{y}_{l,c}) P_{\Theta,l}(t,\underline{y}\,|m)\,, \qquad (2.59)$$

where $\underline{y}_{l,c}$ is the part of $\underline{y}_l$ corresponding to the $c$th codebook and where, this time, we used

$$\gamma_{b,c}(k\,|\underline{y}_{l,c}) = \frac{P_c(\underline{y}_{l,c}\,|k)p_c(k\,|b)}{\sum_{k'} P_c(\underline{y}_{l,c}\,|k')p_c(k'\,|b)}\,. \qquad (2.60)$$

Now, let $\alpha$ be a parameter of $P(\underline{y}_{J,c}\,|k)$. Then,

$$\frac{\partial P_{\Theta}(\underline{y}\,|\boldsymbol{m})}{\partial \alpha} =$$

$$\sum_{l=1}^{L_{\underline{y}}} \frac{1}{P_c(\underline{y}_{J,c}\,|k)}\, \gamma_{b,c}(k\,|\underline{y}_{J,c}) \sum_{t\,\text{full}} P_{\Theta,l}(t,\underline{y}\,|\boldsymbol{m})\, \frac{\partial P(\underline{y}_{J,c}\,|k)}{\partial \alpha} \qquad (2.61)$$

# 3. HMMS: FROM THEORY TO PRACTICE

## 3.1. Introduction

The previous chapter described the theoretical foundations of HMMs. Even though the presentation was strongly oriented towards speech recognition applications, there remains a big step from the theory as presented, to a high-performance speech recognition system.

Indeed, there are a number of practical considerations that the basic theory does not address, but which are important when building a HMM-based system. These range from simple implementation problems to the more difficult question of how to get the most from a limited amount of training data.

This chapter addresses these issues. Our aim is to provide the reader with not only a clear understanding on how to implement the HMM theory, but also a good idea of how to best apply the HMM framework in different speech recognition applications.

## 3.2. Application considerations

### 3.2.1. Initialization and training

Training HMMs consists of finding the parameter vector $\Theta$ which, as much as possible, maximizes the chosen objective function. For MLE training, the objective function is

$$R(\Theta) = \prod_r P_\Theta(y^r | m_r) \,. \tag{3.1}$$

As mentioned in Chapter 2, it is very difficult to find a global maximum for $R(\Theta)$. Instead, training is done through an iterative process which would eventually converge to some local maximum. In practice, it is usually observed that even though recognition rate on a test set initially increases with each iteration, a maximum is quickly reached (often after as little as 3 or 4 iterations), after which the rate starts going down. This phenomenon is called *overtraining*. It is caused in part by the differences between the training and testing data sets and in part by the fact that MLE does not necessarily decrease error rate. Because of this, training is usually stopped after a fixed number of iterations, typically between 2 and 10.

Since the "real" test set — the speech from actual system use — is unavailable at training time, it is difficult to determine the optimal number of iterations. It is a function of a number of factors, such as the amount of training data, the type of output distributions used, the structure of the models, and the initial parameter vector $\Theta$. If a large quantity of training data is available, then the differences between training and testing data sets will probably be small and there will be a smaller possibility of overtraining to peculiarities in the training set. Similarly, continuous or semi-continuous HMMs may require more iterations than discrete HMMs.

Probably one of the most important practical considerations in HMM training is the initial $\Theta$. Indeed, not only will the initial $\Theta$ influence the required number of iterations to use but, more importantly, it will determine which local maximum the procedure converges to. Our experience, as well as that of others [RABI 89c], has shown that the initial $\Theta$ has a very strong influence on the system performance.

Good initialization allows the Baum-Welch algorithm to properly align the models with the desired speech segments. This results in both improved convergence and better models. For simple small vocabulary, isolated word recognition systems using endpoint detection [LAME 81] and word models, alignment is self-evident and uniform or even random initialization will perform satisfactorily for discrete HMMs [RABI 83b]; however, more complex tasks will usually require more sophisticated initialization schemes.

In general, initialization can reliably be done with hand-segmented data. The

speech segments corresponding to each unit model are used to derive proper initial distributions for the model. This is referred to as *bootstraping* the models. For discrete HMMs, this can be done by starting with initial models with uniform distributions and training them on the corresponding speech segments [SCHW 85] using the Baum-Welch algorithm. For example, the SPHINX system uses the phonetically segmented TIMIT database [LAME 86] to bootstrap the phone models. Note that this is also valid for semi-continuous HMMs, since the initial tied mixtures can be properly estimated from the codebook(s) and the training data. For HMMs with Gaussian densities, there is no such thing as a uniform distribution and it may help to have reasonable initial values for the mean and covariance parameters. These can be obtained from the segmented data, using some clustering procedure.

One problem with the use of hand-segmented data is that the actual segmentation process is quite labor-intensive and, in some situations, it is just not possible to do. Some researchers have found that good results can be obtained by linearly segmenting the training data into their corresponding state (or distribution) sequence and then using all frames corresponding to a given distribution to estimate its initial parameters [LEEC 90a]. Our own experience, however, clearly indicates that bootstraping with hand-segmented data produces better results. We feel that, even if it is not possible to segment the training data used for a particular application, it is usually possible to take advantage of segmented databases such as TIMIT to produce better initial models than what could be obtained otherwise.

Following bootstraping, MLE training is done using labeled (but not segmented) speech data. This means that for each training sentence,[1] the content of the sentence is known and the corresponding sentence model can be built. This model is used to compute the forward and backward trellises, which, in turn, are used to increment the counts from which the HMM reestimated parameters are computed.

[1]Here, sentence is used in a broad sense and can mean any type of speech utterance, from an isolated word, to a full paragraph.

### 3.2.2. Silence and optional models

The labeled data used for training usually contains speech signals preceded and followed by silence (or background noise). Moreover, there may or may not be pauses between words in the sentence. In general, the sentence labels only indicate the words spoken. No information is given about the possible presence of silences in the signal.

Suppose a 2-word sentence is used for training. Then, as described above, training is done by building the sentence model from the two word models. If, however, there is a pause between the two words, then it will be absorbed by one (or both) models. If, on the other hand, we choose to use a pause model between the words, and the words were spoken continuously, then the pause model will end up modeling parts of words. Both of these outcomes are undesirable. The problem comes from the ignorance of where the silences are in the signal.

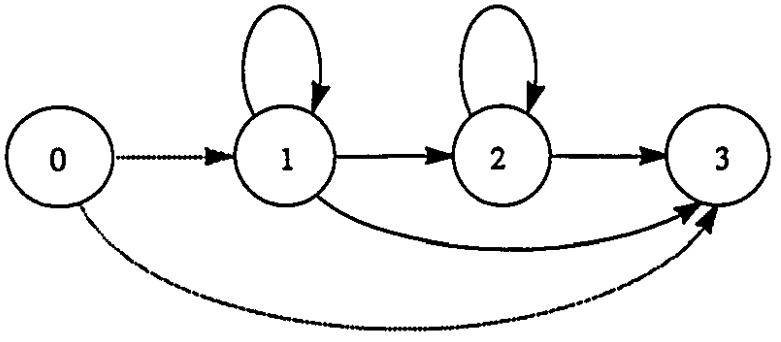


**Figure 3.1:** Optional model

An elegant solution to this problem is to use optional silence models. As shown in Figure 3.1, an optional model has an empty transition from the first to the last state, which basically allows the model to be jumped over. Thus, suppose the distributions of an optional silence model are well-trained and this model is used between two word models during MLE training. If there is actually no silence between the words, then the paths going through the

silence model will have very low probability and their effect on the distributions of the silence model will be negligible. If, on the other hand, there is a silence between the words, then since the silence model's distributions are well trained, the paths going through the model will have higher probability than the other ones.

This is a surprisingly effective technique and it works just as well for recognition as it does for training. Optional models can be used in any situation where it is not possible to say for sure from the speech labels whether or not a given sound is present. For example, the "t" in the word "eight" may or may not be pronounced, which suggests the use of an optional "t". The technique is, however, most effective with silences, for which it is relatively easy to obtain good models. In our systems, use of optional silence models at the beginning and end of sentences has completely eliminated the need for an endpoint detector [LAME 81] and performs much better than an endpoint detector.

### 3.2.3. Output distributions

If vector quantization (VQ) is applied to the feature vectors, then discrete distributions are used. This is attractive since, from the modeling point of view, discrete distributions don't make any (possibly erroneous) modeling assumption. However, these are simply transferred to the vector quantizer, which acts as an *a priori* classifier. Without VQ, continuous distributions are used. There has been much debate about which of discrete or continuous HMMs is superior. For a long time, IBM found discrete parameter HMMs to give better results than continuous ones. Brown [BROW 87] found that, for isolated recognition of the e-set, continuous parameters gave better results. For the SPHINX system, Lee [LEEK 88] chose discrete HMMs because of their lack of modeling assumptions and their efficiency.

Now that semi-continuous HMMs are becoming widely used, this debate seems to be finally resolved in favor of continuous distributions. Indeed, several state-of-the-art discrete systems such as SPHINX [HUAN 89], BYBLOS [KUBA 91] and DECYPHER [MURV 91] have recently been tried in semi-continuous mode and results have generally shown improvements over

the discrete systems.

Semi-continuous HMMs combine the simplicity of discrete HMMs with the robustness of continuous distributions. The transition from discrete to semi-continuous is easy to make: modifications to the HMM programs are relatively straightforward and simple modifications to the codebook design program will allow it to generate good initial mixture components. The resulting system will usually perform better than the original (especially with sparse training data), and the penalty in terms of increased computational complexity is generally not as high as it appears from (2.16). This is because, in practice, most implementations only use the $M$ most probable mixture components for a given frame (with, in general, $M \leq 10$).[3] Most techniques originally developed for discrete HMMs (e.g., speaker adaptation techniques [SCHW 87] and probability smoothing [LEEK 88, SCHW 89]) can also be applied to semi-continuous HMMs.

## 3.3. The training data problem

Among the practical problems faced by designers of speech recognition systems, one of the most challenging is certainly that of limited training data. If unlimited data (and memory and CPU resources) were available, phone, syllable or word models could all be trained perfectly for all contexts, speaker types and environments. Unfortunately, this is never possible in practice, which means that ways have to be found to make the most of whatever data is available.

### 3.3.1. Basic concepts

HMMs are used as parametric speech models. Training them consists of using the available speech samples (the training data) to learn the model parameters. It is implicitly assumed that the training data is representative of the process being modeled. This means that the speech encountered during

[3] Alternatively, some systems [MURV 91] use all mixture components with a probability greater than some fraction of the highest probability.

recognition should be well-modeled by the HMMs trained on the training data. This is, of course, never completely true. Whatever the size of the training data, there will usually be some speech features observed during recognition which could not have been learned from the training data. It is generally observed in practice that performance steadily improves as the available amount of training data increases. For example, in a speaker-independent system, it is desirable to have speech samples from as many types of speakers as possible.

It is important to realize that the representativeness of a given training data set is as much a function of the model used as it is of the data itself. Some models will generalize much better than others from a given training set. This can be illustrated with a simple example. Suppose a given random variable $X$ is normally distributed with unknown mean $\mu$ and variance $\sigma^2$. We want to model the distribution of $X$ from a sample of 200 independent outcomes $x_i$, $i$=1,...,200 of experiments corresponding to $X$. If we suppose that $X$ is normally distributed, then we can compute the estimates

$$\hat{\mu} = \frac{1}{200} \sum_{i=1}^{200} x_i \tag{3.2}$$

$$\hat{\sigma}^2 = \frac{1}{199} \sum_{i=1}^{200} (x_i - \hat{\mu})^2 . \tag{3.3}$$

In general, the estimates obtained will be very good so that we will be able to say that the training sample is representative of the process. This is illustrated in Figure 3.2 where the true distribution is compared to the estimated distribution obtained from a sample of 200 randomly generated numbers (using the true distribution).

Note, however, that if $X$ is not normally distributed, then the estimated distribution may be quite inaccurate, regardless of the training data size. It may thus seem preferable not to make any assumptions about the given distributions. This can always be done by estimating discrete probabilities using an histogram. If the histogram has, say, 50 unit intervals, then 50 probabilities will be computed using relative counts. So, instead of 2 parameters, we now have 50 discrete probabilities to estimate. Because of the small sample size,

most estimates will probably be very bad. Even worse, there will most likely be intervals not represented in the training sample, thus resulting in probability estimates of zero. This is illustrated in Figure 3.3, where the histogram is computed from the same 200 numbers used in Figure 3.2. So in this case, the training sample is clearly not representative.

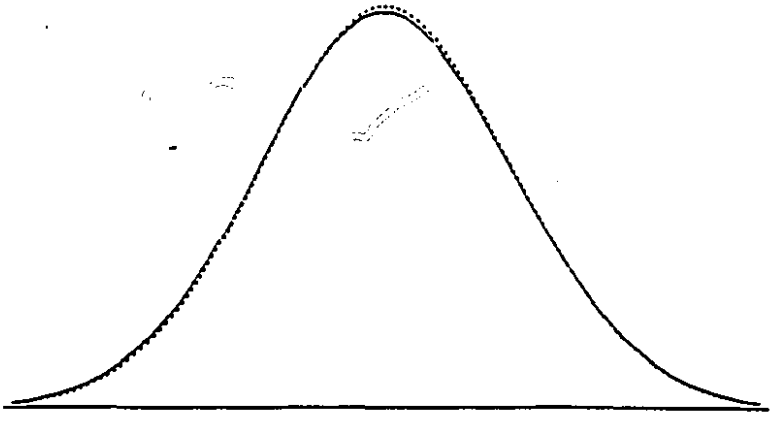

**Figure 3.2:** True distribution (solid line). Estimate (dashed).

In general, the more parameters there are to estimate, the more training data is required to estimate them. Discrete HMMs usually have a large number of parameters and thus require large amounts of training data. Continuous HMMs using single diagonal normal distributions (2.14) have much fewer parameters. Unfortunately, simple distributions may be so different from the "true" distributions that, no matter how much training data is available, performance will always be poor. Mixtures of Gaussian distributions can produce good approximations of complex distributions. However, the number of parameters to estimate is usually proportional to the number of mixture components.

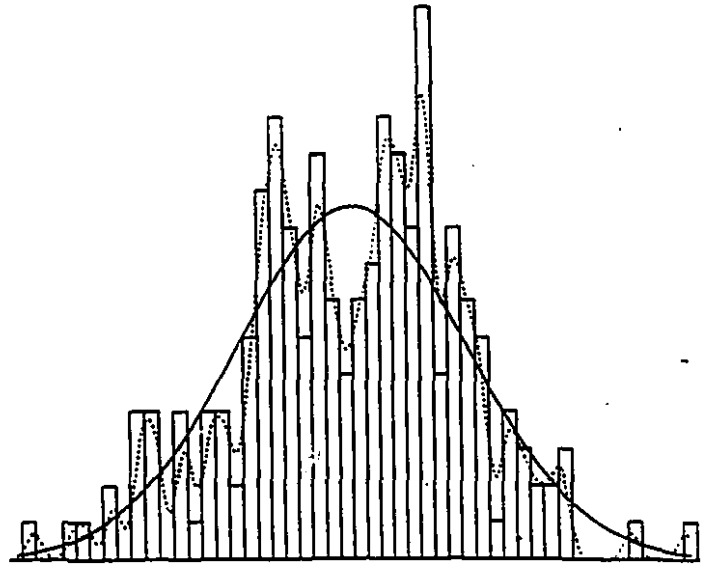

**Figure 3.3:** Histogram estimate, semi-continuous distribution created from the histogram (dashed line), and the true Gaussian density (solid line).

Once again, a compromise may be offered by semi-continuous HMMs. Let us create a codebook of "tied" densities by associating a one-dimensional Gaussian density to each of the 50 histogram bins, using the center of the bin as its mean and $width^2/3$ as its variance (*width* is the bin width). The variance was determined by assuming a uniform distribution within each bin. Then, a "semi-continuous" mixture distribution can be created from these densities by using as mixture weights the histogram heights normalized so that they sum to unity. The distribution, is illustrated in Figure 3.3, is a smoothed version of the histogram estimate.

For both discrete and continuous distributions, insufficient training data will create problems. They are considered in the next two sections.

### 3.3.2. Sparse training data and discrete HMMs

Insufficient training data with discrete HMMs will of course result in poorly estimated probabilities. A more important problem, however, is that a codeword probability may evaluate to zero, which can cause unrecoverable errors if that codeword appears during recognition. A simple solution to this problem,

called *floor smoothing*, is to constrain the discrete probabilities to be greater than or equal to some small constant $\epsilon$ [RABI 83b]. This simple constraint can dramatically improve recognition rates. Moreover, it has been found [RABI 83b] that the performance obtained is not really sensitive to the actual value of $\epsilon$ used within the range $10^{-10} \leq \epsilon \leq 10^{-3}$. Note that, as we might expect, experiments have also shown [LEEK 89a] the technique to become less useful as the training set size increases.

One problem with floor smoothing is that it doesn't distinguish between codewords which have a low probability because of a lack of training data and codewords which actually are very improbable. One solution to this problem, proposed by Lee [LEEK 89a], is a different technique called *co-occurrence smoothing*. The idea is to compute a co-occurrence probability matrix which, for any pair of codewords $(i, j)$, can be interpreted as the probability $CP(i \mid j)$ of generating codeword $i$ if codeword $j$ is generated by the same output distribution. This matrix is computed during the standard Baum-Welch training as

$$CP(i \mid j) = \frac{\sum_b \hat{p}(i \mid b) \sum_r \sum_{(t \mid b_t = b)} \sum_{(l \mid y_l^r = j)} \frac{P_{\Theta,l}(t, y^r \mid m_r)}{P_\Theta(y^r \mid m_r)}}{\sum_{k=0}^{K-1} \sum_b \hat{p}(k \mid b) \sum_r \sum_{(t \mid b_t = b)} \sum_{(l \mid y_l^r = j)} \frac{P_{\Theta,l}(t, y^r \mid m_r)}{P_\Theta(y^r \mid m_r)}}, \tag{3.4}$$

where $K$ is the codebook size and $\hat{p}(k \mid b)$ is given by (2.32). The smoothed probabilities $\hat{p}_s(k \mid b)$ are then obtained from the unsmoothed ones using

$$\hat{p}_s(k \mid b) = \sum_{k'=0}^{K-1} CP(k \mid k') \hat{p}(k' \mid b) . \tag{3.5}$$

Co-occurrence smoothing may produce distributions so smoothed that they have lost much of their discrimination capabilities. To avoid this problem, it may be advantageous to average the smoothed and unsmoothed probabilities using

$$\hat{p}_f(k \mid b) = \lambda_b \hat{p}(k \mid b) + (1 - \lambda_b) \hat{p}_s(k \mid b) , \tag{3.6}$$

where $\hat{p}_f(k|b)$ is the final probability and $\lambda_b$, which is a function of the distribution, may be estimated using deleted interpolation [LEEK 88]. In phoneme recognition experiments, Lee [LEEK 89a] found co-occurrence smoothing to be superior to floor smoothing. Also, once again, the smoothing usefulness diminished as the training set size increased.

### 3.3.3. Sparse training data and continuous HMMs

Experience shows that in most cases, the "true" continuous distributions are quite different from single Gaussian densities.[3] In order to accurately model these distributions, it thus becomes necessary to use mixtures of Gaussian densities. It is not uncommon [DENG 90, LEEC 90a] to see mixtures with more than 20 components. If there is insufficient data to train all Gaussian densities, some of the variances may become very small or even go to zero, which is undesirable. Some proposed solutions to this problem include constraining variances to be greater than a certain minimum value (covariance clipping) [LEEC 90a], tying of covariance matrices in order to provide more training data per matrix [DENG 90] or simply keeping the variances fixed, as in Richter densities [BROW 87].

### 3.3.4. Units

When designing an HMM-based speech recognition system, one of the first things to do is to determine the set of units to use. The best set for a given application depends on a number of factors such as the vocabulary size, the target recognition rate, the amount of training data that will be available, the availability of manually segmented data and so on.

#### 3.3.4.1. Linguistically based units

In certain applications such as some systems for the telephone network [WILP 90], where vocabularies are small, the environment is relatively uniform

[3] This is especially true for speaker-independent models.

and speaker-independence is necessary, the best solution is probably to collect as large a database as possible and to use word models.

In several applications, however, this is not possible. For example, in large-vocabulary systems, it is just not possible to create and train a different model for each word, so subword models must usually be used. The first problem is to find an appropriate speech unit [LEEK 88]. Most such systems use some kind of phoneme-based units. This is a natural choice since there are only around 40 phonemes in English, from which all words can be built. Collecting enough data from one speaker to train those 40 phonemes is usually not a problem. The problem is that a phoneme is actually a very abstract unit and its realization is highly variable, especially in different contexts. This means that, to attain an acceptable level of performance, it is necessary to create different models for different contexts. A common context-dependent unit model is the triphone [SCHW 85], which takes the left and right phonetic contexts into account. Unfortunately, there are a lot of triphones. For example, the DARPA resource management task, which has a 997-word vocabulary, has 2381 different within-word triphones [LEEK 88]. A very large quantity of training data would be required to accurately train so many models.

In his discussion on units, Lee [LEEK 88] listed three important properties that can be attributed to them. The first is *sensitivity*. It measures the level of recognition performance that can be expected from a certain type of unit. For example, context-dependent units, by providing finer modeling, have inherently better discrimination capabilities than context-independent ones. They are thus said to be more sensitive.

The second property is *trainability*. It measures how difficult it is to collect enough data for accurate training of the units. For example, context-independent units like phonemes are much more trainable than context-dependent ones. The third property is *sharability*. It refers to one of the common solutions to the problem of insufficient training data, namely the smoothing of distributions. The rough idea is that since some units are trainable but not very sensitive, and other units are sensitive but not very trainable, we might find a compromise by smoothing the two types of units. This is only possible if the two units smoothed represent the same thing (as, for example, is the case with a triphone model and the corresponding phoneme model) and

if discrete distributions are used (this includes semi-continuous HMMs). This is what sharability means.

### 3.3.4.2. Acoustically based units

So far, we have mostly talked about linguistically based units. However, as we have discussed, the acoustic realization of these units is often quite variable. Even expert phoneticians often disagree about the phonemic identity of a given sound. It may be argued that, in order to obtain better performance, the units used should be based not on abstract and subjective linguistic concepts, but, rather, on acoustical evidence.

One such acoustical unit that was recently proposed by the IBM group is called the *fenone* [BAHL 88a]. They proposed a way to automatically construct the acoustic baseform of a word in terms of fenones. The idea is to associate one fenone to each label in the VQ alphabet and to use a sequence of VQ labels from the pronunciation of a word to determine the fenonic baseform corresponding to that word. This fenonic baseform, determined from only one repetition of a word is called a *singleton baseform*. Since different repetitions of a word usually result in different baseforms, the singleton baseform of a word may not be the most optimal one for that word. The solution they proposed for that problem is to first use singleton baseforms to train the fenones and then, using several utterances of a word to determine the best fenone sequence (or baseform) for that word. This can be done using a search algorithm such as the Stack algorithm [JELI 76], but modified to search over several observation sequences instead of one.

Their argument for introducing these units was based on their observation that, if enough training data is available, word models usually outperform phone-like units. They argued that word models provide much finer coarticulation modeling and they expressed doubts that phone-like units could be made to perform as well. Isolated word, speaker-dependent recognition experiments demonstrated substantial performance improvements over phonetically based models. It is not clear, however, how fenones with word baseforms would perform in continuous speech. Indeed, even though fenones undoubtedly provide fine within-word coarticulation modeling, it is not

obvious how word baseforms could be made to take into account between-word coarticulation effects in continuous speech. A compromise between the acoustical and phonetic approaches has recently been proposed [BAHL 91] by the IBM group as a possible solution to that problem.

### 3.3.5. Using less training data

The requirements of a particular application will usually dictate the choice of unit and the training procedure. If speaker-independence is required, then not only will collection of a large database be necessary, but there will be a performance penalty over a speaker-dependent system.[4] However, any user will afterward be able to use the system without any prior training. On the other hand, some applications may require a speaker-dependent system in order to obtain the desired level of recognition performance. In that case, the user will have to train the system to his/her own voice prior to using it, which may be a lengthy process. Since in general, there is never enough training data, it is important to discuss some of the solutions that have been used to circumvent that problem.

#### 3.3.5.1. Speaker adaptation

Speaker adaptation is a growing area of speech recognition. As the name implies, the purpose of speaker adaptation is to improve the recognition performance of a system by somehow adapting it to the particularities of a speaker. There are several ways in which this can be done.

One of the techniques used is called *HMM clustering*. The idea is that, even though all speakers are different, it should be possible to separate them into a number of groups (or clusters) of speakers sharing common characteristics. For each cluster, a complete set of HMMs is trained. Adaptation takes place during recognition when the system determines which set of models (or cluster) is the most appropriate to use with the given speaker. The clusters can

[4] It is typical to see error rates two to three times higher in speaker-independent systems than in speaker dependent systems.

be determined *a priori* (e.g., by separating male and female speakers [DODD 89]), or they can be determined by automatic techniques. Standard clustering techniques, such as those used for designing vector quantization codebooks [RABI 83a, GRAY 84] [MAKH 85, EQUI 89], are usually used. Some [RABI 89a] start with one cluster, then apply a splitting technique until the desired number of clusters is reached. Others [LEEK 88] start with as many clusters as there are speakers and repeatedly merge them, also until the desired number of clusters is reached. At recognition time, determination of the best cluster can be done once, by finding the cluster resulting in the highest likelihood on a known utterance [LEEK 88], or it can be determined for every unknown utterance by trying every cluster and using the one resulting in the highest likelihood [DODD 89].

There is a different type of speaker adaptation which is more related to the training data problem. The general idea is that if well-trained models for a given application are available (either speaker-dependent or independent), then it should be possible to adapt those models to a particular speaker using much less training data than would be required to train the models from scratch. Adaptation can be supervised or unsupervised. Supervised adaptation uses a small number of known words or sentences from the target speaker, which makes it like regular training, but with less data. By adapting on unknown sentences, unsupervised adaptation is a potentially more flexible technique; however, it is also much more difficult.

Supervised adaptation is the technique that has so far given the best results. One of the early pioneers in this field is BBN. For a number of years, BBN's approach has been to adapt discrete probabilities from the well-trained models of one reference speaker to a target speaker. The adaptation was done by computing a *probabilistic spectral mapping* matrix that is used to transform the reference models into the target models. This matrix was evaluated either using maximum likelihood estimation directly within the Baum-Welch algorithm [SCHW 87] (in which case an initial estimate is necessary), or by using DTW to align the adaptation speech to the same text spoken by the reference speaker, and using this alignment to compute the mapping matrix [FENG 88]. This second technique is not only supervised, but also text-dependent, that is, the adaptation script must be drawn from text also spoken by the reference speaker. BBN reports that the second technique works better, even though

results vary significantly across speakers. Additional improvements were proposed in [FENG 89]. These include using a phoneme-dependent transformation, silence modeling, duration normalization and spectral space normalization. By applying those techniques to the DARPA resource management task, BBN has achieved recognition rates equivalent to the best speaker-independent results. More recently [KUBA 90], they have proposed an adaptation scheme using a reference model trained on a number of speakers normalized to a single prototype space.

AT&T has recently proposed a Bayesian framework for doing supervised speaker adaptation of continuous density HMM parameters [LEEC 90b, GAUV 91]. This is an interesting idea because it is possible to get meaningful prior distributions by either looking at a number of speaker-dependent models or by using all the mixture components from a speaker-independent model, thus taking advantage of the information gained by observing how the parameters vary across speakers.

Note that, in general, techniques developed for either discrete or continuous distributions can also be applied to semi-continuous HMMs. In fact, the mixed nature of semi-continuous HMMs opens interesting new possibilities [RTIS 89]. For example, their discrete distributions (mixture weights) could be used as prior information in order to adapt the tied mixtures to a new speaker.

