THE USE OF DISCRETE DISTRIBUTIONS WITH A VERY LARGE CODEBOOK FOR AUTOMATIC SPEECH RECOGNITION AND SPEAKER VERIFICATION

by

GUOLI YE

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
in Computer Science and Engineering

January 2013, Hong Kong

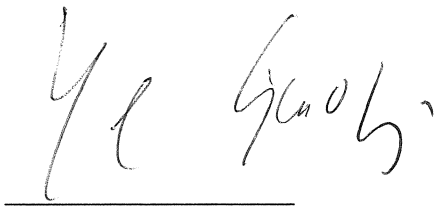
HKUST Library
Reproduction is prohibited without the author's prior written consent

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.



GUOLI YE

THE USE OF DISCRETE DISTRIBUTIONS WITH A VERY LARGE CODEBOOK FOR AUTOMATIC SPEECH RECOGNITION AND SPEAKER VERIFICATION

by

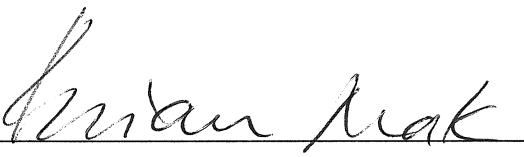
GUOLI YE

This is to certify that I have examined the above Ph.D. thesis

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by

the thesis examination committee have been made.



PROF. BRIAN MAK, THESIS SUPERVISOR

PROF. LEI CHEN, ACTING HEAD OF DEPARTMENT

Department of Computer Science and Engineering

28 January 2013

ACKNOWLEDGMENTS

First of all, I would like to express sincere thanks to my supervisor Prof. Brian Mak, for his kindness, support and patience. His curiosity about a problem, and the optimistic attitude towards research and life has influenced me a lot. I also learnt a lot from his wonderful presentation and writing skills.

I would like to thank members of my PhD thesis examination committee, Prof. James Kwok, Prof. Dit-Yan Yeung, Prof. Chi-Ying Tsui, and Prof. Helen Meng. I would also like to thank Prof. Raymond Wong for serving as the committee member of my thesis proposal defense.

I want to thank my colleagues Benny Ng, Tom Ko, and Dong-Peng Chen, I learnt a lot from them. I would also like to thank Prof. Man-Wai Mak in PolyU for his collaboration on a speaker verification task, who also generously provided me some of the data used in the thesis.

I would like to thank CSE department communication tutor Shauna Dalton, who kindly help proofreading this thesis. I would also like to thank CSE department staff, Connie Lau and Isaac Ma, who are always helpful regarding administrative issues. Thanks also go to the cssystem members, who maintain the cluster machines which are often occupied by my jobs.

Last but not least, I would like to say thanks to my parents and my wife Ye Su. Without their love, nothing would have been possible.

TABLE OF CONTENTS

Title Page i

Authorization Page ii

Signature Page iii

Acknowledgments iv

Table of Contents v

List of Figures ix

List of Tables x

Abstract xi

Chapter 1 Introduction 1

1.1 Problems with Very Large Codebook Discrete Systems 3

1.2 Contributions of the Thesis 4

1.3 Thesis Outline 6

Chapter 2 Introduction to Automatic Speech Recognition (ASR) 8

2.1 Definition of ASR 8

2.1.1 Statistical ASR 9

2.1.2 ASR System Performance Evaluation Criterion 10

2.2 Hidden Markov Model (HMM) for ASR 11

2.2.1 Dynamic Features for HMM 12

2.2.2 Training of and Recognition with HMM in ASR 13

2.3 Different Types of HMM in ASR 14

2.3.1 Continuous Density HMM (CDHMM) 14

2.3.2 Semi-continuous HMM (SCHMM) 16

2.3.3 Discrete HMM (DHMM) 16

2.3.4 Multiple-stream HMM 17

Chapter 3 Related Work in DHMM **20**

3.1 Different Types of Codebooks 20

3.1.1 Codebook by Unsupervised Construction 20

3.1.2 Codebook by Supervised Construction 22

3.2 Robust Estimation of State Output Probability Distribution 23

3.3 Very Large Codebook DHMM 25

Chapter 4 Very Large Codebook DHMM for ASR **27**

4.1 Common Experimental Setup and Definitions 28

4.1.1 Experimental Setup 28

4.1.2 Stream Definition 30

4.1.3 Subvector-quantized (SVQ) & Scalar-quantized (SQ) Codebooks 31

4.1.4 Subvector-partition and Bit-allocation Schemes 34

4.1.5 Subvector vs. Stream 35

4.2 Starting Point 37

4.2.1 High-density DHMM (HDDHMM) System 37

4.2.2 Baseline Model Performance 40

4.3 The Increase of Codebook Size 43

4.4 SQ to SVQ Codebooks 46

4.4.1 Difficulties in Conversion from CDHMM 46

4.4.2 Conversion by Numerical Integration 47

4.4.3 Conversion by Single-point Approximation 52

4.4.4 Experiments 53

4.5 The Use of Fewer Streams 56

4.5.1 Problems in Multiple-stream Systems 56

4.5.2 SVQ-HDDHMM with Fewer Streams 57

4.6 Classical Robust Parameter Estimation Techniques 59

4.6.1 Baum-Welch Re-estimation with Flooring 60

4.6.2 Deleted Interpolation 63

4.7 Subspace-HDDHMM (SHDDHMM) for Robust Parameter Estimation 65

4.7.1 One-stream SHDDHMM with Constrained Bases 66

4.7.2 Training of One-stream SHDDHMM 69

4.7.3 Design Logic of SHDDHMM 69

4.7.4 Multiple-stream SHDDHMM 71

4.7.5 Experiments 72
4.8 Stream Weight Estimation by Iterative Linear Programming (ILP) 77
4.8.1 Modified ILP Method for Stream Weight Estimation 79
4.8.2 Experiments 84
4.9 Summary on WSJ SI-84 Task 85
4.9.1 Review of Various Techniques for HDDHMM 85
4.9.2 Recognition Time 86
4.9.3 Bandwidth Requirement 87
4.10 Comparison with Other Researches 88
4.10.1 Decision Tree Based Acoustic Model 88
4.10.2 DHMM with Maximum Mutual Information (MMI) Tree Based Codebook 90
4.11 Experiments on RM and WSJ SI-284 Tasks 95
4.11.1 Optimal Model Configurations 96
4.11.2 Results 96
4.12 Summary 97

Chapter 5 Very Large Codebook Discrete Model for Speaker Verification (SV) 99

5.1 Introduction to SV 100
5.1.1 The Problem 100
5.1.2 Feature 101
5.1.3 Gaussian Mixture Model Universal Background Model (GMM-UBM) for SV 101
5.1.4 Evaluation Criterion 105
5.2 Related Speedup Methods for GMM-UBM Based Systems 106
5.3 High-density Discrete Model (HDDM) for SV 107
5.4 Experiments 108
5.4.1 SV Performance of Baseline GMM-UBM Systems 109
5.4.2 SQ-HDDM vs. SVQ-HDDM 109
5.4.3 Operating Characteristics of Various Models 111
5.5 Summary 112

Chapter 6 Conclusions and Future Work 114

6.1 Conclusions 114
6.2 Future Work 115

Appendix A L-BFGS Method for Node Splitting in MMI-tree Based Codebook Construction **117**

References **120**

LIST OF FIGURES

2.1 Phone HMM /i/: first-order 3-state left-to-right topology 11

2.2 Word HMM "it": constructed from phone HMMs /i/ and /t/ 11

2.3 Different types of HMM 15

2.4 Examples of 2-stream HMM systems 18

3.1 Codebooks produced by unsupervised and supervised construction. Each method partitions the acoustic space into 2 parts. The dots represent training samples in codebook construction, with colors representing different classes. 22

4.1 SVQ codebooks, where x_t is a d-dimensional feature vector at frame t, and VQ(x_t) is its corresponding full space VQ codeword constructed from SVQ or SQ codewords. Notice that when $L = d$, it becomes SQ codebooks. 32

4.2 An example of a 2-stream system with SVQ codebooks. Each stream is further split into 2 subvectors with 2 SVQ codebooks. 36

4.3 Relationship between model size and recognition error rate for HD-DHMM with SQ codebooks 45

4.4 One-stream SHDDHMM architecture overview. The shaded ellipse represents the global pool of bases, spanning the subspace. Each state output discrete distribution table (i.e., b_i, and b_j), lies in the subspace. The state-dependent weights (the dotted lines) and the global pool of bases (the shaded ellipse) are temporary parameters, which only exist during model training; the final model only stores b_i and b_j. 67

4.5 An example of a 2-stream SHDDHMM, each stream is further split into 2 subvectors. A global pool of bases is stored for each stream independently, and the number of bases in a pool is 3. 73

4.6 Stream weight estimation results for SHDDHMM with iterative linear programming 85

4.7 Operating characteristics of various SI-84 models (finding codeword time is included for discrete models.) 88

4.8 An example of "smoothing by adding 1" technique 94

5.1 An example of computer access control using speaker verification technology 100

5.2 GMM-UBM based speaker verification system 102

5.3 SV operating characteristics of various models, when the UBM and speaker models are pre-loaded to the memory. 113

5.4 SV operating characteristics of various models, when the UBM and speaker models are loaded on-the-fly. 113

LIST OF TABLES

4.1 Experimental settings for different ASR systems 29

4.2 Bit allocation for each stream for HDDHMM in [53] 38

4.3 Baseline model performance on WSJ SI-84 42

4.4 Effect of codebook size for 4s-HDDHMM with SQ codebooks 44

4.5 Comparison of different integration routines 52

4.6 Different subvector-partition and bit-allocation schemes for SVQ-HDDHMM 54

4.7 Performance of 4-stream SVQ-HDDHMM using different subvector-partition bit-allocation schemes and conversion methods 54

4.8 Performance of CDHMM with different stream definitions 56

4.9 Performance of SVQ-HDDHMM with different stream definitions 58

4.10 Performance of parameter re-estimation for SQ-HDDHMM (SQ-a) with flooring 10^{-12} 61

4.11 Performance of deleted interpolation for SQ-HDDHMM (SQ-a) 65

4.12 Performance of 4stream SCHMM with different pool size 72

4.13 Different subvector-partition and bit-allocation schemes for SHDDHMM 74

4.14 SHDDHMM performance with different schemes 75

4.15 Effect of different techniques for HDDHMM on WSJ SI-84 Task 86

4.16 Performance of DHMM with MMI-tree codebook on WSJ SI-84 task 94

4.17 Subvector-partition and bit-allocation schemes for SHDDHMM in RM and WSJ SI-284 96

4.18 Model performance (in WER) on different ASR tasks 97

5.1 EER performance of 1-stream or 2-stream GMMs of varied number of mixtures, M. (S = #streams; all/top5 = GMM likelihood computation using all/top 5 Gaussian components.) 109

5.2 2-stream HDDMs of varied resolutions. (w = subspace dimension. Probabilities are assumed 4-byte floating point numbers. The model size is in megabytes.) 110

5.3 SV performance of various SQ-HDDMs and SVQ-HDDMs converted from 2-stream GMMs with 1024 components 110

THE USE OF DISCRETE DISTRIBUTIONS WITH A VERY LARGE CODEBOOK FOR AUTOMATIC SPEECH RECOGNITION AND SPEAKER VERIFICATION

by

GUOLI YE

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

ABSTRACT

With the advance of semiconductor technology and the popularity of distributed speech/speaker recognition paradigm (e.g., Siri in iPhone4s), here we revisit the use of *discrete model* in automatic speech recognition (ASR) and speaker verification (SV) tasks. Compared with the dominant *continuous density model*, discrete model has inherently attractive properties: it uses non-parametric output distributions and takes only $O(1)$ time to get the probability value from it; furthermore, the features used in the discrete model, compared with that in the continuous model, could be encoded in fewer bits, lowering the bandwidth requirement in distributed speech/speaker recognition architecture. Unfortunately, the recognition performance of a conventional discrete model is significantly worse than that of a continuous one due to the large quantization error and the use of multiple independent streams. In this thesis, we propose to reduce the quantization error of a discrete model by using a very large codebook with *tens of thousands* of codewords. The codebook of the proposed model is about a hundred times larger than that of a conventional discrete model, whose codebook size usually ranges from 256 to 1024. Accordingly, the number of parameters to specify a discrete output distribution grows by a hundred times in the proposed model.

Compared with a discrete model of conventional sized codebook, there are two major challenges in building a very large codebook model. Firstly, given a continuous acoustic feature vector, how do we quickly find its corresponding codeword from a hundred-time larger codebook? Secondly, given the limited amount of training data, how can we robustly train such a high-density model, which has a hundred times more parameters than the conventional model?

To find a codeword for an acoustic vector fast, we employ the *subvector-quantized (SVQ)* codebooks. SVQ codebooks represent a very large codebook in the full feature space by a combinatorial product of per-subvector smaller codebooks. To find a full space codeword is reduced to finding a set of SVQ codewords, which is very fast.

To robustly train such a high-density model, two techniques are explored. The first one is to do model conversion. A discrete model is converted directly from a well-trained continuous model and avoids direct training using the training data. The second one is by subspace modeling. In this technique, the original high-density discrete distribution table is treated a high dimensional vector and assumed to lie in some low dimensional subspace. By this subspace representation, the number of free parameters in the model is reduced by ten and hundred fold. As a result, the model could be trained robustly using the limited amount of data.

Experimental evaluations on both ASR and SV tasks show the feasibility and benefits of the very large codebook discrete model. On the WSJ0 (Wall Street Journal) ASR task, the proposed model shows comparable recognition accuracy as the continuous model with much faster decoding and lower bandwidth requirement. On the NIST (National Institute of Standards and Technology) 2002 SV task, a speedup of 8-25 fold is achieved with almost no loss in verification performance.

CHAPTER 1

INTRODUCTION

Due to its simple parametric form and good generalization ability, the continuous distribution, specifically, Gaussian mixture model (GMM), is commonly used to model acoustic feature distributions, which characterizes either a sub-phoneme in a state-of-the-art Hidden Markov Model (HMM) based automatic speech recognition (ASR) system, or a speaker in the classical GMM-based speaker recognition system. The discrete distribution, which was used in the early history of ASR and speaker recognition, is almost abandoned nowadays, mainly because its recognition accuracy is not as good as that of GMM [35, 55].

This thesis, however, revisits the use of discrete distributions in ASR and speaker recognition. Compared with continuous distributions, discrete distributions are attractive for the following reasons:

- Fast recognition: in GMM-based ASR and speaker recognition systems, the time spent in GMM computation (i.e., to compute the probability of an acoustic feature given the GMM) during recognition is quite high: in ASR, depending on the task, the GMM computation takes about 30$\sim$70% of the total recognition time [28] (the rest of the time is spent searching for promising word sequences); in speaker recognition, almost all the recognition time is dedicated to GMM evaluation. What is worse, the computation time grows linearly with the number of Gaussian components. Nowadays, systems are usually large and complex, with a large mixture of Gaussians, thus resulting in a heavy computation burden in recognition. In a discrete distribution, it pre-stores probabilities to generate each possible discrete feature in a discrete probability table. To get the probability of an acoustic feature only requires an $O(1)$ time table lookup, which is independent of the table size. As a result, what is equivalent to the evaluation of GMM likelihood almost takes no time in discrete distributions, resulting in much faster recognition.

- Simple parameter estimation: the estimation of discrete distributions is merely a bin counting process; while in GMM parameter estimation, the expectation-maximization (E-M) algorithm [20] is usually adopted, which requires several iterations to obtain the local rather than global optimum solution.

- Bandwidth saving in *distributed speech/speaker recognition systems* [73, 63]: distributed systems become more common with the emergence of cloud computing technology (one example is: the Siri in iPhone4s which uses Nuance speech recognition technology [15]). In these systems, recognition components are distributed between clients and servers. Speech features may be extracted locally at a client, and then transmitted to a remote server, where recognition is carried out. Usually, the discrete features used by discrete distributions could be encoded in much fewer bits than the continuous features used in GMM, lowering the bandwidth requirement.

- Non-parametric from: different from the parametric GMM, the discrete distribution uses a non-parametric probability mass function (PMF). In theory, both GMM and PMF could represent arbitrary distribution (given the infinite number of mixtures for GMM, or infinitely large codebook for discrete distributions). In other words, there is no theoretical reason to tell which model is better. However, in the past, limited by the computational resources (e.g., not enough memory to support very large codebooks), the discrete distributions usually showed inferior performance. With the advance of computer technology, it is interesting to revisit the non-parametric discrete distribution. The research in this thesis is also in line with the recent trend to apply non-parametric models to speech processing [19, 21, 75, 71].

Despite all these merits, the recognition error rate using discrete distributions in a speech processing task is much higher compared with GMM. In ASR, it was generally believed that the recognition error rates with conventional discrete distribution systems [1] are a factor of one-and-a-half or two times higher than their continuous density counterpart. In speaker recognition, it was observed in [55] that: under different speaking rates, the speaker identification error rate of discrete systems was consistently $6\sim7$ absolute points higher than that of continuous systems. Large quantization errors have been blamed for the unsatisfactory performance of discrete distribution. Unlike GMM

[1] Throughout the thesis, discrete distribution systems are also called "discrete systems" for simplicity.

which has a smooth probability density function, discrete distribution imposes hard partitions in the feature space, with each partition represented by a codeword. Any different feature vectors falling into the same partition are quantized into the same codeword, which inevitably causes quantization errors. In the past, due to the memory limit, and the amount of training data, the number of codewords used cannot be too large (e.g., a 12 dimensional feature space was typically quantized into 256$\sim$1024 codewords). Such a small number of codewords is far from enough to quantize the acoustic feature space and usually results in a large quantization error, which badly hurts the recognition performance.

With the advance of semiconductor technology and its falling price, and the availability of very large speech corpus, the thesis tries to answer the question: could we improve the recognition accuracy of discrete distribution systems by making use of the rich resources available nowadays? A simple solution is adopted in this thesis:

Reduce the quantization error in discrete distributions by using a very large codebook.

In this thesis, the "very large codebook" refers to a codebook consisting of *tens of thousands* of codewords, in contrast to the conventional codebook with 256$\sim$1024 codewords.

1.1 Problems with Very Large Codebook Discrete Systems

Even though the requirement of large memory may be easily satisfied nowadays, many important techniques used by discrete speech/speaker recognition systems with conventional sized codebooks cannot be applied to systems with codebooks that are hundreds of times larger. Specifically, there are two problems:

- Codebook problem: in conventional discrete systems, a linear codebook is commonly used, where the feature space is vector-quantized (VQ) to a set of codewords by unsupervised k-means clustering. The time to lookup the corresponding codeword given an acoustic vector is proportional to the size of the codebook (i.e., the number of codewords). The linear codebook works well for conventional systems with only 256$\sim$1024 codewords. However, it is not appropriate for systems with 10,000$\sim$65,000 codewords. Firstly, to run k-means with so

many clusters would be very time consuming. Secondly, to find a codeword for an acoustic vector will also be 10$\sim$100 times longer when compared with a conventional system. As a result, recognition speed will be seriously affected.

- Parameter estimation problem: a conventional discrete system uses a simple bin counting method to estimate the discrete distributions. To estimate the output probability of a codeword, it simply counts the number of times the codeword appears in the training data, and then divides it by the total number of training samples. The bin counting method is problematic in very large codebook systems with hundreds of times more codewords: due to the lack of training data and the naturally uneven distribution of training samples, many codewords are rarely seen in the training set. As a result, many codewords have very small or zero output probability in many models. Such a poorly estimated system would perform badly on unseen data.

1.2 Contributions of the Thesis

This thesis investigates various issues when building very large codebook systems, with a focus on the above two problems. The work here is based on our research group's first attempt of a very large codebook system for automatic speech recognition (ASR) back in 2005 [53]. To clearly state the contributions of the thesis, a brief introduction of that system is needed.

In [53], to solve the codebook problem, scalar-quantized (SQ) codebooks are adopted, which represent a very large codebook in full space by a combinatorial product of per-dimensional small SQ codebooks. For example, in a 13-dimensional feature space, a large codebook with 2^{13} codewords could be represented by 13 per-dimensional SQ codebooks, each with only 2 codewords. The codebook construction is to build a set of SQ codebooks, one for each dimension. Finding the codeword for an acoustic feature is reduced to getting a set of SQ codewords from each of the SQ codebooks. Since each SQ codebook is very small (e.g., 2$\sim$4 codewords), both operations could be very fast.

To solve the parameter estimation problem, [53] proposed to convert the discrete model from a well-trained continuous model, which solves the data scarcity problem by avoiding direct training of the large-codebook system. Based on the SQ codebooks,

the conversion turns out to be simple: it only involves per-dimensional integrals with closed form solutions. On a standard English large vocabulary read speech recognition task, called WSJ0 SI-84, the large-codebook system showed a significant improvement compared with conventional discrete systems. However, there is still a big gap compared with continuous system in recognition accuracy.

The contributions of the thesis could be summarized as follows:

- We extend the scalar-quantized (SQ) codebook in [53] to subvector-quantized (SVQ) codebook. That is, a codebook is built per subvector rather than per dimension, with each subvector containing one or more feature dimensions. SVQ codebook was shown to result in smaller quantization error than SQ codebook at any given bit rate [23] in speech coding, and was also applied successfully on some ASR tasks [22, 9]. We verifies that SVQ codebook is also more effective in our large-codebook system and gives higher recognition accuracy. Furthermore, unlike SQ codebook, there is no closed form solution when converting from a continuous system to SVQ-codebook based system. We propose to use numerical integrals by developing a simple yet effective matlab routine.

- A novel large-codebook system based on subspace modeling is proposed, which results in tens even hundreds of fold reduction for the number of free parameters. As a result, training such a large codebook system with limited amount of data becomes feasible. The trained model shows significantly better recognition accuracy than a model that is converted from continuous model without any re-training.

- We significantly narrow the performance gap between discrete and continuous systems in terms of recognition error rate. For example, on the WSJ0 speech recognition task, we reduce the recognition error rate 7.90% of a conventional discrete model to 4.65%, which makes it comparable with the continuous system with error rate 4.46%, with the advantage of 25% speedup during recognition by cutting almost all the GMM computation time of the continuous system.

- We demonstrate the effectiveness of a large codebook system on applications that require many GMM computations, where almost all the recognition time is devoted to GMM computation. For example, on the National Institute of Standards and Technology (NIST) 2002 speaker verification task, the large-codebook

discrete system leads to an overall speedup by a factor of 8$\sim$25 with comparable recognition performance to GMM based system.

- Besides, we systematically investigate the following issues of a large-codebook system:
  - the effect of the codebook size
  - the use of different number of streams in a multi-stream system (the multi-stream system is introduction in Section 2.3.4).
  - the use of a modified iterative linear programming technique to discriminatively estimate the stream weights in the multi-stream system
  - the comparison of our large-codebook system with other proposed discrete systems

Since the research on large-codebook discrete systems is still at its early stage, this thesis is far from providing a perfect solution to it. Instead, we hope to share the experiences and lessons with other researchers in this field.

1.3 Thesis Outline

Since most of the proposed techniques used in thesis are shared by ASR and speaker recognition tasks, and speaker recognition task is generally considered to be simpler than ASR, this thesis will mainly focus on ASR tasks, and treats speaker recognition task as its extension. The detailed outline of the thesis is given below.

Chapter 2 defines the ASR problem, describes the use of HMM in ASR task, and reviews the different types of HMMs that are commonly used or have been used in ASR.

Chapter 3 reviews the related researches on discrete HMM (DHMM), focusing on "codebook" and "robust parameter estimation". Works on very large codebook DHMM by other researchers are also discussed.

Chapter 4 is the main part of this thesis. It investigates various issues when building a very-large-codebook multiple-stream discrete system for ASR. The issues include how to choose the codebook size, codebook type, and the number of streams; how to robustly estimate the large number of model parameters, and how to compensate

for the incorrect stream independence assumption in a multiple-stream system. The resulting very large codebook discrete HMM is compared to continuous density HMM (CDHMM) and other recently proposed discrete systems.

Chapter 5 applies some techniques developed for ASR in Chapter 4 to speedup the speaker verification (SV) system [2], which is built based on Gaussian-Mixture-Model Universal Background Model (GMM-UBM).

Chapter 6 concludes the thesis, with suggestions for future work.

[2] SV is one type of speaker recognition task, the other type is called speaker identification. In general, the methods developed for SV can also be applied to speaker identification.

CHAPTER 2

INTRODUCTION TO AUTOMATIC SPEECH RECOGNITION (ASR)

2.1 Definition of ASR

The task of automatic speech recognition (ASR) is to find the corresponding word sequence given the speech waveform using computer. In this section, the feature extraction from speech waveform is first described, and then a formal definition of ASR follows.

The speech waveform uttered by a speaker is first converted into electrical signal by the input device such as microphones. Sampling and quantization is then applied to the electrical analog signal to generate a digital signal sequence.

Speech analysis is done to the digital signal sequence to generate a sequence of feature vectors $X = \{x_1, x_2, \ldots, x_T\}$. Simply speaking, short speech frames of around 25ms are taken out from the digital signal sequence at intervals of 10ms or so. The length of 25ms for a frame is set with the assumption that the articulatory configuration, such as the vocal tract shape, is quasi-stationary in such a short time. For each time frame t, the feature vector x_t, typically of 20$\sim$60 dimensions, is then extracted. How to extract the feature vectors depends on what kind of speech feature the system wants to use. Various speech features were proposed for speech recognition task, with Mel-Frequency Cepstrum Coefficients (MFCC) [17], Linear Predictive Coding Coefficient (LPC) [54] and Perceptual Linear Predictive Coefficients (PLP) [32] as representative ones. Roughly speaking, the speech feature is considered to be good if it is insensitive to acoustic variations but sensitive to linguistic content.

By adopting the above conventional speech feature extraction process, the task of automatic speech recognition can then be formally defined as: *given a sequence of feature vectors* $X = \{x_1, x_2, \ldots, x_T\}$*, find the corresponding word sequence* $W = \{w_1, w_2, \ldots, w_M\}$ *using computer.*

2.1.1 Statistical ASR

Most successful ASR systems nowadays use statistically-based approaches, which can automatically extract the salient information from the data. Knowledge-based ASR systems also exist in history [80], which require extensive human knowledge about speech. However, in front of the complex speech phenomena, the knowledge from speech experts is usually insufficient, which leads to poorer performance compared with statistically-based ones. Hence, By the late of 1980s, nearly every big speech site had turned to statistical ASR systems.

From the statistical point of view, the ASR task can be rephrased as: find the word sequence $\hat{W} = \{w_1, w_2, \ldots, w_M\}$ that maximizes the posterior probability $P(W|X)$. Applying Bayes's rule, we have:

$$\hat{W} = \arg\max_W P(W|X) = \arg\max_W \frac{P(W)P(X|W)}{P(X)}$$

Since W will not affect $P(X)$, we have:

$$\hat{W} = \arg\max_W P(W)P(X|W) \tag{2.1}$$

$P(X|W)$ and $P(W)$ are two key components in an ASR system:

- $P(X|W)$ is called the *acoustic model*, which represents the probability of observing the speech feature sequence X given the word sequence W. The state-of-the-art acoustic model is built using Hidden Markov Model (HMM), which will be described in details in Section 2.2.

- $P(W)$ is called the *language model*, which represents the *a prior* probability of word sequence W independent of the observed signal sequence X. The probability $P(W)$ could be decomposed as a product of conditional probabilities:

$$P(W) = P(w_1, w_2, \ldots, w_M) = \prod_{i=1}^{M} P(w_i|w_1, \ldots, w_{i-1}), \tag{2.2}$$

  where $P(w_i|w_1, \ldots, w_{i-1})$ is the probability of generating the word w_i, given its entire past history $w_1, \ldots, w_{i-1}$.

  To build a language model based on Equation (2.2) is infeasible: for w_i, there are V^{i-1} different histories, with V be the vocabulary size; so, to specify $P(w_i|w_1, \ldots, w_{i-1})$

completely requires to estimate V^i values. In practice, the vocabulary size V is usually not small, e.g., 5000, which makes V^i a formidable large number even for moderate values of i. A common solution is to assume that w_i depends only on a shorter history of previous $n-1$ words, and $P(w_i|w_1,\ldots,w_{i-1})$ in Equation (2.2) is replaced by $P(w_i|w_{i-n+1},\ldots,w_{i-1})$. The resulting language model is called an n-gram language model. For example, if $n=3$, we have a *trigram* language model $P(w_i|w_{i-2},w_{i-1})$; and if $n=2$, we have a *bigram* language model $P(w_i|w_{i-1})$. Given enough training material, trigram language model is more powerful than bigram language model, as it takes more context information (history) into consideration.

The goodness of a language model can be estimated by using it to compute a measure called "perplexity" on a previously unseen test set. In general, a better language model will give a lower test-set perplexity.

Notice that, in practice, the language model term $P(W)$ is usually scaled by a power α, e.g., $\alpha=16$, to give $P(W)^\alpha$. α is called the *grammar factor*, which is introduced to balance the acoustic and language model scores, as the two have very different dynamic ranges.

The research in this thesis is mainly on the acoustic model $P(X|W)$, which is generally based on Hidden Markov Model (HMM) (see Section 2.2).

2.1.2 ASR System Performance Evaluation Criterion

The ASR system recognition result is usually evaluated by *word error rate* (WER). To compute the *word error rate*, the recognized word sequence is first aligned to the correct word sequence using a string matching algorithm. After the alignment, the number of substitution errors (S), deletion errors (D) and insertion errors (I) can be counted. Let N be the number of words in the correct word sequence, then the *word error rate* can be calculated as:

$$\textit{word error rate} = \frac{D+S+I}{N}$$

.

In this thesis, sometimes, we may also use the term "word accuracy" (or recognition accuracy), which is defined as: 1 - WER.

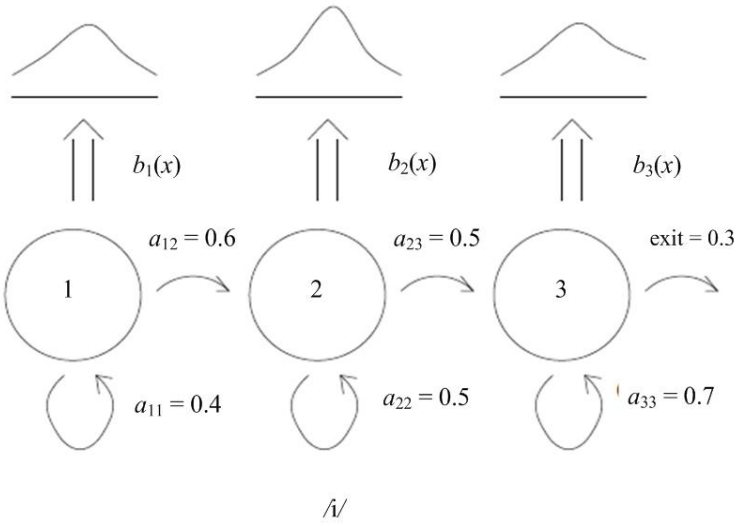


Figure 2.1: Phone HMM /i/: first-order 3-state left-to-right topology

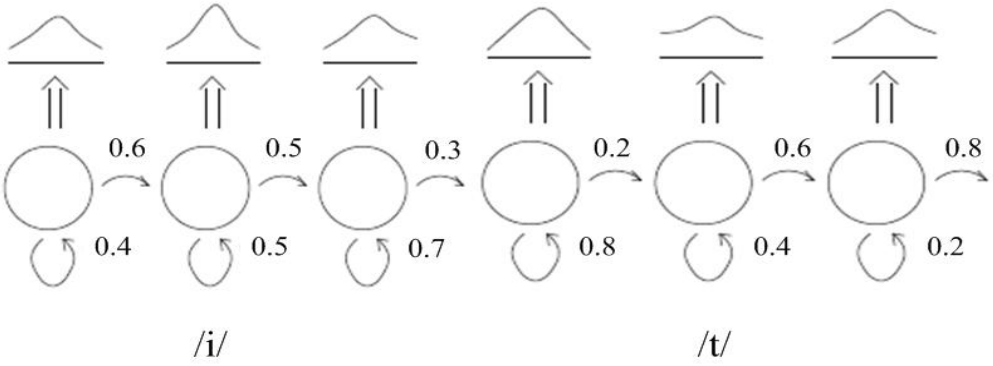


Figure 2.2: Word HMM "it": constructed from phone HMMs /i/ and /t/

2.2 Hidden Markov Model (HMM) for ASR

Hidden Markov model (HMM) is the dominant acoustic model used by modern ASR systems. HMM is a statistical model suitable for characterizing variable length observation sequences [1]. An HMM contains N states, with a_{ij} be the *transition probability* from state i to state j, and $b_j(x_t)$ be the *state output probability distribution* associated with state j. Figure 2.1 shows a 3-state first-order left-to-right HMM (a commonly adopted topology in ASR systems) to model phone /i/. At the first frame, the HMM is in state 1, and will output a feature vector drawn randomly from state output observation distribution $b_1(x_t)$; For each of the following frames, the HMM will first change to a new state randomly according to the transition probabilities from the current state (if the current state has self-loop transition, the new state could be the same as the cur-

[1] In ASR, the observation sequence is the feature vector sequence X.

rent state), and then output a feature vector based on the output distribution of the new state. This feature sequence generation procedure will terminate if the current state is the last state (i.e., state 3 in Figure 2.1), and the next transition is the exit transition.

In Figure 2.1, an HMM is used to model a phone. However, for ASR tasks, an HMM needs to describe the probability distribution $P(X|W)$ in Equation (2.1), that is, to model the word sequence W. Ideally, given infinite amount of resources (e.g., training data, memory, computation power), for every distinct word sequence W, we should train an HMM independently to model $P(X|W)$. However, it is infeasible in reality. Firstly, the number of distinct word sequences is basically infinite, we can not enumerate all of them. Secondly, even if we can enumerate all the word sequences, we can not collect enough training data to robustly train an HMM for each of them. Thus, in practical ASR system, we build an HMM for each phone just as in Figure 2.1 . A word HMM is constructed by the concatenation of phone HMMs (Figure 2.2 shows a word HMM "it" by concatenating two phone HMMs /i/ and /t/). Similarly, the word sequence HMM will be constructed as a concatenation of word HMMs, thus modeling $P(X|W)$. In this thesis, word sequence HMM to model W is denoted as M_W.

In fact, in the state-of-the-art ASR systems, usually, multiple HMMs are built for a single phone, corresponding to the same phone in different contexts. The most common context-dependent model is the triphone HMM [45]. A triphone HMM models the context due to its immediate left and right neighboring phones. For example, /t-i+n/ and /y-i+t/ are two triphones for the same phone i: the former one has left phone /t/ and right phone /n/, while the latter one has left phone /y/ and right phone /t/. Compared with phone HMMs (sometimes also called *monophone* HMMs), triphone HMMs consider the fact that the acoustic realization of a phone is influenced by its neighboring phones, which usually give better recognition performance in practice.

2.2.1 Dynamic Features for HMM

In HMM, it assumes that an observation feature vector x_t depends only on the state that generates it, and not on previous frames of observations. This assumption is generally not true, as the neighboring frames are highly correlated in reality. The simplest way to model correlations between neighboring frames without modifying the HMM assumption is to add dynamic features onto the input feature x_t [27]. Take MFCC features as an example. Let x_t be the original static feature vector at frame t consisting of 12

static MFCC coefficients and 1 energy term. The *delta coefficients* (i.e., the dynamic features), which measure the change of coefficients over time, are computed as below:

- first-order delta coefficients: $\Delta x_t = x_{t+2} - x_{t-2}$
- second-order delta coefficients: $\Delta\Delta x_t = \Delta x_{t+1} - \Delta x_{t-1}$

As a result, the final augmented feature vector at frame t will be a 39-dimensional vector $< x_t, \Delta x_t, \Delta\Delta x_t >$: 13 static, 13 first-order delta, and 13 second-order delta coefficients. Dynamic features significantly improve the recognition performance [35], thus are commonly adopted by modern ASR systems.

2.2.2 Training of and Recognition with HMM in ASR

To train phone HMMs (or triphone HMMs), they are first constructed with some initial parameters. To initialize the model, the transition probabilities from one state to other states could be set uniformly; if we use a Gaussian to model state output distribution $b_j(x_t)$, its initial mean and variance could be set the same as that of the training data. A speech utterance X with correct word sequence transcription W is given as training data. For simplicity, we assume only one utterance in the training data, which could be easily generalized to multiple utterances. Based on the transcription W and a dictionary (containing phone sequence of each word), we could construct a word sequence HMM M_W by concatenating of the corresponding phone HMMs. A common objective in training is to learn HMM parameters to maximize the likelihood of the training data: $P(X|M_W)$, which is realized by an Expectation-Maximization (E-M) type algorithm called Baum-Welch [6] (see Section 4.6.1 for a description of the algorithm for discrete HMM). Baum-Welch algorithm is applied to iteratively update the HMM parameters. It guarantees to increase the likelihood $P(X|M_W)$ after each iteration until a local optimum is reached. Discriminative criterion for HMM training has also been investigated and better performance is reported [81]. *Discriminative training* of HMM is beyond the scope of this thesis.

The recognition (sometimes called *decoding*) is to find the most likely word sequence $\hat{W}$ for a speech utterance X by the formula: $\hat{W} = \arg\max_W P(W)P(X|W)$. Thus, besides the acoustic model $P(X|W)$, a independently-trained language model $P(W)$ needs to be provided during recognition. Given these models, the ideal recognition approach is to compute the score $P(W)P(X|W)$ for every possible word se-

quence W, and choose the one that gives the highest score as the recognized word sequence. However, this approach is infeasible in reality as the number of possible word sequences is basically infinite. In a practical ASR system, the beam-search based Viterbi decoding algorithm is used to get the recognized word sequence, which prunes the un-promising word sequences without calculating their scores. In beam-search decoding, a *beam width* (or *pruning threshold*) is usually set to balance the recognition speed and accuracy. Basically, the smaller the beam width is, the more candidate word sequences are eliminated without evaluation, which leads to faster decoding but higher word error rate.

2.3 Different Types of HMM in ASR

Three types of HMM have been investigated in ASR:

- *continuous density* HMM (CDHMM)
- *semi-continuous* or *tied-mixture* HMM (SCHMM)
- *discrete* HMM (DHMM)

Depending on whether the input feature vector is divided into independent *data streams*, the above 3 types of HMMs could have one-stream or multiple-stream forms. The following 3 subsections will first introduce one-stream HMM for each of the three types. Based on that, multiple-stream HMM is described.

2.3.1 Continuous Density HMM (CDHMM)

In most current CDHMM systems [35], the state output distribution $b_j(x_t)$ is a Gaussian mixture model (GMM) with diagonal-covariances:

$$b_j(x_t) = \sum_{m=1}^{M_j} c_{jm} \mathcal{N}(x_t; \mu_{jm}, \sigma_{jm}^2),$$

where M_j is the number of Gaussian components [2] in state j , $\mathcal{N}(x_t; \mu_{jm}, \sigma_{jm}^2)$ is the mth Gaussian in state j with mean vector μ_{jm}, variance vector σ_{jm}^2, and mixture weight c_{jm}. Since the covariance matrix for each Gaussian is diagonal, variance vector could

[2]Usually, the number of components per state ranges from 8 to 64

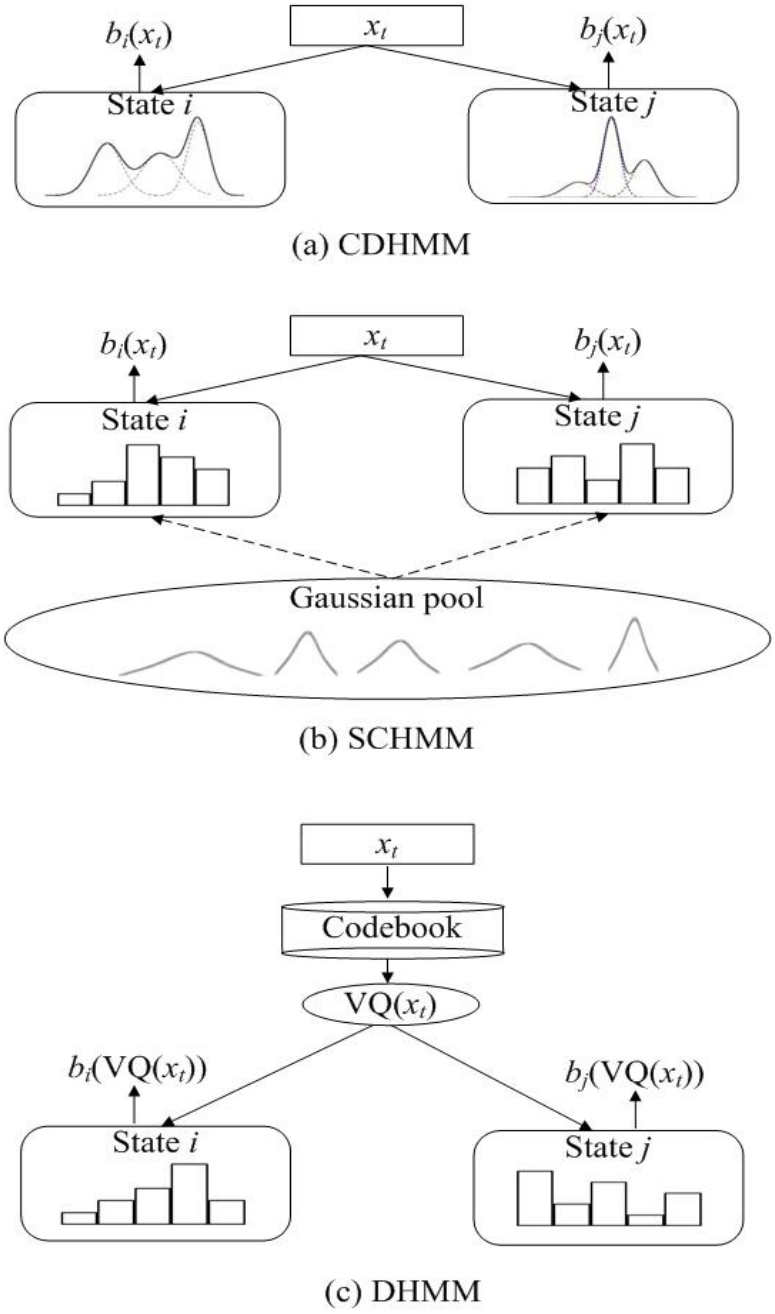


Figure 2.3: Different types of HMM

be used to represent the original covariance matrix. The value in ith dimension of the vector represents the ith diagonal element in the matrix) . An example of CDHMM is shown in Figure 2.3(a).