### 3.3.5.2. Noise adaptation/signal normalization

In order to get the best possible results, the conditions under which a speech recognition system is used usually need to be as similar as possible to those that existed during training. Changes in background noise characteristics, or even a simple change of microphone, can result in substantially deteriorated results. In some applications, this can be a real problem since it may not be possible to have similar conditions during training and recognition. One example is when the recognition environment changes constantly.

The term *noise adaptation* refers to the techniques that deal with changes in background noise characteristics. Most of them use a silence/speech

discriminator to identify the parts in the signal where no speech is present, which are then used to determine the noise characteristics. Some techniques, such as *spectral subtraction* [VANC 89, ACER 90] or *probabilistic vector mapping* [GISH 90], work directly at the signal processing level by transforming the noisy feature vectors into approximately noise-free vectors. Others perform noise adaptation directly on the vector quantizer. One example is [NADA 89], in which a maximum likelihood estimation of the noise-free labels is performed. This last technique, however, only works if the parameters used are filter-bank energies, and only if the background noise is relatively constant.

The term *signal normalization* refers to the techniques that deal with changes in the acoustical environment. Various factors such as room acoustics and microphones can, by modifying the system's overall transfer function, color the speech's power spectrum to the point where degraded recognition results. Signal normalization thus attempts to estimate how the signal was modified with respect to the training data in order to "undo" those modifications [VANC 89, ACER 90].

## 3.4. Speech decoding

It was mentioned in Chapter 2 that the optimum speech decoder in the sense of minimizing the probability of error is the MAP decoder, which chooses $\hat{w}$ such that

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, P(w\,|y)\,. \tag{3.7}$$

Since $P(w\,|y)$ is unknown, however, the HMM-based estimate must be used instead, which means that we must use the sub-optimal decoder

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, P_\Theta(m_w\,|y) = \underset{w}{\operatorname{argmax}}\, \frac{P_\Theta(y\,|m_w)P(w)}{P(y)}$$

$$= \underset{w}{\text{argmax}}\, P_\Theta(y \,|\, m_w) P(w) \,, \tag{3.8}$$

where the last equality results because the maximization does not depend on $P(y)$. Assuming that the HMMs have been trained and that there exists a language model that can compute $P(w)$ for any $w$, then everything is available to find the (sub-optimal) $\hat{w}$. Unfortunately, except in the simplest applications (such as small-vocabulary, isolated word recognition), even (3.8) cannot really be used in practice. Consider, for example, even a simple application such as connected digit recognition, which has an 11-word vocabulary (including "oh" and "zero"). If we were to use (3.8) to find the most probable 7-digit string, we would have to compute $P_\Theta(y \,|\, m_w)$ for a total of 19,487,171 different models $m_w$ corresponding to all possible digit strings $w$.

It is clear, then, that approximations to (3.8) must be used in order to be able to perform speech decoding much more efficiently. Such an approximation is provided by the Viterbi algorithm.

### 3.4.1. The Viterbi algorithm

The Viterbi algorithm was introduced in 1967 as a maximum likelihood decoding technique for convolutional codes [VITE 67]. It is a very general algorithm which is used to find the lowest-cost path in a trellis, where the cost of a path at a given trellis node $n_j$ can be computed as the sum of the cost at the previous node $n_{j-1}$ and the cost incurred to get from node $n_{j-1}$ to node $n_j$. The idea of the algorithm is quite simple.

Let us define the cost $C(t \,|\, m)$ of a path $t$ in a HMM as minus the *a posteriori* log-likelihood of the path, that is,

$$C(t \,|\, m) = -\log P_\Theta(t \,|\, m) P_\Theta(y \,|\, t) \,. \tag{3.9}$$

Let $C_l(i)$ be the cost of the lowest-cost path ending in state $i$ at time $l$. Let $c_l(t,i)$ be the cost of going from state $l_t$ to state $i$ at time $l$, using transition $t$

(this assumes $r_t = i$). If $t$ is empty, then it comes from time $l$ and $c_l(t,i) = -\log q_t$. If $t$ is full, then it comes from time $l-1$ and $c_l(t,i) = -\log q_t b_t(y_l)$. $C_l(i)$ can be computed as

$$C_l(i) = \max \left\{ \max_{t \text{ empty}} \left[ C_l(l_t) + c_l(t,i) \right], \max_{t \text{ full}} \left[ C_{l-1}(l_t) + c_l(t,i) \right] \right\}. \qquad (3.10)$$

The lowest-cost path is the one with highest probability and, using the Viterbi algorithm, it can be found as

$$\max_t C(t|m) = \max_t \left[ -\log P_\Theta(t|m) P_\Theta(y|t) \right] = C_{L_y}(F), \qquad (3.11)$$

where we have assumed $C_0(0)=0$. A trellis recursively computed with (3.10) is called a *Viterbi trellis*. Note that isolated word recognition systems often use $C_{L_y}(F)$ instead of $P_\Theta(y|m)$ to find the best word. This is because in practice it is faster to compute $C_{L_y}(F)$ and the recognition rate obtained is basically the same.

However, the real advantage of the Viterbi algorithm for speech recognition lies not in what it computes, but in its ability to find the best path in a model. This can be done trivially by keeping, within the recursion (3.10), a "back-pointer" $B_l(i)$ to the transition that resulted in the best path and then, using these backpointers to traceback the path from $C_{L_y}(F)$ to $C_0(0)$.

For example, if a word model is made from the concatenation of phoneme models, then the best path in the model can be used to segment an utterance of that word into its individual phonemes. Note that even though the path used is the most likely path in the model, the phoneme sequence given by the path may not be the most likely sequence. This is because the probability of a phoneme sequence must be computed from the sum over all paths in the model corresponding to the sequence (see (2.4)), and not only the most likely path. In practice, however, this rarely makes a big difference.

Similarly, it is usually possible to build a compact general model $m_{gen}$ such

that every path $t$ in every model $m$ in the application is also a legal path in $m_{gen}$.[5] If, conversely, every path in $m_{gen}$ also corresponds to a path in a possible model $m$ in the application, then $m_{gen}$ is said not to over-generate. Using the Viterbi algorithm to find the best path in $m_{gen}$ effectively segments an acoustic sequence $y$ into a "recognized" word sequence. This technique is called *Viterbi decoding*. For example, connected digit recognition is often done using Viterbi decoding with a looped model such as the one in Figure 3.4.

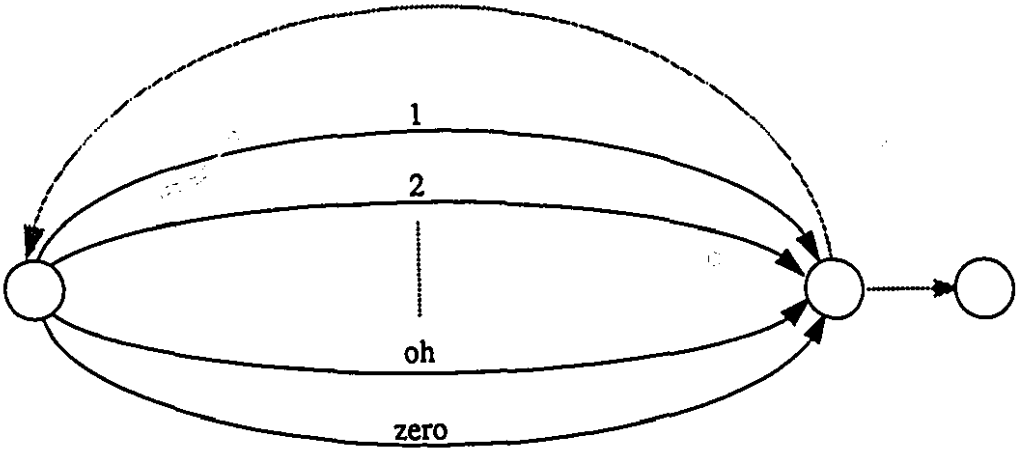


**Figure 3.4:** A looped model used for connected digit recognition

Once again, however, the most likely path will not necessarily produce the most likely word sequence. This is especially true because there is an implicit language model used in the recognition process and this model may be quite different from the true $P(w)$. Remember that the most likely word sequence $w$ is the one that maximizes $P_\Theta(y|m_w)P(w)$. Referring to the looped model in Figure 3.4 and assuming that all transitions from the same node in the word network are equiprobable, we have that $P(w) = 0.5 \cdot (1/11)^n$, where $w$ is the sequence found and $n$ is the number of digits in $w$. Thus, in this implicit language model, the *a priori* probability of a digit string decreases exponentially with the number of digits. More generally, since the implicit language model in a word network arises from word transition probabilities in the

[5] See Chapter 4 for more details.

network, it will be of the form

$$P_{imp}(\boldsymbol{w}) = P(w_1)P(w_2|w_1)\cdots P(w_n|w_{n-1}) , \qquad (3.12)$$

where $w_i$ is the $i$th word in $\boldsymbol{w}$ and $n$ is the number of words in $\boldsymbol{w}$. Such a language model is usually referred to as *bigram* language model.[6] In practice, the acoustic probabilities weigh so heavily in the determination of the best path that the implicit language model has a very limited effect on the recognition process. This means that if the language model is to be useful[7] its contribution to the sentence log likelihood must be increased.[8] For simple applications such as connected digit recognition, a negligible contribution from $P_{imp}(\boldsymbol{w})$ is just as well since, in general, (3.12) will not provide a useful approximation to the "true" language model. For larger applications, however, the use of a good language model becomes essential, which means that more sophisticated search techniques will have to be used.

### 3.4.2. Viterbi search with partial backtrace

One problem with the Viterbi decoding technique described in the previous section is that if it is applied to the recognition of a complete sentence, no word can be recognized before the end of the sentence. For recognition of short sentences, this may not be much of a problem. For some applications, however, this can introduce unacceptable delays. For example, some HMM-based wordspotting systems [ROHL 89, GISH 90, ROSE 90] use looped models of the type illustrated in Figure 3.5 to recognized keywords in

[6]A special case of bigram language model is the *word-pair* model, in which all probabilities are either 0 or 1. Clearly, the complex relationships between words in natural language cannot be reduced to simple bigram probabilities. Nevertheless, bigram language models can be quite effective in many simpler applications.

[7]The potential usefulness of a language model is usually expressed in terms of the reduction in *perplexity* that it allows. The perplexity is a measure of performance of a language model on a given text $w$ [KUHN 90]. It is given by $S(w) = P(w)^{-1/n}$, where $n$ is the number of words in $w$. Roughly speaking, if the perplexity of a language model is $S$, then the speech recognition task is as difficult as it would be if, at any time, $S$ words were equally probable.

[8]For example, in the Lincoln Lab system [PAUL 91], the best recognition rate on the resource management task is obtained when the contribution of the bigram language model to the log likelihood is multiplied by a factor of around 5.

unconstrained speech.[9]

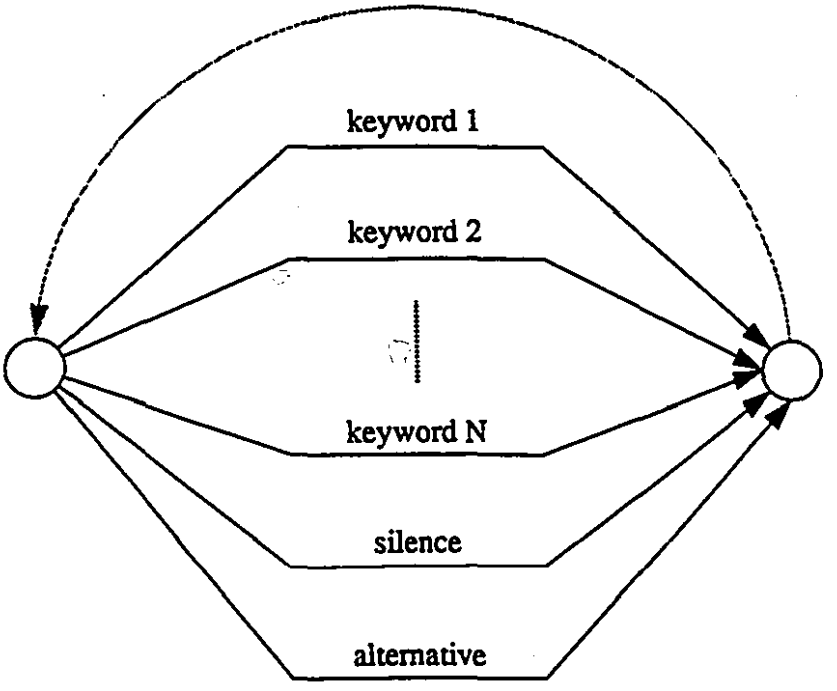


**Figure 3.5:** Looped model for wordspotting.

If Viterbi decoding is used, then a keyword will be reported spotted when the path goes through the corresponding keyword model. The problem is that wordspotting is often applied to non-stop signals which are often several minutes in duration. It does not make much sense to wait until the end of the input signal to start backtracing the best path in the Viterbi trellis; fortunately, this is usually not necessary.

Using a technique called *partial backtrace* [ROSE 90], it is possible to report recognized words as the Viterbi trellis is being computed. The idea is that when the Viterbi algorithm is applied to looped models such as the one in Figure 3.5, there will eventually be a time $l$ such that the paths backtraced from all trellis nodes at that time converge to a common node somewhere in the past. Thus, the path backtraced from that node is common to all active paths at time $l$ and all words on that path can be reported. For wordspotting

[9] In the looped model, the *alternative model* is used to model any non-keyword speech.

application, this partial backtrace resulted in recognition delays usually not exceeding 2 or 3 seconds [ROSE 90].

### 3.4.3. Beam search

When computation time is important, it is possible to make the Viterbi search faster by restricting it to the trellis nodes having a likelihood greater than some fraction of the maximum likelihood in the given column. This is called a *beam search*. Each time a trellis column $l$ is computed, the value $P_l^{\max}$ of the highest log-likelihood of any node in the column is found. Then, only the nodes with a log-likelihood greater than $P_l^{\max} - \Delta$ will be kept in the list of active nodes (the other nodes are *pruned*).[10] The value $\Delta$ is called the *beam width*. The smaller the width, the faster the program will run. In large vocabulary speech recognition experiments, Lee [LEEC 90a] observed that the computation time increased almost linearly with the beam width $\Delta$.[11]

In practice, however, a narrow beam will cause good paths to be eliminated because of poor local acoustic match, which will result in deteriorated recognition rates. As Lee points out, when the overall acoustic match is poor, a larger beam width should be used in order to allow the confusion to be resolved later on in the search. This suggests using a search algorithm with a variable beam width, although it is not clear exactly how this should be done.

### 3.4.4. Language models and N-best algorithms

As today's applications become more and more ambitious (very large vocabularies, continuous speech), it becomes quite difficult to have good recognition based on acoustic information alone. Examples of factors limiting recognition include acoustically similar words (not to mention homonyms), and word deletions and insertions in continuous speech (especially small words). Many of

[10]The pruning strategy causes the search to become suboptimal (the maximum likelihood path may not be found) and we say that it is not *admissible*.

[11]This experimental result cannot be generalized to all applications. Other researchers have found computation time to increase more nearly exponentially with $\Delta$.

the errors will result in recognized sentences which are either syntactically incorrect (even for speech) or meaningless, both of which could be detected if an appropriate language model were available.

If recognition is performed using a frame-synchronous search on a looped model, it will not, in general, be possible to take advantage of such a language model during the search. An alternative solution, however, is to generate the N most acoustically probable sentences (word sequences) and to choose among them using the language model. The number N should be chosen such that the true word sequence will be among the first N choices with high probability.[12] What is needed, then, is an algorithm that can find these N best word sequences (the Viterbi algorithm is not suited for this task).

Two such algorithms have been introduced recently. The first one [SCHW 90] is an exact algorithm (it uses all paths corresponding to the given word sequence) in which N has to be decided *a priori*. The second one [SOON 90], called the *tree-trellis search algorithm* is more computationally efficient. It uses a modified Viterbi algorithm to generate exact heuristics that will be used in a backward A* algorithm. The number N doesn't have to be decided a *priori*; as many sentences as desired can be generated during the backward A* pass; however, it only uses the most likely path in each word sequence, so it is not an exact algorithm.

### 3.4.5. The A* search algorithm

The A* algorithm is a depth-first search algorithm borrowed from the field of artificial intelligence. In speech recognition, the A* algorithm is often called the *stack algorithm* [JELI 76]. It is becoming popular in speech recognition because it does not have the language model integration problems of most frame-synchronous algorithms. However, it is more difficult to apply since partial paths of different lengths must be compared, which means that a func-

[12]This is only necessary in applications, such as dictation, where it is important that *all* words be recognized correctly. For many applications, such as database queries or dialogue systems [ZUE 90], it is only necessary to recognize a number of words sufficient to understand the whole sentence.

tion estimating the cost until the end of the sentence must be available.[13] The tree-trellis search algorithm uses an exact function that was pre-computed by a modified Viterbi search.[14] However, this only applies to the acoustics. A search using a language model would still need an evaluation function for the language model.

## 3.5. Implementation considerations

### 3.5.1. Underflow problems

Looking at equation (2.7) for the forward trellis computation, we realize that $\alpha_l(i)$ decreases geometrically as $l$ increases. The same is true for the backward trellis computation $\beta_l(i)$ as $l$ decreases. For any practical speech application, this will eventually create underflow problems, that is, trellis values will become smaller than the smallest floating point value on the given computer. This problem is usually solved either by scaling or by representing probabilities by their logarithm.

#### 3.5.1.1. Scaling

The scaling procedure [LEVI 83, RABI 89c] is applied as follows. Column-by-column, the forward trellis is computed as described in (2.7). The result of this computation is denoted $\tilde{\alpha}_l(i)$, with $\tilde{\alpha}_0(i) = \alpha_0(i)$. However, after computation of each column, a scaling factor $c_l$ for that column is computed as

$$c_l = \sum_{i=0}^{F} \tilde{\alpha}_l(i) , \tag{3.13}$$

where $F$ is the final state. Then, each element of this column is divided by the scaling factor, resulting in

[13] If the estimated cost is always smaller than or equal to the real cost, then the A* search is admissible.

[14] In that case, the Viterbi search is done forward and the A* search is done backward. The op-

$$\bar{\alpha}_l(i) = \frac{\tilde{\alpha}_l(i)}{c_l} = \frac{\tilde{\alpha}_l(i)}{\sum_{i'=0}^{F} \tilde{\alpha}_l(i')}, \tag{3.14}$$

which may be written in final form as

$$\bar{\alpha}_l(i) = \frac{\alpha_l(i)}{\prod_{j=0}^{l} c_j} = \frac{\alpha_l(i)}{\sum_{i'=0}^{F} \alpha_l(i')}, \tag{3.15}$$

where $\alpha_l(i)$ is the original unscaled trellis. The quantity $\bar{\alpha}_l(i)$ can be interpreted as the *a posteriori* probability of being in state $i$ at time $l$, given the observation sequence. This probability may be useful in wordspotting applications [ROHL 89].

The same scaling is done with the backward trellis $\beta_l(i)$, but this time we use the scaling coefficients $c_l$ calculated in the forward trellis. Using $\bar{\alpha}_l(i)$ and $\bar{\beta}_l(i)$ instead of $\alpha_l(i)$ and $\beta_l(i)$ in (2.10), we obtain

$$P_{\Theta,l}(t,y\,|m) = \begin{cases} c_l\, C\bar{\alpha}_l(l_t)q_t\,\bar{\beta}_l(r_t), & \text{if } t \text{ empty} \\ C\bar{\alpha}_{l-1}(l_t)q_t\,b_t(y_l)\bar{\beta}_l(r_t), & \text{if } t \text{ full,} \end{cases} \tag{3.16}$$

where

$$C = \prod_{l=0}^{L_y} c_l = \frac{\alpha_{L_y}(F)}{\bar{\alpha}_{L_y}(F)} = \frac{P_\Theta(y\,|m)}{\bar{\alpha}_{L_y}(F)} \tag{3.17}$$

From (3.16) and (3.17), we can now compute $P_{\Theta,l}(t,y\,|m)/P_\Theta(y\,|m)$ as

---

posite is also possible.

$$\frac{P_{\Theta,l}(t,y\,|m)}{P_{\Theta}(y\,|m)} = \begin{cases} \dfrac{c_l\,\bar{\alpha}_l(l_t)q_t\,\bar{\beta}_l(r_t)}{\bar{\alpha}_{L_y}(F)}, & \text{if } t \text{ empty} \\ \dfrac{\bar{\alpha}_{l-1}(l_t)q_t\,b_t(y_l)\bar{\beta}_l(r_t)}{\bar{\alpha}_{L_y}(F)}, & \text{if } t \text{ full.} \end{cases} \tag{3.18}$$

This scaling process should bring the probabilities of a given column to within the dynamic range of the computer. In practice, scaling need not be done at every time instant (or for every column). Computing time can therefore be saved by performing scaling only when necessary, leaving the scaling coefficients for the unscaled columns to 1.0. Note, however, that since the backward trellis uses the scaling coefficients from the forward trellis, it is possible to have underflow problems in the backward trellis even if the forward trellis is scaled properly. This means that scaling should be done more often than is required by the forward trellis.

### 3.5.1.2. Logarithmic probabilities

In some cases, especially with continuous HMMs, the dynamic range of probabilities within a given trellis column can exceed the range of floating point numbers in a given computer. This means that there will be underflow problems, regardless of scaling. In that case, one solution [BROW 87] is to use logarithmic probabilities throughout the trellis computations. One immediate consequence is that multiplications become additions, which, if anything, is an advantage. Handling additions is slightly more complicated. Let $p_1$ and $p_2$ be probabilities, with $p_1 \geq p_2$ and let $u$ be the logarithm base. The problem of computing $\log_u(p_1+p_2)$ from $\log_u p_1$ and $\log_u p_2$ can be solved with the following expression:

$$\log_u(p_1+p_2) = \log_u p_1 + \log_u(1 + u^{\log_u p_2 - \log_u p_1}) \tag{3.19}$$

Note that if $\log_u p_2 - \log_u p_1$ goes below a certain value (which depends on the number of significant digits available), then $u^{\log_u p_2 - \log_u p_1}$ becomes negligible compared to 1 and the expression becomes simply $\log_u(p_1+p_2) \approx \log_u p_1$.

In practice, logarithms are usually expressed as integers, which allows all computations to be performed using integer arithmetic. Thus, if $\tilde{p}$ is the integer logarithmic representation of $p$, then

$$\tilde{p} = \left\lfloor \log_u p + 0.5 \right\rfloor, \tag{3.20}$$

where $\lfloor x \rfloor$ means the greatest integer smaller than or equal to $x$. Since integer representation rounds the logarithm value to the nearest integer, an error of at most 0.5 is introduced, which, if $u > 0$, means a maximum relative error of $100(u^{0.5} - 1)$ percent. In our experiments, we usually take $u = 1.001$, which gives a maximum relative error of around 0.05%, or about 3.5 significant digits. Also, we use integer values from -134217727 (our value for $\log_u 0$, or $-\infty$) to 0, which gives a dynamic range of over 58000 orders of magnitude.

Integer arithmetic also allows the computation of $\log_u (1 + u^{\log_u p_2 - \log_u p_1})$ to be implemented as a table look-up, using $\log_u p_1 - \log_u p_2$ as index. Thus, the computation of $\log_u (p_1+p_2)$ from $\log_u p_1$ and $\log_u p_2$ can be implemented with at most two comparisons, one integer subtraction, one table look-up and one integer addition.

Logarithmic probabilities can offer a much increased dynamic range at the cost of some lost precision; however, experience shows that with HMMs, dynamic range is much more critical than precision. Moreover, for most applications, the increased dynamic range completely eliminates the need for scaling.[15] Since most of our programs are much faster with logarithm probabilities, we are now using them exclusively.

### 3.5.2. In-place computation of the trellises

The term central to most HMM-related computations is $P_{\Theta,l}(t, \mathbf{y}^r | m_r)$. It appears in all MLE reestimation formulas, in the probability derivative

[15]Some applications, which require the trellis computations to be done on very long input signals may require scaling to be performed from time to time even with logarithmic probabilities. For example, wordspotting systems often fall in that category.

computations, and in all MMIE-related computations in Chapter 4. Recall from Chapter 2 that $P_{\Theta,l}(t,\mathbf{y}^r|\mathbf{m}_r)$ is computed as

$$P_{\Theta,l}(t,\mathbf{y}^r|\mathbf{m}_r) = \begin{cases} \alpha_l(l_t)q_t\beta_l(r_t), & \text{if } t \text{ empty} \\ \alpha_{l-1}(l_t)q_t b_t(\underline{y}_l^r)\beta_l(r_t), & \text{if } t \text{ full.} \end{cases} \tag{3.21}$$

Thus, in order to compute $P_{\Theta,l}(t,\mathbf{y}^r|\mathbf{m}_r)$, both the forward and backward trellises are needed. This, however, may be a problem since the trellis matrices are typically the largest data structures in an HMM-based system. Suppose, for example, that the system can process a maximum of 5 seconds of speech at a time. At a frame period of 10 ms, this means a maximum of 500 frames of speech. Suppose, furthermore, that the largest model in a task has 200 states. Then, a trellis for this system will have a maximum size of 200×500, or 100,000 elements. This means that if 4 bytes are used per trellis element, then 400,000 bytes of memory will be required for each trellis.

Now, although in practice, the probabilities $P_{\Theta,l}(t,\mathbf{y}^r|\mathbf{m}_r)$ are usually needed for all $l$ and all $t$, their values can be computed in any desired order. This allows computations to be carried out using a single trellis matrix $M$. In order to see this, first remember that backward trellis elements $\beta_l(i)$ are computed recursively in decreasing order of column number $l$ and, within one column, in decreasing order of state number $i$.[16] This is illustrated in Figure 3.6. The idea is to first compute the forward trellis, storing $\alpha_l(i)$ in $M[l,i]$, and then to compute both the backward trellis and $P_{\Theta,l}(t,\mathbf{y}^r|\mathbf{m}_r)$ in the backward pass, storing $\beta_l(i)$ in $M[l,i]$ in the process.

[16]Remember from Chapter 2 the assumption that for any empty transition $t$, we must have $l_t < r_t$

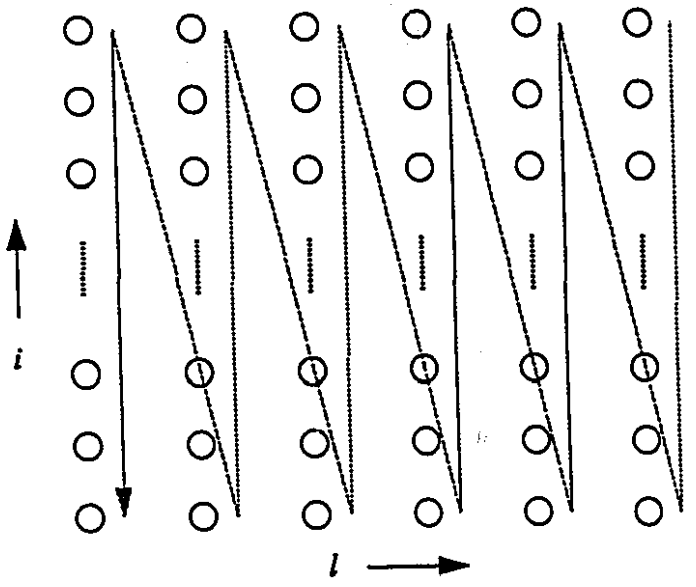


**Figure 3.6:** Order of evaluation of backward trellis elements.

Suppose that somewhere in the recursion, the backward trellis element $\beta_l(i)$ is the next one to be computed. This means that all matrix locations $M[l',i']$ such that either $l'>l$ or $l'=l$ and $i'>i$, contain backward trellis elements and all other locations (in particular, $M[l,i]=\alpha_l(i)$) contain forward trellis elements. From (3.21), we see that all values of $P_{\Theta,l}(t,\mathbf{y}^r|m_r)$ computed with $\alpha_l(i)$ can be computed at this time, after which $\beta_l(i)$ can be computed and stored in $M[l,i]$, replacing $\alpha_l(i)$ which will not be needed any more. Moving to the next backward trellis element to evaluate, the process is repeated until the last element is reached.