In theory, given enough number of mixtures, GMM could approximate any continuous density function arbitrarily close. Thus, when the amount of training data is sufficient, CDHMM with a large number of mixtures offers the best recognition accuracy. However, the GMM computation time grows linearly with the number of Gaussian mixtures. Large and complex systems usually use a large mixture of Gaussians

per state, thus resulting in a heavy computation burden in recognition.

2.3.2 Semi-continuous HMM (SCHMM)

SCHMM [38, 7] could be viewed as a special instance of CDHMM, whose Gaussians are not state dependent, but shared by all states. As a result, its state output distribution is:

$$b_j(x_t) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(x_t; \mu_m, \sigma_m^2),$$

where $\mathcal{N}(x_t; \mu_1, \sigma_1^2), \ldots, \mathcal{N}(x_t; \mu_M, \sigma_M^2)$ are a global pool of M Gaussians shared by all states. The mixture weights c_{jm} in SCHMM are state dependent, thus differentiating the output distribution of one state from another. An example of SCHMM is shown in Figure 2.3(b).

The number of Gaussians M in SCHMM usually ranges from 512$\sim$2048. Thus, from a state's perspective, the GMM in SCHMM has more components than that in CDHMM. However, for a typical ASR system with thousands of distinct states, the total number of distinct Gaussians in CDHMM systems is usually much larger: for CDHMM, this number is equal to the number of states times the number of components per state, which will be amount to tens of thousands in general; while for SCHMM, due to Gaussian sharing, the number is the same as the Gaussian pool size M. As a result, the number of free parameters in SCHMM can be reduced compared with CDHMM, and the model could be trained robustly even when the amount of training data is limited.

2.3.3 Discrete HMM (DHMM)

In DHMM [67], the continuous feature space is divided into N disjoint partitions, with each partition represented by a *codeword*. Usually, this procedure is carried out by k-means or LBG [49] clustering algorithm. The set of N codewords is called a *codebook*. For each original continuous observation feature x_t (e.g., MFCC feature), the partition which x_t belongs to is first located, and then x_t is replaced by the codeword $VQ(x_t)$; this step is called the *vector quantization* (VQ) of x_t. As a result, $VQ(x_t)$ becomes the observation feature in DHMM, which is a discrete symbol because $VQ(x_t)$ can only be one of the N codewords in the codebook. Accordingly, the state output distribution

$b_j(VQ(x_t))$ is a discrete probability table with size N, which stores the probabilities to output each codeword from state j. An example of DHMM is shown in Figure 2.3(c).

DHMM was used in the early history of speech recognition. It is very fast in recognition: to get the state output probability, DHMM only needs $O(1)$ table lookup from the discrete probability table, while CDHMM has to do time-consuming GMM computation. However, due the large quantization error, the recognition accuracy of conventional DHMM is usually much worse than that of CDHMM and SCHMM. As a result, DHMM systems are almost abandoned in modern ASR applications.

2.3.4 Multiple-stream HMM

In multiple-stream HMM, the input feature vector x_t is divided into a number of *data streams*, with the assumption that different streams are independent (known as *stream independence assumption* in ASR community). Each stream in a state is modeled independently by a different output distribution. Given a feature vector, the output probability from a state becomes a product of output probabilities from each stream in the state.

Specifically, in S-stream HMM systems, a feature vector x_t is divided into S sub-vectors $x_t^{(1)}, \ldots, x_t^{(S)}$ [3]. Because of the stream independence assumption, the state output probability $b_j(\cdot)$ for continuous and discrete models are given below in Equation (2.3) and (2.4) respectively.

$$b_j(x_t) = \prod_{s=1}^{S} b_j^{(s)}(x_t^{(s)}), \tag{2.3}$$

$$b_j(VQ(x_t)) = \prod_{s=1}^{S} b_j^{(s)}(VQ^{(s)}(x_t^{(s)})), \tag{2.4}$$

where $b_j^{(s)}(\cdot)$ is the output distribution from the sth stream in state j, and $VQ^{(s)}(x_t^{(s)})$ denotes the corresponding VQ codeword for $x_t^{(s)}$ found by the codebook in the sth stream. Figure 2.4 gives examples of 2-stream systems for CDHMM, SCHMM and DHMM. Notice that, for DHMM and SCHMM, multiple codebooks or Gaussian pools are stored, each for one stream.

[3] Throughout the thesis, the superscript s is used to denote the stream index.

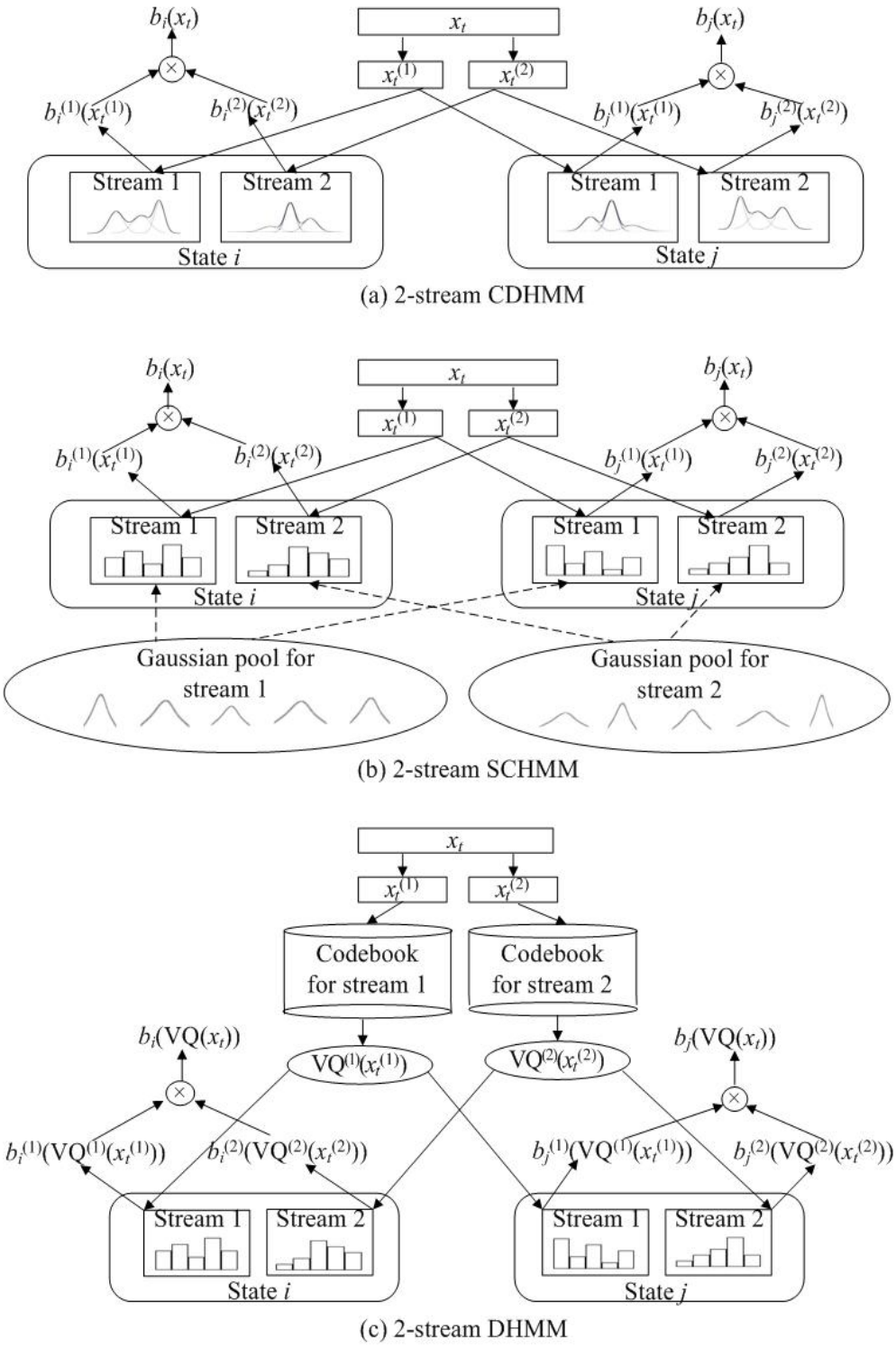


Figure 2.4: Examples of 2-stream HMM systems

In practice, multiple stream systems are usually adopted for DHMM and SCHMM, which could cut the recognition error rate by more than 10% [35] compared with 1-stream systems. The justifications to use multiple streams are as follows: limited by the amount of training data, memory, and recognition time, DHMM and SCHMM

usually use at most thousands of codewords (or Gaussians) to represent the full acoustic feature space. However, such a small number is far from enough to represent the high-dimensional feature space in single stream system (e.g., in state-of-the-art ASR systems, we use 39-dimensional MFCC features). By splitting the single stream into multiple lower dimensional streams, and representing each stream with thousands of codewords or Gaussians independently, a better representation of acoustic space could be expected, which boosts the recognition performance.

The disadvantage of multiple-stream HMM is that it does not model the correlations among different streams and assumes they are independent. However, in practice, streams are not perfectly independent. The loss of stream correlation information may result in degradation in recognition accuracy.

How to split a feature vector into multiple streams is tricky. On the one hand, more streams are desired to better represent the acoustic space. On the other hand, one has to make sure the divided streams are as independent as possible, and more streams usually imply greater loss of correlation information among the streams. In the ASR community, there exist some commonly used stream partitions, defined using the heuristic to put conceptually similar features together in one stream. E.g., for the 39-dimensional MFCCs, consisting of 12MFCCs, 1 energy, and their first- and second-order time derivatives, one "common" stream definition is to divide it into 4 streams as below, where e stands for energy.

- Stream 1: 12MFCCs
- Stream 2: 12Δ MFCCs
- Stream 3: 12$\Delta\Delta$ MFCCs
- Stream 4: $e + \Delta e + \Delta\Delta e$

CHAPTER 3

RELATED WORK IN DHMM

The first two sections in this chapter will review the researches related to codebook and robust parameter estimation in DHMM, the two key problems for large codebook systems as shown in Section 1.1. Since most of the research was proposed for discrete systems with conventional codebook size, a brief discussion on its applicability to very large codebook system is given at the end of each section. The last section in this chapter introduces the previous works on very large codebook systems.

3.1 Different Types of Codebooks

A codebook, consisting of N codewords, partitions the feature space into N disjoint parts. The codebook construction procedure involves clustering a set of training vectors x_t's into N disjoint sets based on some criterion, with each cluster represented by a codeword. Depending on whether the class label information of x_t is utilized during construction (e.g., the class label could be the phoneme identity of x_t), the codebook construction could be classified into two types: unsupervised construction and supervised construction.

3.1.1 Codebook by Unsupervised Construction

In unsupervised construction, given a fixed codebook size, the general criterion of a good codebook is to have small quantization error. Various kinds of codebooks based on unsupervised clustering had been proposed in history. Below we introduce some representative ones that are closely related to the works in this thesis.

Linear codebook [12, 31]: it is probably the most commonly used codebook in conventional DHMM systems (also called vector quantized (VQ) codebook). Generally, an unsupervised clustering algorithm based on Euclidean distance (e.g., k-means clustering, or LBG) is applied on a set of training feature vectors to partition them into N clusters. The centroids of each cluster then serve as the codewords in the VQ

codebook. For an input continuous feature vector x_t, to find its codeword requires the comparison with each codeword in the codebook in $O(N)$ time. The reason for calling it *linear* codebook is that the time to find a codeword grows linearly with the size of the codebook.

Subvector-quantized (SVQ) codebook: SVQ codebook was initially introduced for high-quality, low-bit rate speech coding task [61] (called "split vector quantization" in speech coding), and later applied to ASR task [9, 78]. SVQ codebook could be used to significantly reduce the time of finding a codeword in DHMM systems. In an SVQ codebook, a d-dimensional feature vector x_t is split into L (with $L \leq d$) subvectors as: $x_t = [x_{1t}, x_{2t}, \ldots, x_{Lt}]$, with one codebook per subvector. The full-space codebook is constructed as a product of the codebooks in the subvectors. As a result, the full-space codeword $VQ(x_t)$ could be denoted as: $VQ(x_t) = [VQ_1(x_{1t}), VQ_2(x_{2t}), \ldots, VQ_L(x_{Lt})]$, where $VQ_i(x_{it})$ denotes the corresponding SVQ codeword for x_{it} found by the codebook in the ith subvector. To find the codeword for x_t is thus reduced to finding the L subvector codewords, which requires $O(\sum_{i=1}^{L} n_i)$ time, with n_i be the codebook size in subvector i. This is significantly faster than that of finding a codeword in the linear codebook with the same size, which requires $O(\prod_{i=1}^{L} n_i)$ time.

Scalar-quantized (SQ) codebook [53]: it is a special case of SVQ codebook when each subvector contains only one feature dimension. It other words, the number of subvectors L is equal to the feature vector dimension d. SQ codebook was used in our research group's first attempt of very large codebook system [53].

Tree-structured VQ codebook [84]: tree-structured codebook is also used to reduce the codebook finding time. To construct it, a binary tree is usually adopted. At the beginning, all the training feature vectors are pooled at the root node of the tree to form a single cluster. During tree construction, the training vectors in each node are split into 2 clusters (e.g. by k-means clustering), with each cluster represented by a child node. The mean of the training data in each node is recorded for later binary search. The node splitting procedure stops when a desired number of N leaves is reached, which serve as the codewords. To find a codeword for x_t, it starts from the root node, and iteratively compares with the means in the two children to decide which sub-tree to go, until it reaches a leaf node, which is the codeword of x_t. Obviously, this binary search procedure takes $O(\log N)$ time, which is much faster than that of a linear codebook.

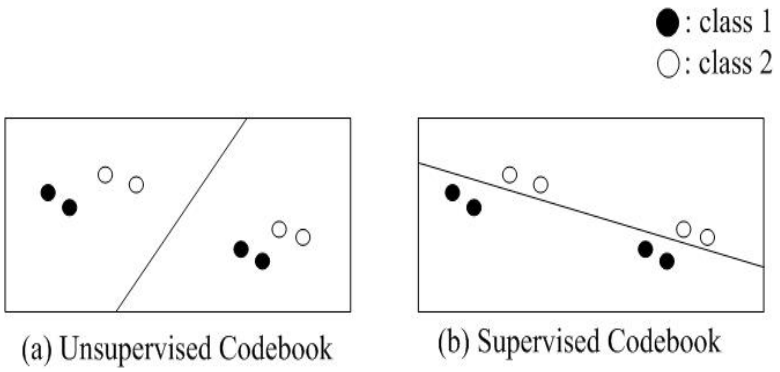


Figure 3.1: Codebooks produced by unsupervised and supervised construction. Each method partitions the acoustic space into 2 parts. The dots represent training samples in codebook construction, with colors representing different classes.

Although both SVQ and tree-structured codebooks are much faster to lookup compared with linear codebook, they also incur higher quantization error. As a result, linear codebook is more commonly used in conventional DHMM system, where the codebook size is small and the codeword lookup time is not a burden.

3.1.2 Codebook by Supervised Construction

Rather than minimizing the quantization error, one may also try to *maximize the mutual information* (MMI) between a codeword and its class in supervised codebook construction, where the class label of each training vector is available [60, 58, 26, 24, 77, 1, 2, 3] (the class label could be a phone label, for example). Due to the classification nature of ASR, codebook construction based on MMI criterion is believed to be more related to the recognition accuracy. Figure 3.1 shows the difference between codebooks produced by unsupervised and supervised construction.

Since the class entropy itself is independent of the codebook, to maximize the mutual information between the codeword and the class is equivalent to minimizing the conditional entropy of the class given the codeword. This conditional entropy is denoted as *entropy impurity* in classification and regression trees (CART) [25]. In other words, we want the class labels of the feature vectors belonging to the same codeword to be as pure as possible. In the ideal case, all features vectors in a codeword belong to only one class. Interestingly, it was proven independently by [60] and [58] that: under some assumption, jointly estimating[1] the codebook and HMM parameters under

[1]In conventional DHMM, the parameter estimation is carried out in 2 steps: VQ codebook is first constructed, and then HMM parameters are estimated.

the maximum likelihood criterion will naturally lead to the above MMI criterion.

To build the MMI-based codebook, usually a binary decision tree is adopted [2]. At the beginning of the MMI tree construction, all training feature vectors with their associated class labels are pooled at the root node. A node is split into two children by finding the hyperplane that partitions the data in the parent to give the maximum decrease of entropy impurity. The splitting stops when the decrease of entropy impurity falls below a pre-defined threshold. The leaves of the tree are the codewords. To find a codeword for x_t, one starts from the root node, and traverses through the tree by checking which side of the hyperplane it lies in at each node, until one reaches a leaf node, which is then the codeword of x_t. Like a tree-structured VQ codebook constructed in unsupervised manner, finding a codeword on an MMI-based tree codebook also takes $O(\log N)$ time.

For DHMM systems with conventional codebook size, using an the MMI-based codebook shows better recognition accuracy compared with linear codebook [58, 26, 24] (and also from our later experimental results in Section 4.10.2). However, when the codebook size is very large (e.g., tens of thousands), the MMI-based codebook does not guarantee to perform better than linear codebook [5].

To summarize, except linear codebook, many codebooks (e.g., SVQ, SQ, VQ-tree, MMI-tree) have the desirable properties for very large codebook systems: they are fast to construct and lookup. This thesis mainly explores the use of SVQ and SQ codebooks for very large codebook systems. However, comparisons with tree-based codebooks are also carried out.

3.2 Robust Estimation of State Output Probability Distribution

In DHMM, the state output distribution stores the probabilities to generate each codeword from the state. The discrete distribution $b_j(\cdot)$ in state j could be denoted as a vector with size N, where $b_j[i]$ stores the probability value of generating the ith codeword. When the training data is insufficient, or the codebook size N is too large, b_j usually cannot be trained robustly, resulting in poor recognition performance. For example, the ith codeword is not seen in the training data of state j, thus $b_j[i] = 0$;

[2] There is also some work which used MMI neural network [58].

however, it occurs in the testing data. Four major methods that improve the robustness of DHMM distributions are described below.

(1) Flooring: to robustly estimate b_j, the simplest method is to set a flooring threshold (e.g., 1e-6) [47]. After training, if the value of $b_j[i]$ (i.e., the probability of the ith codeword from state j) falls below the flooring threshold, it is replaced by the threshold. By flooring, the codewords which may not even be seen in the training data could also get some small probability values, avoiding zero probabilities for unseen testing data. Due to the simplicity, flooring is commonly applied in DHMM systems.

(2) Smoothing matrix: in [72, 43], a smoothing matrix A of size $N \times N$ was introduced to enhance the robustness of b_j with respect to unseen data by transforming b_j as follows: $b_j^* = A \times b_j$, where b_j^* is the output distribution of state j after smoothing. Smoothing matrix makes use of the relationship among codewords to help robustly estimate the probabilities of the codewords with rare training samples. The values in a column of the smoothing matrix sum up to 1, with $A[i][j]$ denotes: if the jth codeword is seen, what is the probability of seeing the ith codeword? Different methods were proposed to obtain the smoothing matrix. One of the methods is called parzen smoothing, where the value in $A[i][j]$ is set based on the Euclidian distance between codeword i and j. Basically, the smaller the distance is, the larger $A[i][j]$ will be, which is based on the assumption that two codewords close in distance should have a strong relationship. Another smoothing method is called co-occurrence smoothing, which sets $A[i][j]$ according to the number of times codeword i and j co-occur in the training data. The smoothing matrix methods showed some improvement in simple recognition task. However, they are hard to generalize to tasks with very large codebook size N (e.g., tens of thousands of codewords), as the number of parameters $N \times N$ in smoothing matrix will be too large to estimate.

(3) Interpolation: the basic idea in interpolation [46, 36, 37] is to interpolate the detailed but not robustly trained model with a coarse but robust model. In practice, we may interpolate b_j in a context-dependent DHMM (e.g., triphone DHMM) with the corresponding one in a context-independent DHMM (e.g., monophone DHMM) by deleted interpolation [40].

(4) State tying: state tying also serves as an efficient way to improve the robustness of b_j against unseen data [45, 39, 85]. Generally speaking, if two states are similar based on some similarity measure, they will be merged together by tying the param-

eters and sharing their training data. By reducing the number of free parameters, this effectively increases the average amount of training data for each parameter. Parameter tying is a common technique in training ASR systems, both for CDHMM and DHMM.

To summarize, both flooring and state tying are "standard" settings in current ASR systems, which are also utilized by the baseline system in the thesis. We also explore the deleted interpolation for very large codebook systems. The $N \times N$ number of parameters in smoothing matrix makes it unsuitable for very large codebook systems, where N could be as large as 65536 in our experiment. Finally, as shown in our later experiments in Section 4.6, using the above techniques alone are still not enough to robustly estimate the parameters in very large codebook systems. As a result, new techniques based on subspace modeling are proposed in the thesis.

3.3 Very Large Codebook DHMM

Very few works on DHMM use large codebooks in the literature. This is probably because DHMM was most popular in 1980's, when the memory resources were quite limited, and couldnot afford large models. The only few works with large codebooks that we are aware of are in [5, 53, 24].

In [53], our group proposed the *high-density discrete hidden Markov model* (HDDHMM) to reduce the quantization error of DHMM with a codebook of 16K codewords per stream. To make the codebook construction and lookup fast, SQ codebooks are used instead of linear codebook. To solve the data scarcity problem during model parameter estimation, HDDHMM was directly converted from CDHMM by integration without any training. The resulting HDDHMM showed a significant improvement compared with conventional DHMM. However, there was still an significantly increase in WER compared with CDHMM on large vocabulary continuous speech recognition task.

In [24], Droppo investigated the effectiveness of three impurity functions, including principal component analysis (PCA), linear discriminative analysis (LDA), and MMI, for constructing VQ codebook by an acoustic decision tree. Different codebook sizes were tried with a maximum of 64K. The MMI-tree-based codebook was shown to be the best among the three, and the discrete system built with it showed comparable performance as that of GMM. However, the performance was only evaluated on frame

classification in a a simple task. Furthermore, the robust parameter estimation issue was not touched in [24] even though enough training data could be collected for such a simple task.

In [5], both VQ and MMI based tree-structured codebook were investigated for a very large codebook system with 16$\sim$64 K codewords. To robustly estimate the model parameters, three types of tree-structured smoothing techniques (mixture smoothing, smoothing by adding 1, and Gaussian smoothing) were explored. It was found that different smoothing methods yielded comparable performances, and MMI-based tree-structured codebook did not seem to be better than VQ-based ones in such a large codebook system. Unfortunately, limited by the resources, the evaluation set used in [5] is not a standard one, and the very CDHMM baseline is not strong.

The works in this thesis are mainly based on [53] with SVQ or SQ codebooks. However, as a comparison, we also implement the very large codebook system based on MMI-tree codebook and the smoothing by adding 1 technique.

CHAPTER 4

VERY LARGE CODEBOOK DHMM FOR ASR

This chapter investigates the use of very large codebook for ASR tasks. To be consistent with our previous works in [53, 83], the very large codebook DHMM is also called *high-density discrete hidden Markov model* (HDDHMM), where the term *high-density* refers to discrete distributions with very large codebooks.

Five issues are explored for very large codebook DHMMs:

- Codebook size: how large should the codebook be?
- Codebook type: which type of codebook should be chosen?
- Number of streams: how many streams should be used? Specifically, given such a large codebook, could we use fewer streams?
- Robust parameter estimation: how to train HDDHMM, which has hundreds of times more parameters compared with conventional DHMM?
- How to compensate for the loss due to incorrect stream independence assumption in multiple-stream systems?

Since the above five issues are correlated, the ideal way to explore them is to try all possible combinations of settings. However, the search space required is too large to be feasible. As a result, a *coordinate ascent* research methodology is applied, by exploring one issue at a time while keeping the other four fixed.

The work in this chapter is based on our group's previous works from 2005 [53] and we start by describing that system. Then, by exploring the five issues one at a time, we show incremental improvements based on that baseline system. Comparisons with other discrete systems are also carried out in this chapter.

In the next section, to facilitate the discussion, we would first like to describe the common experimental setup, as well as giving definitions, before going into the details of our work.

4.1 Common Experimental Setup and Definitions

4.1.1 Experimental Setup

Three clean read English corpora are used for speaker-independent (SI) ASR experiments in the chapter:

- Resource management (RM): it consists of utterances which are commands to a computer system relating to shipping, with 3.8 hours of training data.
- Wall Street Journal (WSJ) SI-84 (also known as WSJ0 corpus): it consists of utterances selected from North American Business news, with 14 hours of training speech.
- WSJ SI-284: it is a superset of SI-84 with data from both WSJ0 and WSJ1 corpus, with 66 hours of training speech.

For each corpus, different acoustic models are built and compared. The common experimental settings are described below.

Acoustic Feature

The acoustic feature vectors are extracted every 10 ms over a window of 25 ms, each comprising 39 dimensions. *Cepstral mean normalization* is performed on the features to reduce the variations introduced by different channels.

The 39 coefficients of a feature vector are: 1-12th static MFCC coefficients and log energy e, as well as their first-(Δ) and second-order ($\Delta\Delta$) time derivatives, which could be denoted as

$$12\text{MFCCs} + e + 12\Delta\text{MFCCs} + \Delta e + 12\Delta\Delta\text{MFCCs} + \Delta\Delta e$$

For discrete systems (e.g., DHMM and HDDHMM), we further convert each 39-dimensional feature vector to discrete features based on some vector quantization (VQ) codebooks.

Acoustic Models

The acoustic models are speaker and gender independent cross-word triphone HMMs, built using decision tree state tying [85]. Except the 1-state *short pause* HMM, each

Table 4.1: Experimental settings for different ASR systems

Settings	RM	WSJ SI-84	WSJ SI-284
Training Set	SI-109	SI-84	SI-284
#Training Utterances	3,990	7,138	36,861
#Training Speakers	109	83	283
Development Set	speaker-dependent dev set	mainly from si_dt_05	
#Development Utterances	1,200	465	
Test Set	feb89, oct89, feb91, sep92	nov92	
#Test Utterances	1,200	330	
Language Model	word-pair bigram	trigram	
Vocabulary Size	1K	5K	
Perplexity on Test Set	60	57	
#Triphone HMMs	6,817	15,449	
#Tied States	1,589	3,132	

HMM has a 3-state left-to-right topology. In WSJ models, the only state in the *short pause* HMM is tied to the middle state of the *silence* HMM.

Different acoustic models built on the same corpus use the same set of triphones, and share the state tying structure with the same number of tied states. Also, WSJ SI-84 and SI-284 models share the triphones and state tying structure, and only differ by the complexity of state output distributions (e.g., for CDHMM systems, the number of Gaussian mixtures in a state in SI-284 is twice of that in SI-84).

Software & Hardware Environment

The HTK software [84] is used for model training and decoding. Since HTK does not support HDDHMM directly, the HTK codes are modified. All experiments are run on a Linux machine with a 3GHz CPU and 4GB RAM. All recognition results are shown in word error rate (WER). Unless specified, WER stands for word error rate on the test set. For some results, we may report the error rate on the development or training set, which will then be denoted as *Dev WER* or *Train WER* explicitly.

Details of Experimental Settings

The details about data, acoustic model characteristics, and the evaluation tasks (including test set and language model) for the three systems are summarized in Table 4.1.

As can be seen, the RM system is relatively simple, and is evaluated on a medium-vocabulary task with a vocabulary of 1000 words; while the WSJ systems are more complex, and are evaluated on a large vocabulary task with a vocabulary of 5000 words. In general, the larger the vocabulary, the more difficult the task is. Both the RM and WSJ are closed vocabulary tasks; it means all the words in the test set are included in the vocabulary of the language model.

The Choice of System/Task for Exploration

To explore different issues of HDDHMM is a time-consuming procedure. As a result, we need to first choose one system/task, and carry out investigations on it. Based on the settings/experiences learned from it, we then evaluate other systems/tasks.

The WSJ SI-84 system with a 5K vocabulary task is chosen for the following reasons:

- It is a large vocabulary task with a reasonably large amount of training data. Yet, its training set is not as large as that in WSJ SI-284 so that we could carry out experiments quickly.
- The previous work in [53], which serves as the starting point of the thesis, was evaluated on the same WSJ SI-84 task. It is better to continue on the same task, to show any improvements.
- It is a benchmark task with a lot of published results. So it is easy to compare with other researchers' work. For example, one of the most recent research works on DHMM [77, 1, 2, 3], which we want to compare with, worked on the same WSJ SI-84 task.

In the rest of the thesis, for simplicity, the WSJ SI-84 5K-vocabulary task is denoted as "WSJ SI-84 task", and the other two systems/tasks are called "RM task" and "WSJ SI-284 task" respectively.

4.1.2 Stream Definition

The following common stream definitions will be used in this thesis:

- S1 (1 stream)
  Stream 1: 12MFCCs $+ e +$ 12ΔMFCCs $+ \Delta e +$ 12$\Delta\Delta$MFCCs $+ \Delta\Delta e$

- S4 (4 streams)
  Stream 1: 12MFCCs
  Stream 2: 12Δ MFCCs
  Stream 3: 12$\Delta\Delta$ MFCCs
  Stream 4: $e + \Delta e + \Delta\Delta e$

- S3-a (first definition for 3 streams)
  Stream 1: 12MFCCs + e
  stream 2: 12Δ MFCCs + Δe
  stream 3: 12$\Delta\Delta$ MFCCs + $\Delta\Delta e$

- S3-b (second definition for 3 streams)
  Stream 1: 12MFCCs + 12Δ MFCCs
  Stream 2: 12$\Delta\Delta$ MFCCs
  Stream 3: $e + \Delta e + \Delta\Delta e$

- S3-c (third definition for 3 streams)
  Stream 1: 12MFCCs
  Stream 2: 12Δ MFCCs + 12$\Delta\Delta$ MFCCs
  Stream 3: $e + \Delta e + \Delta\Delta e$

- S3-d (fourth definition for 3 streams)
  Stream 1: 12MFCCs + 1$\sim$6$th\Delta\Delta$ MFCCs
  Stream 2: 12Δ MFCCs + 7$\sim$12$th\Delta\Delta$ MFCCs
  Stream 3: $e + \Delta e + \Delta\Delta e$

- S2 (2 streams)
  Stream 1: 12MFCCs + e + 12Δ MFCCs + Δe
  Stream 2: 12$\Delta\Delta$ MFCCs + $\Delta\Delta e$

4.1.3 Subvector-quantized (SVQ) & Scalar-quantized (SQ) Codebooks

Subvector-quantized (SVQ) and scalar-quantized (SQ) codebooks are used by HD-DHMM in the rest of the chapter. It is better to describe them first.

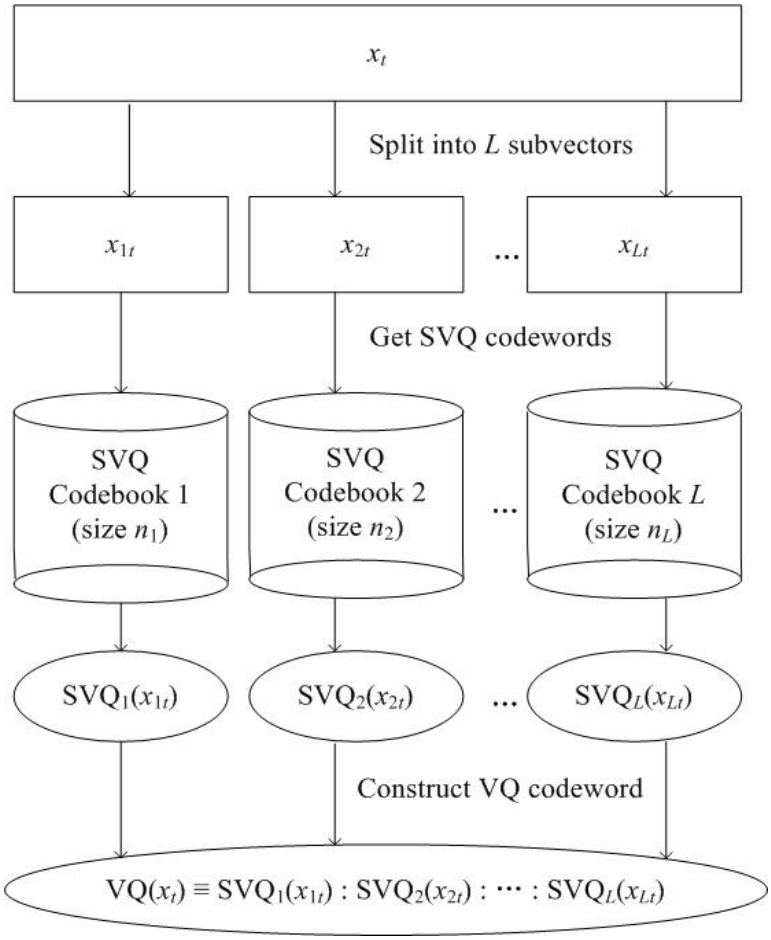


Figure 4.1: SVQ codebooks, where x_t is a d-dimensional feature vector at frame t, and VQ(x_t) is its corresponding full space VQ codeword constructed from SVQ or SQ codewords. Notice that when $L = d$, it becomes SQ codebooks.

SVQ Codebooks

Figure 4.1 shows the architecture of SVQ codebooks. The codebook construction and lookup are introduced in detail in the following two paragraphs.

Codebook construction: to construct the SVQ codebooks, each d-dimensional acoustic vector x_t is split into L subvectors $x_t = [x_{1t}, x_{2t}, \ldots, x_{Lt}]$. In the thesis, we assume that the feature dimensions in a subvector are consecutive). Subvectors of each split are vector-quantized (usually by k-means clustering) to n_i codewords to create an SVQ codebook for the split i, with $i = 1, \ldots, L$. The full-space codebook is represented by the combinatorial product of the per-dimension SVQ codebooks. As a result, the full d-dimensional acoustic space is divided into $N = \prod_{i=1}^{L} n_i$ bins. Each HDDHMM state probability distribution becomes a discrete distribution consisting of the probabilities of an acoustic observation falling into each of the N bins. For instance, if $d = 12$ (for the typical static MFCC vectors), and x_t is split into 6 subvectors with each split vector-quantized to $n_1 = n_2 = \ldots = n_6 = 4$ codewords, then there will be $4^6 = 4096$ bins.

Codebook lookup: to find the codeword for a new acoustic vector x_t, it will first be split into L subvectors. The subvector in the ith split is then compared with the codewords in the corresponding SVQ codebook to find its SVQ codeword in $O(n_i)$ time. After getting all the L SVQ codewords of x_t, its full-space VQ codeword is constructed as the product of SVQ codewords, one from each of the L splits as:

$$VQ(x_t) \equiv SVQ_1(x_{1t}) : SVQ_2(x_{2t}) : \cdots : SVQ_L(x_{Lt}),$$

where $VQ(\cdot)$ and $SVQ_i(\cdot)$ represent the VQ codeword given the full-space acoustic vector and the SVQ codeword in the ith split given the acoustic subvector in that split respectively. A VQ codeword (or bin) constructed above represents a d-dimensional convex polytope.

Since construction of a full-space codeword from its SVQ codewords takes almost no time, the time to get the full-space codeword for a given input feature vector is determined by the cost to find its L SVQ codewords, which is $O(\sum_{i=1}^{L} n_i)$. Thus, with the use of SVQ codebooks, finding codeword for an input vector is significantly quicker than if a linear VQ codebook with the same codebook size is used, which takes $O(\prod_{i=1}^{L} n_i)$ time. Continuing with our example of using a 4-codeword SVQ codebook for each of the 6 subvectors, finding a codeword with SVQ codebooks is reduced to

only $4 \times 6 = 24$ subvector comparison operations, independent of the total number of 4096 bins in the 12-dimensional feature space.

Finally, the constructed full-space codeword is used as an index to get the state output probability by a table lookup, which takes only $O(1)$ time.

SQ Codebooks

SQ codebooks are in fact a special case of SVQ codebooks, with each subvector containing only one dimension. In other words, the number of splits L is equal to the feature dimension d for SQ codebooks. Compared with the more general SVQ codebooks (with a subvector dimension larger than 1), SQ codebooks differ in the following ways:

- Since each subvector is scalar, scalar-quantization (SQ) rather than vector-quantization is performed during codebook construction.
- An SQ codeword (or bin) represents an interval in one dimension space, and the constructed full-space VQ codeword represents a d-dimensional hypercube. While in the SVQ representation, each SVQ codeword (or bin) represents a multi-dimensional convex polytope, and the full-space VQ codeword represents a d-dimensional convex polytope. Since convex polytope is more general in shape compared with hypercube, SVQ codebooks are generally more effective to represent the feature space, given the same number of codewords in the full space.
- Since each split is scalar rather than vector, binary search could be used to find an SQ codeword in ith dimension with $\log_2(n_i)$ time, where n_i is the SQ codebook size in dimension i. As a result, to find a codeword given x_t takes $O(\sum_{i=1}^{d} \log_2(n_i))$ time, which is even faster than SVQ codebooks.

4.1.4 Subvector-partition and Bit-allocation Schemes

To specify the partition of a full-space vector into subvectors and the size of each SVQ codebook, we introduce the *subvector-partition and bit-allocation scheme*, which is denoted by a sequence of "$w(b)$" patterns.

In the scheme, the basic building block is $w(b)$, where w is a subvector dimension, b is the number of bits allocated to the subvector, and $[w(b)]^C$ is equivalent to repeated patterns of $w(b)$ $\underbrace{w(b)\cdots w(b)}_{C\ in\ total}$.

The terminology *bit* is borrowed from speech coding. If b bits are allocated to quantize a subvector, then its SVQ codebook has a size of 2^b. The number of codewords in the full feature space could be computed as 2^B, where B is the total number of bits accumulated from each subvector.

Let us use an example to illustrate the scheme. If a 12-dimensional MFCC vector x_t is split into four 3-dimensional subvectors, 6 bits are allocated to the first subvector, 4 bits are allocated to the second subvector, and 3 bits are allocated to each of the last two subvectors, then the subvector partition and bit allocation scheme is denoted by the expression $3(6)3(4)[3(3)]^2$. The total number of bits used in the scheme is: $6+4+3\times 2=16$, which implies the original 12-dimensional feature space is partitioned into 2^{16} bins.

For SQ codebooks, since each subvector is scalar with a single dimension, the subvector dimension w in the scheme will always be 1. In this case, we may omit w. Continuing with our 12-dimensional MFCC vector example, if SQ codebooks are used, and each of the first two dimensions is allocated 3 bits, while each of the remaining 10 dimensions is allocated one bit, we could simply denote it as: $[(3)]^2[(1)]^{10}$ rather than $[1(3)]^2[1(1)]^{10}$.

4.1.5 Subvector vs. Stream

The HDDHMM systems in this chapter usually employ multiple streams (see Section 2.3.4). Since both *stream* and *subvector* involve splitting vectors into components, it is easy to confuse the two concepts. We try to explain their differences by showing the use of SVQ codebooks in multiple-stream DHMM systems. Figure 4.2 shows an example of a 2-stream system, with each stream consisting of 2 SVQ codebooks. Generally speaking,

- The acoustic feature vector x_t is first divided into S streams: $x_t=[x_t^{(1)},x_t^{(2)},\ldots,x_t^{(S)}]$. Different streams are assumed to be independent, and each stream s of an HMM state j has its own high-density discrete distribution $b_j^{(s)}(\cdot)$.