# 4. MAXIMUM MUTUAL INFORMATION ESTIMATION OF HMM PARAMETERS

## 4.1. Introduction

In HMM-based speech recognition, the purpose of training is to find the HMM parameter set $\Theta$ which will result in the decoder with the lowest possible recognition error rate. This is done by maximizing some objective function $R(\Theta)$.

There are two important and difficult problems to consider. The first is to determine a meaningful objective function. This function should be such that, whenever $R(\hat{\Theta}) > R(\Theta)$, then $\hat{\Theta}$ produces a better decoder than $\Theta$. Once a function $R(\Theta)$ has been chosen, the second problem (the estimation problem) is to find the parameter set $\Theta$ which maximizes it. These two problems are interrelated. An objective function is useless if it makes the estimation problem impossible. Also to be considered is the fact that a typical HMM parameter set usually has several thousands of parameters, which makes it very unlikely that a globally optimal $\Theta$ will be found. This means that even with a good function, it is possible to have poor results if the estimation procedure converges to a bad local maximum.

By far the most common HMM parameter estimation technique is MLE which we described in Chapter 2. Its most obvious quality is the existence of a reestimation formula $f(\cdot)$ such that, if $\hat{\Theta} = f(\Theta)$, then we will have $R(\hat{\Theta}) \geq R(\Theta)$, with equality only when $\Theta$ is a local maximum (or, possibly, a saddle point) of $R(\Theta)$. In practice, very few iterations are necessary to obtain the desired results (usually under 5). The existence of this reestimation formula is the main reason for the introduction of HMMs in speech recognition, and it is also largely responsible for their success and popularity. Recall from Chapter 2 that the objective function typically used in MLE is

$$R(\Theta) = \prod_r P_\Theta(y^r | m_r) , \tag{4.1}$$

where, as before, $m_r \equiv m_{w_r}$ is the model corresponding to the $r$th observation sequence $y^r$. Thus, MLE tries to increase the *a posteriori* probability of the training data, given the model corresponding to the data. The models not corresponding to the data are never taken into account. It is not intuitively obvious how (4.1) relates to the objective of reducing the error rate. A tentative explanation was offered by Nádas [NADA 83], who has shown that, if certain assumptions are met, (4.1) will, in fact, produce the best decoder. As mentioned earlier, these assumptions are, however, always violated in any practical speech recognition application.

More recently, a different type of estimation, maximum mutual information estimation (MMIE) has been proposed [BAHL 86]. The objective function used in MMIE is

$$R(\Theta) = \prod_r P_\Theta(m_r | y^r) = \prod_r \frac{P_\Theta(y^r | m_r) P(m_r)}{\sum_w P_\Theta(y^r | m_w) P(m_w)} , \tag{4.2}$$

where we have assumed that the language model $P(w)=P(m_w)$ is available and where $\sum_w$ means a sum over all possible word sequences in the application. MMIE tries to increase the *a posteriori* probability of the model corresponding to the training data, given the data. Since this is also the probability used in MAP decoding, the relationship between MMIE and error rate is much more intuitive than it is with MLE. Unfortunately, contrary to MLE, there are no known reestimation formulas with theoretically proven convergence, which means that general purpose optimization techniques such as gradient descent are usually used. This is a problem because convergence can be quite slow and each iteration of gradient descent is at least as expensive as a standard Baum-Welch iteration.

There have been attempts at empirically justifying the use of MMIE, usually using very simple experiments where all the parameters are known. Some of them [BROW 87, NADA 88] demonstrate that, for certain types of estimation

problems, MMIE will converge to the optimal decoder even if incorrect modeling assumptions are made, while MLE will not. These experiments thus tend to demonstrate that MMIE is more robust than MLE to incorrect modeling assumptions. Since most of HMMs' modeling assumptions about speech are plainly wrong, this seems to be an argument in favor of MMIE. It is, however, also possible to build simple experiments [GOPA 88] in which neither MMIE nor MLE will converge to the optimal decoder, but another type of estimator will.

It is not at all clear how the results of these simple experiments apply to practical speech recognition problems. Optimization algorithms will not, in general, converge to the global optimum, and, in any case, it is probably not possible to realize an HMM-based optimal decoder for speech recognition. In the end, the most convincing justifications for MMIE are probably going to come from experimental evidence.

There have been a number of comparisons between MLE and MMIE over the past few years. The IBM speech recognition group was the first to report results with MMIE [BAHL 86]. In their case, MMIE allowed them to reduce their error rate by 18% in a 2000-word speaker-dependent isolated word recognition system. Shortly thereafter, Brown [BROW 87] reported improvements from using MMIE for isolated word recognition of the e-set; however, he found that MMIE actually degraded results with discrete HMMs. He explained this apparent contradiction by the fact that, since discrete distributions don't make any assumptions, there is nothing to be gained from MMIE.

Merialdo [MERI 88] successfully applied MMIE to speaker-dependent phoneme recognition in continuous speech, using discrete HMMs. He accelerated gradient descent convergence by biasing the gradient expression in order to reduce emphasis from the low-valued discrete probabilities. In addition, by also using for training the looped model used for recognition, he avoided having to approximate the denominator of (4.2) by a small number of terms, as is often done in practice [BAHL 86, BROW 87]. More recently, Chow [CHOW 90] marginally improved the performance of BBN's BYBLOS system on the DARPA resource management corpus by using MMIE to estimate codebook exponents.

From this short history, it is not possible to draw definite conclusions regarding the usefulness of MMIE. This thesis will demonstrate that, in some situations, there can be real benefits from MMIE. Before we do this, however, this chapter will explain the MMIE theory and describe the estimation techniques used in the experiments.

## 4.2. Basic concepts

Let W be a random variable designating the message (words, phonemes, etc.) in some spoken speech and let Y be the random variable designating an observation sequence. Let $w$ and $y$ be possible outcomes of W and Y, respectively. Using a communications theory viewpoint [BAHL 83], we can say that a message $w$ is encoded into $y$. A measure of the average amount of uncertainty about W, given the knowledge of Y is $H(\mathrm{W}|\mathrm{Y})$, the conditional entropy of W given Y, is defined as

$$H(\mathrm{W}|\mathrm{Y}) = -\sum_{w,y} P(w,y)\log P(w|y) = -E[\log P(w|y)]\,. \tag{4.3}$$

In any speech application, the "true" $P(w|y)$ is unknown. In our case, we approximate it by an HMM-based parametric distribution $P_\Theta(m_w|y)$, where $\Theta$ is the HMM parameter vector and $m_w$ is the model corresponding to $w$. We have

$$\begin{aligned} H_\Theta(\mathrm{W}|\mathrm{Y}) &= -E[\log P_\Theta(m_w|y)] = -\sum_{w,y} P(w,y)\log P_\Theta(m_w|y) \\ &= -\sum_{w,y} P(w,y)\log \frac{P_\Theta(m_w|y)}{P(w,y)} - \sum_{w,y} P(w,y)\log P(w|y) \\ &\geq -\sum_{w,y} P(w,y)\left\{\frac{P_\Theta(m_w|y)}{P(w,y)}-1\right\} + H(\mathrm{W}|\mathrm{Y}) = H(\mathrm{W}|\mathrm{Y})\,, \end{aligned} \tag{4.4}$$

where we used the fact that $\log x \leq x-1$, with equality only when $x=1$. Thus, in (4.4), we have equality only when $P_\Theta(m_w|y) = P(w|y)$. This means that

minimizing $H_\Theta(\mathrm{W}\,|\mathrm{Y})$ attempts to make $P_\Theta(m_w\,|y)$ as similar as possible to $P(w\,|y)$. Since $P(w,y)$ is unknown, $H_\Theta(\mathrm{W}\,|\mathrm{Y})$ can only be estimated. This is done by replacing the expectation in (4.4) by the sample average

$$\hat{H}_\Theta(\mathrm{W}\,|\mathrm{Y}) = -\frac{1}{N}\sum_{r=1}^{N}\log P_\Theta(m_r\,|y^r) = -\frac{1}{N}\log\prod_{r=1}^{N}P_\Theta(m_r\,|y^r)\,, \tag{4.5}$$

where $m_r \equiv m_{w_r}$ is the model corresponding to the $r$th sequence in the training set $T=\{(w_r,y^r),r=1,\cdots,N\}$ and $N$ is the number of sequences in $T$. For historical reasons, estimation of HMM parameters aimed at minimizing $\hat{H}_\Theta(\mathrm{W}\,|\mathrm{Y})$ above is called *maximum mutual information estimation* (MMIE). To explain this, let us first define the average mutual information $I(\mathrm{W};\mathrm{Y})$ between the words spoken, W, and the corresponding observation sequence, Y, as

$$\begin{aligned} I(\mathrm{W};\mathrm{Y}) &= \sum_{w,y}P(w,y)\log\frac{P(w,y)}{P(w)P(y)} = \sum_{w,y}P(w,y)\log\frac{P(w\,|y)}{P(w)} \\ &= H(\mathrm{W}) - H(\mathrm{W}\,|\mathrm{Y})\,, \end{aligned} \tag{4.6}$$

where $H(\mathrm{W}) = \sum_{w}P(w)\log P(w)$ is the entropy of W. Assuming that $P(w)$ (the language model) is known, replacing $P(w\,|y)$ by its HMM estimate, and using sample averages instead of expectations, we obtain

$$\hat{I}_\Theta(\mathrm{W};\mathrm{Y}) = \frac{1}{N}\sum_{r=1}^{N}\log P(w_r) - \hat{H}_\Theta(\mathrm{W}\,|\mathrm{Y})\,; \tag{4.7}$$

so that maximizing $\hat{I}_\Theta(\mathrm{W};\mathrm{Y})$ with respect to $\Theta$ is equivalent to minimizing $\hat{H}_\Theta(\mathrm{W}\,|\mathrm{Y})$.

There is another interpretation of MMIE that is offered by (4.5). Assuming that all pairs $(w,y)\in T$ are independent, MMIE aims at maximizing the *a posteriori* probability of the "good models", given their observation sequences. This probability has to take all models into account, which makes MMIE much more complex than MLE. To derive a parametric expression for

$P_\Theta(m_w|y)$, we can use the probabilities as defined in Chapter 2 to obtain

$$P_\Theta(m_w|y) = \frac{P_\Theta(y|m_w)P(w)}{\sum_{w'} P_\Theta(y|m_{w'})P(w')}, \tag{4.8}$$

which, combined with (4.5), gives the objective function $R(\Theta)$ in (4.2).

It is important to realize here that, from the MMIE point of view, we are only interested in $P_\Theta(m_w|y)$, which, as expressed in (4.8), is really a *parametric family* of probability distributions, with parameter vector $\Theta$. The functions $P_\Theta(y|m_w)$ and $P(w)$ are of interest only to the extent that they are used in (4.8). In fact, they don't even have to be distributions. This is going to be important later on when modifications are introduced to the expression used to compute $P_\Theta(y|m_w)$ which will result in it not retaining the properties of a distribution.

It has always been clear that, because of the frame independence assumption inherent to the HMM formulation, $P_\Theta(y|m_w)$ is a very bad approximation to $P(y|w)$. It is hoped that $P_\Theta(m_w|y)$ in (4.8) may, in fact, be a much better approximation to $P(w|m)$.

## 4.3. MMIE in practice

For several reasons, using MMIE is much harder than using MLE. Let $R(\Theta)$ be the criterion to maximize in MMIE. Then, from (4.5) and (4.8), we have

$$R(\Theta) = \prod_r \frac{P_\Theta(y^r|m_r)P(m_r)}{\sum_w P_\Theta(y^r|m_w)P(m_w)}. \tag{4.9}$$

One thing that makes MMIE harder is the sum over $w$ in the denominator which can have a very large (but always finite, in practice) number of terms.

Let us define a model $m_{gen}$ such that $P_\Theta(y|m_{gen}) = \sum_w P_\Theta(y|m_w)P(m_w)$. It

is always theoretically possible to design such a model by having every possible model $m$ in parallel, with an initial state from which an empty transition goes to the first state of every model, and a final state to which an empty transition from the last state of every model goes. Setting the probability of the empty transition from the initial state to every model $m$ to $P(m)$ results in the desired model. Because of its sheer size, however, such a model will usually be unwieldy to use in practice.

It may be possible to use graph reduction techniques which remove redundancies to reduce the size of the model somewhat, although doing this without modifying the language model probabilities will be very difficult. In some applications (such as connected digit recognition) for which the language model is secondary, it may be possible to create a very compact model $m_{gen}$ in which there is a path corresponding to every path in every possible model $m$ in the application. One example is the looped model used for connected digit recognition (see Figure 3.4). In general, if recognition is done by applying a Viterbi search on some model $m_{gen}$, then the same model could be used for the denominator of (4.9); however, such a model could be much too big for a practical implementation of MMIE training.

If it is not possible to build a satisfactory $m_{gen}$ of a reasonable size, then the denominator of (4.9) will usually be approximated by using a much smaller number of models [BAHL 86, BROW 87, CHOW 90]. In this case, the sum should be taken over the most probable models, which can be determined using a so-called "N-best" algorithm [SCHW 90, SOON 90].

Another reason why MMIE is harder is that there are no reestimation formulas of the type derived in Section 3 for MLE. This may impose the use of general optimization techniques such as gradient descent, in which the gradient is computed as

$$\frac{\partial \log R(\Theta)}{\partial \theta} =$$

$$\sum_r \left\{ \frac{1}{P_\Theta(y^r|m_r)} \frac{\partial P_\Theta(y^r|m_r)}{\partial \theta} - \frac{\sum_m P(m) \frac{\partial P_\Theta(y^r|m)}{\partial \theta}}{\sum_m P(m) P_\Theta(y^r|m)} \right\}, \tag{4.10}$$

where $\theta$ is one of the parameters in $\Theta$ and the expressions for the partial derivatives can be obtained from Chapter 2. Using $m_{gen}$, (4.10) reduces to

$$\frac{\partial \log R(\Theta)}{\partial \theta} =$$

$$\sum_r \left\{ \frac{1}{P_\Theta(y^r|m_r)} \frac{\partial P_\Theta(y^r|m_r)}{\partial \theta} - \frac{1}{P_\Theta(y^r|m_{gen})} \frac{\partial P_\Theta(y^r|m_{gen})}{\partial \theta} \right\}. \tag{4.11}$$

### 4.3.1. A reestimation formula for discrete HMMs

Traditionally, MMIE training has been done using a gradient descent on $-\log R(\Theta)$, which, because of slow convergence, can be very time-consuming. Recently however, Gopalakrishnan *et al.* [GOPA 89] introduced the following reestimation formula for rational objective functions (such as $R(\Theta)$) associated with discrete HMMs:

$$\hat{\theta} = \frac{\theta \left( \frac{\partial \log R(\Theta)}{\partial \theta} + D \right)}{\sum_{\theta'} \theta' \left( \frac{\partial \log R(\Theta)}{\partial \theta'} + D \right)}, \tag{4.12}$$

where the sum is taken over all parameters belonging to the same distribution as $\theta$ and $D$ is a constant to be determined. Using (4.11) and (2.51), we have, for discrete output probabilities

$$\frac{\partial \log R(\Theta)}{\partial \theta} = \frac{1}{\theta}(c_\theta - c_\theta^{gen}) , \tag{4.13}$$

where $c_\theta$ represents the standard MLE count for parameter $\theta$ and $c_\theta^{gen}$ is the corresponding count obtained using the general model. That is, if $\theta$ is the parameter corresponding to $p(k|b)$, then

$$\begin{aligned} c_\theta &= \sum_r \sum_{(t|b_t=b)} \sum_{(l|y_l^r=k)} \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} , \\ c_\theta^{gen} &= \sum_r \sum_{(t|b_t=b)} \sum_{(l|y_l^r=k)} \frac{P_{\Theta,l}(t,y^r|m_{gen})}{P_\Theta(y^r|m_{gen})} . \end{aligned} \tag{4.14}$$

Let us define a quantity $\psi_{\Theta,l}(t,y^r)$ as

$$\psi_{\Theta,l}(t,y^r) = \frac{P_{\Theta,l}(t,y^r|m_r)}{P_\Theta(y^r|m_r)} - \frac{P_{\Theta,l}(t,y^r|m_{gen})}{P_\Theta(y^r|m_{gen})} . \tag{4.15}$$

Then, from (4.12) and (4.13), we obtain

$$\hat{p}(k|b) = \frac{\sum_r \sum_{(t|b_t=b)} \sum_{(l|y_l^r=k)} \psi_{\Theta,l}(t,y^r) + Dp(k|b)}{\sum_{k'} \left[ \sum_r \sum_{(t|b_t=b)} \sum_{(l|y_l^r=k')} \psi_{\Theta,l}(t,y^r) \right] + D} \tag{4.16}$$

What needs to be determined is the value of $D$. It is clear from (4.16) that the greater $D$, the less $\hat{p}(k|b)$ will differ from $p(k|b)$; thus, for fast convergence, $D$ needs to be as small as possible. Gopalakrishnan *et al.* have shown that there is a value $D(\Theta)$ such that, for any $D > D(\Theta)$, (4.16) is guaranteed to converge. However, as we shall see, $D(\Theta)$ is usually so large that using $D > D(\Theta)$ renders (4.16) practically useless. For smaller $D$, there is no theoretically proven convergence; however, Gopalakrishnan *et al.* report that using

$$D = \max_{\theta} \left\{ -\frac{\partial \log R(\Theta)}{\partial \theta}, 0 \right\} + \epsilon \,, \tag{4.17}$$

where $\epsilon$ is a small positive constant, results in fast convergence. Even though our experiments using (4.16) with (4.17) also consistently demonstrated convergence, we generally found that convergence was too slow to be useful. Following an argument of Merialdo [MERI 88], we conjectured that by removing emphasis from the low-valued parameters in the gradient vector, convergence could be improved. Merialdo had found that when a parameter $\theta$ is very small, the division by $\theta$ in (4.13) often causes the corresponding gradient coordinate to have a large magnitude. The consequence is that search is often concentrated on coordinates corresponding to very low-valued parameters, but since those values are small, they are also unreliably estimated. Merialdo argues that the search should put more emphasis on better estimated, high-valued parameters.

In his gradient descent based MMIE training experiments, Merialdo improved convergence by replacing (4.13) by

$$\frac{\partial \log R(\Theta)}{\partial \theta} \approx \frac{c_{\theta}}{\sum_{\theta' \in b(\theta)} c_{\theta'}} - \frac{c_{\theta}^{gen}}{\sum_{\theta' \in b(\theta)} c_{\theta'}^{gen}} \,, \tag{4.18}$$

where the notation $\sum_{\theta' \in b(\theta)}$ means a summation over all parameters $\theta'$ belonging to the same distribution as $\theta$. We observed similar improvement in our own MMIE training experiments with gradient descent. We naturally thought that (4.18) could also improve convergence when (4.16) is used instead of gradient descent. This proved to be indeed the case. All our experiments demonstrate that convergence, indeed, is dramatically improved.[1] We experimented with different variants of (4.18) based on the same idea and observed similar convergence behavior. In all cases, however, as $R(\Theta)$ gets near its optimum (1.0), divergence is often observed.

[1] See Chapter 5

### 4.3.2. The Corrective MMIE training algorithm

The *Corrective MMIE training* algorithm, based on the above results, is initialized with the HMMs obtained after a pre-determined number of MLE iterations. Subsequently, each iteration is a two-step process. First, recognition is performed on the training set and then reestimation is done using only those sentences that were incorrectly recognized. The set of incorrectly recognized strings is called the *reestimation set*. The aim here is to correct as many errors as possible from the training set, hoping that this will improve results on the test set.

Note that for correctly recognized sentences, the two contributions to the counts in (4.14) will be similar so that their effect will tend to cancel out, leaving most of the contribution with the incorrect ones. In practice, the results obtained by training only on errors are similar to the ones obtained by training on the full training set, but at a much lower computational cost. Reestimation is done using (4.12), (4.17) and (4.18), and the HMM parameters obtained are smoothed with the ones from the previous iteration using a weight that is dependent on the number of errors in the training set.

### 4.3.3. Extension to Gaussian densities

The reestimation formula (4.12) only applies to discrete distributions.[2] It is known [BROW 87], however, that MMIE can result in substantially improved recognition results when continuous HMMs are used. It would thus be useful to have an equivalent of (4.12) for continuous densities. This section examines this problem for the case of diagonal covariance Gaussian densities. These densities can appear by themselves; as part of a Gaussian mixture in CDHMMs; or they can appear as the tied continuous densities in SCHMMs. For this section, we consider the simplest case, that of a single diagonal Gaussian distribution. Application of the results to more complex Gaussian-based distributions is relatively straightforward.

[2]This includes transition probabilities, as well as the weights used in mixture distributions.

### 4.3.3.1. Looking for the fixed point

It is possible to gain insight into the reestimation problem by simply taking the derivative $\log R(\Theta)$ with respect to its parameters and setting them equal to zero. Let $\mu_i$ and $\sigma_i$ be parameters of a diagonal Gaussian distribution $b$. From (4.11), (2.55) and (2.56), we have

$$\frac{\partial \log R(\Theta)}{\partial \mu_i} = \sum_r \sum_{(t\,|\,b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r)\, \frac{(y_{l,i} - \mu_i)}{{\sigma_i}^2}\,, \tag{4.19}$$

$$\frac{\partial \log R(\Theta)}{\partial \sigma_i} = \sum_r \sum_{(t\,|\,b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r) \left\{ \frac{1}{\sigma_i} + \frac{(y_{l,i} - \mu_i)^2}{{\sigma_i}^3} \right\}, \tag{4.20}$$

where $y^r_{l,i}$ is the $i$th parameter from the $l$th observation vector in $\mathbf{y}^r$. Setting the derivatives to zero and solving for $\mu_i$ and ${\sigma_i}^2$, we obtain:

$$\mu_i = \frac{\sum_r \sum_{(t\,|\,b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r)\, y^r_{l,i}}{\sum_r \sum_{(t\,|\,b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r)}\,, \tag{4.21}$$

$${\sigma_i}^2 = \frac{\sum_r \sum_{(t\,|\,b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r)\, (y^r_{l,i} - \mu_i)^2}{\sum_r \sum_{(t\,|\,b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r)}\,. \tag{4.22}$$

Since the unknowns appear on both sides of (4.21) and (4.22), these equations don't offer a solution to the estimation problem. They are, however, interesting because they suggest a recursion that could be used in the estimation process. Unfortunately, not only is there no proof of convergence, but, since $\psi_{\Theta,l}(t,\mathbf{y}^r)$ can be negative, there is not even a guarantee that the variance estimate is positive.

### 4.3.3.2. A heuristic reestimation formula

A Gaussian density can be approximated with arbitrary precision by a discrete distribution. Let $b_i = N(y, \sigma_i, \mu_i)$ be the $i$th one-dimensional Gaussian density in the diagonal Gaussian density $b$. Partition the real axis (domain of the density) into three non-overlapping intervals $I_1 = (-\infty, \mu_i - \nu\sigma_i)$, $I_2 = [\mu_i - \nu\sigma_i, \mu_i + \nu\sigma_i]$ and $I_3 = (\mu_i + \nu\sigma_i, +\infty)$. Choose $\nu$ such that all points $\{y_{l,i}^r | r=1, \cdots, N, l=1, \cdots, L_r\}$ in the training data fall in the second interval $I_2$. This is always possible since the training data is available and the range of $y_{l,i}^r$ is finite. Now, let us partition $I_2$ into $M$ non-overlapping subintervals $I_{2k}$, $k=1, \ldots, M$ of width $\Delta = 2\nu\sigma_i / M$. This construction is illustrated in Figure 4.1.

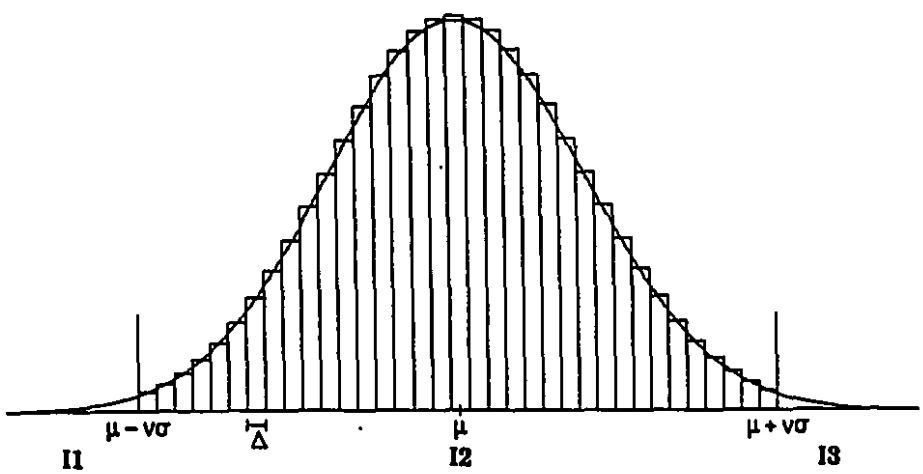


**Figure 4.1:** Construction described in the text.

Given a continuous random variable $Y_i$, we can define a discrete distribution by the $M$ probabilities $a_i(k) = P(Y_i \in I_{2k})$ and we can set these probabilities to

$$a_i(k) = \frac{N(\tilde{y}_k, \sigma_i, \mu_i)\Delta}{\sum_k N(\tilde{y}_k, \sigma_i, \mu_i)\Delta},$$

where $\tilde{y}_k$ is the mid-point of the interval $I_{2k}$. Each of these probabilities corresponds to the surface of the corresponding bin in Figure 4.1. It is easy to see that for any $y \in I_{2k}$, $\lim_{\Delta \to 0} y = \tilde{y}_k$ and

$$\lim_{\Delta \to 0} N(\tilde{y}_k, \sigma_i, \mu_i) = N(y, \sigma_i, \mu_i) \tag{4.22}$$

$$\lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \sum_k N(\tilde{y}_k, \sigma_i, \mu_i)\Delta = 1 , \tag{4.23}$$

which means that for any $k$ and any $y \in I_{2k}$, we can find $\Delta$ and $\nu$ such that $a_i(k)/\Delta$ approximates $b_i(y) = N(y, \sigma_i, \mu_i)$ with any desired precision. Let the partitions be the same for all densities and let $k(y)$ be a scalar quantizer mapping $y$ to its partition, that is, if $y \in I_{2i}$, then $k(y)=i$. If, in $P_\Theta(y^r|m)$ and $P_{\Theta,l}(t, y^r|m)$, we replace Gaussian densities $N(y, \sigma_i, \mu_i)$ by $a_i(k(y))$, we get $P_{\Theta_d}(y^r|m)$ and $P_{\Theta_d,l}(t, y^r|m)$, where $\Theta_d$ is the parameter vector of a discrete HMM. Observe that

$$\lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \frac{P_{\Theta_d,l}(t, y^r|m)}{P_{\Theta_d}(y^r|m)} = \frac{\Delta^{L_r} P_{\Theta,l}(t, y^r|m)}{\Delta^{L_r} P_\Theta(y^r|m)} = \frac{P_{\Theta,l}(t, y^r|m)}{P_\Theta(y^r|m)} , \tag{4.24}$$

$$\lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \frac{P_{\Theta_d}(y^r|m_w)P(m_w)}{\sum_{w'} P_{\Theta_d}(y^r|m_{w'})P(m_{w'})} = \frac{P_\Theta(y^r|m_w)P(m_w)}{\sum_{w'} P_\Theta(y^r|m_{w'})P(m_{w'})} . \tag{4.25}$$

This means that, in the limit, the MLE counts for the discrete HMMs are the same as the counts for the continuous ones and that $R(\Theta_d) = R(\Theta)$.