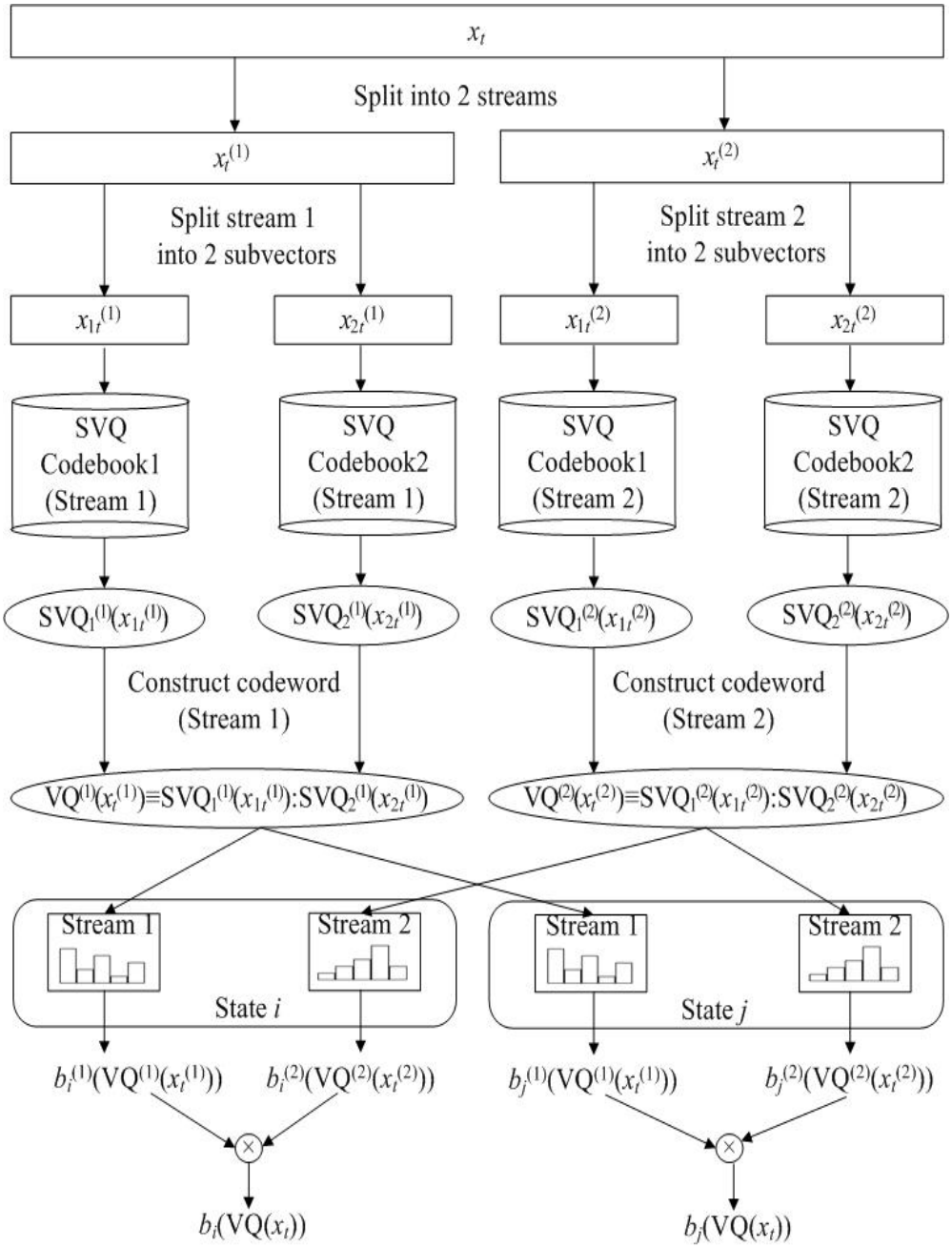


Figure 4.2: An example of a 2-stream system with SVQ codebooks. Each stream is further split into 2 subvectors with 2 SVQ codebooks.

- SVQ codebooks are constructed per stream. For each stream, for example, the sth stream, the feature vector $x_t^{(s)}$ is further split into $L^{(s)}$ subvectors: $x_t^{(s)} = [x_{1t}^{(s)}, x_{2t}^{(s)}, \ldots, x_{L^{(s)}t}^{(s)}]$.

  The full space codeword in stream s is constructed as a product of the $L^{(s)}$ SVQ codewords in that stream as:

$$VQ^{(s)}(x_t^{(s)}) \equiv SVQ_1^{(s)}(x_{1t}^{(s)}) : SVQ_2^{(s)}(x_{2t}^{(s)}) : \cdots : SVQ_{L^{(s)}}^{(s)}(x_{L^{(s)}t}^{(s)}),$$

  where $SVQ_i^{(s)}(\cdot)$ denotes the SVQ codeword for the ith subvector of stream s. The constructed codeword $VQ^{(s)}(x_t^{(s)})$ is used as an index to get the state output probability from stream s in state j by a table lookup.

Finally, to distinguish between stream and subvector, we use superscript s to denote stream index, and subscript $_i$ (usually put immediately before the other subscript t, which denotes the frame index) to denote the subvector index throughout this thesis.

4.2 Starting Point

This section describes the work back in 2005 [53], which is our previous group's first attempt at HDDHMM, and serves as the starting point of this thesis. We show in details how they tackled the five issues in HDDHMM at that time.

4.2.1 High-density DHMM (HDDHMM) System

Codebook Type: Scalar-quantized (SQ) Codebook

There were two reasons SQ codebooks were chosen for HDDHMM in [53].

- Both codebook construction and codeword lookup are very fast. Fast codebook operations are essential for HDDHMM with a very large codebook to succeed.
- To robustly estimate the parameters, HDDHMM was converted directly from a well-trained CDHMM without further training. Based on SQ codebooks, the conversion from CDHMM to HDDHMM involves only simple per-dimensional integration with closed-form solutions. Details will be given in the later part of this section.

Table 4.2: Bit allocation for each stream for HDDHMM in [53]

Stream	Feature	Per-dimensional Bit Allocation	# Bins
1	12MFCCS	$[(2)]^2[(1)]^{10}$	16384
2	12ΔMFCCS	$[(1)]^{12}$	4096
3	12ΔΔMFCCS	$[(1)]^{12}$	4096
4	$e + \Delta e + \Delta\Delta e$	$[(3)]^3$	512

Number of streams: 4

HDDHMM is a 4-stream HMM using the 4-stream partition S4 (see Section 4.1.2) with the assumption that different streams are independent. Each stream of an HMM state has its own high-density discrete distribution.

Although multi-stream systems fail to model the correlations among different streams, it is impossible to build a 1-stream HDDHMM in this case: for a typical 39-dimensional MFCC feature vector used in the task, even a 2-codeword SQ codebook for each dimension will lead to a codebook size of 2^{39} in the full space in a 1-stream system, which implies that each HMM state needs to store a discrete distribution table of size 2^{39}!

Modeling Stream Correlations? No

The system does not model stream correlations. In other words, it is a typical multi-stream system that assumes different streams are independent.

Codebook Size: By Experience

The SQ codebook size (represented in bits) for each dimension as well as the number of codewords (bins) per stream are shown in Table 4.2, following the notation of *subvector-partition and bit-allocation scheme* defined in Section 4.1.4.

As a preliminary study, the SQ codebook size (or bit-allocation) was not determined systematically in this system. Basically, the heuristics below are followed to allocate bits for each dimension:

- For stream 1, 2 and 3, first allocate one bit (i.e., 2 codewords) per dimension. Consider the fact that 1st and 2nd MFCC coefficients are usually more important for speech recognition, 1 more bit is allocated to them.

- For stream 4, since it only contains 3 dimensions, more bits may be allocated for each dimension. Thus, 3 bits are used per dimension.

The number of bins (codewords) in a stream could be computed by 2^B, where B denotes the total number of bits in that stream. Notice that the number of bins per stream also determine the discrete distribution table size in that stream of an HMM state. In Table 4.2, the number of bins in the 4 streams are 16384, 4096, 4096, and 512. That is, the 4 streams of a state in the HDDHMM store discrete distribution tables of size 16384, 4096, 4096 and 512 respectively. That is why the model is called *high-density* DHMM (HDDHMM), as opposed to conventional DHMM where the discrete distribution table for each stream of a state has a size of at most 1024.

Robust Parameter Estimation: Conversion of CDHMM to HDDHMM

For such high-density DHMM, there are insufficient training data in WSJ to train the model robustly. In the system, a simple algorithm has been investigated to convert the continuous density of a CDHMM state to a reliable estimate of the discrete distribution of the corresponding HDDHMM state.

The conversion procedure is summarized as follows:

- STEP 1 : To create an S-stream HDDHMM, an S-stream CDHMM is first created with appropriate parameter tying.
- STEP 2 : Scalar quantization (SQ) is performed on each dimension to give the desired number of codewords per dimension.
- STEP 3 : For each SQ codeword, determine the lower and upper bounds. Denote these bounds for the ith dimension by (l_i, u_i).
- STEP 4: Each bin in a d-dimensional stream is represented by a multiplicative combination of the per-dimension SQ codewords from its d dimensions. Thus, each bin is a hypercube with bounds $\{(l_1, u_1), (l_2, u_2), \ldots, (l_d, u_d)\}$ for its d sides.
- STEP 5 : The corresponding S-stream HDDHMM will have the same topology as that of the S-stream CDHMM, except that, for each stream, the CDHMM

state probability density functions are now replaced by HDDHMM's state distributions computed by integration. That is, if the pdf for the sth stream of the jth CDHMM state with a mixture density of $M_j^{(s)}$ Gaussian components is

$$b_j^{(s)}(x_t^{(s)}) = \sum_{m=1}^{M_j^{(s)}} c_{jm}^{(s)} \mathcal{N}(x_t^{(s)}; \mu_{jm}^{(s)}, (\sigma_{jm}^{(s)})^2,$$

the bin with bounds $\{(l_1, u_1), (l_2, u_2), \ldots, (l_d, u_d)\}$ on the corresponding HDDHMM stream of state j has the following probability (assuming the stream contains d dimensions):

$$\begin{aligned}
& \sum_{m=1}^{M_j^{(s)}} c_{jm}^{(s)} \int_{l_1}^{u_1} \int_{l_2}^{u_2} \ldots \int_{l_d}^{u_d} \mathcal{N}(x_t^{(s)}; \mu_{jm}^{(s)}, (\sigma_{jm}^{(s)})^2) dx_t^{(s)} \\
= & \sum_{m=1}^{M_j^{(s)}} c_{jm}^{(s)} \int_{l_1}^{u_1} \mathcal{N}(x_{1t}^{(s)}; \mu_{1jm}^{(s)}, (\sigma_{1jm}^{(s)})^2) dx_{1t}^{(s)} \\
& \int_{l_2}^{u_2} \mathcal{N}(x_{2t}^{(s)}; \mu_{2jm}^{(s)}, (\sigma_{2jm}^{(s)})^2) dx_{2t}^{(s)} \\
& \ldots \int_{l_d}^{u_d} \mathcal{N}(x_{dt}^{(s)}; \mu_{djm}^{(s)}, (\sigma_{djm}^{(s)})^2) dx_{dt}^{(s)},
\end{aligned}$$

where $\mathcal{N}(x_{it}^{(s)}; \mu_{ijm}^{(s)}, (\sigma_{ijm}^{(s)})^2)$ represents the univariate Gaussian density of the ith dimension of the mth component of the sth stream in the jth CDHMM state. The integral for each dimension can be computed using the $erf(\cdot)$ function as follows:

$$\begin{aligned}
& \int_{l_i}^{u_i} \mathcal{N}(x_{it}^{(s)}; \mu_{ijm}^{(s)}, (\sigma_{ijm}^{(s)})^2) dx_{it}^{(s)} \\
= & \; 0.5 \left[erf\left(\frac{u_i - \mu_{ijm}^{(s)}}{\sqrt{2}(\sigma_{ijm}^{(s)})^2} \right) - erf\left(\frac{l_i - \mu_{ijm}^{(s)}}{\sqrt{2}(\sigma_{ijm}^{(s)})^2} \right) \right]
\end{aligned}$$

4.2.2 Baseline Model Performance

The performance for different baseline acoustic models are evaluated on the WSJ SI-84 task (see Section 4.1.1). The experimental settings are basically the same as in [53],

except that we use the trigram rather than the bigram language model, since trigram language model is more commonly used in state-of-the-art ASR systems.

We compare 4-stream HDDHMM with 1-stream CDHMM, 4-stream CDHMM, and conventional 4-stream DHMM. The standard 4 stream definition S4 (see Section 4.1.2) is adopted in the 4-stream models. The common model configurations are shown in Section 4.1.1.

For each model, we try best effort to figure out its optimal setting using the development set. The details of the models with optimal configurations are given below.

- 1s-CDHMM: 1-stream continuous density HMMs (CDHMM) with 16 Gaussian mixture components per state.
- 4s-CDHMM: 4-stream CDHMM, which is trained as follows: a 1-stream CDHMM with only 1 Gaussian mixture per state is developed which has the same HMM topology as the baseline 1s-CDHMM. The 1-stream 1-mixture CDHMM system is then converted to a 4-stream 1-mixture CDHMM system, and the number of mixtures is grown to 16 for each stream until we have a 4-stream 16-mixture CDHMM system.
- 4s-DHMM: 4-stream DHMM has 512 codewords for each MFCC stream, and 256 codewords for the energy stream.
- 4s-HDDHMM: 4-stream HDDHMM, converted from 4s-CDHMM, has 16,384 codewords for the static MFCC stream, 4096 codewords for both delta and delta-delta MFCC streams, and 512 codewords for the energy stream (as shown in Table 4.2).

For DHMM and HDDHMM, *flooring* (see Section 3.2) is applied to avoid the state output probability of some codewords being too small. The flooring thresholds for DHMM and HDDHMM are set to be 10^{-6} and 10^{-12} respectively, which are found to work well in practice.

The baseline model recognition results as well as the model size are shown in Table 4.3. The CDHMM results are comparable to others in the literature for the same task [68, 48].

Table 4.3: Baseline model performance on WSJ SI-84

Acoustic Model	WER	Memory (in MB)
1s-CDHMM	4.46%	16
4s-CDHMM	5.25%	16
4s-DHMM	7.90%	11
4s-HDDHMM	6.71 %	157

Recognition Performance

As expected, compared with 1s-CDHMM, 4s-DHMM shows a significant degradation in recognition performance. There is also a small degradation in the recognition performance with the use of 4s-CDHMM, probably due to the fact that the multi-stream CDHMM cannot model the correlation among features from different streams.

4s-HDDHMM significantly improves the recognition accuracy of 4s-DHMM by 1.19% absolute, due to the use of a large codebook. However, its performance is worse than 4s-CDHMM by 1.48% absolute, and is worse than 1s-CDHMM by 2.25% absolute.

Model Size

HDDHMM requires much more memory space to store its state output distributions than other models. Before describing how to compute the size for each model, we need to introduce the parameter storage (defined by the HTK toolkits [84]) as it differs between continuous and discrete models.

Parameter Storage: for CDHMM, the Gaussian means, variances and mixture weights are stored in a 4-byte float, while the situation for discrete systems is different. The output probabilities of DHMM or HDDHMM are stored as logs in a scaled 2-byte short integer format such that if $b_j^{(s)}(VQ^{(s)}(x_t^{(s)}))$ is the stored discrete probability for codeword $VQ^{(s)}(x_t^{(s)})$ in stream s of state j, the true probability is given by

$$\exp(-b_j^{(s)}(VQ^{(s)}(x_t^{(s)}))/c)$$

, where c is a constant set by the user, which implies that the range of probabilities from 1.0 down to $\exp(32767/c)$ could be stored in the model. Storage in the form of scaled logarithms allows discrete probability HMMs to be implemented very efficiently since

HTK tools mostly use $\log$ arithmetic and direct storage in logarithms avoids the need for a run-time conversion.

The model sizes are computed as follows:

- 1s-CDHMM: $3132 \times 16 \times (1 + 39 \times 2) \times 4 = 16$MB (3132 tied states, each with 16 components; each component consists of a scalar weight, a 39-dimensional Gaussian mean vector, and a 39-dimensional Gaussian variance vector. Each of these quantities is stored in a 4-byte float).

- 4s-CDHMM: $3132 \times 16 \times (4 + 39 \times 2) \times 4 = 16$MB (similar computation as in the case of 1s-CDHMM, except that the number of mixture weights should be multiplied by 4, as each of the 4 streams in a state has its own set of mixture weights).

- 4s-DHMM: $3132 \times (512 + 512 + 512 + 256) \times 2 = 11$MB (3132 tied states, each state with 4 streams; the discrete distribution table size of each stream is 512, 512, 512 and 256. Each probability value is stored in a 2-byte short integer).

- 4s-HDDHMM: $3132 \times (16384 + 4096 + 4096 + 512) \times 2 = 157$MB (similar computation as in the case of 4s-DHMM).

4.3 The Increase of Codebook Size

The codebook size per stream and the bit-allocation per dimension in baseline HDDHMM were decided in an ad-hoc manner. Also limited by the memory in year 2005 (with 1G Hz total memory), larger codebook size were not tried in [53]. Based on the baseline HDDHMM, this section systematically increases the codebook size, and show its effect on recognition performance.

For 4-stream HDDHMM, the number of codewords in each stream could be different. Furthermore, for the same number of codewords in a stream, there exists many different bit-allocation schemes. It is impossible to try all possible configurations. As a compromise, the following heuristic-based strategy is employed.

- First decide the total number of bits for each stream. We force each of the first 3 streams to have the same number of bits, and the total number of bits in the 4th stream is one bit less compared with the first 3 streams as it contains only 3

Table 4.4: Effect of codebook size for 4s-HDDHMM with SQ codebooks

Scheme	Codebook Size (4 streams)	Bit-alloc (Stream 1)	Bit-alloc (Stream 2)	Bit-alloc (Stream 3)	Bit-alloc (Stream 4)	Memory (MB)	Test WER	Dev WER
SQ-a	16384, 4096, 4096, 512	$[(2)]^2[(1)]^{10}$	$[(1)]^{12}$	$[(1)]^{12}$	$[(3)]^3$	157	6.71%	9.22%
SQ-b	16384, 16384, 16384, 8192	$[(2)]^2[(1)]^{10}$	$[(2)]^2[(1)]^{10}$	$[(2)]^2[(1)]^{10}$	$(5)[(4)]^2$	359	6.5%	8.96%
SQ-c	32768, 32768, 32768, 16384	$[(2)]^3[(1)]^9$	$[(2)]^3[(1)]^9$	$[(2)]^3[(1)]^9$	$[(5)]^2(4)$	718	6.46%	8.55%
SQ-d	65536, 65536, 65536, 32768	$[(2)]^4[(1)]^8$	$[(2)]^4[(1)]^8$	$[(2)]^4[(1)]^8$	$[(5)]^3$	1437	6.26%	8.12%

dimensions. For example, we decide that the total number of bits for each of the 4 streams is 15, 15, 15, and 14 respectively, giving a codebook size of 2^{15}, 2^{15}, 2^{15}, and 2^{14} in each stream.

- For the energy stream with e, Δe, and $\Delta\Delta e$, we want each of the 3 dimensions to contain roughly the same number of bits. Specifically, the total number of bits in the stream is divided by 3, and if:
  - the remainder = 0, each of the 3 dimensions will get exactly the same number of bits.
  - the remainder = 1, e will get one more bit than Δe, and $\Delta\Delta e$.
  - the remainder = 2, e and Δe will get one more bit than $\Delta\Delta e$.

  In our example since the total number of bits is decided to be 14, the number of bits for e, Δe and $\Delta\Delta e$ is 5, 5 and 4 respectively.

- For the other 3 streams, we want them to use the same bit-allocation scheme (i.e., the bit allocations for the 12 dimensions in stream 1, 2 and 3 are the same). Furthermore, the number of bits for each dimension (from left to right) in a stream should be in a non-increasing order. That is, we want to put more bits to lower-order MFCCs, assuming that lower-order MFCCs are more important for recognition tasks. Continue with our running example, for each stream with 15 bits, we will try the following bit-allocation schemes [1].
  - $[(2)]^3[(1)]^9$
  - $(3)(2)[(1)]^{10}$
  - $(4)[(1)]^{11}$

[1] a dimension must have at least one bit, otherwise the discrimination information from this dimension will be lost.

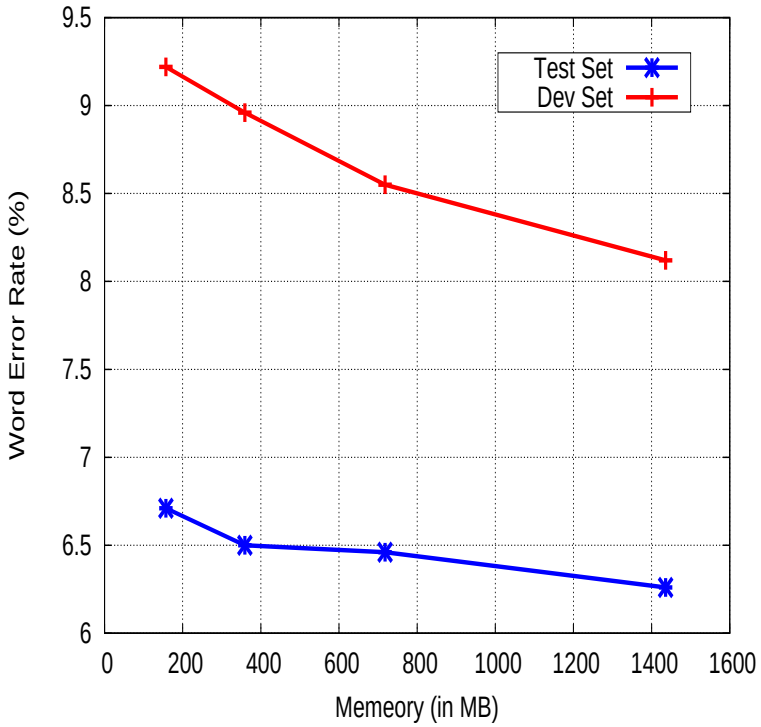


Figure 4.3: Relationship between model size and recognition error rate for HDDHMM with SQ codebooks

The effect of a codebook size for 4s-HDDHMM is shown in Table 4.4. Scheme SQ-a (first row) is the configuration of our baseline HDDHMM in Section 4.2. The other schemes follow the strategy described above; that is, we first fix the total number of bits (codebook size) per stream, and try different bit-allocation schemes. For each scheme, the bit allocation shown in the table is the one that gives the best WER on the development (dev) set. From SQ-b to SQ-d, the codebook size doubles each time. We stop with the codebook size {65536, 65536, 65536, 32768} (i.e., SQ-d), as it already results in a model size of 1,437 MB. The relationship between the model size and recognition performance is also shown in Figure 4.3.