Using these observations, we can make a heuristic extension of (4.12) to the case of diagonal Gaussian densities. The idea is that, with (4.12), we can use MMIE to obtain new discrete probability estimates $\hat{a}_i(k)$, from which we can get an estimate of the new mean and variance as

$$\hat{\mu}_i = \lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \sum_k \hat{a}_i(k)\tilde{y}_k, \quad \hat{\sigma}_i^{\,2} = \lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \sum_k \hat{a}_i(k)(\tilde{y}_k - \hat{\mu}_i)^2 . \tag{4.26}$$

Since it will usually not be the case that $\hat{a}_i(k) \propto N(\bar{y}_k, \hat{\sigma}_i, \hat{\mu}_i)$ for any $\hat{\sigma}_i$ and $\hat{\mu}_i$, it is difficult to determine exactly what (4.26) computes. In fact, what we have is a discrete distribution obtained with MMIE, from which the parameters of a Gaussian density are computed using MLE. Nothing guarantees that this will in fact increase the MMIE function $R(\Theta)$. Nonetheless, it could lead to a useful expression. Observe that

$$\lim_{\Delta \to 0} \sum_k a_i(k) \bar{y}_k = \mu_i, \tag{4.27}$$

$$\lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \sum_k a_i(k) \bar{y}_k{}^2 = \sigma_i{}^2 + \mu_i{}^2. \tag{4.28}$$

Using this with (4.16) and the fact that $\lim_{\Delta \to 0} y^r_{l,i} = \bar{y}_k$, (4.26) gives

$$\hat{\mu}_i = \frac{\sum_r \sum_{(t \mid b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t, \mathbf{y}^r)\, y^r_{l,i} + D\mu_i}{\sum_r \sum_{(t \mid b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t, \mathbf{y}^r) + D} \tag{4.29}$$

and

$$\hat{\sigma}_i{}^2 = \frac{\sum_r \sum_{(t \mid b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t, \mathbf{y}^r)\, y^r_{l,i}{}^2 + D\left(\sigma_i{}^2 + \mu_i{}^2\right)}{\sum_r \sum_{(t \mid b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t, \mathbf{y}^r) + D} - \hat{\mu}_i{}^2. \tag{4.30}$$

Several observations can be made here. First note that the reestimation formulas do not depend on the actual value of $\Delta$ used in the approximation. Note also that the variance estimate is not guaranteed to be positive so $D$ must be at least as large as the minimum value that guarantees a positive estimate.

In our experiments, we start from the value for $D$ that guarantees a positive

denominator and we repeatedly double its value until all estimates (4.30) are positive. The value we use is twice this last value. We observed fast convergence in all our experiments with (4.29) and (4.30).

### 4.3.3.3. Revisiting the reestimation formula

Even though (4.29) and (4.30) experimentally exhibit good convergence properties, the heuristic derivation used in the last section is not really satisfying. This section proposes a new derivation for Gopalakrishnan's formula (4.12), which will allow a formal extension to the continuous case.

For this section, new definitions are needed. We use $\boldsymbol{t}$ to designate any sequence of transition components, regardless of whether or not the sequence could have been generated by any model in the application. For example, if there are a total of $N$ transitions components in the application's parameter vector, then there are $N^L$ such sequences of length $L$. We say that a sequence $\boldsymbol{t}$ is a *possible* sequence if it could have been generated by one of the models in the application. Otherwise it is called an *impossible* sequence.

We must also introduce the concept of a discrete *distribution component*, which refers to one of the elements of one of the discrete distributions in the application. If $K$ is the size of the discrete alphabet, then a discrete distribution has $K$ such components, each of which is associated with a different alphabet symbol. Each component $a$ belongs to one and only one discrete distribution $b$, which we express as $a \in b$. We use $b(a)$ to the designate the distribution to which $a$ belongs and $k(a)$ to designate the alphabet symbol $k$ associated with $a$. If there are $B$ output distributions in the applications using a size-$K$ alphabet, then there are $BK$ distribution components corresponding to that alphabet. The probability associated to $a$ is $p_a = p(k(a)|b(a))$. For any distribution $b$, component probabilities must respect the constraint $\sum_{a \in b} p_a = 1$. We use $\boldsymbol{a}$ to designate any sequence of distribution components. To any sequence $\boldsymbol{a}$ correspond not only a unique sequence of distributions, but also a unique observation sequence $\boldsymbol{y}$. Again, we can distinguish between possible and impossible sequences.

Finally, let:

$L_r$ be the length of $\boldsymbol{y}^r$ (and also the minimum length of any transition sequence having generated $\boldsymbol{y}^r$)

$L_r^{\max}$ be the maximum length of any possible transition sequence having generated $\boldsymbol{y}^r$

$\boldsymbol{t}^r$ be any transition sequence corresponding to $\boldsymbol{y}^r$, possible or impossible. Note that $P(\boldsymbol{t}^r|\boldsymbol{m})$ will be zero if $\boldsymbol{t}^r$ is not a possible transition sequence in $\boldsymbol{m}$.

$L_{t^r}$ be the length of $\boldsymbol{t}^r$, a transition sequence having generated $\boldsymbol{y}^r$. In what follows, we assume that $L_r \leq L_{t^r} \leq L_r^{\max}$.

$\boldsymbol{a}^r$ be any sequence of discrete distribution components corresponding to $\boldsymbol{y}^r$ (This correspondence only affects the length of the sequence. Only one such sequence is a possible sequence)

$a_l$ be the $l$th element of $\boldsymbol{a}$

Let us assume that there are $N$ repetitions, that is, $r$ can take values from 1 to $N$. We want to maximize

$$\prod_{r=1}^{N} \frac{P_{\hat{\Theta}}(\boldsymbol{y}^r|\boldsymbol{m}_r)/P_{\hat{\Theta}}(\boldsymbol{y}^r|\boldsymbol{m}_{gen})}{P_{\Theta}(\boldsymbol{y}^r|\boldsymbol{m}_r)/P_{\Theta}(\boldsymbol{y}^r|\boldsymbol{m}_{gen})} \tag{4.31}$$

with respect to the parameters of $\hat{\Theta}$. It is relatively easy to see that this is equivalent to maximizing

$$F(\hat{\Theta},\Theta) = \prod_{r=1}^{N} P_{\hat{\Theta}}(\boldsymbol{y}^r|\boldsymbol{m}_r) - \prod_{r=1}^{N} \frac{P_{\Theta}(\boldsymbol{y}^r|\boldsymbol{m}_r)}{P_{\Theta}(\boldsymbol{y}^r|\boldsymbol{m}_{gen})} P_{\hat{\Theta}}(\boldsymbol{y}^r|\boldsymbol{m}_{gen}) . \tag{4.32}$$

This is a very complex expression. In order to be able to work with it, we will have to express it differently. First, observe that since discrete distributions are normalized to unity, both $\sum_r q_r = \sum_r \hat{q}_r$ and $\sum_a p_a = \sum_a \hat{p}_a$ are constant expressions; this means that

$$A = \prod_{r=1}^{N} \left[ \left( \sum_{\tau} q_{\tau} \right)^{L_r} + \cdots + \left( \sum_{\tau} q_{\tau} \right)^{L_r^{\max}} \right] \left( \sum_{a} p_a \right)^{L_r}$$

is also a constant. Maximizing an expression with a constant added to it is the same as maximizing the expression itself. Hence, maximizing (4.32) is the same as maximizing

$$F'(\hat{\Theta},\Theta) = \prod_{r=1}^{N} P_{\hat{\Theta}}(\mathbf{y}^r|\mathbf{m}_r) - \prod_{r=1}^{N} \frac{P_{\Theta}(\mathbf{y}^r|\mathbf{m}_r)}{P_{\Theta}(\mathbf{y}^r|\mathbf{m}_{gen})} P_{\hat{\Theta}}(\mathbf{y}^r|\mathbf{m}_{gen}) + DA \ , \qquad (4.33)$$

where $D$ is a constant to be specified later. Now, note that $A$ can be rewritten as

$$A = \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \prod_{r=1}^{N} \prod_{i=1}^{L_{tr}} \hat{q}_{t_i^r} \prod_{l=1}^{L_r} \hat{p}_{a_l^r} \ , \qquad (4.34)$$

where the notation $\sum_{t^1 \cdots t^N}$ means a sum over all possible combinations of $\{t^1 \cdots t^N\}$. Let us define $\delta(\boldsymbol{t},\boldsymbol{a}|\boldsymbol{m},\boldsymbol{y})$ to be 1 if $\boldsymbol{t}$ and $\boldsymbol{a}$ are possible given both model $\boldsymbol{m}$ and output sequence $\boldsymbol{y}$, and 0 otherwise. Then, using (2.4), we have

$$\prod_{r=1}^{N} P_{\Theta}(\mathbf{y}^r|\mathbf{m}_r) = \prod_{r=1}^{N} \left\{ \sum_{t^r} \sum_{a^r} \delta(t^r,a^r|\mathbf{m}_r,\mathbf{y}^r) \prod_{i=1}^{L_t} q_{t_i} \prod_{l=1}^{L_r} p(y_l^r|b_{t_{n_l}}) \right\}$$

$$= \sum_{t^1} \cdots \sum_{t^N} \sum_{a^1} \cdots \sum_{a^N} \prod_{r=1}^{N} \delta(t^r,a^r|\mathbf{m}_r,\mathbf{y}^r) \prod_{i=1}^{L_{tr}} q_{t_i} \prod_{l=1}^{L_r} p(y_l^r|b_{t_{n_l}^r})$$

$$= \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \prod_{r=1}^{N} \delta(t^r,a^r|\mathbf{m}_r,\mathbf{y}^r) \prod_{i=1}^{L_{tr}} q_{t_i} \prod_{l=1}^{L_r} p(y_l^r|b_{t_{n_l}^r}) \ . \qquad (4.35)$$

This means that, using (4.34), (4.33) can be written

$$F'(\hat{\Theta},\Theta) = \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N}$$

$$\left\{ \prod_{r=1}^{N} \delta(t^r,a^r|m_r,y^r) - \prod_{r=1}^{N} \frac{P_\Theta(y^r|m_r)}{P_\Theta(y^r|m_{gen})} \delta(t^r,a^r|m_{gen},y^r) + D \right\} \prod_{r'=1}^{N} \prod_{i=1}^{L_{t^{r'}}} \hat{q}_{t_i^{r'}} \prod_{l=1}^{L_{r'}} \hat{p}_{a_l^{r'}}$$

$$= \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \prod_{r'=1}^{N} F'(r',t_1^N,a_1^N,\hat{\Theta},\Theta) , \tag{4.36}$$

where, in order to lighten the notation, we used $t_1^N \equiv t^1 \cdots t^N$ and $a_1^N \equiv a^1 \cdots a^N$, and where

$$F'(r',t_1^N,a_1^N,\hat{\Theta},\Theta) =$$

$$\left\{ \prod_{r=1}^{N} \delta(t^r,a^r|m_r,y^r) - \prod_{r=1}^{N} \frac{P_\Theta(y^r|m_r)}{P_\Theta(y^r|m_{gen})} \delta(t^r,a^r|m_{gen},y^r) + D \right\}^{1/N} \prod_{i=1}^{L_{t^{r'}}} \hat{q}_{t_i^{r'}} \prod_{l=1}^{L_{r'}} \hat{p}_{a_l^{r'}} .$$

(4.37)

Note that

$$F'(\Theta,\Theta) = \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \prod_{r'=1}^{N} F'(r',t_1^N,a_1^N,\Theta,\Theta) = DA . \tag{4.38}$$

If we choose $D > \prod_{r=1}^{N} P_\Theta(y^r|m_r)/P_\Theta(y^r|m_{gen})$, then $F'(r,t_1^N,a_1^N,\hat{\Theta},\Theta)$ is always positive and maximizing $F'(\hat{\Theta},\Theta)$ becomes equivalent to maximizing

$$\log \frac{F'(\hat{\Theta},\Theta)}{F'(\Theta,\Theta)} = \log \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \prod_{r'=1}^{N} \frac{F'(r',t_1^N,a_1^N,\Theta,\Theta)}{F'(\Theta,\Theta)} \frac{F'(r',t_1^N,a_1^N,\hat{\Theta},\Theta)}{F'(r',t_1^N,a_1^N,\Theta,\Theta)}$$

$$\geq \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \left( \prod_{r'=1}^{N} \frac{F'(r', t_1^N, a_1^N, \Theta, \Theta)}{F'(\Theta, \Theta)} \right) \log \left( \prod_{r'=1}^{N} \frac{F'(r', t_1^N, a_1^N, \hat{\Theta}, \Theta)}{F'(r', t_1^N, a_1^N, \Theta, \Theta)} \right) \quad (4.39)$$

Using the fact that

$$\log \prod_{r'=1}^{N} \frac{F'(r', t_1^N, a_1^N, \hat{\Theta}, \Theta)}{F'(r', t_1^N, a_1^N, \Theta, \Theta)} = \sum_{r'=1}^{N} \sum_{l'=1}^{L_{r'}} \log \frac{\hat{p}_{a_{l'}^{r'}}}{p_{a_{l'}^{r'}}} + \sum_{r'=1}^{N} \sum_{i'=1}^{L_{t^{r'}}} \log \frac{\hat{q}_{t_{i'}^{r'}}}{q_{t_{i'}^{r'}}},$$

equation (4.39) becomes

$$\log \frac{F'(\hat{\Theta}, \Theta)}{F'(\Theta, \Theta)} \geq \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N}$$

$$\prod_{r=1}^{N} \frac{\delta(t^r, a^r | m_r, y^r)}{DA} \prod_{i=1}^{L_{t^r}} q_{t_i^r} \prod_{l=1}^{L_r} p_{a_l^r} \left( \sum_{r'=1}^{N} \sum_{l'=1}^{L_{r'}} \log \frac{\hat{p}_{a_{l'}^{r'}}}{p_{a_{l'}^{r'}}} + \sum_{r'=1}^{N} \sum_{i'=1}^{L_{t^{r'}}} \log \frac{\hat{q}_{t_{i'}^{r'}}}{q_{t_{i'}^{r'}}} \right)$$

$$- \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \prod_{r=1}^{N} \frac{P_\Theta(y^r | m_r)}{P_\Theta(y^r | m_{gen})} \frac{\delta(t^r, a^r | m_{gen}, y^r)}{DA} \prod_{i=1}^{L_{t^r}} q_{t_i^r} \prod_{l=1}^{L_r} p_{a_l^r}$$

$$\left( \sum_{r'=1}^{N} \sum_{l'=1}^{L_{r'}} \log \frac{\hat{p}_{a_{l'}^{r'}}}{p_{a_{l'}^{r'}}} + \sum_{r'=1}^{N} \sum_{i'=1}^{L_{t^{r'}}} \log \frac{\hat{q}_{t_{i'}^{r'}}}{q_{t_{i'}^{r'}}} \right)$$

$$+ \sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \frac{1}{A} \prod_{i=1}^{L_{t^r}} q_{t_i^r} \prod_{l=1}^{L_r} p_{a_l^r} \left( \sum_{r'=1}^{N} \sum_{l'=1}^{L_{r'}} \log \frac{\hat{p}_{a_{l'}^{r'}}}{p_{a_{l'}^{r'}}} + \sum_{r'=1}^{N} \sum_{i'=1}^{L_{t^{r'}}} \log \frac{\hat{q}_{t_{i'}^{r'}}}{q_{t_{i'}^{r'}}} \right). \quad (4.40)$$

Equation (4.40) is a three-part sum, in which every part is divided into a sum over distribution components and a sum over transition probabilities. Distribution components and transition probabilities can be optimized separately. We will only consider distribution components. First, let us look at the first part of (4.40). We have

$$\sum_a \log \frac{\hat{p}_a}{p_a} \sum_{r'=1}^{N} \sum_{l'=1}^{L_{r'}} \sum_{t^1 \cdots t^N} \sum_{(a^1 \cdots a^N | a_{l'}^{r'} = a)} \prod_{r=1}^{N} \frac{\delta(t^r, a^r | m_r, y^r)}{DA} \prod_{i=1}^{L_{t^r}} q_{t_i^r} \prod_{l=1}^{L_r} p_{a_l^r}$$

$$= \sum_a \log \frac{\hat{p}_a}{p_a} \sum_{r'=1}^{N} \sum_{(l' | y_{l'}^{r'} = k(a))} \sum_{(t | a \in b_t)} \frac{1}{DA} \frac{P_{\Theta,l'}(t, y^{r'} | m_{r'})}{P_\Theta(y^{r'} | m_{r'})} \prod_{r=1}^{N} P_\Theta(y^r | m_r) . \tag{4.41}$$

The development for the second part of (4.40) is similar and left to the interested reader. Now, for the third part, observe that, in the case of distribution components we have

$$\sum_{t^1 \cdots t^N} \sum_{a^1 \cdots a^N} \frac{1}{A} \prod_{r=1}^{N} \prod_{i=1}^{L_{t^r}} q_{t_i^r} \prod_{l=1}^{L_r} p_{a_l^r} \sum_{r'=1}^{N} \sum_{l'=1}^{L_{r'}} \log \frac{\hat{p}_{a_{l'}^{r'}}}{p_{a_{l'}^{r'}}}$$

$$= \frac{1}{A} \sum_{r'=1}^{N} \sum_{l'=1}^{L_{r'}} \sum_a \sum_{t^1 \cdots t^N} \sum_{(a^1 \cdots a^N | a_{l'}^{r'} = a)} \prod_{r=1}^{N} \prod_{i=1}^{L_{t^r}} q_{t_i^r} \prod_{l=1}^{L_r} p_{a_l^r} \log \frac{\hat{p}_a}{p_a}$$

$$= \frac{1}{A} \sum_{r'=1}^{N} \sum_{l'=1}^{L_{r'}} \sum_a \sum_{t^1 \cdots t^N} \sum_{(a^1 \cdots a^N | a_{l'}^{r'} = a)} \prod_{r=1}^{N} \prod_{i=1}^{L_{t^r}} q_{t_i^r} \prod_{r=1}^{N} \prod_{l=1}^{L_r} p_{a_l^r} \log \frac{\hat{p}_a}{p_a}$$

$$= \sum_{r'=1}^{N} L_{r'} \sum_a \frac{p_a}{\sum_{a'} p_{a'}} \log \frac{\hat{p}_a}{p_a} . \tag{4.42}$$

This means that for discrete output probabilities, we want to have

$$\sum_a \log \frac{\hat{p}_a}{p_a} \left\{ \sum_{r'=1}^{N} \sum_{(l' | y_{l'}^{r'} = k(a))} \sum_{(t | a \in b_t)} \left\{ \frac{P_{\Theta,l'}(t, y^{r'} | m_{r'})}{P_\Theta(y^{r'} | m_{r'})} \prod_{r=1}^{N} P_\Theta(y^r | m_r) \right. \right.$$

$$\left. \left. - \frac{P_{\Theta,l'}(t, y^{r'} | m_{gen})}{P_\Theta(y^{r'} | m_{gen})} \prod_{r=1}^{N} \frac{P_\Theta(y^r | m_r)}{P_\Theta(y^r | m_{gen})} P_\Theta(y^r | m_{gen}) \right\} \right\}$$

$$+ DA \sum_{r'=1}^{N} L_{r'} \frac{p_a}{\sum_{a'} p_{a'}} \Bigg\} \geq 0 . \tag{4.43}$$

Then, dividing everywhere by the constant $\prod_{r=1}^{N} P_\Theta(y^r | m_r)$, and defining a new constant $D'$ as

$$D' = \frac{DA \sum_{r'=1}^{N} L_{r'}}{\left(\sum_{a'} p_{a'}\right) \prod_{r=1}^{N} P_\Theta(y^r | m_r)} , \tag{4.44}$$

(4.43) reduces to

$$\sum_a \log \frac{\hat{p}_a}{p_a} \left\{ \sum_{r=1}^{N} \sum_{(t | a \in b_t)} \sum_{(l | y_l^r = k(a))} \psi_{\Theta,l}(t, y^r) + D' p_a \right\}$$

$$= \sum_b \sum_{a \in b} \log \frac{\hat{p}_a}{p_a} \left\{ \sum_{r=1}^{N} \sum_{(t | b_t = b)} \sum_{(l | y_l^r = k(a))} \psi_{\Theta,l}(t, y^r) + D' p_a \right\} \geq 0 . \tag{4.46}$$

Using the notation from Chapter 2 for discrete probabilities, (4.46) can be expressed as

$$\sum_b \sum_k \log \frac{\hat{p}(k|b)}{p(k|b)} \left\{ \sum_{r=1}^{N} \sum_{(t | b_t = b)} \sum_{(l | y_l^r = k)} \psi_{\Theta,l}(t, y^r) + D' p(k|b) \right\} \geq 0 \tag{4.47}$$

Using the constrained optimization technique used for the MLE case, we find the following reestimation formula:

$$\hat{p}(k|b) = \frac{\sum_{r=1}^{N} \sum_{(t | b_t = b)} \sum_{(l | y_l^r = k)} \psi_{\Theta,l}(t, y^r) + D' p(k|b)}{\sum_{k'} \left\{ \sum_{r=1}^{N} \sum_{(t | b_t = b)} \sum_{(l | y_l^r = k')} \psi_{\Theta,l}(t, y^r) \right\} + D'} , \tag{4.48}$$

which is the Gopalakrishnan reestimation formula. Now, suppose as before that $p(k\,|\,b)/\Delta = N(\tilde{y}_k, \sigma_b, \mu_b)$, that is, $p(k\,|\,b)/\Delta$ is an approximation to a Gaussian density. Then we know that as $\Delta \to 0$ and $\nu \to \infty$, the discrete counts in (4.47) become equal to the continuous ones. Suppose further that we also want $\hat{p}(k\,|\,b)/\Delta = N(\tilde{y}_k, \hat{\sigma}_b, \hat{\mu}_b)$. As before, $\tilde{y}_k$ is the mid-point in the $k$th interval. Using (4.47), we must maximize

$$F(\hat{\mu}_b, \hat{\sigma}_b, \mu_b, \sigma_b) = \lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \sum_k \left\{ -\log \hat{\sigma}_b - \frac{(\tilde{y}_k - \hat{\mu}_b)^2}{2\hat{\sigma}_b^{\,2}} \right\} \left\{ \sum_{r=1}^{N} \sum_{(t\,|\,b_t \equiv b)} \sum_{(l\,|\,y_l^r \in k)} \psi_{\Theta,l}(t, \mathbf{y}^r) + D' p(k\,|\,b) \right\} \tag{4.49}$$

with respect to $\hat{\mu}_b$ and $\hat{\sigma}_b$. Taking the derivatives with respect to $\hat{\mu}_b$ and $\hat{\sigma}_b$ and setting them equal to zero, we have

$$\begin{aligned} &\frac{\partial F(\hat{\mu}_b, \hat{\sigma}_b, \mu_b, \sigma_b)}{\partial \hat{\mu}_b} \\ &= \lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \sum_k \left\{ \sum_{r=1}^{N} \sum_{(t\,|\,b_t \equiv b)} \sum_{(l\,|\,y_l^r \in k)} \psi_{\Theta,l}(t, \mathbf{y}^r) + D' p(k\,|\,b) \right\} \frac{(\tilde{y}_k - \hat{\mu}_b)}{\hat{\sigma}_b^{\,2}} \\ &= \sum_{r=1}^{N} \sum_{(t\,|\,b_t \equiv b)} \sum_l \psi_{\Theta,l}(t, \mathbf{y}^r) \frac{(y_l^r - \hat{\mu}_b)}{\sigma^2} + D' \frac{(\mu_b - \hat{\mu}_b)}{\hat{\sigma}_b^{\,2}} = 0, \end{aligned} \tag{4.50}$$

$$\begin{aligned} &\frac{\partial F(\hat{\mu}_b, \hat{\sigma}_b, \mu_b, \sigma_b)}{\partial \hat{\sigma}_b} \\ &= \lim_{\substack{\Delta \to 0 \\ \nu \to \infty}} \sum_k \left\{ \sum_{r=1}^{N} \sum_{(t\,|\,b_t \equiv b)} \sum_{(l\,|\,y_l^r \in k)} \psi_{\Theta,l}(t, \mathbf{y}^r) + D' p(k\,|\,b) \right\} \left\{ -\frac{1}{\hat{\sigma}_b} + \frac{(\tilde{y}_k - \hat{\mu}_b)^2}{\hat{\sigma}_b^{\,3}} \right\} \end{aligned}$$

$$= \sum_{r=1}^{N} \sum_{(t|b_t \equiv b)} \sum_{l} \psi_{\Theta,l}(t,\mathbf{y}^r) \left( \frac{1}{\hat{\sigma}_b} + \frac{y_l^{r2} - 2y_l^r \hat{\mu}_b + \hat{\mu}_b{}^2}{\hat{\sigma}_b{}^3} \right)$$

$$+ D' \left( \frac{1}{\hat{\sigma}_b} + \frac{\sigma_b{}^2 + \mu_b{}^2 - 2\mu_b \hat{\mu}_b + \hat{\mu}_b{}^2}{\hat{\sigma}_b{}^3} \right) = 0 . \qquad (4.51)$$

Solving the above equations leads to

$$\hat{\mu}_b = \frac{\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r)\, y_l^r + D'\mu_b}{\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r) + D'} , \qquad (4.52)$$

$$\hat{\sigma}_b{}^2 = \frac{\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r)\, y_l^{r2} + D'\left(\sigma_b{}^2 + \mu_b{}^2\right)}{\sum_r \sum_{(t|b_t \equiv b)} \sum_{l=1}^{L_r} \psi_{\Theta,l}(t,\mathbf{y}^r) + D'} - \hat{\mu}_b{}^2 , \qquad (4.53)$$

which is what we had obtained with the heuristic development.

### 4.3.3.4. On the value of $D$

Little has been said so far about the constant $D'$ used in the derived reestimation formulas. As we mentioned, for the previous development to be valid, we must have $D > \prod_{r=1}^{N} P_\Theta(\mathbf{y}^r|\mathbf{m}_r)/P_\Theta(\mathbf{y}^r|\mathbf{m}_{gen})$. It is, however, clear from (4.44) that this makes $D'$ an extremely large number. This, in turn, means that using (4.48) within its theoretically proven convergence region will make it practically useless. Moreover, in the development for Gaussian parameters, as $\Delta \to 0$, $D' \to \infty$, which, in effect, means that convergence is only proven for infinitesimal steps. Thus, the reestimation formulas are in fact very similar to gradients.

Nonetheless, even if we cannot theoretically say anything about (4.48), (4.52) and (4.53) when $D'$ is small, these formulas may have great practical value. It turns out that indeed they do. All our experiments using a biased gradient in (4.48) resulted in very fast convergence. Similarly, (4.52) and (4.53) have consistently exhibited good convergence in practice.

## 4.4. MMIE refinements

In order to make $P_\Theta(m_w|y)$ a better model of $P(w|y)$ and thus, hopefully, improve recognition, it is possible to introduce refinements into (4.8). The most obvious refinements are those aimed at reducing the effect of some of the known deficiencies of HMMs such as the diverse independence assumptions (frame independence, codebook independence, etc.). Modifications introduced into the terms used in the computation of $P_\Theta(y|m)$ will leave (4.8) a probability distribution.

### 4.4.1. Global codebook exponents

One of the simplest possible refinements appears in HMMs using multiple codebook output distributions. In those distributions, it is assumed that the parameter sets quantized by each codebook are independent of one another and their probabilities are simply multiplied. This assumption is usually necessary in order to have a tractable number of parameters to estimate. There are two important deficiencies with output distributions of this type. First, the parameter sets are usually not independent (though they may be relatively uncorrelated) and second, they may not carry the same amount of information about the speech being modeled and so, may not have the same importance for recognition.

Apart from modeling the joint distribution, which is not practical, not much can be done about the first problem. The second problem, however, can be tackled by weighting the contribution of each parameter set using *codebook*

*exponents*,[3] as follows:

$$b(\underline{y}) = \prod_{c=1}^{NC} [P_c(\underline{y}_c \mid b)]^{\lambda_c} , \tag{4.54}$$

where $P_c(\underline{y}_c)$ is the distribution associated with the $c$th codebook and $\lambda_c$ is the corresponding codebook exponent. There is an additional advantage to (4.54). It has generally been found that transition probabilities have negligible effect on the overall recognition process. Because of that, some systems simply don't use them at all. The main reason, however, why their effect is negligible is that the dynamic range of transition probabilities is very small compared to that of output probabilities.[4] Moreover, as output distributions become more complex (in number of codebooks, number of parameters, etc), this difference in dynamic ranges increases. The $\lambda_c$s in (4.54) can optimize the dynamic ranges difference in order to improve the model.