The information contained in both the table and figure could be summarized as follows:

- Comparing the largest codebook size {65536, 65536, 65536, 32768} (SQ-d) with baseline codebook size (SQ-a), it reduces the WER by 0.45% absolute and 1.1% absolute on the test and dev sets respectively.

- Increasing the codebook size improves the recognition performance. As shown in Figure 4.3, an almost linear relationship is observed between the model size and recognition error rate.

- Unfortunately, the slope for the linear relationship is not large (especially for the test set). Thus, to get a comparable performance as a continuous model, simply increasing the codebook size is not a feasible solution. In Figure 4.3, if we fit the best straight line to the results on the test set, the linear function is: WER = -0.0003 $\times$ memory + 6.6891. To achieve the CDHMM WER 4.46%, a memory the size of 7,430 MB is required, which is beyond the memory capacity of our system. Furthermore, whether this linear trend could be kept by continuing to grow the codebook size is an issue.

In the following sections, we start to explore other dimensions of HDDHMM, with the codebook sizes fixed at $\{65536, 65536, 65536, 32768\}$.

4.4 SQ to SVQ Codebooks

This section generalizes the SQ codebook used in the baseline model to SVQ codebooks (with subvector dimension larger than 1). To distinguish HDDHMM with SQ codebooks and SVQ codebooks, we call them *SQ-HDDHMM* and *SVQ-HDDHMM* respectively.

Unlike SQ codebooks which partition the full feature space into hypercubes whose faces are perpendicular to the axis of the feature dimension, SVQ codebooks partition the full feature space into convex polytopes. As a result, given the same number of codewords, SVQ codebooks are more efficient in representing the feature space than SQ codebooks. Better performance of SVQ over SQ was proven in the speech coding task [61], and some ASR tasks [9, 78].

4.4.1 Difficulties in Conversion from CDHMM

The major difficulty of using SVQ codebooks for HDDHMM lies in the conversion from the CDHMM. Similar to SQ-HDDHMM of Section 4.2.1, to robustly estimate the model parameters of SVQ-HDDHMM, it may be directly converted from CDHMM without further re-estimation. Each GMM probability density function (pdf) in CDHMM is converted to a discrete distribution of the corresponding SVQ-HDDHMM by finding the probability of each VQ codeword. The computation boils down to integrating a GMM pdf over the convex polytope representing each VQ codeword. Suppose a VQ codeword in the sth stream is $v^{(s)}$, which is constructed from $L^{(s)}$ SVQ codewords

as: $v^{(s)} \equiv v_1^{(s)} : v_2^{(s)} : \cdots : v_{L^{(s)}}^{(s)}$, where $v_i^{(s)}$ denotes the SVQ codeword for the ith subvector of the stream s. Let us also denote the convex polytopes for full-space VQ codewords and subvector SVQ codewords by $\Omega(v^{(s)})$ and $\Omega(v_i^{(s)})$ respectively. Hence, the probability of generating the VQ codeword $v^{(s)}$ from stream s in state j of SVQ-HDDHMM could be converted from the corresponding pdf in diagonal-covariance CDHMM as follows:

$$
\begin{aligned}
b_j^{(s)}(x_t^{(s)} \in \Omega(v^{(s)})) &= \sum_{m=1}^{M_j^{(s)}} c_{jm}^{(s)} \int_{x_t^{(s)} \in \Omega(v^{(s)})} \mathcal{N}(x_t^{(s)}; \mu_{jm}^{(s)}, (\sigma_{jm}^{(s)})^2) dx_t^{(s)} \\
&= \sum_{m=1}^{M_j^{(s)}} c_{jm}^{(s)} \int_{x_{1t}^{(s)} \in \Omega(v_1^{(s)})} \mathcal{N}(x_{1t}^{(s)}; \mu_{1jm}^{(s)}, (\sigma_{1jm}^{(s)})^2) dx_{1t}^{(s)} \\
&\quad \int_{x_{2t}^{(s)} \in \Omega(v_2^{(s)})} \mathcal{N}(x_{2t}^{(s)}; \mu_{2jm}^{(s)}, (\sigma_{2jm}^{(s)})^2) dx_{2t}^{(s)} \\
&\quad \cdots \int_{x_{L^{(s)}t}^{(s)} \in \Omega(v_{L^{(s)}}^{(s)})} \mathcal{N}(x_{L^{(s)}t}^{(s)}; \mu_{L^{(s)}jm}^{(s)}, (\sigma_{L^{(s)}jm}^{(s)})^2) dx_{L^{(s)}t}^{(s)}
\end{aligned}
$$

where $x_{it}^{(s)}$, $\mu_{ijm}^{(s)}$, and $(\sigma_{ijm}^{(s)})^2$ are the subvectors of $x_t^{(s)}$, $\mu_{jm}^{(s)}$, and $(\sigma_{jm}^{(s)})^2$ in the ith partition of the sth stream respectively.

Unlike SQ-HDDHMM, where the integral is carried out in a one-dimensional space with closed-form solutions, there is no closed-form solution for the integrals over the convex polytope of each SVQ codeword in the subvector space:

$$
\int_{x_{it}^{(s)} \in \Omega(v_i^{(s)})} \mathcal{N}(x_{it}^{(s)}; \mu_{ijm}^{(s)}, (\sigma_{ijm}^{(s)})^2 dx_{it}^{(s)} \tag{4.1}
$$

This thesis proposes two ways to solve the conversion problems, one is by numerical integration, the other by single-point approximation.

4.4.2 Conversion by Numerical Integration

This section explores the numerical integration for Formula (4.1).

To integrate over a polytope is a #P-hard problem [18]. Two commonly used solutions are standard numerical integration and Monte Carlo (MC) integration [74], both belonging to the numerical integration methods. When the integrand belongs to a

specific function class, e.g., polynomial functions, fast integration methods were also proposed [18].

Generally speaking, in numerical integration, the integrand is evaluated at a finite set of points called *integration points* and a weighted sum of these values is used to approximate the integral.

In MC integration, the integration points are randomly sampled from the integration region. By law of large numbers, given a sufficient number of points, the approximation will converge to the correct integral value. In fact, the approximation error of MC integration is proportional to $1/\sqrt{n}$, where n is the number of integration points. Since the error does not depend on the dimension of space where integral is carried out, it makes the method very attractive for high-dimensional polytope integration.

Unlike the MC methods, in standard numerical integration, the integration points are chosen carefully. These methods usually start by solving the one-dimensional integrals, and the multiple dimensional integrations could be broken down into repeated one-dimensional integrals. In standard numerical integration, an original one-dimensional integrand is approximated by an easy-to-integrate interpolating function, which has the same function value as the original integrand at the integration points. Typically, the interpolating functions are polynomials. The form of the interpolating function and the choice of integration points (together, they are called the *quadrature rules*) distinguish one method from another. Several commonly used quadrature rules with increasing approximation power are listed below.

- Rectangle rule: the interpolating function is a constant function (i.e., polynomial of degree zero), and the integration point is chosen to be the middle point of the interval.

- Trapezoidal rule: the interpolating function is an affine function (i.e., polynomial of degree one), and the integration points are chosen to be the two endpoints of the integration interval.

- Simpson's rule: the interpolating function is a quadratic function, and the integration points are chosen to be the middle point as well as the two endpoints of the integration interval.

- Newton-Cotes quadrature's rule: given n equally-spaced integration points, this rule approximates the integral with a polynomial of degree $n-1$. If the integrand

itself is a polynomial of degree $n-1$ or less, a perfect approximation without errors will be obtained. In fact, rectangle, trapezoidal, and Simpson's are all special instances of Newton-Cotes's rule.

- Gaussian quadrature's rule: in the above 3 rules, the intervals between integration points are regular. If we allow them to vary, it will lead to Gaussian quadrature's rule, which could exactly approximate the integral for a polynomial of degree $2n-1$ or less by carefully choosing n integration points.

To get an accurate approximation in practice, the integration interval in standard numerical integration is usually broken down into a number of small sub-intervals, with approximation computed for each sub-interval. The sum of integrals from each subinterval is used to approximate the original integral. Depending on how an interval is broken up, two different rules were proposed in the past:

- Composite rule: the interval is divided into pre-defined sub-intervals, and the integral is carried out at each sub-interval.

- Adaptive rule: the numerical integration is first carried out on the full interval, and the approximation error is estimated. If the error exceeds a pre-defined tolerance, the interval will be divided into two, and the adaptive rule is applied to both subintervals in a recursive manner. In practice, the adaptive rule is usually found to be more efficient than the composite rule.

Unfortunately, in standard numerical integration, given the same number of integration points n, the approximation error increases exponentially when the space dimension d increases. For example, the approximation error for integration with a composite Simpson's rule is proportional to $1/n^{4/d}$. As a result, the standard numerical integrations are more applicable to lower-dimensional polytope integration.

Choice of Integration Method

In our experiments, standard numerical integration rather than MC integration, is used for the following reasons:

- The space for integral is not high-dimensional, which makes it more suitable to apply standard integration. As a rough estimate, one could compare the approximation errors of standard integration with the composite Simpson' rule and

MC integration are $1/\sqrt{n}$ vs. $1/n^{4/d}$. It could easily be deduced that when the integral dimension d is not greater than 8, standard integration will give better approximate error for the same number of integration points n. In our thesis, the integral is carried out for polytopes in subvector space (see Formula (4.1)), with each subvector consisting of at most 6 dimensions in our experiments. As a result, the standard numerical integration is expected to be better.

- There exists routines for standard numerical integration in Matlab, which could compute integrals to a known accuracy.

Practical Problems for Standard Numerical Integration with Matlab

Matlab already provides the following routines for standard numerical integration:

- *quad*: a one-dimensional integration routine, using the Simpson quadrature rules. It is a fast routine, but not very accurate.
- *quadgk*: a one-dimensional integration routine, using the adaptive Gauss-Kronrod quadrature rules [2]. quadgk is usually more accurate than quad, but much slower.
- *dblquad*: a two-dimensional integration routine, which iteratively calls a one-dimensional integration routine to evaluate the double integral. The one-dimensional integration routine could be quad or quadgk.
- *triplequad*: a three-dimensional integration routine, which iteratively calls a one-dimension integration routine to evaluate the triple integral. The one-dimensional integration routine could be quad or quadgk.

However, we are not satisfied with the above routines for the following reasons:

- For a multiple-dimensional integration, it is either too slow or too inaccurate, depending on which one-dimensional routine it calls. For example, for the three-dimensional integration routine triplequad, if it calls quad, the result is fast but not accurate; however, if it calls quadgk, the result is accurate but takes a long time.
- It only supports integration of up to 3 dimensions.

[2] Gauss-Kronrod quadrature is an extension of Gauss quadrature, in which the integration points are chosen so that an accurate approximation can be computed by re-using the information produced by the computation of a less accurate approximation.

Self-Developed Integration Routine

To solve the above problems, we developed our own Matlab routines to evaluate integrals. Similar to dblquad and triplequad, our routines also iteratively make use of one-dimensional integration routines such as quad or quadgk. The differences compared with the Matlab in-built routines are:

- In dblquad and triplequad, the maximum allowable integral dimension is 3; while our routine generalizes it to any dimension using a recursive function.
- Unlike dblquad or triplequad which is allowed to call only one type of integration routine (either quad or quadgk) during an integral, our routine uses both quad and quadgk simultaneously in a multiple-dimensional integral. Specifically, we only used quadgk for the integration of the innermost dimension to guarantee accuracy, and quad for the remaining dimensions to save time. Thus, it obtains a good trade-off between integration speed and accuracy.

To show the effectiveness of our routines, we carry out integration on both two- and three-dimensional space. In the two-dimensional case, we partition a two-dimensional subvector feature space into 16 clusters, each represented by a polytope. A Gaussian is integrated over each of the 16 polytopes. While for 3-dimensional integration, a 3-dimensional space is partitioned into 64 polytopes, and a Gaussian is integrated over each of them. Notice that some polytopes may be un-bounded.

We compare different routines below:

- dblquad/triplequad (quad): Matlab routine dblquad/triplequad which iteratively calls quad.
- dblquad/triplequad (quadgk): Matlab routine dblquad/triplequad which iteratively calls quadgk.
- my routine (d dimension): my routine for d-dimensional integrals, which uses quad and quadgk at different level of integrals.

The effectiveness of the various integration routines are judged by the following two criterions.

Table 4.5: Comparison of different integration routines

# Dimensions	Routine	Integration Time Per Polytope (in seconds)	Sum of Integrals
2	dblquad (quad)	0.25	0.9979
2	dblquad (quadgk)	8.56	0.9999
2	mine (2 dimension)	0.38	0.9982
3	triplequad (quad)	0.52	0.9725
3	triplequad (quadgk)	252.09	0.9998
3	mine (3 dimension)	1.47	0.9942

- Integration time per polytope: on average, how long it takes to finish the integral over a polytope. The faster, the better.
- Sum of integral values: the sum of the integral values over all the polytopes. Ideally, the sum should be equal to 1. Thus, the closer it is to one, the more accurate the integration is.

The effectiveness of different routines are shown in Table 4.5 and the error tolerance is set to be 1e-5. In Matlab, the integration error is estimated as the value difference between two different approximation methods for an integral over the same region. From Table 4.5, we observe that:

- My routine shows a good tradeoff between accuracy and speed in integration over both 2- and 3-dimensional space. However, particularly in the 3-dimensional case, our routine should be the only choice for the later experiments which require tens of millions of integration operations, as the other two routines are either very inaccurate (triplequad (quad) sums up to only 0.9725) or too slow (triplequad (quadgk) uses 252 seconds to compute an integral).
- The integration time grows significantly with the dimensionality of the space. In my routine, for example, it is 4 times slower to integrate a polytope in 3-dimensional space than a 2-dimensional one.

4.4.3 Conversion by Single-point Approximation

Converting an SVQ-HDDHMM from CDHMM using numerical integration is very slow. Even using my routine, it takes about 1.47 seconds to integrate over a polytope

in 3-dimensional space (see Table 4.5). In our experiments, there have been tens of millions of such integral operations, which makes the conversion very slow.

Instead of doing numerical integration, this section uses a single function value evaluated at the SVQ codeword of the polytope as an approximation, which was also used in discrete-mixture HMM systems in [78]. In other words, the integration formula in (4.1) is replaced by:

$$\frac{\mathcal{N}(v_i^{(s)}; \mu_{ijm}^{(s)}, (\sigma_{ijm}^{(s)})^2)}{\sum_{a_i^{(s)}} \mathcal{N}(a_i^{(s)}; \mu_{ijm}^{(s)}, (\sigma_{ijm}^{(s)})^2)} \tag{4.2}$$

where $v_i^{(s)}$ is the SVQ codeword of the polytope in the ith subvector of stream s, and $a_i^{(s)}$ is any SVQ codeword in the ith subvector of stream s. The denominator actually serves as a normalization term, which makes the sum of output probabilities over all the codewords to be one.

Since this method only evaluates at one point, it is very fast when compared with numerical integration.

4.4.4 Experiments

We carry out experiments to evaluate the use of SVQ codebooks in HDDHMM, and the two conversion methods. The common experimental configurations are as follows:

- A 4-stream system with codebook size {65536, 65536, 65536, 32768}.
- Subvector partitioning policy: there are too many ways to partition a vector into subvectors. In this experiment, we force a vector to be partitioned into subvectors of consecutive dimensions, and each subvector contains the same number of dimensions. Furthermore, we use the same subvector partition for each of the first 3 streams. For the 4th energy stream of only 3 dimensions, we believe that the codebook size 32768 is large enough and there is no need to use the more efficient SVQ codebooks. Thus, SQ codebooks are kept for stream 4, with one dimension per subvector.
- Bit allocation policy: since the numerical integration for model conversion is slow, it is costly for us to try different bit allocations for a given subvector

Table 4.6: Different subvector-partition and bit-allocation schemes for SVQ-HD-DHMM

Scheme	Stream 1	Stream 2	Stream 3	Stream 4
SQ-d	$[1(2)]^4[1(1)]^8$	$[1(2)]^4[1(1)]^8$	$[1(2)]^4[1(1)]^8$	$[1(5)]^3$
SVQ-a	$[2(4)]^2[2(2)]^4$	$[2(4)]^2[2(2)]^4$	$[2(4)]^2[2(2)]^4$	$[1(5)]^3$
SVQ-b	$3(6)3(4)[3(3)]^2$	$3(6)3(4)[3(3)]^2$	$3(6)3(4)[3(3)]^2$	$[1(5)]^3$
SVQ-c	$4(8)[4(4)]^2$	$4(8)[4(4)]^2$	$4(8)[4(4)]^2$	$[1(5)]^3$
SVQ-d	$6(10)6(6)$	$6(10)6(6)$	$6(10)6(6)$	$[1(5)]^3$

Table 4.7: Performance of 4-stream SVQ-HDDHMM using different subvector-partition bit-allocation schemes and conversion methods

Scheme	Conversion Method	WER
SQ-d	Closed-form Integration	6.26%
SVQ-a	Numerical Integration	6.41%
SVQ-b	Numerical Integration	5.64%
SQ-d	Single Point Approximation	6.86%
SVQ-a	Single Point Approximation	6.54%
SVQ-b	Single Point Approximation	6.35%
SVQ-c	Single Point Approximation	6.22%
SVQ-d	Single Point Approximation	6.09%

partition. As a result, the bit-allocation scheme for SVQ-HDDHMM is borrowed from the optimal bit-allocation scheme SQ-d in SQ-HDDHMM (see Section 4.3). Specifically, for a particular subvector partition scheme, the number of bits assigned to a subvector is set to be the total number of bits in the same subvector of SQ-d.

- The subvector-partition and bit-allocation schemes used in this experiment are shown in Table 4.6. To be consistent with other SVQ schemes in notation, the dimension of each subvector is also given for SQ-d, although the value is always 1. From SQ-d to SVQ-d, the subvector dimension increases from 1 to 2, 3, 4 and 6 respectively. The subvector dimension 6 is the largest we try, which already results in 1024 SVQ codewords in some subvectors.

- The error tolerance of numerical integration is set to 1e-5.

The recognition performance for different subvector-partition and bit-allocation schemes using the 2 conversion methods is shown in Table 4.7, where SQ-d with closed-form integration is actually the optimal SQ-HDDHMM described in Section 4.3.

The following observations can be made:

- The larger the subvector dimension, the better the performance. In *single-point approximation*-based conversion, WER decreases monotonically with increasing subvector dimension. In an integration-based conversion, SVQ-b (subvector dimension of 3) is also significantly better than SVQ-a (subvector dimension of 2) and SQ-d (subvector dimension of 1). It is a little unexpected that SVQ-a is worse than SQ-d in integration-based methods, which may be explained by (1) the closed-form integration in SQ-d is more accurate than numerical integration in SVQ-a; and (2) the subvector dimension is only 2 in SVQ-a, and each subvector has at most 4 SVQ codewords. Thus, even if SVQ is actually better than SQ, it may not clearly show up in such a small subvector dimension.

- The numerical integration is more accurate than single-point approximation, although the latter is much faster. For the same scheme, integration always performs better than single-point integration.

- The best SVQ-HDDHMM result 5.64% is given by SVQ-b with numerical integration, which is 0.62% absolute point better than the best SQ-HDDHMM performance (6.26% absolute).

- Although numerical integration is accurate, we cannot try larger subvector dimensions. In fact, 3 dimensions per subvector in SVQ-b is already the maximum we could try in our systems. The time spent in SVQ-b with numerical integration could roughly be estimated as the multiplication of the following two factors:

  - Time to integrate a Gaussian over a 3-dimensional polytope: 1.47 seconds (see Table 4.5).
  - The number of such integral operations during the conversion from CDHMM: 14.4 millions, which could be obtained by simply counting how many times the formula 4.1 is called during model conversion.

  As a result, the model conversion time for SVQ-b with numerical integration will be 211,680, 000 seconds, or 5,880 hours, which takes us 3 days when running over 80 computers simultaneously. Since the integration time increases significantly with the subvector dimension, even extending it to SVQ-c is very difficult.

Table 4.8: Performance of CDHMM with different stream definitions

# Streams	Stream Definition	WER
1	S1	4.46%
2	S2	4.60%
3	S3-a	5.12%
3	S3-b	4.95%
3	S3-c	4.89%
3	S3-d	4.88%
4	S4	5.25%

4.5 The Use of Fewer Streams

Until now, HDDHMM uses 4 streams, assuming different streams are independent. The use of 4 streams follows the common practice in conventional DHMM, which is reported to give a WER reduction of more than 10% when compared with single-stream systems [35]. The major reason to use multiple streams, as mentioned in Section 2.3.4, is that the codebook in conventional DHMM consists of about a thousand codewords, which are far from enough to represent the high-dimensional (e.g., 39-dimensional MFCCs) feature space. By splitting the feature vectors into multiple lower-dimensional streams and representing each by a codebook independently, a better representation of the acoustic space is obtained.

4.5.1 Problems in Multiple-stream Systems

Despite better modeling power, multiple-stream systems fail to model the correlations among different streams, as they are assumed to be independent. However, no matter how carefully one partitions the acoustic features into streams, they cannot be perfectly independent. Thus, the lack of stream correlation modeling may hurt the recognition performance.