Since we don't know of any reestimation formula for the exponents, gradient descent will be used to estimate them. Let us derive the derivative expressions for the two following cases:

1) Discrete with multiple codebooks:

$$b(\underline{y}) = \prod_{c=1}^{NC} [p_c(y_c \mid b)]^{\lambda_c} . \tag{4.55}$$

Using (2.49), we obtain

---

[3]The term "codebook" is sometimes used to designate the parameter set quantized by a given codebook. In that context, "codebook exponent" means that the exponent is applied to the probability of the parameter set corresponding to a given codebook.

[4]This is similar to the mismatch between acoustic probabilities and language model probabilities, which requires that the language model contribution to the log likelihood be multiplied by a certain factor. This factor is usually determined empirically. MMIE is again a good framework for determining that factor automatically.

$$\frac{\partial P_\Theta(\underline{y}^r|\underline{m}_r)}{\partial p_c(k|b)} = \frac{\lambda_c}{p_c(k|b)} \sum_{(t|b_t=b)} \sum_{(l|y_{l,c}=k)} P_{\Theta,l}(t,\underline{y}^r|\underline{m}_r) , \tag{4.56}$$

$$\frac{\partial P_\Theta(\underline{y}^r|\underline{m}_r)}{\partial \lambda_c} = \sum_t \sum_{l=1}^{L_y} P_{\Theta,l}(t,\underline{y}^r|\underline{m}_r) \log p_c(y_{l,c}|b) , \tag{4.57}$$

where natural logarithms are assumed.

2) Multiple codebooks semi-continuous:

$$b(\underline{y}) = \prod_{c=1}^{NC} \left[ \sum_{k=0}^{K_c-1} P_c(\underline{y}_c|k) p_c(k|b) \right]^{\lambda_c} . \tag{4.58}$$

We now have

$$\frac{\partial P_\Theta(\underline{y}^r|\underline{m}_r)}{\partial p_c(k|b)} = \frac{\lambda_c}{p_c(k|b)} \sum_{(t|b_t=b)} \sum_{l=1}^{L_y} \frac{P_c(\underline{y}_{l,c}|k)p_c(k|b)}{\sum_{k'} P_c(\underline{y}_{l,c}|k')p_c(k'|b)} P_{\Theta,l}(t,\underline{y}^r|\underline{m}_r)$$

$$= \frac{\lambda_c}{p_c(k|b)} \sum_{(t|b_t=b)} \sum_{l=1}^{L_y} \gamma_{b,c}(k|\underline{y}_{l,c}) P_{\Theta,l}(t,\underline{y}^r|\underline{m}_r) , \tag{4.59}$$

where the $\gamma_{b,c}(k|\underline{y}_{l,c})$ is the same as in (2.60), and

$$\frac{\partial P_\Theta(\underline{y}^r|\underline{m}_r)}{\partial \lambda_c} = \sum_{t \text{ full}} \sum_{l=1}^{L_y} P_{\Theta,l}(t,\underline{y}^r|\underline{m}_r) \log \left[ \sum_{k=0}^{K_c-1} P_c(\underline{y}_{l,c}|k) p_c(k|b_t) \right]. \tag{4.60}$$

Let $\alpha$ be a parameter of $P_c(\underline{y}_{l,c}|k)$, then

$$\frac{\partial P_\Theta(\underline{y}^r|\underline{m}_r)}{\partial \alpha} =$$

$$\sum_{(t|b_t=b)} \sum_{l=1}^{L_y} \frac{\lambda_c}{P_c(\underline{y}_{l,c}|k)} \gamma_{b,c}(k|\underline{y}_{l,c}) P_{\Theta,l}(t,\underline{y}^r|\underline{m}_r) \frac{\partial P_c(\underline{y}_{l,c}|k)}{\partial \alpha} . \qquad (4.61)$$

### 4.4.2. Frame-dependent weighting

One of the weaknesses of HMMs as speech models is the fact that all speech frames have the same weight in the computation of $P_\Theta(\underline{y}^r|\underline{m}_r)$. This clearly cannot be correct; we know intuitively that linguistic information is not uniformly distributed across the speech signal. For example, there is much redundancy in long vowel sounds and each frame from that vowel probably carries less information than frames from, say, a plosive. In order to determine how much new information $\underline{y}_l$ carries, however, previous observations have to be taken into account.

Suppose we have a labeler $\kappa(\underline{y})$ which classifies a speech frame into a number of categories, according to some criteria. This labeler could, for example, be based on prosodic information. In particular, information about stress might especially useful. Indeed, it has been found [ZUE 85] that, in addition to being more robust, acoustic observations around stressed syllables contain more information about the words spoken than those around unstressed syllables.

We can use that labeler to weigh the contribution of each speech frame $\underline{y}_l$ in the computation of $P_\Theta(\underline{y}^r|\underline{m}_r)$ according to the category $\kappa(\underline{y}_l)$ to which it was assigned. This could be done by expressing output distributions as

$$b(\underline{y}_l) = [\bar{b}(\underline{y}_l)]^{\lambda(\kappa(\underline{y}_l))} , \qquad (4.62)$$

where $\bar{b}(\underline{y}_l)$ is any standard output distribution and $\lambda(\kappa(\underline{y}_l))$ is the weight applied to $\bar{b}(\underline{y}_l)$. If $\lambda(\kappa(\underline{y}_l)) \geq 0$, it is easy to see that the larger $\lambda(\kappa(\underline{y}_l))$ is, the larger the contribution of $\underline{y}_l$ to the final value of $\log P_\Theta(\underline{y}^r|\underline{m}_r)$ will be. As an extreme case, if $\lambda(\kappa(\underline{y}_l)) = 0$, then $\underline{y}_l$ will not have any effect on $\log P_\Theta(\underline{y}^r|\underline{m}_r)$. From the communication theory viewpoint, the interpretation would be that $\underline{y}_l$ should not be considered for speech recognition. This, for

example could happen if $\underline{y}_l$ doesn't carry any useful information about $w$ that is not already in the other observations.

MMIE can be used to learn $\lambda(\kappa)$, where $\kappa$ is one of the frame categories. The labeler itself, however, has to be determined a priori. MMIE will only be useful if the category $\kappa(\underline{y}_l)$ of $\underline{y}_l$ can somehow be correlated with the "usefulness" of $\underline{y}_l$ for speech recognition purposes. We assume that a good labeler is available and find the derivative expressions (using 2.49) with respect to $\lambda(k)$, the weight of frame category $k$, to be

$$\frac{\partial P_\Theta(\boldsymbol{y}^r|m_r)}{\partial\lambda(k)} = \frac{1}{\lambda(k)} \sum_t \sum_{(l|\kappa(\underline{y}_l)=k)} P_{\Theta,l}(t,\boldsymbol{y}^r|m_r) \log b_t(\underline{y}_l) , \quad (4.63)$$

and the derivative with respect to $\alpha$, a parameter of $\bar{b}(\underline{y}_l)$, to be

$$\frac{\partial P_\Theta(\boldsymbol{y}^r|m_r)}{\alpha} = \sum_{(t|b_t\equiv b)} \sum_{l=1}^{L_y} \frac{\lambda(\kappa(\underline{y}_l))}{\bar{b}(\underline{y}_l)} P_{\Theta,l}(t,\boldsymbol{y}^r|m_r) \frac{\partial\bar{b}(\underline{y}_l)}{\partial\alpha} . \quad (4.64)$$

As a generalization of both codebook exponents and frame-dependent weighting, we can use an output distribution of the form

$$b(\underline{y}) = \left[\prod_{c=1}^{NC} [P_c(\underline{y}_c|b)]^{\lambda_c(\kappa_1(\underline{y}_l))}\right]^{\lambda(\kappa_2(\underline{y}_l))} , \quad (4.65)$$

where $\kappa_1(\underline{y}_l)$ is the frame labeler used to determine the frame-dependent codebook exponents and $\kappa_2(\underline{y}_l)$ is the labeler used to determine the frame-dependent distribution exponent. The derivatives become

$$\frac{\partial P_\Theta(\boldsymbol{y}^r|m_r)}{\partial\lambda_c(k_1)} = \sum_t \sum_{(l|\kappa_1(\underline{y}_l)=k_1)} \lambda(\kappa_2(\underline{y}_l)) P_{\Theta,l}(t,\boldsymbol{y}^r|m_r) \log P_c(\underline{y}_{l,c}|b_t) , \quad (4.66)$$

$$\frac{\partial P_\Theta(\boldsymbol{y}^r|m_r)}{\partial\lambda(k_2)} = \frac{1}{\lambda(k_2)} \sum_t \sum_{(l|\kappa_2(\underline{y}_l)=k_2)} P_{\Theta,l}(t,\boldsymbol{y}^r|m_r) \log b_t(\underline{y}_l) , \quad (4.67)$$

and let $\alpha$ be a parameter of $P_c(\underline{y}_c \,|b)$. Then

$$\frac{\partial P_\Theta(\boldsymbol{y}^r|\boldsymbol{m}_r)}{\partial \alpha} =$$

$$\sum_{(t\,|b_t \neq b)} \sum_{l=1}^{L_y} \frac{\lambda_c(\kappa_1(\underline{y}_l))\,\lambda(\kappa_2(\underline{y}_l))}{P_c(\underline{y}_{l,c}\,|k)}\, P_{\Theta,l}(t,\boldsymbol{y}^r|\boldsymbol{m}_r)\, \frac{\partial P_c(\underline{y}_{l,c}\,|b)}{\partial \alpha} \qquad (4.68)$$

# 5. CONNECTED DIGIT RECOGNITION EXPERIMENTS

## 5.1. The TI/NIST connected digit task

### 5.1.1. Database description

The connected digit recognition experiments were performed using the adult speaker portion of the TI/NIST connected digit database [LEON 84]. This large database contains speech from a total of 326 speakers (111 men, 114 women and 101 children), coming from 21 geographical regions of the continental United States (approximately 5 men, 5 women and 5 children per region). The database vocabulary is made of the digits '1' to '9', plus 'oh' and 'zero', for a total of 11 words. Each speaker in the database provided two repetitions of each digit in isolation and 55 digit strings, evenly distributed into lengths 2, 3, 4, 5 and 7. This makes a total of 77 digit strings, or 253 digits per speaker. Each string is stored in a separate signal file, with some silence (or background noise) preceding and following the speech signal. Approximately half the speakers have been assigned to the training set, the remaining half being the testing set.

This is a clean database in the sense that it has high quality sound and high signal to noise ratio (SNR). It was originally sampled at 20 kHz using a 16 bit A/D and a 10 kHz antialiasing filter. It has subsequently been downsampled by NIST to 10 kHz, the version we use.

Following the lead of other researchers [BOCC 86, BUSH 87, DODD 89, RABI 89a, RABI 89b], we only use the adult portion of the database. This reduces it to 225 speakers (111 men and 114 women), 112 of which (55 men, 57 women) are used for training and 113 (56 men, 57 women) for testing. This is the standard set used by most researchers, which makes result comparisons

relatively meaningful. Out of a total of 17 325 signal files (there is one digit string per signal file) in the database, 20 contained errors and could not be used. Of these, 8 were in the training set and 12 in the testing set. As a result, our training set contained 8616 digit strings (28 302 digits) and our testing set contained 8689 digit strings (28 543 digits).

There are usually two types of recognition experiments performed with the TI/NIST connected digit database. In the first type, *known-length* recognition, the number of digits in each digit string is assumed to be known a priori. In this case, the number of errors is computed by comparing in sequence the digits in the true and the recognized strings and counting the number of mismatches. Thus, all recognition errors are assumed to be *substitution errors*. In the second, more difficult type, *unknown-length* recognition, the string length is assumed unknown. This means that the true and the recognized strings do not necessarily contain the same number of digits. The number of errors is computed by first doing an optimal dynamic programming based alignment between the true and the recognized strings [PICO 86]. This alignment produces three types of errors: insertions, deletions and substitutions. In both cases, results are usually reported in terms of word and string error rates (or recognition rates).[1]

In this work, only unknown length recognition experiments were performed. The word error rate was computed using

$$\text{word error rate} = \frac{\text{insertions + deletions + substitutions}}{\text{total number of words}} \times 100\% \ ,$$

and the string error rate is simply

$$\text{string error rate} = \frac{\text{number of strings with one or more errors}}{\text{total number of strings}} \times 100\% \ .$$

[1]Note that our scoring algorithm was compared to the NIST scoring algorithm, and they were found to give identical results.

### 5.1.2. Previous results

Digits — telephone numbers, prices, serial numbers, code numbers, etc. — are an inherent part of our everyday life. The number of potential applications to digit recognition is almost limitless. It is not surprising, then, that the problem of digit recognition has enjoyed a large popularity within the speech recognition community. This is especially true since, thanks to the wide availability of the large TI/NIST connected digit database, it is possible to compare the recognition rates obtained at different sites.

Results on the TI/NIST database have been reported in the literature ever since its creation. In 1985, Kopec and Bush [KOPE 85] reported a 2% error rate using only the isolated digits in the adult portion of the database. In 1986, Bush and Kopec [BUSH 86] reported results of connected digit recognition experiments using about half of the adult portion of the database. Separating male and female talkers, they achieved 3.5% and 2.2% string error rate for unknown length and known length recognition, respectively.

The next year, the same authors reported results obtained using the entire adult portion of the database. Using separate models for male and female talkers, they achieved a 4% string error rate (around 1.5% word error rate).

In 1988, Rabiner *et al.* reported a 2.94% string error rate on the same task. This result was obtained using 4 models per digit and Gaussian mixture distributions. The next year, they improved their performance to 2.84% string error rate by experimenting with different clustering procedures.

The best results reported so far in the literature are probably those of George Doddington [DODD 89]. He obtained a 1.5% string (0.5% word) error rate using *phonetically sensitive discriminants.* His system uses an 18-element feature vector obtained from a 32-element feature vector via principal component analysis. He uses separate male and female models in which each state corresponds, on average, to one frame of speech. For each state, a linear discriminant transformation matrix is computed using "in-class data" and "confusion data". The resulting complexity is about equivalent to a single full covariance density per state.

## 5.2. Baseline system

### 5.2.1. Signal processing

The speech in our version of the TI/NIST connected digit database was sampled at 10 kHz. This speech signal $s(n)$ is first pre-emphasized using the difference equation $\bar{s}(n) = s(n) - 0.95s(n-1)$. The effect of pre-emphasis is to spectrally flatten the signal. The pre-emphasized signal is then blocked into 256-point frames, with 100 points separating the beginning of two consecutive frames. The frame rate is thus 100 frames per second. No endpoint detection is performed.

Each frame is analyzed with a 256-point FFT, using a Hamming window. A bank of 20 mel-scaled, triangular band-pass filters is applied to the FFT spectrum [DAVI 80] and the logarithm of the energy in each band is computed. Then, cepstral coefficients $C_i$ are extracted by applying a cosine transform to the 20 log-energies, using the equation

$$C_i = \sum_{k=0}^{19} X_k \cos\left[i(k+0.5)\frac{\pi}{20}\right], \tag{5.1}$$

where $X_k$ is the log-energy in the $k$th band. It was found [HUNT 89] that this cepstral transformation is close to a principal component analysis and that the resulting cepstral coefficients are nearly uncorrelated, with most of the useful information in the first coefficients. In our system, only the 6 cepstral coefficients $C_1$ through $C_6$ are used. The coefficient $C_0$, the total log-energy across the channels, is not used. Instead, we use a different parameter $E$, which we compute directly from the signal points $\bar{s}(n)$ in a frame (after both pre-emphasis and windowing) using

$$E = 10 \log_{10}\left[\frac{1}{N}\sum_{n=0}^{N-1} [w(n)\bar{s}(n)]^2\right], \tag{5.2}$$

where $N$ is the frame size (256 in our case), $w(n)$ is the Hamming window, and $\bar{s}(0)$ is the first point in the frame.

For each of these 7 "static" parameters, a corresponding "dynamic" parameter is computed. The dynamic parameter is used to describe how the static parameter changes over time. It is defined as the slope of the linear regression of the static parameter, computed over a 5-frame window centered on the current frame as

$$\Delta P(l) = 0.1 \sum_{k=-2}^{2} kP(l+k) , \tag{5.3}$$

where $P(l)$ is any of the static parameters at frame $l$, and $\Delta P(l)$ is the corresponding dynamic parameter. Thus, in total, 14 parameters are extracted from each frame.

### 5.2.2. Vector quantization

Three VQ codebooks [LEEK 88] are created from the entire training set (1509735 frames), using the binary-split VQ training algorithm [RABI 83a, GRAY 84] with a Euclidean distortion measure. The first codebook, of size 128, is used for the 6 cepstral coefficients $C_1$ through $C_6$. The second, also of size 128, is used for the 6 cepstral slopes $\Delta C_1$ through $\Delta C_6$. The third, of size 32, is used for both the log-energy $E$ and its slope $\Delta E$.

The VQ codebooks thus created have two different purposes. In the baseline system, they are used to quantize the 14-dimensional continuous parameter vectors into 3-dimensional discrete vectors. For semi-continuous HMMs, the codebooks are used to compute an initial estimate for the mean and variance vectors of the tied mixtures.

### 5.2.3. HMMs

The baseline system is a standard discrete HMM system with one model per digit. The output distributions used are of the *discrete with multiple codebooks* type (see (2.12)), where in our case the number of codebooks is 3.

The word models are built from a set of sub-word units, using the following lexicon:

| | | | | | |
|---|---|---|---|---|---|
| 1 | w-ax | n-tail | | | |
| 2 | t | uw | | | |
| 3 | th | r-iy | | | |
| 4 | f | ow-r | | | |
| 5 | f | ay | v5 | | |
| 6 | s | ih-k | pau | k-s | |
| 7 | s | eh | v7 | ax | n-tail |
| 8 | ey | pau | t8 | | |
| 9 | n-head | ay | n-tail | | |
| oh | ow | | | | |
| zero | zz | iy-r-ow | | | |

**Table 5.1:** Digit lexicon

There are a total of 24 unit models, including the pause model (pau) and the silence model (sil). For sub-word units, the HMM structure has been chosen to allow precise temporal modeling of acoustic events. The basic HMM building blocks are the *duration* block (Figure 5.1) and the *head/tail* block (Figure 5.2). All transitions in any of these blocks share the same output distribution. Each subword unit model is built from the concatenation of one head block, a unit-dependent number of duration blocks (none, in some cases), and, if duration blocks are used, a tail block. This is shown in Table 5.2.

| model | head | duration | tail | model | head | duration | tail |
|---|---|---|---|---|---|---|---|
| w-ax | 1 | 6 | 1 | n-tail | 1 | 3 | 1 |
| t | 1 | 3 | 1 | uw | 1 | 8 | 1 |
| th | 1 | 2 | 1 | r-iy | 1 | 5 | 1 |
| f | 1 | 2 | 1 | ow-r | 1 | 11 | 1 |
| ay | 1 | 8 | 1 | v-5 | 1 | | |
| s | 1 | 3 | 1 | ih-k | 1 | 3 | 1 |
| k-s | 1 | 3 | 2 | eh | 1 | 4 | 1 |
| ax | 1 | 1 | 1 | v-7 | 1 | | |
| ey | 1 | 7 | 1 | t-8 | 1 | | |
| n-head | 1 | 2 | 1 | ow | 1 | 8 | 1 |
| z | 1 | 2 | 1 | iy-r-ow | 1 | 10 | 1 |

**Table 5.2:** Structure of unit models

The number of duration blocks in each unit is a function of the average time duration of each unit. Note that duration blocks without a self-loop could

have been used; however, we did not want to constrain the duration of an acoustic event to a given maximum (determined from the training set).

Silence and pause models, intended to model near-stationary processes that can, in principle be of any length, have a simpler structure (Figure 5.3). In fact, in this case, a 2-state model such as the head/tail block would have done just as well.

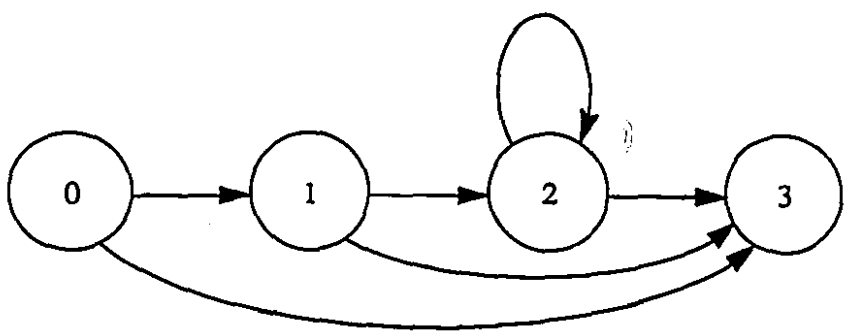


**Figure 5.1:** Duration block

This unit set is somewhat arbitrary and obviously task-specific. It was originally chosen because speech data from 13 speakers in the training set had previously been manually segmented according to these units. The set of manually segmented speech has now been extended to 78 speakers from the training set.[2] We use these speakers with the corresponding labels to bootstrap the unit models with 3 iterations of Maximum Likelihood Estimation (MLE) training. After bootstraping, the silence and pause models are converted into optional models. This means that a new state is added at the beginning of the models, with equiprobable empty transitions going to the previous first state and to the last state (see Figure 3.1).

[2]Interestingly, even though our experience shows proper model bootstraping to be very important in order to get good results, it doesn't seem necessary to use large amounts of data for that purpose. We found basically no difference in our final results when the bootstraping set was increased from 13 to 78 speakers.

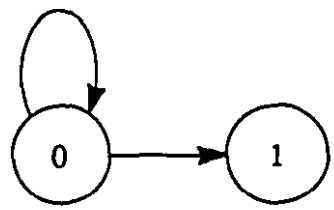


**Figure 5.2: Head/tail blocks**

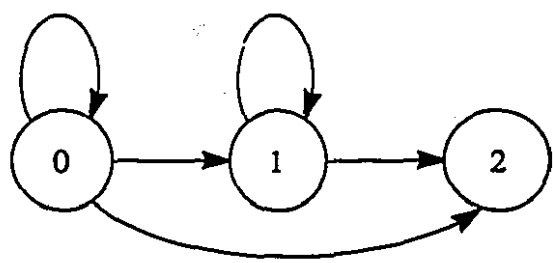


**Figure 5.3: Silence and pause models**

Subsequently, three iterations of MLE training are performed on the entire training set. In each iteration, a model corresponding to each digit string in the training set is built such that it reflects the way the digit string would appear in the looped model used for recognition. For example, the model corresponding to the digit string 5-9-6 is shown in Figure 5.4. The Baum-Welch algorithm is applied to this model, using the observation sequence from the corresponding string. The unit models obtained after MLE training are called the *MLE models*.

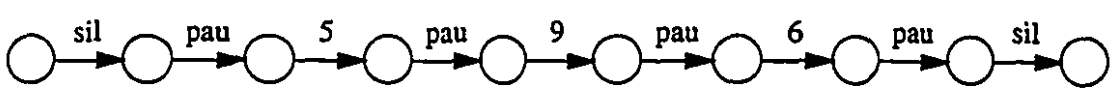


**Figure 5.4: Model used for digit string 5-9-6**

Unknown string length recognition is performed by applying the Viterbi algorithm with the looped model (Figure 5.5). As described, the baseline system has a word error rate of 1.40% and a string error rate of 4.01%. Table 5.3 shows the system's recognition performance on the test set as the number of MLE training iterations is increased. In this and the following tables, word and string error rates are in percentage, while insertions (ins), deletions (del) and substitutions (sub) are given in absolute numbers.

| Iterations | word | string | ins | del | sub |
|---|---|---|---|---|---|
| 3 | 1.40 | 4.01 | 55 | 107 | 237 |
| 4 | 1.40 | 4.04 | 54 | 110 | 236 |
| 5 | 1.39 | 4.01 | 52 | 112 | 232 |
| 6 | 1.36 | 3.91 | 50 | 110 | 227 |
| 7 | 1.37 | 3.96 | 51 | 112 | 229 |
| 8 | 1.37 | 3.94 | 51 | 113 | 227 |
| 9 | 1.36 | 3.90 | 50 | 112 | 225 |

**Table 5.3:** Baseline system's recognition performance on the test set as the number of training iterations is increased.

As seen from the table, after 3 iterations, the performance doesn't really change if the number of iterations is increased. It is also interesting to see that performance on the training set, though somewhat better than on the test set, also levels off very quickly. This is shown in Table 5.4. This, as MMIE results will clearly show, illustrates the discrepancy between MLE training and the objective of reducing the error rate.

| Iterations | word | string |
|---|---|---|
| 3 | 1.14 | 3.27 |
| 4 | 1.12 | 3.23 |
| 5 | 1.10 | 3.15 |
| 6 | 1.10 | 3.16 |
| 7 | 1.08 | 3.10 |
| 8 | 1.08 | 3.09 |
| 9 | 1.08 | 3.09 |

**Table 5.4:** Baseline system's performance on the training set

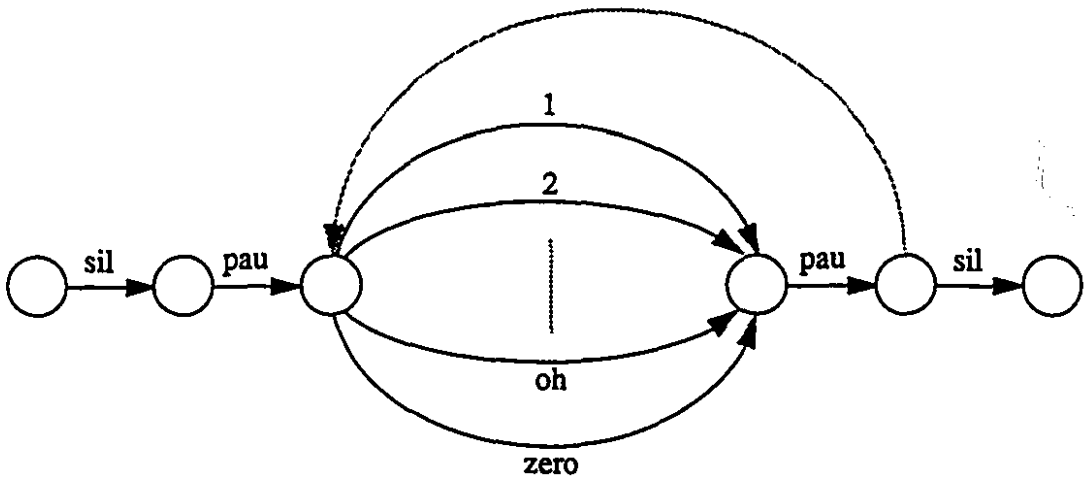


**Figure 5.5:** Looped model.

## 5.3. Semi-continuous HMMs

All SCHMM experiments reported assume diagonal covariance Gaussian densities. Since, as mentioned earlier, the cepstral coefficients are relatively uncorrelated, this assumption should not dramatically affect the system's performance; it will, however, substantially reduce both the number of parameters to estimate and the computation time.

Trained models taken from a system using discrete with multiple codebook distributions can trivially be converted into models for a multiple codebook SCHMM system. In fact, the models can stay the same. Only the means and variances of the tied mixture components need to be estimated. Using the codebook used in the discrete case, initial estimates can be obtained by assigning all frames from the training set to the closest codeword in the codebook. Then, for each codeword, means and variances of the corresponding mixture can be computed using all frames assigned to it.

It is usually better, however, to train semi-continuous HMMs following the steps used in the discrete case. That is, start with initial models with uniform distributions and initial mixtures components as described above, and perform bootstraping followed by MLE training (using the same number of iterations). This has the advantage of jointly optimizing (with MLE) the codebook probabilities and the mixture parameters. Table 5.5 compares the two approaches.

Unless specified otherwise, all semi-continuous experiments (training and recognition) were performed by considering only the 3 most probable components (densities) in the mixture. Thus, referring to equation (2.17), we assume $P_c(\underline{y}_c|k)=0$ if $k$ is not one of the 3 most probable mixture components. This may affect performance but it substantially reduces the execution time.

| | word | string | ins | del | sub |
|---|---|---|---|---|---|
| discrete models | 1.40 | 4.00 | 55 | 107 | 237 |
| direct conversion to SCHMM | 1.44 | 4.14 | 55 | 120 | 202 |
| full SCHMM training | 1.22 | 3.51 | 47 | 99 | 202 |

**Table 5.5:** Semi-continuous recognition on the test set with 3 iterations of MLE training.

## 5.4. MMIE experiments

There were two aims pursued in our MMIE experiments. The first aim was to evaluate the effectiveness of the reestimation techniques proposed in Chapter 4. In our opinion, an effective reestimation formula should be such that in most cases, it will produce a sizable improvement in the value of the function being optimized. For example, in MLE training, the fact that the reestimation formulas from the Baum-Welch algorithm are mathematically guaranteed not to produce a degradation is very comforting. What is most important, however, is the experimental evidence showing that in practice, good parameter values are usually obtained after a small number of iterations. That, of course, is not to say that faster convergence cannot be obtained otherwise.[3] In the case of the MMIE reestimation formulas proposed in this thesis, there is no mathematical guarantee of convergence. What we are looking for, then, is to experimentally demonstrate their effectiveness.

The second aim was to evaluate the usefulness of MMIE training for speech recognition. In both discrete and semi-continuous experiments, discrete distributions (codebook and transition probabilities) were reestimated using

[3]In fact, it is not impossible that estimation techniques borrowed from the field of neural networks, such as *on-line* estimation, could lead to faster convergence.

(4.12), with $D$ computed using (4.17), as proposed by Gopalakrishnan *et al.* [GOPA 89].

In the connected digit case, the general model $m_{gen}$ used for training (see Chapter 4) is simply the looped model used for recognition (Figure 5.5). For each digit string in the training set, a typical MMIE training iteration thus consists of one pass of the Baum-Welch algorithm using the "correct" string model, and one pass using the looped model.

### 5.4.1. Convergence experiments for discrete distributions

The following experiment was aimed at evaluating the effect of biasing the gradient expression, as proposed by Merialdo [MERI 88]. Using $c_\theta$ and $c_\theta^{gen}$ as defined in (4.14), the gradient expressions compared were:

1) Exact expression

$$\frac{\partial \log R(\Theta)}{\partial \theta} = \frac{1}{\theta}(c_\theta - c_\theta^{gen})$$

2) Weighted (1)

$$\frac{\partial \log R(\Theta)}{\partial \theta} \approx \frac{1}{\theta}(c_\theta - c_\theta^{gen})\,(c_\theta + c_\theta^{gen})$$

3) Weighted (2)

$$\frac{\partial \log R(\Theta)}{\partial \theta} \approx \frac{1}{\theta}(c_\theta - c_\theta^{gen})\,\frac{(c_\theta + c_\theta^{gen})}{\sum_{\theta' \in b(\theta)} (c_{\theta'} + c_{\theta'}^{gen})}$$

4) Merialdo

$$\frac{\partial \log R(\Theta)}{\partial \theta} \approx \frac{c_\theta}{\sum_{\theta' \in b(\theta)} c_{\theta'}} - \frac{c_\theta^{gen}}{\sum_{\theta' \in b(\theta)} c_{\theta'}^{gen}}$$

5) Modified Merialdo

$$\frac{\partial \log R(\Theta)}{\partial \theta} \approx \frac{(c_\theta - c_\theta^{gen})}{\sum_{\theta' \in b(\theta)} (c_{\theta'} + c_{\theta'}^{gen})}$$

For each of these five expressions, several full MMIE iterations[4] were performed on a subset of the complete training set made from all the speech data from 10 male and 10 female speakers. The value of the logarithm of the global criterion $R(\Theta)$ (using a base of 1.001) was computed after each iteration. The initial value of the criterion was -406055. The results are summarized in the Table 5.6.

| Iteration | Exact | Weighted (1) | Weighted(2) | Merialdo | Mod. Merialdo |
|---|---|---|---|---|---|
| 1 | -395443 | -314734 | -282458 | -127088 | -141359 |
| 2 | -386397 | -176154 | -235354 | -28792 | -33944 |
| 3 | -377779 | -263393 | -183231 | -67025 | -11027 |
| 4 | -369426 | -104312 | -136283 | -5152 | -22813 |
| 5 | -360547 | -120408 | -94958 | -9710 | -3690 |
| 6 | -355778 | -89411 | -75643 | -105 | -7231 |
| 7 | -351413 | -50689 | -56996 | -4213 | -85998 |
| 8 | -347043 | -29394 | -24143 | -9391 | -6242 |
| 9 | -342651 | -28624 | -29685 | -11360 | -41501 |
| 10 | -338328 | -27232 | -16629 | -39859 | -3336 |
| 11 | -334071 | -20965 | -9322 | -96657 | -351 |
| 12 | -329960 | -1529 | -17517 | -72 | -121 |
| 13 | -325923 | -39432 | -22446 | -127 | -35856 |
| 14 | -321776 | -152421 | -5506 | -116314 | -11779 |
| 15 | -317774 | -12861 | -577 | -78 | -533 |
| 16 | -313603 | -766 | -2177 | -23295 | -120592 |
| 17 | -309468 | -97174 | -2085 | -2775 | -45407 |

**Table 5.6:** Convergence of gradient expressions

The convergence is graphically illustrated in Figure 5.6 for the exact expression, the Merialdo expression, and the "weighted (2)" expression.

[4]For these convergence experiments, MMIE training was done using all digit strings in the described training set. This is in contrast with the corrective MMIE training algorithm and it is what is meant by "full MMIE iterations".

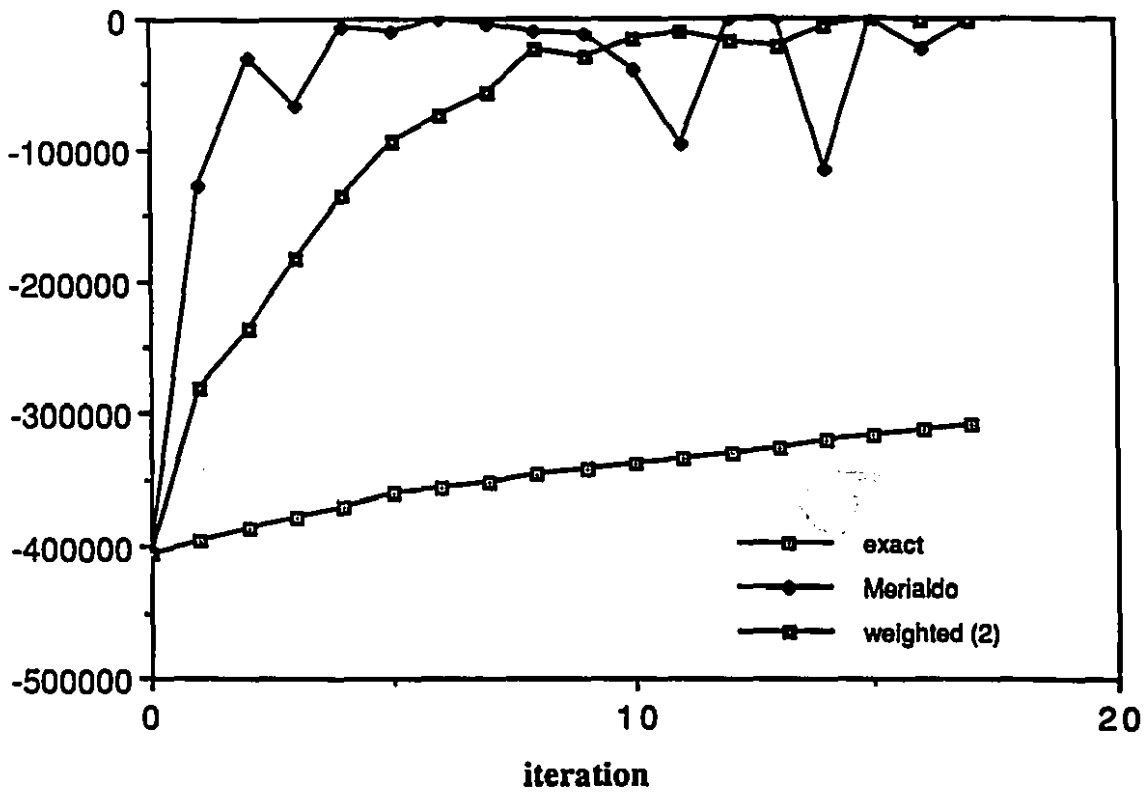


**Figure 5.6:** Value of $\log_{1.001} R(\Theta)$ as a function of the iteration number

However difficult it is to draw definite conclusions from these results, some important trends are noticeable. First, it seems clear that using the exact gradient expression results in hopelessly slow convergence. Second, with all other expressions, we observe that as the criterion gets close to the optimum value of zero, training becomes chaotic and divergence is often observed. In fact, some iterations can result in large degradations, although results are always substantially better than with the exact expression.

It is unclear at this point whether this behavior is caused by the modified gradient expressions, which become inaccurate near the optimum, or whether it is caused by the use of (4.17) to determine the value of $D$ used in (4.12). Remember that (4.12) is only proven to converge for a $D$ probably much larger than the one actually used. This, on the other hand, seems unlikely since (4.21) and (4.22) show that at a fixed point of $R(\Theta)$ (any local optimum), the value of $D$ is zero.

In order to clarify this point, we performed an experiment using the exact gradient expression, but starting with the models obtained after the 16th iteration with the "Weighted (2)" formula. The initial value of $\log_{1.001} R(\Theta)$ was -2177. The value of $\log_{1.001} R(\Theta)$ after each iteration is shown in Table 5.7. As can be seen, the value of $R(\Theta)$ stays much more stable near the optimum (the variations observed are consistent with the precision of the computations).

| Iteration | $\log_{1.001} R(\Theta)$ |
|---|---|
| 1 | -85 |
| 2 | -67 |
| 3 | -77 |
| 4 | -102 |
| 5 | -164 |
| 6 | -247 |
| 7 | -149 |
| 8 | -84 |

**Table 5.7:** Convergence of exact expression near the optimum

The chaotic behavior of $\log_{1.001} R(\Theta)$ near the optimum seems, therefore, to be caused by the use of a modified gradient expression. Even though this behavior is clearly undesirable, it does not necessarily undermine the usefulness of the modified expressions. Indeed, the initial reductions of $\log_{1.001} R(\Theta)$ are quite spectacular compared with those obtained with the exact expression. The problem really seems to appear when $\Theta$ is already relatively satisfactory. Note that since, in the MMIE corrective training algorithm, only incorrectly recognized strings are used for training, then $\Theta$ is always far from satisfactory (it in fact caused recognition errors for every single digit string in the training set). It is thus quite possible that, in the context of this algorithm, the modified expressions will perform as desired.

In order to verify this, we first decided to limit all further investigations to the Merialdo gradient expression. Then in all our discrete HMM experiments with corrective MMIE training, we looked at the value of $\log_{1.001} R(\Theta)$ before and after reestimation for each iteration of the training algorithm. The results obtained are given in the section on recognition experiments. They confirm our speculation that, in the context of corrective MMIE training, the modified reestimation formula is indeed very effective.

### 5.4.2. Convergence experiments with the Gaussian reestimation formula

This experiment uses multiple codebooks semi-continuous HMMs to look at the convergence of the reestimation formulas (4.52) and (4.53). The formulas are used to reestimate the means and variances of each of the tied Gaussian densities in the codebook. The models used in this experiment are discrete HMMs obtained after 3 MLE iterations on the entire training set and the initial tied mixtures were computed from the codebook as explained before. The training set is the same as the one used in the previous convergence experiments (10 male and 10 female speakers).

In order to verify the effectiveness of the continuous reestimation formulas, only the parameters of the tied Gaussian densities were modified during training. A heuristic way of determining the constant $D$ in the formulas was used. We started with the minimum value guaranteeing a positive denominator in (4.53). We then repeatedly doubled its value until all new variance estimates were positive. The value of $D$ used was double that final value. The initial value of the optimization function $\log_{1.001} R(\Theta)$ was -302278. The experiment is summarized in Table 5.8.

| Iteration | $\log_{1.001} R(\Theta)$ |
|---|---|
| 1 | -169153 |
| 2 | -98848 |
| 3 | -70639 |
| 4 | -57420 |
| 5 | -39332 |
| 6 | -23169 |
| 7 | -14225 |
| 8 | -13184 |

**Table 5.8:** Convergence of continuous reestimation formulas

As can be seen from the Table, the convergence is steady and relatively impressive, considering that all discrete distribution parameters remained constant. Again, it is probably possible to have a faster convergence using gradient descent but the technique proposed here offers both simplicity and convergence within a small number of iterations.

### 5.4.3. Recognition experiments with discrete HMMs

As is the case for all training techniques of an error-correcting nature [APPL 89, LEEK 90], the corrective MMIE training algorithm has an important problem. As the number of errors on the training set becomes negligible, the algorithm runs out of data to train on. The procedure must thus stop because of a lack of training data.

There is another, closely related problem. Since the training set used for reestimation is in fact a very small fraction of the full training set, the parameters learned may not be as globally useful as if the whole training set were used for estimation.

In our experiments with the corrective MMIE training algorithm, we used a simple solution to both the above problems. In each iteration, the new models are smoothed with those from the previous iteration, using a weight for the old model that increases by increments of 0.1 from 0.0 to 0.8 (for a total of 9 iterations). Thus, as used, the weights are not a function of the reestimation set size.

#### 5.4.3.1. Applying MMIE to the baseline system

This first experiment uses the models obtained after 3 MLE iterations and applies 9 iterations of corrective MMIE training. Table 5.9 shows the results obtained on the test set after each iteration.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| 1 | 1.16 | 3.38 | 51 | 88 | 191 |
| 2 | 1.05 | 3.10 | 46 | 76 | 178 |
| 3 | 1.00 | 2.98 | 42 | 72 | 171 |
| 4 | 0.95 | 2.87 | 38 | 66 | 167 |
| 5 | 0.95 | 2.87 | 40 | 60 | 171 |
| 6 | 0.95 | 2.84 | 41 | 60 | 169 |
| 7 | 0.91 | 2.76 | 38 | 59 | 164 |
| 8 | 0.91 | 2.75 | 38 | 59 | 163 |
| 9 | 0.92 | 2.79 | 40 | 56 | 166 |

**Table 5.9:** Applying MMIE to the baseline system. Error rate on the test set after each iteration of corrective MMIE training.

As can be seen from the table, straightforward application of corrective MMIE training reduced the word error rate by around 32% from 1.36% to 0.92% and the string error rate by around 28% from 3.90% to 2.79%.[5]

Table 5.10 shows the performance of the training algorithm on the training set. The string recognition rate gives the percentage of the total number of strings from the training set that was not used for reestimation. For example, in the sixth iteration, the reestimation set contains only 0.2% of the number of strings in the full training set. The last two columns show the value of $\log_{1.001} R(\Theta)$, computed on the reestimation set before and after reestimation. They demonstrate the effectiveness of the modified reestimation formula.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | |
|---|---|---|---|---|
| | word | string | before | after |
| 1 | 1.14 | 3.27 | -5267094 | -2181203 |
| 2 | 0.58 | 1.73 | -2190945 | -905775 |
| 3 | 0.28 | 0.88 | -909609 | -388468 |
| 4 | 0.20 | 0.62 | -447990 | -161837 |
| 5 | 0.11 | 0.34 | -167403 | -60264 |
| 6 | 0.06 | 0.20 | -60479 | -18845 |
| 7 | 0.03 | 0.09 | -18076 | -11830 |
| 8 | 0.05 | 0.16 | -26546 | -8507 |
| 9 | 0.03 | 0.08 | -8658 | -6969 |

**Table 5.10:** Error rate on the training set during each iteration. Also shown is the value of $\log_{1.001} R(\Theta)$, computed before and after reestimation on the reestimation set.

### 5.4.3.2. Global codebook exponents

As discussed in Chapter 4, each of the three sets of parameters may carry different amounts of useful (for recognition) information. It may be appropriate, then, to weigh the contribution of each of the parameter sets using so-called "codebook exponents". This is a type of refinement that fits quite naturally within the MMIE framework. We treat exponents as a set of parameters separate from the other parameters. Within the same iteration, each of the two parameter sets is estimated independently, assuming the other set fixed. Note that even though each estimate separately optimizes $R(\Theta)$, this may not

[5]Comparisons are made with the best MLE results in Table 5.3.

be true of the combined estimate. In order to avoid this problem, in our original experiments we estimated exponents in odd-numbered iterations and the other parameters in even-numbered iterations. It turns out that in practice this is unnecessary; the combined estimate works just as well and is twice as fast.

Exponents are estimated using a simple line search in the gradient direction. At each iteration, the initial step size is chosen so that no exponent changes by more than 10% of its original value. If this doesn't increase $R(\Theta)$, the step size is slowly reduced until a value for the exponents is found such that $R(\Theta)$ is greater than its original value.

The first experiment looks at the usefulness of *global codebook exponents*, thus called because the same fixed exponents are applied to all frames and distributions. Such exponents are attractive since they add very little complexity to the programs (they can be precomputed in the distributions).

### 5.4.3.2.1. First pass

Starting with the models obtained after 3 standard MLE iterations and codebook exponents initialized at 1.0, we obtained the recognition results as shown in Table 5.11.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| 1 | 1.10 | 3.25 | 46 | 84 | 183 |
| 2 | 1.00 | 2.98 | 42 | 70 | 174 |
| 3 | 0.92 | 2.76 | 39 | 56 | 169 |
| 4 | 0.84 | 2.54 | 31 | 50 | 160 |
| 5 | 0.86 | 2.59 | 35 | 48 | 162 |
| 6 | 0.88 | 2.65 | 36 | 49 | 167 |
| 7 | 0.86 | 2.60 | 38 | 43 | 164 |
| 8 | 0.83 | 2.51 | 36 | 42 | 160 |
| 9 | 0.85 | 2.58 | 38 | 42 | 163 |

**Table 5.11:** Error rate on the test set after each iteration of corrective MMIE training (using global codebook exponents).

This is an encouraging result since it corresponds to a word (string) error rate improvement of 8% (8%) over the previous result, and a 38% (34%) improvement over standard MLE training.

Table 5.12 shows the performance of the training algorithm on the training set, as well as the codebook exponents obtained after each iteration. As before, the value of $\log_{1.001} R(\Theta)$, computed on the reestimation set before and after reestimation with (4.12) and (4.18),[6] clearly demonstrates the effectiveness of the formula. Also, from the exponent values obtained, it seems clear that the second set of parameters ($\Delta C_1$ through $\Delta C_6$) contains more useful information than the other two sets. This indicates the usefulness of dynamic parameters.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | | Exponents | | |
|---|---|---|---|---|---|---|---|
| | word | string | before | after | 1 | 2 | 3 |
| 1 | 1.14 | 3.27 | -5767094 | -2181207 | 0.9000 | 1.0146 | 0.9182 |
| 2 | 0.57 | 1.68 | -1984268 | -825008 | 0.8190 | 1.0184 | 0.8712 |
| 3 | 0.28 | 0.87 | -764405 | -293995 | 0.7535 | 1.0242 | 0.8573 |
| 4 | 0.17 | 0.53 | -290326 | -78038 | 0.7007 | 1.0600 | 0.8057 |
| 5 | 0.10 | 0.27 | -80842 | -33795 | 0.7356 | 1.0239 | 0.7574 |
| 6 | 0.07 | 0.22 | -49859 | -15936 | 0.6989 | 1.0338 | 0.7633 |
| 7 | 0.04 | 0.12 | -19249 | -3970 | 0.6864 | 1.0204 | 0.7938 |
| 8 | 0.04 | 0.08 | -14746 | -2004 | 0.7045 | 1.0203 | 0.7700 |
| 9 | 0.04 | 0.12 | -7311 | -4661 | 0.7072 | 1.0282 | 0.7577 |

**Table 5.12:** Error rate on the training set during each iteration (with global codebook exponents). Also shown is the value of $\log_{1.001} R(\Theta)$, computed before and after reestimation on the reestimation set, as well as the exponents obtained after each iteration.

### 5.4.3.2.2. Second pass

We have discussed in Chapter 3 the importance of initial values used when training HMMs. As can be seen from the last results, the exponent values obtained at the end of training are quite different from the initial values. This suggest that these initial values are probably not the best with which to start training.

It may thus be interesting to repeat the whole training process (including bootstraping and MLE training), using, as initial exponent values, those obtained at the end of the first pass. Even though bootstraping and MLE training do not modify the exponent values, training will take the exponents into account and the parameters learned will be different from those learned

[6] Assuming the exponents fixed.

without exponents.

Table 5.13 shows the results obtained on the test set in this second training pass. The results confirm the importance of good initial exponents. The word (string) error rate is now 12% (14%) better than in the first pass and 45% (43%) better than with standard MLE training. Note that even the results after both bootstraping and standard MLE training are noticeably better than those obtained without exponents.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| boot | 1.59 | 4.57 | 71 | 92 | 290 |
| MLE-3 | 1.21 | 3.48 | 50 | 89 | 207 |
| 1 | 0.96 | 2.80 | 36 | 63 | 175 |
| 2 | 0.86 | 2.51 | 31 | 54 | 160 |
| 3 | 0.83 | 2.45 | 33 | 47 | 157 |
| 4 | 0.83 | 2.46 | 32 | 50 | 155 |
| 5 | 0.80 | 2.38 | 32 | 46 | 151 |
| 6 | 0.80 | 2.37 | 34 | 45 | 148 |
| 7 | 0.77 | 2.29 | 32 | 45 | 143 |
| 8 | 0.75 | 2.23 | 36 | 39 | 139 |
| 9 | 0.75 | 2.23 | 36 | 38 | 140 |

**Table 5.13:** Error rate on the test set after each iteration (second training pass with global codebook exponents).

Table 5.14 shows the performance of the training algorithm on the training set, as well as the codebook exponents obtained after each iteration.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | | Exponents | | |
|---|---|---|---|---|---|---|---|
| | word | string | before | after | 1 | 2 | 3 |
| 1 | 0.98 | 2.83 | -3761588 | -1493435 | 0.6365 | 1.0091 | 0.7139 |
| 2 | 0.50 | 1.49 | -1401493 | -413552 | 0.5792 | 0.9916 | 0.6832 |
| 3 | 0.22 | 0.65 | -408220 | -121876 | 0.5329 | 0.9881 | 0.6661 |
| 4 | 0.13 | 0.39 | -125959 | -19131 | 0.4978 | 0.9840 | 0.6195 |
| 5 | 0.07 | 0.24 | -41297 | -2198 | 0.5277 | 0.9710 | 0.6092 |
| 6 | 0.03 | 0.09 | -6452 | -748 | 0.5013 | 0.9814 | 0.6263 |
| 7 | 0.03 | 0.10 | -15023 | -881 | 0.4812 | 0.9756 | 0.6469 |
| 8 | 0.02 | 0.07 | -2175 | -106 | 0.4957 | 0.9754 | 0.6417 |
| 9 | 0.02 | 0.06 | -7933 | -2146 | 0.4858 | 0.9719 | 0.6535 |

**Table 5.14:** Error rate on the training set during each iteration (second training pass with global codebook exponents). Also shown is the value of $\log_{1.001} R(\Theta)$, computed before and after reestimation on the reestimation set, as well as the exponents obtained after each iteration.

### 5.4.3.3. Frame-dependent codebook exponents

The previous results have shown what can be gained by weighting the contribution of each parameter set in the probability computation. The exponents obtained are values which, on average, lead to better discrimination than unity exponents. It is not clear, however, that the same weighting is appropriate for all types of sounds.

In order to verify whether different codebook exponents should be used with different types of sounds, we have used a recurrent neural network (RNN) developed at CRIM to label the speech frames as one of three categories: sonorant/nasal, silence/noise and fricative/plosive. Then, for each frame, the exponents used were dependent on the category to which the frame was assigned by the RNN.

#### 5.4.3.3.1. First pass

This experiment is exactly the same as the first pass with global exponents, except that now there are 3 sets of three codebook exponents to train. Table 5.15 shows the results obtained on the test set.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| 1 | 1.08 | 3.18 | 48 | 84 | 175 |
| 2 | 0.95 | 2.82 | 42 | 69 | 161 |
| 3 | 0.87 | 2.64 | 35 | 57 | 156 |
| 4 | 0.83 | 2.50 | 35 | 48 | 154 |
| 5 | 0.83 | 0.50 | 38 | 45 | 154 |
| 6 | 0.81 | 2.45 | 34 | 46 | 152 |
| 7 | 0.81 | 2.42 | 33 | 45 | 153 |
| 8 | 0.77 | 2.31 | 33 | 42 | 145 |
| 9 | 0.78 | 2.36 | 36 | 42 | 145 |

**Table 5.15:** Error rate on the test set after each iteration (first training pass with frame-dependent codebook exponents).

This is a word (string) error rate improvement of 8% (9%) over what was obtained after the first pass with global codebook exponents. This is encouraging, even though it may not be significant. Table 5.16 shows the performance of the training algorithm on the training set.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | |
|---|---|---|---|---|
| | word | string | before | after |
| 1 | 1.14 | 2.27 | -5267094 | -2181231 |
| 2 | 0.57 | 1.68 | -2005903 | -846674 |
| 3 | 0.26 | 0.82 | -781706 | -338863 |
| 4 | 0.17 | 0.52 | -338471 | -145867 |
| 5 | 0.12 | 0.34 | -141112 | -67254 |
| 6 | 0.07 | 0.22 | -75117 | -25147 |
| 7 | 0.06 | 0.19 | -31277 | -9497 |
| 8 | 0.02 | 0.07 | -9488 | -5283 |
| 9 | 0.03 | 0.08 | -6647 | -3064 |

**Table 5.16:** Error rate on the training set during each iteration (first training pass with frame-dependent codebook exponents). Also shown is the value of $\log_{1.001} R(\Theta)$, computed before and after reestimation on the reestimation set.

Table 5.17 shows the exponents obtained at the end of the 9th iteration, for each of the 3 categories. We can see that there are noticeable differences between the categories. In particular, the silence/noise category puts almost equal weights on all parameter sets, which is quite different from the global codebook exponents obtained in the previous experiment.

| category | exponent | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| sonorant/nasal | 0.6679 | 1.0585 | 0.7903 |
| silence/noise | 0.9822 | 0.9794 | 1.0432 |
| fricative/plosive | 0.8559 | 1.0814 | 0.8277 |

**Table 5.17:** Exponents obtained at the end of the first pass of training with codebook-dependent exponent.

### 5.4.3.3.2. Second pass

The complete training process, (from bootstraping) is done all over again, using the exponents from Table 5.17 as initial exponents. Table 5.18 shows the results obtained on the test set. They show a word (string) error rate improvement of 6% (8%) over the results from the first pass, and of 3% (3%) over the results obtained with global codebook exponents after the second pass.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| bootstrap | 1.50 | 4.30 | 64 | 86 | 279 |
| MLE-3 | 1.14 | 3.22 | 46 | 81 | 197 |
| 1 | 0.91 | 2.65 | 39 | 61 | 160 |
| 2 | 0.80 | 2.37 | 36 | 50 | 143 |
| 3 | 0.77 | 2.26 | 37 | 44 | 138 |
| 4 | 0.76 | 2.23 | 31 | 45 | 141 |
| 5 | 0.76 | 2.21 | 39 | 42 | 136 |
| 6 | 0.77 | 2.29 | 42 | 41 | 138 |
| 7 | 0.73 | 2.14 | 32 | 41 | 134 |
| 8 | 0.73 | 2.16 | 36 | 42 | 130 |
| 9 | 0.73 | 2.16 | 37 | 40 | 132 |

**Table 5.18:** Error rate on the test set after each iteration (second training pass with frame-dependent codebook exponents).