Table 4.8 shows CDHMM systems with different stream definitions (see Section 4.1.2 for the detailed definition), where S1 and S4 correspond to our baseline 1stream- and 4stream-CDHMM in Table 4.3. All models have the same number of triphones and state tying structure, and each stream in a state consists of 16 Gaussian mixture components. As a result, the WER difference among the models could be attributed to the use of different stream partitions.

As could be seen from Table 4.8, the fewer streams give better recognition performance. In other words, more streams lead to greater loss of correlation information among features in different streams.

It is worth pointing out that the argument that supports the use of multiple streams in DHMM (i.e., to increase the representation power), may not be applicable to CDHMM, as the latter model uses continuous distribution and does not need to quantize the feature space.

4.5.2 SVQ-HDDHMM with Fewer Streams

Investigating SVQ-HDDHMM with fewer streams is motivated by

- As shown in the last section, fewer streams imply smaller loss of correlation information.
- The codebook in SVQ-HDDHMM consists of tens of thousands of codewords, which may be able to represent higher dimensional space compared with conventional codebooks with only about a thousand codewords.

Table 4.9 shows the performance of SVQ-HDDHMM with different stream definitions, where S4 corresponds to the best SVQ-HDDHMM system in Section 4.4.4. Each SVQ-HDDHMM is converted by numerical integration from its corresponding CDHMM counterpart in Table 4.8. The codebook size and the subvector-partition and bit-allocation schemes are decided based on the following heuristics:

- Codebook size
  - On the one hand, more bits will be allocated to streams of more dimensions. Take the 2-stream system S2 as an example. Stream 1 has 26 dimensions and we allocate 18 bits to it (i.e., its codebook size is $2^{18} = 262144$); stream 2 has only 13 dimensions and we allocate 16 bits to it (i.e., its codebook size is 65536).
  - On the other hand, the codebook size cannot grow unchecked, as the model size increases linearly with the codebook size (see Section 4.2.2 for the detailed calculation of model size). Since the 4stream definition S4 already results in a model size of 1.4G bytes, the maximum codebook size we

Table 4.9: Performance of SVQ-HDDHMM with different stream definitions

Stream Def.	Codebook Size (4 streams)	Bit-alloc (Stream 1)	Bit-alloc (Stream 2)	Bit-alloc (Stream 3)	Bit-alloc (Stream 4)	WER
S4	65536, 65536, 65536, 32768	$3(6)3(4)[3(3)]^2$	$3(6)3(4)[3(3)]^2$	$3(6)3(4)[3(3)]^2$	$[1(5)]^3$	5.64%
S3-a	65536, 65536, 65536	$3(6)4(4)[3(3)]^2$	$3(6)4(4)[3(3)]^2$	$3(6)4(4)[3(3)]^2$	N/A	7.66%
S3-b	262144, 65536, 32768	$3(3)[3(2)]^3 3(3)[3(2)]^3$	$3(6)3(4)[3(3)]^2$	$[1(5)]^3$	N/A	6.48%
S3-c	65536, 262144, 32768	$3(6)4(4)[3(3)]^2$	$3(3)[3(2)]^3 3(3)[3(2)]^3$	$[1(5)]^3$	N/A	6.48%
S3-d	65536, 65536, 32768	$3(6)[3(2)]^5$	$3(6)[3(2)]^5$	$[1(5)]^3$	N/A	6.58%
S2	262144, 65536	$4(3)[3(2)]^3 4(3)[3(2)]^3$	$3(6)4(4)[3(3)]^2$	N/A	N/A	9.15%

try is 262144, which is used to represent a 24- (in S3-b or S3-c) or 26-dimensional (in S2) space.

- Subvector-partition and bit-allocation scheme:
  - Subvector dimension: except the energy stream, the dimension of a subvector in other streams is either 3 or 4. As learned from Section 4.4.4, SVQ-HDDHMM shows better performance when the subvector dimension is at least 3. However, we cannot afford too large subvector dimensions (e.g., larger than 4) as the model conversion will then be very slow.
  - Bit allocations: similar to the previous experiments, subvectors with lower order MFCC coefficients will get more bits, as the lower order MFCCs are more important for ASR.
- Experiences from 4-stream SVQ-HDDHMM (see Section 4.4.4) are well utilized. For example, when we design the scheme S3-b, we find that its second stream are exactly the same as the third streams of S4, so we copy the allocation scheme from the third stream of S4 to the second stream of S3-b.

From Table 4.9, we can see that:

- The recognition performances of both 3-stream and 2-stream SVQ-HDDHMM are worse than that of 4-stream SVQ-HDDHMM.
- The SVQ-HDDHMM performance becomes worse when the number of streams is reduced. The SVQ-HDDHMM behavior is the opposite of its parent CDHMM's. This could be explained as follows: fewer streams indeed lead to smaller loss of correlation information among streams, improving the recognition results.

However, the improvement is offset by an inadequate representation of high-dimensional space by a limited codebook size. For example, although the codebook of the 2-stream system using S2 stream definition is large (262144 for the first stream), it is still not large enough to represent the 24-dimensional space of the first stream.

As a result, in the rest of this thesis, HDDHMMs are constructed with 4 streams.

4.6 Classical Robust Parameter Estimation Techniques

Until now, the best WER 5.64% (see Table 4.7) for HDDHMM is obtained by using SVQ codebooks, whose state output distribution parameters are converted directly from CDHMM by numerical integration, without any further re-estimation. Although model conversion provides a way for HDDHMM parameter estimation by avoiding the direct training of such a high-density model, it is not an "optimal" solution for the following reasons:

- The resulting HDDHMM is only an approximation of CDHMM, as it is simply converted from the CDHMM counterpart. This deviates from one of our motivations to use HDDHMM: we expect the non-parametric discrete distribution could bring us something different (hopefully good), compared with the continuous parametric GMM.
- It is slow. As can be seen in Section 4.4.4, to generate the optimal SVQ-HDDHMM (SVQ-b) by numerical integration takes 3 days with 80 computers.
- Numerical integration cannot afford large subvector dimensions, as the conversion time increases significantly with the dimension. As a result, the maximum we try is 3 dimensions per subvector (as in SVQ-b). However, given the same number of codewords in full space, larger subvector dimensions usually result in smaller quantization, with better recognition performance.

This section and the next section will investigate the training issues of HDDHMM. In this section, we show that the classical robust parameter estimation techniques, which were developed for conventional DHMM, do not work well for an HDDHMM with hundreds more parameters. In the next section, a novel HDDHMM structure

(called “subspace HDDHMM”) based on subspace modeling is proposed, which results in tens or even hundreds of reductions in the number of free parameters. As a result, the robust parameter estimation becomes feasible.

The classical robust parameter estimation techniques (see Section 3.2) that may be extended for HDDHMM are:

- Flooring
- Interpolation
- State tying

State tying is actually used by all the acoustic models in the thesis, including HDDHMM. Thus, this section focuses on the use of flooring and interpolation in HDDHMM.

The experiments in this section are carried out on the baseline system SQ-HDDHMM (i.e., SQ-a in Table 4.4), with WER 6.71% and codebook size 16384, 4096, 4096 and 512 for each of the 4 streams. This is the HDDHMM system with the smallest codebook. The reason we work on this system is because we want to show that classical techniques are not applicable to HDDHMM. Thus, if they do not work in this relatively ”small” codebook system, they are unlikely to work in larger codebook systems.

4.6.1 Baum-Welch Re-estimation with Flooring

In actual fact, the flooring technique has already been applied to our baseline SQ-HDDHMM. After conversion from CDHMM, when some state output probability value is smaller than a predefined flooring threshold 10^{-12}, it is replaced by the threshold.

Here, we plan to further re-estimate the converted HDDHMM by the Baum-Welch algorithm for a few iterations, with flooring applied. Baum-welch is an Expectation-Maximization (E-M) type algorithm for state-of-the-art HMM training (see Section 2.2.2), which iteratively updates the HMM parameters and guarantees the increase in likelihood of the training data given the model after each iteration. The procedure in one iteration of Baum-welch for a discrete model could be described as follows:

Table 4.10: Performance of parameter re-estimation for SQ-HDDHMM (SQ-a) with flooring 10^{-12}

Iteration	Test WER	Train WER
0	6.71%	6.27%
1	67.40%	0.32%
2	73.94%	0.31%
3	77.53%	0.30%
4	79.77%	0.30%

- Expectation Step (E-step): based on the current model, the state posterior probabilities given each feature (sample) $VQ(x_t)$ are estimated. Each sample $VQ(x_t)$ is then assigned to an HMM state based on the estimated probabilities. Since the posterior probability values are usually not exactly 0 or 1, this assignment is a soft assignment, in which a sample could be assigned to more than one state with different fractional count. For example, assuming only two states i and j are in the system, if the posterior probabilities of state i and j given a sample are 0.2 and 0.8 respectively, then 0.2 of this sample is assigned to state i, and 0.8 of it belongs to state j.

- Maximization Step (M-step): under the maximum likelihood criterion, estimate a new output distribution for each state based on the samples collected in E-step, and replace the current state output distribution with the new estimated one. Specifically, for a state, to estimate its probability to output a codeword v, it simply counts the number of samples in this state whose value $VQ(x_t)$ is v, and then divides it by the total number of samples in this state.

The flooring is applied in the M-step: when the computed probability value falls below a pre-defined flooring threshold, it is replaced by it. As a result, some codewords, which may not even appear in a state's training samples, will get non-zero probability, which makes the model generalize well to unseen testing data.

Experiments

Four iterations of further re-estimation are carried out for the baseline SQ-HDDHMM (i.e., SQ-a), with the results shown in Table 4.10. The flooring threshold is set to be 10^{-12}, which was found to be good for the baseline model (see Section 4.2.2).

In the Table, iteration 0 represents the baseline SQ-HDDHMM, which is converted directly from CDHMM without any re-estimation. As can be seen, further re-estimation results in a serious performance degradation on the testing data. On the other hand, the performance on the training set is almost perfect with near-to-zero WER, which clearly shows an over-fitting of the models to the training set.

The overfitting problem is due to the fact that the training data is too small to robustly train such a high density model. The data scarcity problem is further demonstrated by the following analysis.

Analysis of Data Scarcity

To show how serious the data sparsity is, for the above M-step, we compute the average number of samples (frames) used to estimate a single output probability value for each stream as follows:

$$\frac{\text{total number of train samples}}{\text{total number of output probability values}}$$

In our baseline 4-stream SQ-HDDHMM, we have:

- The total number of training samples (frames) is 4,851,811.
- The total number of output probability values in each stream is: 3132×16384, 3132×4096, 3132×4096 and 3132×512 (3132 tied states. each of the 4 streams in a state stores a discrete distribution table with size 16384, 4096, 4096 and 512 respectively).

Thus, on average, for each stream, we only have 0.1, 0.38, 0.38 and 3.03 samples to estimate a single output probability value, which is far from enough.

Finally, for a HDDHMM, we also compute the ratio of output probability values that get floored (i.e., probability value less than 10^{-12}) in each stream as:

$$\frac{\text{total number of output probability values that get floored}}{\text{total number of output probability values}}$$

The ratios for the models at iteration 0 and 1 in Table 4.10 are:

- Ratios for 4 streams in model at iteration 0: 9%, 0%, 0%, 7%.

- Ratios for 4 streams in model at iteration 1: 94%, 98%, 96%, 65%

As can be seen, the ratios of very small probability values (i.e., smaller than 10^{-12}) are much higher in the re-estimated model (iteration 1) compared with the one that converted directly from CDHMM (iteration 0), which may explain the WER difference between iteration 0 and iteration 1. The high ratio for the model in iteration 1 is obviously due to the lack of training data; while for iteration 0, the model conversion by numerical integration has a certian smoothing effect on the probability values. As a result, the number of very small probability values in iteration 0 is reduced compared with iteration 1.

4.6.2 Deleted Interpolation

The basic idea in deleted interpolation is to smooth a less well trained, but more detailed model with a robustly-trained but coarse one. Specifically for DHMM, the state output distribution in a triphone model (detailed one) could be interpolated with a corresponding one in a monophone model (coarse one) as follows:

$$b_j^{[DI]}(\cdot) = \lambda_j b_j^{[TRI]}(\cdot) + (1 - \lambda_j) b_j^{[MONO]}(\cdot)$$

where λ_j is the state dependent interpolation weight, $b_j^{[TRI]}(\cdot)$, $b_j^{[MONO]}(\cdot)$, and $b_j^{[DI]}(\cdot)$ are the output distributions for state j in a triphone model, the corresponding monophone model, and the interpolated model respectively.

To make the interpolated model generalize well to the unseen test set, the interpolation weight is estimated using cross-validation data with the E-M algorithm. We use cross-validation because oth triphone and monophone models are trained using the same data set under maximum likelihood criterion, with the former giving a higher likelihood. If the interpolation weights are also estimated on that data set, the weights for the triphone model are always one as it fits the data better.

To do deleted interpolation, the training set is first divided into I parts. Triphone and monophone systems are trained from each combination of $I - 1$ parts, with the deleted part serving as the unseen data to estimate the interpolation weights (hence called "deleted interpolation"). To get the final weight, the I sets of interpolation weights are averaged. The detailed E-M procedure to estimate the weight λ_j for state j is listed below:

- STEP 1: Initialize λ_j, and the maximum number of iterations.

- STEP 2: Update λ_j by:

$$\lambda_j' = \frac{1}{\sum_{i=1}^{I} N_j^i} \sum_{i=1}^{I} \sum_{k=1}^{N_j^i} \frac{\lambda_j b_j^{[TRI-i]}(VQ(x_j^{ik}))}{\lambda_j b_j^{[TRI-i]}(VQ(x_j^{ik})) + (1-\lambda_j) b_j^{[MONO-i]}(VQ(x_j^{ik}))},$$

  where $b_j^{[TRI-i]}(\cdot)$ is the output discrete distribution for tied state j in the triphone models, estimated by the entire training set except the deleted part (i.e., part i). Similarly, $b_j^{[MONO-i]}(\cdot)$ is the output distribution for the corresponding monophone state, estimated by the same $I-1$ parts of data. N_j^i indicates the number of training samples for state j in part i; $VQ(x_j^{ik})$ indicates the k-th sample for state j in part i.

- STEP 3: If the new value λ_j' is equal to the previous value λ_j, or the maximum number of iterations arrives, we stop. Otherwise, go to Step2.

Notice that, the above procedure assumes the system only contains one stream. For multiple stream systems, the deleted interpolation should be carried out for each stream independently, and each stream s in a state will have its own interpolation weight $\lambda_j^{(s)}$.

Experiments

The deleted interpolation technique is applied to smooth the baseline triphone SQ-HDDHMM (i.e., SQ-a with WER 6.71%) with the corresponding monophone SQ-HDDHMM. The triphone model has 3132 tied states, while the monophone model has 120 states. Both models consist of 4 streams, with codebook size 16384, 4096, 4096, and 512 respectively.

The experimental settings are as follows:

- $I = 2$: the SI-84 training set with 7138 utterances is divided into 2 equal-size parts, each part with 3596 utterances.

- $\lambda_j^{(s)} = 0.5$: the initial deleted interpolation weights, which are stream and state dependent, are all set to be 0.5. That is, the monophone and triphone models are initialized to be of the same importance.

Table 4.11: Performance of deleted interpolation for SQ-HDDHMM (SQ-a)

WER	Average Weight (Stream 1)	Average Weight (Stream 2)	Average Weight (Stream 3)	Average Weight (Stream 4)
6.87%	0.81	0.90	0.89	0.94

- The maximum number of iterations is set to be 30.
- The triphone SQ-HDDHMM with output distribution $b_j^{[TRI-i]}(\cdot)$, which should be built based on the entire training set except part i, is generated as follows: a triphone CDHMM is first trained using all except the ith part of the training data, and then converted by integration to SQ-HDDHMM.
- The monophone SQ-HDDHMM with output distribution $b_j^{[MONO-i]}(\cdot)$ is converted from the triphone SQ-HDDHMM counterpart. Specifically, the output distribution for a state in the monophone model is a weighted average of all the output distributions in the corresponding triphone states whose basephone is that monophone, where the weight for a triphone state is proportional to the number of training samples it contains.

The result by deleted interpolation after 30 iterations is shown in Table 4.11. For each stream, we also show the average interpolation weight over all the 3132 tied states. As could be seen from the table:

- As expected, the interpolation weights are biased toward triphone models.
- The smoothing by deleted interpolation does not show any improvement on top of the baseline model: the interpolated model gets a WER 6.87%, which is basically comparable with the baseline WER 6.71%.

4.7 Subspace-HDDHMM (SHDDHMM) for Robust Parameter Estimation

The number of parameters in HDDHMM is too many to be trained robustly, even applying classical robust parameter estimation methods does not alleviate the problem. In this section, a novel HDDHMM structure based on subspace modeling and SVQ

codebooks is proposed, which reduces the number of free parameters by tens or even hundreds. As result, the model parameters could be estimated robustly with a limited amount of training data. The resulting model is called "subspace HDDHMM" (SHDDHMM).

4.7.1 One-stream SHDDHMM with Constrained Bases

Similar to the previous HDDHMM, SHDDHMM also uses multiple streams. For simplicity, we only describe one-stream SHDDHMM in this and the next section. The extension to multiple streams are shown in Section 4.7.4. The overview structure of a one-stream SHDDHMM is shown in Figure 4.4. Notice that SVQ codebooks are utilized in SHDDHMM.

Basic Structure

The basic structure of a one-stream SHDDHMM could be simply written as:

$$b_j(VQ(x_t)) = \sum_{m=1}^{M} c_{jm} B_m(VQ(x_t)) \tag{4.3}$$

where j is an HMM state; b_j is the output discrete distribution table of state j with $b_j(\cdot)$ being the discrete distribution function; $B_1, \cdots, B_M$ are a pool of M discrete distribution tables shared globally by all the states; $c_{j1}, \cdots, c_{jM}$ are the state-dependent weights, satisfying the constraint that $\sum_{m=1}^{M} c_{jm} = 1$. By viewing a discrete distribution table as a vector, the vector $b_j \in \Re^N$ thus lies in the subspace spanned by M bases $B_1, \cdots, B_M$ [3]. It is called "subspace" since the number of bases M in the model is usually significantly smaller than the dimension N of vector b_j, which is the discrete state output distribution table size or codebook size (one recent related work is called *subspace Gaussian mixture model* (SGMM) [64, 65], which uses subspace techniques to derive the means of the state in CDHMM).

Constrained Bases

Since the codebook size N (i.e., the length of each basis vector) is usual very large (tens of thousands), even if we use the subspace representation, there are still too many

[3] Strictly speaking, due to the constraint on c_{jm}'s, b_j only lies in part of the subspace spanned by bases $B_1, \cdots, B_M$.

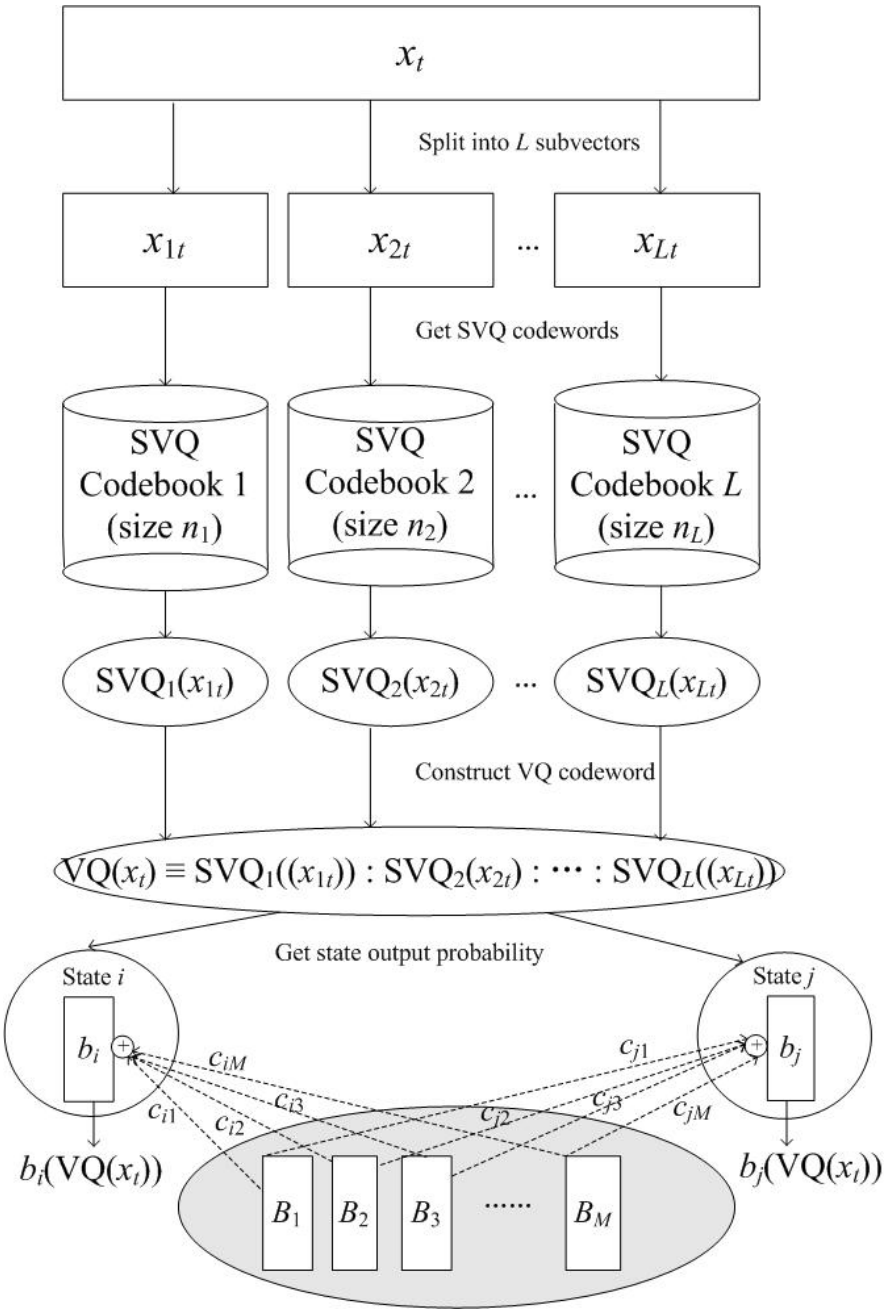


Figure 4.4: One-stream SHDDHMM architecture overview. The shaded ellipse represents the global pool of bases, spanning the subspace. Each state output discrete distribution table (i.e., b_i, and b_j), lies in the subspace. The state-dependent weights (the dotted lines) and the global pool of bases (the shaded ellipse) are temporary parameters, which only exist during model training; the final model only stores b_i and b_j.

parameters in the system. Thus, similar to [78], we make the assumption that the SVQ codewords of different subvectors are conditionally independent given the basis. This assumption imposes a constraint on each basis as:

$$B_m(VQ(x_t)) = \prod_{i=1}^{L} B_{im}(SVQ_i(x_{it})) \tag{4.4}$$

where $B_{1m} \in \Re^{n_1}, \cdots, B_{Lm} \in \Re^{n_L}$ are the SVQ discrete distribution tables in split 1 to L respectively, which are used to construct the original basis $B_m \in \Re^N$. Thus, the "real" free parameters in SHDDHMM with constrained bases are c_{jm}'s and B_{im}'s, which have a significantly fewer number of parameters compared with the original system parameters b_j's.