Although this last improvement is quite marginal, we think the results are nonetheless significant with regard to demonstrating the concept of frame-dependent codebook weighting. Indeed, we could not expect to gain much information from the classes determined by the RNN.[7] Yet, it seems clear, both from the significant differences between the exponents for each category (Tables 5.17 and 5.20) and from the small recognition improvement, that the training algorithm was capable of learning how to usefully adapt codebook weighting to the sound category.

Table 5.19 shows the performance of the training algorithm on the training set.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | |
|---|---|---|---|---|
| | word | string | before | after |
| 1 | 0.98 | 2.82 | -3747997 | -1477450 |
| 2 | 0.46 | 1.40 | -1410572 | -557900 |
| 3 | 0.19 | 0.59 | -353049 | -67140 |
| 4 | 0.08 | 0.28 | -82773 | -16484 |
| 5 | 0.05 | 0.15 | -19463 | -8912 |
| 6 | 0.03 | 0.09 | -26985 | -5813 |
| 7 | 0.03 | 0.09 | -10384 | -876 |
| 8 | 0.02 | 0.07 | -2987 | -777 |
| 9 | 0.02 | 0.08 | -9053 | -950 |

**Table 5.19:** Error rate on the training set during each iteration (second training pass with frame-dependent codebook exponents). Also shown is the value of $\log_{1.001} R(\Theta)$, computed before and after reestimation.

[7]Moreover, the RNN used is several years old and its accuracy is unknown.

Table 5.20 shows the exponents obtained at the end of the 9th iteration, for each of the 3 categories.

| category | exponent | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| sonorant/nasal | 0.5208 | 1.0067 | 0.6702 |
| silence/noise | 0.9283 | 0.9622 | 1.0102 |
| fricative/plosive | 0.7448 | 1.1276 | 0.6955 |

**Table 5.20:** Exponents obtained at the end of the second pass of training with codebook-dependent exponent.

### 5.4.3.4. Frame-dependent weighting

This experiment attempts to verify the hypothesis that speech frames are not equally useful for recognizing words. The first thing needed is an automatic procedure that produces a frame labeling which somehow correlates with the frame's relative usefulness. If this is available, then MMIE can be used to learn the weight that should be applied to each category.

Such a labeler could for example be based on the assumption that redundant speech frames (those which are very similar to immediately preceding frames) are less useful than frames which are very different from those preceding it. In other words, speech frames should be more important when the signal changes rapidly. We decided to categorize this using the output from the second codebook, which encodes the dynamic changes in the cepstral coefficient. This thus results in 128 different categories. One advantage of this is that no additional signal processing is required. Instead, the same discrete parameter is used in two different ways. Note that global (frame-independent) codebook exponents are also used.

#### 5.4.3.4.1. First pass

This experiment is exactly the same as the first pass with global exponents, except that in addition to the 3 codebook exponents, a set of 128 exponents is also trained. The initial models used are the ones obtained after 3 iterations of standard MLE training. All exponents are initialized at 1.0. Table 5.21 shows the results obtained on the test set.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| 1 | 1.07 | 3.15 | 47 | 81 | 178 |
| 2 | 0.98 | 2.87 | 43 | 68 | 168 |
| 3 | 0.92 | 2.75 | 41 | 59 | 164 |
| 4 | 0.82 | 2.45 | 30 | 47 | 157 |
| 5 | 0.81 | 2.47 | 31 | 46 | 155 |
| 6 | 0.80 | 2.44 | 27 | 47 | 155 |
| 7 | 0.82 | 2.50 | 30 | 47 | 156 |
| 8 | 0.83 | 2.54 | 29 | 47 | 162 |
| 9 | 0.82 | 2.49 | 31 | 45 | 158 |

**Table 5.21:** Error rate on the test set after each iteration (first training pass with frame-dependent weighting).

Table 5.22 shows the performance of the training algorithm on the training set.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | | Exponents | | |
|---|---|---|---|---|---|---|---|
| | word | string | before | after | 1 | 2 | 3 |
| 1 | 1.14 | 3.27 | -5267094 | -2181219 | 0.9000 | 1.0146 | 0.9182 |
| 2 | 0.57 | 1.67 | -1912998 | -778350 | 0.8190 | 1.0186 | 0.8719 |
| 3 | 0.25 | 0.78 | -674851 | -300591 | 0.7535 | 1.0284 | 0.8524 |
| 4 | 0.16 | 0.50 | -270240 | -89697 | 0.7007 | 1.0528 | 0.8073 |
| 5 | 0.07 | 0.22 | -77966 | -17452 | 0.7091 | 1.0496 | 0.7586 |
| 6 | 0.04 | 0.13 | -16612 | -9525 | 0.6775 | 1.0389 | 0.7968 |
| 7 | 0.06 | 0.20 | -54941 | -7735 | 0.6852 | 1.0537 | 0.7649 |
| 8 | 0.02 | 0.06 | -7851 | -2288 | 0.6885 | 1.0295 | 0.7879 |
| 9 | 0.02 | 0.07 | -12809 | -2693 | 0.6748 | 1.0313 | 0.7956 |

**Table 5.22:** Error rate on the training set during each iteration (first training pass with frame-dependent weighting). Also shown is the value of $\log_{1.001} R(\Theta)$, computed before and after reestimation on the reestimation set, as well as the codebook exponents obtained after each iteration.

### 5.4.3.4.2. Second pass

The complete training process is started from bootstraping, using as initial exponents those obtained at the end of the first pass. Table 5.23 shows the results obtained on the test set.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| boot | 1.54 | 4.45 | 61 | 95 | 283 |
| MLE-3 | 1.18 | 3.35 | 45 | 93 | 199 |
| 1 | 0.91 | 2.62 | 33 | 66 | 161 |
| 2 | 0.84 | 2.45 | 30 | 59 | 150 |
| 3 | 0.75 | 2.22 | 27 | 48 | 139 |
| 4 | 0.79 | 2.36 | 27 | 49 | 149 |
| 5 | 0.78 | 2.34 | 33 | 48 | 142 |
| 6 | 0.79 | 2.36 | 28 | 52 | 145 |
| 7 | 0.80 | 2.37 | 34 | 48 | 145 |
| 8 | 0.81 | 2.41 | 37 | 47 | 147 |
| 9 | 0.83 | 2.49 | 42 | 46 | 149 |

**Table 5.23:** Error rate on the test set after each iteration (second training pass with frame-dependent weighting).

Table 5.24 shows the performance of the training algorithm on the training set.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | | Exponents | | |
|---|---|---|---|---|---|---|---|
| | word | string | before | after | 1 | 2 | 3 |
| 1 | 1.02 | 2.95 | -3326158 | -1073830 | 0.6073 | 1.0110 | 0.7359 |
| 2 | 0.46 | 1.39 | -992536 | -254541 | 0.5538 | 0.9681 | 0.6697 |
| 3 | 0.18 | 0.56 | -259860 | -45035 | 0.5095 | 0.9687 | 0.6538 |
| 4 | 0.08 | 0.27 | -44383 | -8218 | 0.5452 | 0.9129 | 0.6172 |
| 5 | 0.04 | 0.14 | -13243 | -1082 | 0.5657 | 0.9274 | 0.5802 |
| 6 | 0.07 | 0.22 | -42068 | -2021 | 0.5479 | 0.9208 | 0.6092 |
| 7 | 0.02 | 0.08 | -4984 | -1084 | 0.5394 | 0.9375 | 0.5848 |
| 8 | 0.01 | 0.05 | -4173 | -891 | 0.5232 | 0.9386 | 0.5938 |
| 9 | 0.01 | 0.03 | -2729 | -55 | 0.5337 | 0.9242 | 0.5850 |

**Table 5.24:** Error rate on the training set during each iteration (second training pass with frame-dependent weighting). Also shown is the value of $\log_{1.001} R(\Theta)$, computed before and after reestimation on the reestimation set, as well as the codebook exponents obtained after each iteration.

Clearly, as applied here, frame-dependent weighting is not useful at all. Note that the training algorithm did just as well as before (even better) on the training set. This should not be surprising since the additional 128 parameters simply provide additional degrees of freedom to the training algorithm. The problem is that whatever was learned has no generalization value. This seems clear by looking at the 128 exponents which, after 18 iterations of training (2 passes), did not change much from their original values.

These results, however, do not necessarily undermine the concept of frame-dependent weighting. The problem is probably one of finding an appropriate

labeling criterion. As mentioned in the last chapter, an interesting possibility might be to classify each speech frame in terms of how stressed the signal in the frame is.

### 5.4.3.5. Increasing the amount of training data

As mentioned earlier, one problem with the corrective MMIE training algorithm is that, as the recognition rate on the training set increases, the size of the reestimation set becomes very small (it may even completely vanish). One proposed way [LEEK 90] to alleviate this problem is to attempt to generate potential errors and then train on them.

Another possibility is to train not only on the incorrectly recognized strings in the training set, but also on those which had a close second choice. This is what we have chosen to do for this experiment. Using the N-best algorithm from Soong and Huang [SOON 90], we generated the 2 most likely digit strings for each string in the training set. All strings with an incorrect first choice were automatically selected for training. We then computed the average difference between the likelihood of the first and second choices for all the other strings, and added to the reestimation set all strings which had a difference smaller than 20% of the average difference. This guaranteed that the reestimation set was not reduced to a token number of strings.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| 1 | 0.98 | 2.87 | 41 | 61 | 177 |
| 2 | 0.88 | 2.61 | 41 | 61 | 177 |
| 3 | 0.84 | 2.49 | 37 | 48 | 155 |
| 4 | 0.83 | 2.44 | 31 | 51 | 155 |
| 5 | 0.80 | 2.36 | 32 | 51 | 145 |
| 6 | 0.84 | 2.49 | 32 | 52 | 155 |
| 7 | 0.83 | 2.45 | 34 | 51 | 151 |
| 8 | 0.79 | 2.34 | 30 | 51 | 145 |
| 9 | 0.79 | 2.34 | 30 | 51 | 145 |

**Table 5.25:** Error rate on the test set after each iteration of N-best training with global codebook exponents. The initial models used are the ones obtained after 3 iterations of MLE training in the second pass of training with global codebook exponents.

In order to save time, we started training with the MLE models obtained in

the second pass of the experiment with global codebook exponents. The initial exponents for parameter sets 1, 2 and 3 were thus 0.7072, 1.0282 and 0.7577, respectively. The recognition results on the test set are shown in Table 5.25.

The results are not quite as good as those obtained with the standard corrective MMIE training algorithm, although the difference may not be significant. One explanation, however, for this deterioration may be found in Table 5.26, which describes the performance of the algorithm during training. Indeed, note from the "before" and "after" columns that the modified reestimation formula doesn't seem as effective when correctly recognized strings are added to the reestimation set. This is especially true for the two iterations which had no incorrectly recognized strings as part of the reestimation set. These two iterations resulted in substantial degradations in the value of $\log_{1.001} R(\Theta)$.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | | fraction of |
|---|---|---|---|---|---|
| | word | string | before | after | training set |
| 1 | 0.98 | 2.84 | -10299253 | -5520558 | 0.054 |
| 2 | 0.47 | 1.42 | -6167800 | -3858200 | 0.036 |
| 3 | 0.19 | 0.58 | -4497131 | -3126228 | 0.033 |
| 4 | 0.09 | 0.29 | -3893807 | -1672855 | 0.030 |
| 5 | 0.03 | 0.10 | -3557109 | -2643450 | 0.028 |
| 6 | 0.01 | 0.03 | -3498021 | -2428867 | 0.027 |
| 7 | 0.00 | 0.00 | -3407676 | -91779282 | 0.026 |
| 8 | 0.00 | 0.00 | -3274827 | -91779892 | 0.026 |
| 9 | 0.01 | 0.02 | -2986947 | -1568665 | 0.024 |

**Table 5.26:** Error rate on the training set during each iteration (N-best training with global codebook exponents). Also shown is the value of $\log_{1.001} R(\Theta)$, computed before and after reestimation on the reestimation set, as well as the fraction of the total number of strings from the training set that was used for reestimation.

### 5.4.4. Recognition experiments with semi-continuous HMMs

Earlier, we presented the results of a convergence experiment with the reestimation formulas for continuous densities proposed in this thesis. This section now looks at whether their use can translate into better recognizers. This will also be an opportunity to compare the performance of semi-continuous and discrete HMMs.

### 5.4.4.1. One model

This experiment starts with what has so far been called the "second pass" of training. We use codebook exponents obtained after an earlier MMIE training experiment with semi-continuous HMMs as our initial exponents. The initial exponents are, in order, 0.6941, 1.0222 and 0.7287. Table 5.27 shows the results obtained on the test set. The result at the end of training corresponds to a word (string) error rate reduction of 46% (43%) compared to the results obtained after 3 standard (i.e., no exponents) MLE iterations (see Table 5.5). It also corresponds to word (string) improvement of 12% (10%) over the results obtained with discrete HMMs in the same conditions.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| boot | 1.44 | 0.96 | 68 | 85 | 257 |
| MLE-3 | 0.96 | 2.85 | 37 | 70 | 168 |
| 1 | 0.71 | 2.15 | 26 | 62 | 126 |
| 2 | 0.72 | 2.14 | 36 | 46 | 121 |
| 3 | 0.61 | 1.84 | 24 | 39 | 112 |
| 4 | 0.61 | 1.85 | 24 | 36 | 114 |
| 5 | 0.67 | 2.00 | 37 | 37 | 116 |
| 6 | 0.65 | 1.98 | 33 | 33 | 120 |
| 7 | 0.63 | 1.90 | 35 | 34 | 110 |
| 8 | 0.64 | 1.96 | 41 | 34 | 109 |
| 9 | 0.66 | 2.01 | 47 | 30 | 112 |

**Table 5.27:** Error rate on the test set after each iteration using semi-continuous HMMs with global codebook exponents. The initial models used are the ones obtained after 3 iterations of MLE training in the second pass of training with global codebook exponents.

Table 5.28 shows the performance of the training algorithm on the training set. For semi-continuous experiments, the value of $\log_{1.001} R(\Theta)$ is computed after reestimation of both the discrete and continuous parameters (with the exponents unchanged). Observe that the error rate on the training set is always lower than in the corresponding discrete case. This means that the size of the reestimation set is also always smaller. The last training iteration was done using 3 digit strings (out of 8616), which is probably not a reasonable thing to do.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | | Exponents | | |
|---|---|---|---|---|---|---|---|
| | word | string | before | after | 1 | 2 | 3 |
| 1 | 0.89 | 2.56 | -3278371 | -993317 | 0.624713 | 0.981255 | 0.680461 |
| 2 | 0.39 | 1.18 | -923233 | -262261 | 0.568489 | 0.965428 | 0.638151 |
| 3 | 0.15 | 0.49 | -261732 | -42987 | 0.523010 | 0.907878 | 0.627088 |
| 4 | 0.08 | 0.26 | -65732 | -5070 | 0.559621 | 0.880285 | 0.617109 |
| 5 | 0.05 | 0.16 | -20751 | -834 | 0.535705 | 0.907287 | 0.580082 |
| 6 | 0.06 | 0.15 | -15611 | -556 | 0.535857 | 0.886985 | 0.609086 |
| 7 | 0.03 | 0.08 | -10805 | -579 | 0.518053 | 0.883373 | 0.633449 |
| 8 | 0.02 | 0.07 | -5908 | -394 | 0.502511 | 0.888198 | 0.647999 |
| 9 | 0.01 | 0.03 | -1211 | -669 | 0.512561 | 0.888812 | 0.637920 |

**Table 5.28:** Error rate on the training set during each iteration (semi-continuous HMMs with global codebook exponents).

### 5.4.4.2. Separate male and female models

This last experiment looks at how to perform MMIE training when multiple models per unit are used, and at how this can improve results. It also looks at the effect of increasing the number of MLE iterations before MMIE training. This last point seemed relevant since our other experiences with semi-continuous HMMs (in particular for wordspotting applications) tend to show that SCHMMs require more MLE training iterations than do discrete HMMs. The fact that the continuous mixture components are shared by all distributions may in part explain why this should be the case.

We use separate male and female models, which means that there are now twice as many models as previously. Note, however, that the same tied mixtures are used for both male and female models. Since the information about the sex of speakers is available in the database, it is relatively straightforward to train models on speakers of the corresponding sex. This is what we do for both bootstraping and MLE training. Table 5.29 shows how the error rate on the test set changes as the number of MLE iterations is increased.

| Iteration | word | string | ins | del | sub |
|---|---|---|---|---|---|
| 3 | 0.75 | 2.36 | 39 | 77 | 99 |
| 4 | 0.74 | 2.27 | 39 | 75 | 96 |
| 5 | 0.70 | 2.18 | 38 | 72 | 91 |
| 6 | 0.72 | 2.21 | 39 | 72 | 94 |
| 7 | 0.73 | 2.23 | 40 | 69 | 99 |
| 8 | 0.71 | 2.16 | 38 | 69 | 95 |
| 9 | 0.72 | 2.19 | 38 | 67 | 100 |

**Table 5.29:** Error rate on the test set after additional MLE training iterations.

After seeing these results, we decided to perform two sets of MMIE training experiments: one after 3 iterations of MLE training (as previously) and one after 6 iterations.

It is not immediately clear how MMIE training should be done when several models per unit are used, especially when, as is our case, the clusters (male & female) are determined *a priori* and the information about each speaker's cluster is available from the database. The question is whether or not we should enforce the sex of speakers in the training process. If we did, it would, in effect, add sex recognition to the problem of digit recognition. Since this is not useful for our purpose, we decided not to use the information about sex during MMIE training. Suppose one of the digit training sequences contains $w$=5-9-6. Then the "good" model used for training is the one illustrated in Figure 5.7. In all cases, the model $m_{gen}$ will be the one illustrated in Figure 5.8.

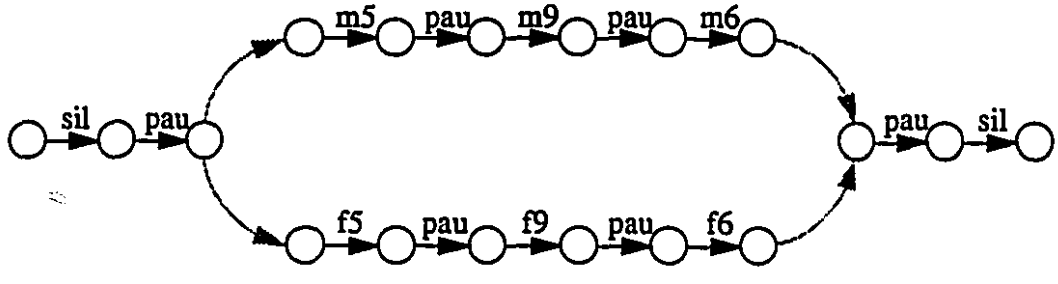


Figure 5.7: Model $m_w$ for $w$=5-9-6.

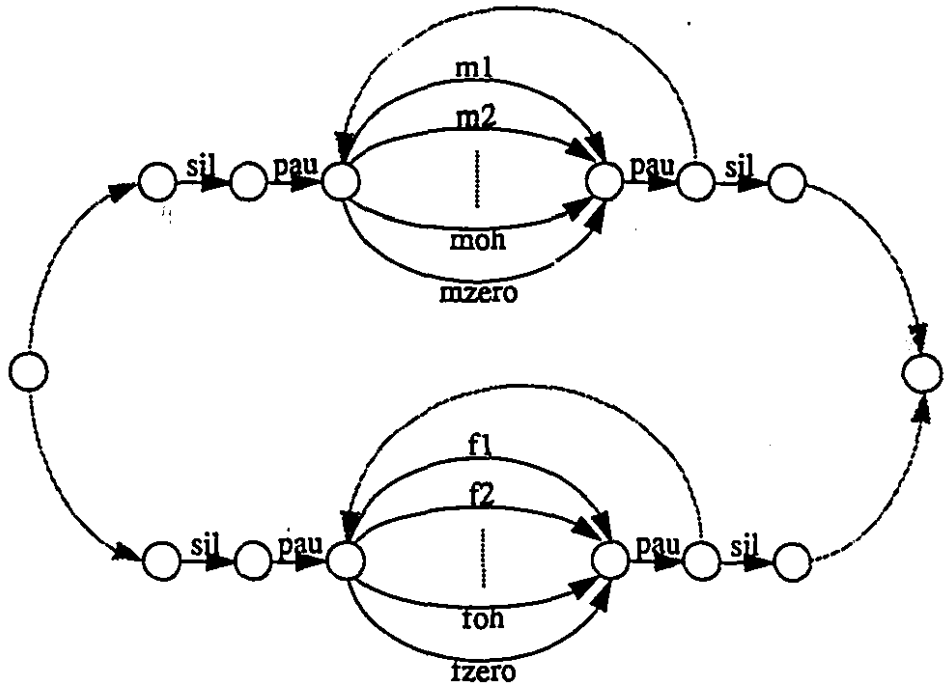


**Figure 5.8:** Model $m_{gen}$ used for training with separate male and female models.

Table 5.30 shows the recognition results on the test set after each iteration of MMIE training. The initial models used were those obtained after 3 MLE iterations. In order to determine if taking into account a larger number of mixture components can improved results, we performed two sets of recognition experiments: One using the best 3 components and one using the best 6 components. In both cases, the models used are those trained using only the best 3 components.

| Iter | best 3 codewords | | | | | best 6 codewords | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | word | string | ins | del | sub | word | string | ins | del | sub |
| 1 | 0.66 | 2.00 | 39 | 62 | 86 | 0.64 | 1.96 | 35 | 62 | 85 |
| 2 | 0.63 | 1.93 | 42 | 52 | 87 | 0.61 | 1.86 | 37 | 51 | 85 |
| 3 | 0.57 | 1.71 | 39 | 38 | 86 | 0.55 | 1.67 | 39 | 41 | 76 |
| 4 | 0.59 | 1.80 | 43 | 39 | 87 | 0.56 | 1.74 | 43 | 36 | 82 |
| 5 | 0.56 | 1.73 | 45 | 29 | 86 | 0.52 | 1.59 | 45 | 26 | 76 |
| 6 | 0.51 | 1.62 | 37 | 26 | 83 | 0.49 | 1.53 | 38 | 23 | 79 |
| 7 | 0.53 | 1.65 | 47 | 22 | 82 | 0.57 | 1.77 | 45 | 29 | 88 |
| 8 | 0.49 | 1.52 | 41 | 20 | 78 | 0.48 | 1.50 | 42 | 16 | 80 |
| 9 | 0.51 | 1.59 | 45 | 21 | 79 | 0.49 | 1.51 | 45 | 16 | 78 |

**Table 5.30:** Error rate on the test after each MMIE training iteration (semi-continuous HMMs with global codebook exponents, separate male and female models). The initial models used are those obtained after 3 MLE iterations.

Table 5.31 shows the performance of the training algorithm on the training set.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | | Exponents | | |
|---|---|---|---|---|---|---|---|
| | word | string | before | after | 1 | 2 | 3 |
| 1 | 0.58 | 1.72 | -3094683 | -1125788 | 0.907877 | 1.019112 | 0.900000 |
| 2 | 0.24 | 0.75 | -1021651 | -442032 | 0.826168 | 1.018070 | 0.852678 |
| 3 | 0.12 | 0.38 | -435294 | -199868 | 0.777467 | 1.030542 | 0.784464 |
| 4 | 0.10 | 0.30 | -183330 | -42012 | 0.746291 | 1.041456 | 0.729552 |
| 5 | 0.04 | 0.14 | -46524 | -8139 | 0.701514 | 1.009579 | 0.709844 |
| 6 | 0.04 | 0.13 | -13680 | -190 | 0.666438 | 0.987393 | 0.691150 |
| 7 | 0.01 | 0.03 | -14427 | -1 | 0.658413 | 1.026889 | 0.679909 |
| 8 | 0.03 | 0.09 | -10323 | -184 | 0.647485 | 1.020205 | 0.700306 |
| 9 | 0.01 | 0.03 | -3838 | -72 | 0.657150 | 1.040609 | 0.689485 |

**Table 5.31:** Error rate on the training set during each iteration (semi-continuous HMMs with global codebook exponents, separate male and female models). The initial models used are those obtained after 3 MLE iterations.

Now, using as initial models those produced with 6 MLE iterations, we obtained the results on the test set as shown in Table 5.32, and results on the training set as shown in Table 5.33. In this case, training had to be stopped after 7 iterations since, as the recognition on the training set was 100%, there was not any data to use for training. Although it is difficult to determine which combination works best, it is clear that the training algorithm performs really well and the results obtained on the test set are very good.

| Iter | best 3 codewords | | | | | best 6 codewords | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | word | string | ins | del | sub | word | string | ins | del | sub |
| 1 | 0.62 | 1.92 | 35 | 59 | 83 | 0.60 | 1.82 | 31 | 56 | 83 |
| 2 | 0.53 | 1.65 | 31 | 45 | 75 | 0.49 | 1.54 | 25 | 41 | 75 |
| 3 | 0.53 | 1.63 | 37 | 34 | 79 | 0.48 | 1.47 | 36 | 30 | 70 |
| 4 | 0.52 | 1.62 | 35 | 35 | 77 | 0.44 | 1.39 | 27 | 30 | 70 |
| 5 | 0.51 | 1.61 | 30 | 32 | 85 | 0.47 | 1.45 | 25 | 30 | 78 |
| 6 | 0.56 | 1.71 | 34 | 34 | 92 | 0.50 | 1.55 | 32 | 32 | 79 |
| 7 | 0.51 | 1.58 | 31 | 29 | 86 | 0.50 | 1.55 | 35 | 29 | 78 |

**Table 5.32:** Error rate on the test after each MMIE training iteration (semi-continuous HMMs with global codebook exponents, separate male and female models). The initial models used are those obtained after 6 MLE iterations.

| Iteration | Error | | $\log_{1.001} R(\Theta)$ | | Exponents | | |
|---|---|---|---|---|---|---|---|
| | word | string | before | after | 1 | 2 | 3 |
| 1 | 0.48 | 1.44 | -2801591 | -928698 | 0.900765 | 1.018678 | 0.900000 |
| 2 | 0.20 | 0.63 | -859489 | -326391 | 0.819696 | 1.027538 | 0.835109 |
| 3 | 0.10 | 0.33 | -300554 | -75060 | 0.754120 | 1.037853 | 0.774689 |
| 4 | 0.10 | 0.30 | -120180 | -13408 | 0.762586 | 1.026837 | 0.720461 |
| 5 | 0.04 | 0.12 | -28092 | -1275 | 0.716831 | 1.027858 | 0.738279 |
| 6 | 0.01 | 0.02 | -9150 | 1 | 0.728267 | 0.985781 | 0.701365 |
| 7 | 0.02 | 0.06 | -10359 | -94 | 0.720362 | 0.975759 | 0.729420 |
| 8 | 0.00 | 0.00 | - | - | - | - | - |

**Table 5.33:** Error rate on the training set during each iteration (semi-continuous HMMs with global codebook exponents, separate male and female models). The initial models used are those obtained after 6 MLE iterations.

## 5.5. Summary of results

Table 5.34 compares the results obtained in the different experiments in this chapter. In all cases, we report the best result for a given type of experiment. These result exhibit a clear superiority of semi-continuous HMMs over discrete HMMs. This merely confirms what several other researchers have recently found out. It also confirms the usefulness of more specific models (male and female in our case).

These results are only side-effects of our experiments. What we wanted to verify was the usefulness of MMIE as a training framework for HMMs. There seems to be little doubt about this.

| Experiment | word | string | ins | del | sub |
|---|---|---|---|---|---|
| Discrete MLE | 1.36 | 3.90 | 50 | 112 | 225 |
| Discrete MMIE | 0.91 | 2.75 | 38 | 59 | 163 |
| Discrete MMIE + exponents | 0.75 | 2.23 | 36 | 38 | 140 |
| Discrete MMIE + CD-exponents | 0.73 | 2.16 | 37 | 40 | 132 |
| SCHMM MLE | 1.22 | 3.51 | 47 | 99 | 202 |
| SCHMM MMIE + exponents | 0.61 | 1.84 | 24 | 39 | 112 |
| SCHMM 2 models MLE | 0.70 | 2.18 | 39 | 72 | 94 |
| SCHMM 2 models MMIE + exponents | 0.44 | 1.39 | 27 | 30 | 70 |

**Table 5.34:** Summary of results

# 6. CONCLUSION

Looking back at everything that has been accomplished within such simple a framework as HMMs, it is difficult not to be a little surprised. Yet, it is now clear that the statistical approach to speech recognition, as implemented by HMMs, is indeed very powerful. Some of the progress realized over the past years with HMMs is, of course, the result of trial and error; however, much of the success has resulted from a better understanding of their strengths and weaknesses.

This thesis has thoroughly reviewed the theory and the practice of HMMs, as they relate to the problem of speech recognition. Along the way, we described some of the improvements which have helped to make HMMs such a successful technology. We felt this was important in order to provide the necessary insight into the problem.

As we have seen, there are many different ways of making HMMs work better. For example, improving the front-end by finding better speech features (or descriptors) to extract from the signal and by making it more robust to changes in the acoustical environment; finding ways of quickly adapting HMM parameters to changes in speaker or environmental characteristics; improving language modeling at all levels (syntactic, semantic and pragmatic) and integrating the language models into the search strategies (realizing that many applications do not require that every single word be correctly recognized); finding better speech units; and improving training techniques. Each of these constitutes an area of research in its own right and each promises to improve, yet again, the performance of HMM-based systems.

We have chosen to work on the training problem and, because we felt it was more intuitively appealing than MLE, to concentrate on the MMIE framework. In the process, our hope was not to replace MLE with MMIE. After all, as is the case for MLE, MMIE needs good initial models to perform well,

and MLE seems to be a method well-suited to produce these models. In fact, our hope was not even to establish MMIE as an indispensable part of everybody's training process (although for us, in many cases, it is). Instead, what we wanted to do was to add tools to those already available to designers of speech recognition systems and to demonstrate their effectiveness. We especially hoped that this work would increase general understanding of HMMs and add further insight into the speech recognition problem.

## 6.1. Contributions

We have shown that using MMIE following MLE can result in significantly improved recognition rates, compared to MLE alone. In fact, using MMIE with the techniques presented in this thesis has allowed us to obtain, with a fairly simple system, recognition rates better than the best results published to date on the TI/NIST connected digit task.

More significant, however, than the absolute recognition rates obtained, is the general applicability of the techniques presented. It is quite probable that better results could be obtained using MLE alone in a more complex system (full covariance densities, more specific models, second derivative parameters, etc.); however, there is little doubt that, given enough training data, even these MLE results could be substantially improved using the MMIE techniques described in this thesis.

We have introduced an efficient new training algorithm, "Corrective MMIE training", which has allowed us to obtain these improvements with a small number of iterations, each of which is usually faster than a standard MLE training iteration.

This algorithm is the result of a modification that we introduced into a reestimation formula for discrete distributions proposed by Gopalakrishnan *et al.*, and of the idea of only using errors in the training set for reestimating the HMM parameters. Taken separately, none of these ideas would have performed very well; however, taken together they led to systematically fast convergence in practice.

We proposed a new derivation for the discrete reestimation formula. This allowed us to derive new reestimation formulas for the case of Gaussian densities with diagonal covariance. We demonstrated convergence in practice and also effectively used the procedures in a speech recognition system with semi-continuous distributions. This resulted in a joint optimization of the codebook of tied densities and of the discrete distributions of mixture weights and transition probabilities.

We proposed a method for performing MMIE training with multiple model per word and we demonstrated its effectiveness in practice. We proposed a way of using an N-best search algorithm to generate more training data. This technique could also be used elsewhere, such as in *corrective training*. Moreover, this could also be used to generate a good approximation to the denominator of the MMIE objective function.

We have shown how HMMs can be improved by the introduction of a small number of additional parameters and how MMIE, contrary to MLE, can be used to learn these new parameters. We have shown that this can be very effective in a speech recognition system.

## 6.2. Discussion and future work

In many ways, the results reported are very encouraging. There are not many experiments that we have made with MMIE which have not resulted in substantial improvements over MLE alone. Yet, in order to correctly assess their real significance, it is important to take into account the characteristics of the task that was used for the experiments.

One of the first characteristics that comes to mind is the very small vocabulary. This is an important point because it allows the use of a very simple looped model ($m_{gen}$) to represent all possible models in the task. A similar model for, say, a triphone-based, large vocabulary continuous speech application would either have to be extremely big or to grossly over-generate. Suppose, for example, that we use for $m_{gen}$ a big looped model with all triphones in parallel. Then, even if the rules about the legal triphone transitions are observed, most of the paths in that looped model will not correspond to legal

English words (not to mention sentences). Moreover, those that do will not use anything close to the "true" language model.

For the connected digit task, the language model was probably not an important issue. The implicit language model in $m_{gen}$ was obviously wrong, but not dramatically so. Moreover, as some simple demonstrations have shown, MMIE is probably robust to incorrect modeling assumptions such as this one. There is no point, however, in training a system to make distinctions it will never need to do, as would be the case with the triphone looped model. It is, of course, possible to improve the model somewhat using bigram probabilities for triphone transitions, but this is as far as it is possible to go with a looped model. Training with such a model could probably improve recognition of connected phones, but it is not clear that this would translate into better word accuracy.

For such applications, then, it may be better to approximate the denominator of (4.9) with a summation over the word sequences which could most easily be confused with the true sentence. These word sequences could be found using one of the recently introduced "N-best" algorithms and the approximation would probably be very good.

Another important characteristic of the connected digit task is the large size of the database and the fact that the training set was designed to be "representative" of the entire task. This makes it more difficult for us to explain the sizable differences between our results on the training and on testing sets. Yet, since a large proportion of errors usually comes from a small number of "bad" speakers (such as HM, CS or LE), we believe the training set may still not be "representative" enough. More training data, then, might be necessary. This brings us back to the question of how much data is enough training data. As we discussed in Chapter 3, the answer to that question depends to a large extent on the characteristics of the model used; this is certainly as valid for MMIE as it is for MLE.

Our experience shows that, if enough information is available in the training data, MMIE will be very good at using that information in order to reduce the error rate on the training set to almost nothing. This, however, doesn't mean that results will be similar on a different set. In fact, as many researchers

have often found out, the opposite may very well happen. If it does, it can of course result from mismatches between training and testing sets. In our task, however, it is more likely that this would be the result of attempting to learn parameters which just don't have real global significance for the task. A case in point is the frame-dependent weighting experiment that we performed: It worked wonderfully on the training set, but degraded results on the test set.

In reality, some HMM parameters with global usefulness may not require a large amount of training data to be well-estimated. Take, for instance, the global codebook exponents that were used in most of our experiments. Initially, we trained them on a very small number of male speakers from the training set. It turns out that the exponents obtained that way were quite similar to those we now obtain with our "integrated" training procedure. This, we think, is because the values obtained really reflect the relative usefulness (for our task) of the various sets of parameters, which applies similarly to all speakers. These exponents, however useful we now know them to be, do not fit well within the MLE framework and could not have been learned in it. Codebook exponents are very similar to the factor that is often used to increase the contribution of the language model to the log likelihood of a sentence. They both arise because of incorrect modeling assumptions in HMMs. So far, however, the language model factor has been determined empirically. This could be done automatically with MMIE.

This brings us to a discussion of one of the ways in which MMIE could help us most to improve our speech recognizers. We discussed in Chapter 1 the importance of the front-end and how knowledge about HMMs should be used in its design. The goal is, of course, to extract features which will allow HMMs, as a speech recognition tool, to perform as well as possible. It is quite possible that the best way of achieving this would be to *not* consider feature extraction and HMMs a separate entities, but rather as two parts of an integrated recognition tool.

After all, it is the HMMs themselves which are the most capable of determining what they need from the front-end in order to improve their recognition performance. This seems a natural application for MMIE training. The idea is as follows. Feature extraction is a mathematical transformation that takes signal samples as input and produces features as output. Even though the

commonly used transformations (FFT or LPC-based cepstral analysis, delta parameters, etc.) perform reasonably well, they might be far from optimal for speech recognition. It is possible, however, to implement feature extraction using neural networks as *function approximators*. Then, integrating the neural network within the HMMs, the gradient of the MMIE objective function with respect to its parameters could be computed, thus allowing it to be optimized at the same time as the other HMM parameters.

There are two important points to make. First, as in most such optimization problems, the initialization is very important. An obvious solution is to initialize the neural network in such a way that it approximates a type of feature extraction which is known to perform well. Second, since many parameters will be estimated, it is probable that a large amount of training data will be required in order to obtain an optimum which can be generalized. Such experiments, if done extensively, could substantially increase our knowledge about the types of speech features which are most useful to HMMs. This, indeed, would be a valuable result.

Our experience with MMIE training so far has been that there usually is something (often substantial) to be gained from its direct use in training. As a result, corrective MMIE training is now well integrated into our HMM software package and is part of our standard training procedure.

# REFERENCES


[ACER 90] A. Acero, "Acoustical and Environmental Robustness in Automatic Speech Recognition", *Ph.D. thesis,* Carnegie Mellon University, Pittsburg, September 1990

[APPL 89] T.H. Applebaum and B.A. Hanson, "Enhancing the discrimination of speaker independent Hidden Markov Model with corrective training", *Proc. ICASSP-89,* pp. 302-305, Glasgow, 1989

[BAHL 83] L.R. Bahl, F. Jelinek and R.L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 2, March 1983

[BAHL 86] L.R. Bahl, P.F. Brown, P.V. de Souza and R.L. Mercer, "Maximum Mutual Information Estimation of Hidden Markov Model Parameters for Speech Recognition", *Proc. ICASSP-86,* pp. 49-52, Tokyo, 1986

[BAHL 88a] L.R. Bahl, P.F. Brown, P.V. de Souza, R.L. Mercer and M.A. Picheny, "Acoustic Markov Models Used in the Tangora Speech Recognition System", *Proc. ICASSP-88,* paper S11.3, pp. 497-500, New-York, 1988

[BAHL 88b] L.R. Bahl, P.F. Brown, P.V. de Souza and R.L. Mercer, "A New Algorithm for the Estimation of Hidden Markov Model Parameters", *Proc. ICASSP-88,* pp. 493-496, New-York, 1988

[BAHL 91] L.R. Bahl, P.V. de Souza, P.S. Gopalakrishnan, D. Nahamoo, and M.A. Picheny, "Context Dependent Modeling of Phones in Continuous Speech Using Decision Trees", *Proceedings of the DARPA Speech Recognition Workshop, February 1991*

[BAUM 72] L.E. Baum, "An Inequality and Associated Maximization Technique in Statistical Estimation of Probabilistic Functions of Markov Processes", *Inequalities* 3:1-8, 1972

[BELL 89] J.R. Bellegarda and D. Nahamoo, "Tied Mixtures Continuous Parameter Models for Large Vocabulary Isolated Speech Recognition", *Proc. ICASSP-89,* pp. 13-16, Glasgow, 1989

[BELL 90] J.R. Bellegarda and D. Nahamoo, "Tied Mixtures Continuous Parameter Modeling for Speech Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. ASSP-38, no. 12, December 1990

[BOCC 86] E.L. Bocchieri and G.R. Doddington, "Frame-Specific Statistical Features for Speaker-Independent Speech Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. ASSP-34, no. 4, August 1986

[BROW 87] P.F. Brown, "The Acoustic-Modeling Problem in Automatic Speech Recognition", *Ph.D. Thesis,* Carnegie Mellon University, Pittsburg, May 1987

[BUSH 86] M.A. Bush and G.E. Kopec, "Network-Based Connected Digit Recognition Using Explicit Acoustic-Phonetic Modeling", *Proc. ICASSP-86,* Tokyo, pp. 1097-1100, April 1986

[BUSH 87] M.A. Bush and G.E. Kopec, "Network-Based Connected Digit Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. ASSP-35, no. 10, October 1987

[CARD 91] R. Cardin, Y. Normandin, and R. De Mori, "High Performance Connected Digit Recognition Using Maximum Mutual Information Estimation", to be published in *Proc. ICASSP-91,* Toronto, May 1991

[CHOW 86] Y.L. Chow *et al,* "The Role of Word-Dependent Coarticulatory Effects in a Phoneme-Based Speech Recognition System", *Proc. ICASSP-86,* Tokyo, pp. 1593-1596, April 1986

[CHOW 87] Y.-L. Chow, M.O. Dunham, O.A. Kimball, M.A. Krasner, G.F. Kubala, J. Makhoul, P.J. Price, S. Roucos, and R.M. Schwartz, "BYBLOS: The BBN Continuous Speech Recognition System", *Proc. ICASSP-87,* pp. 89-92, Dallas, 1987

[CHOW 90] Y.L. Chow, "Maximum Mutual Information Estimation of HMM Parameters for Continuous Speech Recognition using The N-Best Algorithm", *Proc. ICASSP-90,* paper S13.6, Albuquerque, April 1990

[COHE 89] J.R. Cohen, "Application of an Auditory Model to Speech Recognition", *Journal of the Acoustical Society of America*, 85(6), June 1989

[COX 89] S.J. Cox and J.S. Bridle, "Unsupervised Speaker Adaptation by Probabilistic Spectrum Fitting", *Proc. ICASSP-89*, paper S6.11, Glasgow, 1989

[DAVI 80] S.B. Davis and P. Mermelstein, "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-28, no. 4, August 1980

[DENG 90] L. Deng, M. Lenning, F. Seitz and P. Mermelstein, "Large Vocabulary Word Recognition Using Context-Dependent Allophonic Hidden Markov Models", *Computer Speech and Language*, vol. 4, no. 4, October 1990

[DEVI 82] P.A. Devijver and J. Kittler, "Pattern Recognition: A Statistical Approach", *Prentice-Hall International*, London, 1982

[DODD 89] G.R. Doddington, "Phonetically Sensitive Discriminants for Improved Speech Recognition", *Proc. ICASSP-89*, paper S10b.11, Glasgow, 1989

[EPHR 87] Y. Ephraim, A. Dembo, and L.R. Rabiner, "A Minimum Discrimination Information Approach for Hidden Markov Modeling", *Proc. ICASSP-87*, paper 1.8.1, pp. 25-28, Dallas, 1987

[EPHR 88] Y. Ephraim and L.R. Rabiner, "On the Relations Between Modeling Approaches for Information Sources", *Proc. ICASSP-88*, pp. 24-27, New-York, 1988

[EPHR 89] Y. Ephraim, A. Dembo and L.R. Rabiner, "A Minimum Discrimination Information Approach for Hidden Markov Modeling", *IEEE Transactions on Information Theory*, vol. 35, no. 5, September 1989

[EQUI 89] W.H. Equitz, "A New Vector Quantization Clustering Algorithm", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-37, no. 10, October 1989

[FENG 88] M.-W. Feng, "Improved Speaker Adaptation using Text Dependent Spectral Mapping", *Proc. ICASSP-88*, paper S3.9, New-York, 1988

[FENG 89] M.-W. Feng, R. Schwartz, F. Kubala, J. Makhoul, "Iterative Normalization for Speaker-Adaptive Training in Continuous Speech Recognition", *Proc. ICASSP-89,* pp. 612-615, Glasgow, 1989

[FURU 86] S. Furui, "Speaker-Independent Isolated Word Recognition Using Dynamic Features of Speech Spectrum", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. ASSP-34, no. 1, February 1986

[FURU 89] S. Furui, "Unsupervised Speaker Adaptation Method Based on Hierarchical Spectral Clustering", *Proc. ICASSP-89,* paper S6.9, Glasgow, 1989

[GAUV 91] J.L. Gauvain and C.H. Lee, "Bayesian Learning of Gaussian Mixture Densities for Hidden Markov Models" *Proceedings of the DARPA Speech Recognition Workshop, February 1991*

[GISH 90] H. Gish, Y.L. Chow, and J.R. Rohlicek, "Probabilistic Vector Mapping of Noisy Speech Parameters for HMM Word Spotting", *Proc. ICASSP-90,* paper S2.21, Albuquerque, April 1990

[GOPA 88] P.S. Gopalakrishnan, D. Lanevsky, A. Nádas, D. Nahamoo, M.A. Picheny, "Decoder Selection based on cross-entropies", *Proc. ICASSP-88,* pp. 20-23, New-York, 1988

[GOPA 89] P.S. Gopalakrishnan, D. Kanevsky, A. Nádas, and D. Nahamoo, "A Generalization of the Baum Algorithm to Rational Objective Functions", *Proc. ICASSP-89,* paper S12.9, Glasgow, 1989

[GRAY 84] R.M. Gray, "Vector Quantization", *IEEE ASSP Magazine,* April 1984

[GUPT 87] V.N. Gupta, M. Lenning, and P. Mermelstein, "Integration of Acoustic Information in a Large Vocabulary Word Recognizer", *Proc. ICASSP-87,* paper 17.2.1, pp. 697-700, Dallas, 1987

[HON 90] H.W. Hon and K.F. Lee, "On Vocabulary-Independent Speech Modeling", *Proc. ICASSP-90,* paper S14.2, Albuquerque, April 1990

[HUAN 89] X.D. Huang and M.A. Jack, "Semi-Continuous Hidden Markov Models for Speech Signals", *Computer Speech and Language,* vol. 3, no. 3, July 1989

[HUAN 90] X. Huang, K.F. Lee, and H.W. Hon, "On Semi-Continuous Hidden Markov Modeling", *Proc. ICASSP-90*, paper S13.3, Albuquerque, April 1990

[HUNT 89] M.J. Hunt and C. Lefebvre, "Distance Measures for Speech Recognition", *Aeronautical Note NAE-AN-57*, National Research Council Canada, NRC no. 30144, March 1989

[JELI 76] F. Jelinek, "Continuous Speech Recognition by Statistical Methods", *Proceedings of the IEEE*, vol. 64, no. 4, April 1976

[JUAN 85] B.-H. Juang and L.R. Rabiner, "Mixture Autoregressive Hidden Markov Models for Speech Signals", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-33, no. 6, December 1985

[KOPE 85] G.E. Kopec, and M.A. Bush, "Network-Based Isolated Digit Recognition Using Vector Quantization", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-33, no. 4, August 1985

[KUBA 88] F. Kubala *et al.*, "Continuous Speech Recognition Results of the BYBLOS System on the DARPA 1000-Word Resource Management Database", *Proc. ICASSP-88*, pp. 291-294, New-York, April 1988

[KUBA 90] F. Kubala, R. Schwartz, and C. Barry, "Speaker Adaptation from a Speaker-Independent Training Corpus", *Proc. ICASSP-90*, paper S3.2, Albuquerque, April 1990

[KUBA 91] F. Kubala, S. Austin, C. Barry, J. Makhoul, P. Placeway, and R. Schwartz, "BYBLOS Speech Recognition Benchmark Results", *Proceedings of the DARPA Speech Recognition Workshop, February 1991*

[KUHN 90] R. Kuhn, and R. De Mori, "A Cache-Based Natural Language Model for Speech Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990

[LAME 81] L.F. Lamel, L.R. Rabiner, A.E. Rosenberg, and J.G. Wilpon, "An Improved Endpoint Detector for Isolated Word Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-29, no. 4, August 1981

[LAME 86] L.F. Lamel, R.H. Kassel, and S. Seneff, "Speech Database Development: Design and Analysis of the Acoustic-Phonetic

Corpus", *Proceedings of the DARPA Speech Recognition Workshop*, 1986

[LEEC 89b] C.H. Lee and L.R. Rabiner, "A Frame-Synchronous Network Search Algorithm for Connected Word Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-37, no. 11, November 1989

[LEEC 90a] C.H. Lee, L.R. Rabiner, R. Pieraccini, and J.G. Wilpon, "Acoustic Modeling for Large Vocabulary Speech Recognition", *Computer Speech and Language*, vol. 4, no. 2, April 1990

[LEEC 90b] C.H. Lee, C.H Lin, and B.H. Juang, "A Study on Speaker Adaptation of Continuous Density HMM Parameters", *Proc. ICASSP-90*, paper S3.4, Albuquerque, April 1990

[LEEK 88] K.-F. Lee, "Large-Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System", *Ph.D. thesis*, Carnegie Mellon University, Pittsburg, April 1988

[LEEK 89a] K.-F. Lee, and H.-W. Hon, "Speaker-Independent Phone Recognition Using Hidden Markov Models", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-37, no. 11, November 1989

[LEEK 89b] K.-F. Lee, H.-W. Hon, M.-Y. Hwang, S. Mahajan, and R. Reddy, "The SPHINX Speech Recognition System", *Proc. ICASSP-89*, paper S9.3, Glasgow, May 1989

[LEEK 90] K.-F. Lee, and S. Mahajan, "Corrective and Reinforcement Learning for Speaker-Independent Continuous Speech Recognition", *Computer Speech and Language*, vol. 4, no. 3, July 1990

[LEON 84] R. G. Leonard, "A Database for Speaker-Independent Digit Recognition", *Proc. ICASSP-84*, paper 42.11, 1984

[LEVI 83] S.E. Levinson, L.R. Rabiner, and M.M. Sondhi, "An Introduction to the Application of the Theory of Probabilistic Function of a Markov Process to Automatic Speech Recognition", *The Bell System Technical Journal*, vol. 62, no. 4, Apr. 1983.

[MAKH 85] J. Makhoul and S. Roucos, "Vector Quantization in Speech Coding", *Proceedings of the IEEE*, vol. 73, no. 11, November 1985

[MERI 88] B. Merialdo, "Phonetic Recognition using Hidden Markov Models and Maximum Mutual Information Training", *Proc. ICASSP-88*, paper S3.4, New-York, 1988

[MURV 91] H. Murveit, J. Butzberger, and M. Weintraub, "Speech Recognition in SRI's Resource Management and ATIS Systems", *Proceedings of the DARPA Speech Recognition Workshop, February 1991*

[MYER 81] C. S. MYERS and L. R. Rabiner, "A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-29, no. 2, April 1981

[NADA 83] A. Nádas, "A Decision Theoretic Formulation of a Training Problem in Speech Recognition and a Comparison of Training by Unconditional Versus Conditional Maximum Likelihood", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-31, no. 4, August 83, pp. 814-817

[NADA 88] A. Nádas, D. Nahamoo, and M.A. Picheny, "On a Model-Robust Training Method for Speech Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-36, no. 11, September 1988, pp. 1432-1436

[NADA 89] A. Nádas, D. Nahamoo, and M.A. Picheny, "Speech Recognition Using Noise-Adaptive Prototypes", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-37, no. 11, November 1989, pp. 1495-1503

[PAUL 89] D.B. Paul, "The Lincoln Robust Continuous Speech Recognizer", *Proc. ICASSP-89*, paper S9.4, Glasgow, May 1989

[PAUL 91] D.B. Paul, "New Results with the Lincoln Tied-Mixture HMM CSR System", *Proceedings of the DARPA Speech Recognition Workshop, February 1991*

[PICO 86] J. Picone, K.M. Goudie-Marshall, G.R. Doddington, and W. Fisher, "Automatic Text Alignment for Speech System Evaluation", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-34, no. 4, August 1986

[RABI 83a] L.R. Rabiner, M.M. Sondhi, and S.E. Levinson, "Note on the Properties of a Vector Quantizer for LPC Coefficients", *The Bell Systems Technical Journal*, vol. 62, no. 8, October 1983

[RABI 83b] L.R. Rabiner, S.E. Levinson, and M.M. Sondhi, "On the Application of Vector Quantization and Hidden Markov Models to Speaker-Independent Isolated Word Recognition", *The Bell Systems Technical Journal*, vol. 62, no. 4, April 1983

[RABI 85a] L. R. Rabiner and S. E. Levinson, "A Speaker-Independent, Syntax-Directed, Connected Word Recognition System Based on Hidden Markov Models and Level Building", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-33, no. 3, June 1985.

[RABI 85b] L.R. Rabiner, B.H. Juang, S.E. Levinson, M.M. Sondhi, "Recognition of Isolated Digits Using Hidden Markov Models with Continuous Mixture Densities", *AT&T Technical Journal*, vol. 64, no. 6, July-August 1985

[RABI 86] L. R. Rabiner and B. H. Juang, "An Introduction to Hidden Markov Models", *IEEE ASSP Magazine*, January 1986

[RABI 88] L.R. Rabiner, J.G. Wilpon and F.K. Soong, "High Performance Connected Digit Recognition, Using Hidden Markov Models", *Proc. ICASSP-88*, paper S3.6, New-York, April 1988

[RABI 89a] L.R. Rabiner, C.H. Lee, B.H. Juang, and J.G. Wilpon, "HMM Clustering for Connected Word Recognition", *Proc. ICASSP-89*, paper S8.5, Glasgow, 1989

[RABI 89b] L. R. Rabiner, J.G. Wilpon, and F.K. Soong, "High Performance Connected Digit Recognition Using Hidden Markov Models", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-37, no. 8, August 1989

[RABI 89c] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *Proceedings of the IEEE*, vol. 77, no. 2, February 1989

[ROHL 89] J. R. Rohlicek, W. Russel, S. Roukos and H. Gish, "Continuous Hidden Markov Modeling for Speaker-Independent Word Spotting", *Proc. ICASSP-89*, April 1989

[ROSE 90] R. C. Rose and D. B. Paul, "A Hidden Markov Model Based Keyword Recognition System", *Proc. ICASSP-90*, Albuquerque, April 1990

[RTIS 89] D. Rtischev, "Speaker Adaptation in a Large-Vocabulary Speech Recognition System", *Masters Thesis*, MIT, 1989

[SCHW 85] R. Schwartz, Y. Chow, O. Kimball, S. Roucos, M. Krasner, J. Makhoul, "Context-Dependent Modeling for Acoustic-Phonetic Recognition of Continuous Speech", *Proc. ICASSP-85,* April 1985

[SCHW 87] R. Schwartz, Y.-L. Chow and F. Kubala, "Rapid Speaker Adaptation Using Probabilistic Spectral Mapping", *Proc. ICASSP-87,* pp. 633-636, Dallas, 1987

[SCHW 89] R. Schwartz et al., "Robust Smoothing Methods for Discrete Hidden Markov Models", *Proc. ICASSP-89,* pp. 548-551, Glasgow, 1989

[SCHW 90] R. Schwartz and Y.-L. Chow, "The N-Best Algorithm: An Efficient and Exact Procedure for Finding the N Most Likely Sentence Hypotheses", *Proc. ICASSP-90,* paper S2.12, Albuquerque, April 1990

[SENE 88] S. Seneff, "A Joint Synchrony/Mean-Rate Model of Auditory Speech Processing", *Journal of Phonetics*, 16, 1988

[SOON 90] F.K. Soong and E.-F. Huang, "A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition", Japan, December 1990

[VANC 87] D. Van Compernolle, "Increased noise immunity in large vocabulary speech recognition with the aid of spectral subtraction", *Proc. ICASSP-87,* Dallas, 1987

[VANC 89] D. Van Compernolle, "Noise adaptation in a hidden Markov model speech recognition system", *Computer Speech and Language 3,* 1989

[VITE 67] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", *IEEE Transactions on Information Theory,* vol. 13, no. 2, April 1967

[WEIN 89] M. Weintraub et al, "Linguistic Constraints in Hidden Markov Model Based Speech Recognition", *Proc. ICASSP-89,*, paper S13.2, Glasgow, 1989

[WILP 90] J.G. Wilpon, L.R. Rabiner, C.-H. Lee, and E.R. Goldman, "Automatic Recognition of Keywords in Unconstrained Speech Using Hidden Markov Models", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. ASSP-38, no. 11, November 1990

[ZUE 85] V. Zue, "The Use of Speech Knowledge in Automatic Speech Recognition", *Proceedings of the IEEE*, vol. 73, no. 11, November 1985

[ZUE 90] V. Zue *et al.*, "The VOYAGER Speech Understanding System: Preliminary Development and Evaluation", *Proc. ICASSP-90,*, paper S2.9, Albuquerque, April 1990