Parameters in SHDDHMM

In SHDDHMM, the parameters for training are different from that for decoding. The weight parameters c_{jm}'s and the basis parameters B_{im}'s only exist during the training stage, which make training with a limited amount of data feasible. After training, we use these parameters to compute the state output discrete distribution b_j's by Equation (4.3) and (4.4). Finally, we discard the weights and basis parameters, and only store the state output distributions. Thus, the final SHDDHMM for decoding has the same architecture of a conventional DHMM, but just with a much larger codebook. In other words, the attractive properties of DHMM are naturally inherited by SHDDHMM (e.g., it takes only $O(1)$ table lookup to get the state output probability).

Let us use an example to illustrate the reduction of system parameters during training. Let $I = 3132$ be the number of states in our tied-state system, $M = 1024$ be the number of bases, $L = 2$ so that a feature vector is split into 2 subvectors, and the SVQ codebook sizes for them are $n_1 = 256 = n_2 = 256$ respectively. Thus, the equivalent full-space codebook size, $N \equiv n1 \times n2 = 65536$. In the original system, there are I states, each with a discrete table of size N, giving a total of $NI = 205,258,752$ parameters. On the other hand, in the proposed model, there will be only $MI = 3,207,168$ parameters for the weights c_{jm}'s, and $M(n_1 + n_2) = 524,288$ parameters for the constrained bases B_{im}'s, resulting in a total of 3,731,456 model parameters, which is 55-fold reduction compared with the original system.

4.7.2 Training of One-stream SHDDHMM

Instead of flat-start training, a 1-stream SHDDHMM with M bases is first initialized from SCHMM, and is then reestimated. The basic steps of training SHDDHMM are listed:

STEP 1: A 1-stream SCHMM with a global pool of M diagonal-covariance Gaussians is first trained. In this SCHMM, the output distribution for state j takes the form: $b_j(x_t) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(x_t; \mu_m, \sigma_m^2)$, where c_{jm}'s are the state dependent mixture weights, and $\mathcal{N}(x_t; \mu_1, \sigma_1^2), \ldots, \mathcal{N}(x_t; \mu_M, \sigma_M^2)$ are the M diagonal-covariance Gaussians in the global pool.

STEP 2: Initialization of SHDDHMM: The mixture weights c_{jm}'s of SHDDHMM are copied from SCHMM. The SVQ discrete distribution tables B_{im}'s of SHDDHMM are initialized using the statistics from the training data based on SCHMM as:

$$B_{im}(l) = \frac{\sum_{t:SVQ_i(x_{it})=l} \gamma_t(m)}{\sum_{t=1}^{T} \gamma_t(m)} \tag{4.5}$$

where l is an SVQ codeword, $B_{im}(l)$ is the probability value from the entry indexed by l of SVQ discrete table B_{im}, $\gamma_t(m)$ is the posterior probability of observation x_t belonging to mixture m given the SCHMM and the training observation sequence $x_1, \ldots, x_T$.

STEP 3: Retraining of SHDDHMM: The updating of SVQ discrete distribution tables in SHDDHMM uses exactly the same Equation (4.5), except that the posterior probability $\gamma_t(m)$ is based on SHDDHMM of the previous iteration, rather than SCHMM. SHDDHMM weights c_{jm}'s are also updated.

STEP 4: Generation of final SHDDHMM: After the reestimation is complete, the state output discrete distribution b_j's are computed using the weights c_{jm}'s and SVQ discrete tables B_{im}'s by Equation (4.3) and (4.4). The final SHDDHMM only stores the state output discrete distributions, and discards the weights and SVQ discrete tables.

4.7.3 Design Logic of SHDDHMM

In this section, we propose and answer a set of questions to explain why the SHDDHMM is designed like this.

- **Why use bases with the constraints that: the SVQ codewords of different subvectors are independent?**

  To reduce the number of free parameters for robust parameter estimation. Without this constraint, the number of free parameters to represent a basis will be $\prod_{i=1}^{L} n_i$, while those with constrains only need $\sum_{i=1}^{L} n_i$ parameters (n_i be the size of SVQ codebook in subvector i as shown in Figure 4.4).

- **In each constrained-basis, SVQ codewords from different subvectors are assumed to be independent. However, they are actually correlated in reality. How to model the correlations?**

  The correlations among different subvectors are modeled by the mixture structure as shown in Equation (4.3). Usually, there are thousands of bases in SHDDHMM, a mixture of such a large number of bases can model the correlations quite well despite the fact that each basis does not model it.

- **Why use SVQ codebooks?**

  Linear codebooks are definitely not appropriate, as they are very slow to find the codeword for an input feature vector in large codebook systems like SHDDHMM. Both tree-structured codebooks and SVQ codebooks find codewords quickly. However, using SVQ codebooks could naturally assume the independence among different subvectors in a basis to reduce the number of free parameters for robust parameter estimation, while it is difficult for tree-structured codebooks to make such assumptions.

- **Why initialize the SHDDHMM parameters using data statistics based on SCHMM?**

  Firstly, the training problem of SHDDHMM is not a convex problem. That is, the model parameters will get trapped at some local maximum point during the training process. As a result, the choice of a good initial point in the model parameter space is very important. SCHMM has similar model structure as SHDDHMM, thus it could be used to initialize the SHDDHMM parameters. Furthermore, the data statistics based on a well-trained SCHMM are quite reliable, and initialization from it should be effective.

  Secondly, besides this initialization method, we also tried to borrow the model conversion technique in SVQ-HDDHMM (see Section 4.4) to initialize SHD-

DHMM . However, they do not work well in practice: they are either applied too slowly on large subvector dimensions (the numerical integration technique), or not accurate in providing a good initial point (the single-point approximation technique).

- **Is the model related to semi-continuous HMM (SCHMM)?**

  SHDDHMM shares some resemblance with the SCHMM. As a result, many techniques developed for SCHMM, such as adapting mixture weights for speaker adaptation [36], are also applicable to SHDDHMM. Despite the similarity, there are two key differences between the two models. Firstly, SHDDHMM is a discrete model, while SCHMM is a continuous model. Secondly, unlike SCHMM, the density pool and state-dependent weights of SHDDHMM only exist during the training stage. The final SHDDHMM for decoding only stores the state output discrete distributions, and discards the pool and mixture weights.

- **Is the model related to discrete mixture HMM (DMHMM)?**

  Discrete mixture HMM (DMHMM) [76] is a variant of DHMM, which replaces the one-dimensional Gaussian density in each mixture component of a CDHMM state by a discrete distribution, but otherwise keeps the mixture structure of CDHMM intact. As a result, in a DMHMM state, a discrete distribution table is stored for every dimension in each mixture. Later, people also generalized the "one-dimension" concept in DMHMM to "subvector" [22, 78].

  SHDDHMM is related to DMHMM. However, in the final model, for each state, DMHMM still keeps a mixture of discrete distribution tables, while SHDDHMM only stores a single discrete table. As a result, to get the state output probability during decoding, DMHMM needs to evaluate each discrete mixture and sum up their probabilities, while SHDDHMM only takes one table lookup in $O(1)$ time, irrespective of the number of mixtures. Furthermore, the "bases" in DMHMM are state dependent, while SHDDHMM uses a global pool of bases, which results in a lower number of free parameters.

4.7.4 Multiple-stream SHDDHMM

The extension of SHDDHMM from one stream to multiple streams is actually quite straight forward. For a multiple-stream SHDDHMM, Equation (4.3) and (4.4) for

Table 4.12: Performance of 4stream SCHMM with different pool size

# Gaussians Per Stream (4 streams)	Test WER	Dev WER
(256, 256, 256, 256)	5.85%	7.83%
(512, 512, 512, 512)	5.38%	7.20%
(1024, 1024, 1024, 1024)	**4.82%**	**6.86%**
(2048, 2048, 2048, 2048)	4.84%	7.09%

one stream should be replaced by the following two equations, with S being the total number of streams, and the superscript s denoting the sth stream.

$$b_j(VQ(x_t)) = \prod_{s=1}^{S} \sum_{m=1}^{M^{(s)}} c_{jm}^{(s)} B_m^{(s)}(VQ^{(s)}(x_t^{(s)})) \tag{4.6}$$

$$B_m^{(s)}(VQ^{(s)}(x_t^{(s)})) = \prod_{i=1}^{L^{(s)}} B_{im}^{(s)}(SVQ_i^{(s)}(x_{it}^{(s)})) \tag{4.7}$$

Figure 4.5 shows an example of a 2-stream SHDDHMM, with each stream storing a pool of bases independently. The training procedure for multiple-stream SHDDHMM is also similar to the one stream case, which now uses a multiple-stream SCHMM to initialize the model.

4.7.5 Experiments

Experiments are carried out to evaluate 4stream-SHDDHMM. The common settings are shown below:

- A 4-stream system using SVQ codebooks. The codebook size in each stream is 65536, 65536, 65536 and 32768 (i.e., the total number bits for each stream is 16, 16, 16, and 15).
- 4-stream SCHMM of different Gaussian pool sizes are constructed as shown in Table 4.12. The optimal one (as shown in boldface), has a WER 4.82% with 1024 diagonal-covariance Gaussians per pool. This result was obtained using the top 128 Gaussians during decoding. To compute the state likelihood of a feature vector x_t using only the top few Gaussians rather than all of them is

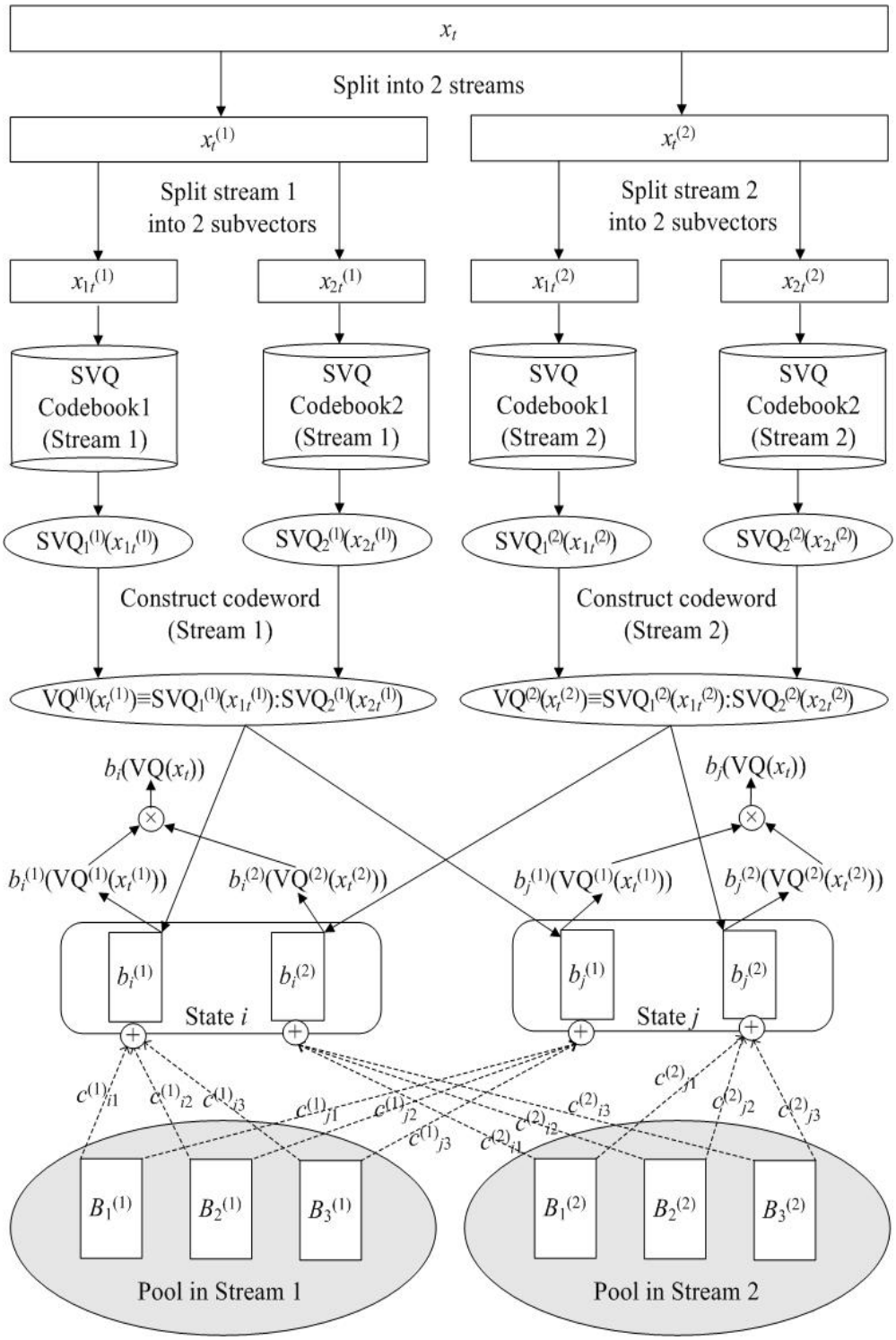


Figure 4.5: An example of a 2-stream SHDDHMM, each stream is further split into 2 subvectors. A global pool of bases is stored for each stream independently, and the number of bases in a pool is 3.

Table 4.13: Different subvector-partition and bit-allocation schemes for SHDDHMM

Scheme	Stream 1	Stream 2	Stream 3	Stream 4
S-a	$[1(2)]^4[1(1)]^8$	$[1(2)]^4[1(1)]^8$	$[1(2)]^4[1(1)]^8$	$[1(5)]^3$
S-b	$3(6)3(4)[3(3)]^2$	$3(6)3(4)[3(3)]^2$	$3(6)3(4)[3(3)]^2$	$[1(5)]^3$
S-c	$6(9)6(7)$	$6(9)6(7)$	$6(9)6(7)$	$[1(5)]^3$

a common practice for SCHMM, which could significantly speed up decoding while maintaining the recognition accuracy.

- SHDDHMM is initialized from the optimal SCHMM, thus the pool size per stream is also 1024.

- Three subvector-partition and bit-allocation schemes used in the experiment are depicted in Table 4.13. Each scheme describes a different subvector partition and its corresponding bit allocation gives the best WER of that partition. In general, the subvector dimension increases from scheme S-a to S-c. The detailed strategies for subvector partition and bit allocations are described below:

  – The subvector partition policy is similar to SVQ-HDDHMM in Section 4.4.4. Basically, all the subvectors for the first 3 streams have the same dimensions, while the last stream puts only 1 dimension into each subvector. Since the initialization and training are efficient for SHDDHMM, a subvector dimension as large as 6 (scheme S-c) could be used. This is unlike SVQ-HDDHMM with numerical integration, where at most 3 dimensions per subvector are allowed in practice.

  – For a specific subvector partition, a set of promising bit allocations are tried out, with the heuristics to put more bits into the MFCCs of lower indices. This utilizes the fact that MFCCs of lower indices are more important for recognition tasks. The bit allocation shown in the table for each scheme is the one that gives the best WER under that subvector partition.

Experiment Results

Table 4.14 lists the SHDDHMM results with the three different subvector-partition and bit-allocation schemes. For reestimation, the column labeled "no" gives results of the model that is initialized from an SCHMM without any reestimation, whereas the

Table 4.14: SHDDHMM performance with different schemes

Scheme	WER (Reestimation?)	
	no	yes
S-a	6.82%	6.11%
S-b	5.62%	5.32%
S-c	5.53%	**4.91%**

column labeled "yes" gives the results of the final reestimated SHDDHMM. From the table, we observe that:

- From scheme S-a to S-c, the WER keeps dropping. Consistent with the SVQ-HDDHMM result (see Section 4.4.4), the performance is usually better when the subvector dimension is larger, as a larger dimension results in smaller quantization error in full space. Despite that, we cannot use subvectors of too large dimensions, as the number of free parameters and the time needed to find the codeword also increase with the subvector dimensions. Scheme S-c with 6 dimensions per subvector is the maximum subvector dimension we tried.

- Initialization of SHDDHMM from SCHMM is effective (i.e., even without reestimation, the model is reasonably good), which provides a good starting point for re-estimation.

- For each scheme, model re-estimation always reduces the WER significantly.

- The best WER 4.91% of SHDDHMM (S-c in Table 4.14) is similar to that of its SCHMM counterpart 4.82% (in Table 4.12). Since SCHMM could be viewed as a continuous version of SHDDHMM (as they both have a global pool of similarly structured bases), it shows that the quantization error inherent in a discrete model is largely alleviated in SHDDHMM.

- Compared with the best SVQ-HDDHMM of the same codebook size (SVQ-b with WER 5.64% by numerical integration in Table 4.7), SHDDHMM S-c could reduce the WER by 0.73% absolute.

- Finally, the performance of both SCHMM and SHDDHMM is slightly worse than the baseline 1-stream CDHMM with WER 4.46% (see Table 4.3). The small performance degradation is probably due to the incorrect stream independence assumption underlying 4 stream SCHMM and SHDDHMM (we try to

alleviate the problem in the next section). However, these degradations are not statistically significant (according to the NIST matched pair sentence segment word error test [62, 30] with confidence level 0.95).

Comparison of Model Parameters

To show the effectiveness of parameter reduction by SHDDHMM, the number of free parameters [4] in each of the following models is calculated:

- Baseline 4s-DHMM: $3132 \times (512 + 512 + 512 + 256) = 5,612,544$ (3132 tied states, each state with 4 streams; the discrete distribution table size in each stream are 512, 512, 512 and 256 respectively.).
- Best SVQ-HDDHMM (SVQ-b in Table 4.7): $3132 \times (65536 + 65536 + 65536 + 32768) = 718,405,632$ (3132 tied state, each state with 4 streams, the discrete distribution table sizes in each stream are 65536, 65536, 65536 and 32768 respectively.)
- Best SHDDHMM (S-c in Table 4.14): $3132 \times 4 \times 1024 + 1024 \times ((512 + 128) \times 3 + 32 \times 3) = 14,893,056$, where $3132 \times 4 \times 1024$ is the number of parameters for weights $c_{jm}^{(s)}$ in Equation 4.6 (3132 tied states, each state with 4 streams, each stream in a state is associated with 1024 weights), and $1024 \times ((512 + 128) \times 3 + 32 \times 3)$ is the number of parameters for the constrained bases $B_{im}^{(s)}$ in Equation 4.7 (each stream has a pool with 1024 bases. For the first 3 streams, each basis is composed of two SVQ discrete distribution tables with size 512 and 128 respectively; For the 4th stream, each basis is composed of 3 SQ discrete distribution tables of size 32 for each).

Some observations from the above calculations are:

- There are about 128 times more parameters in SVQ-HDDHMM compared with baseline 4s-DHMM, which makes it hard to train robustly.
- SHDDHMM reduces the number of free parameters in SVQ-HDDHMM by 48 fold, which is now only 2.6 times more compared with 4s-DHMM, thus robust parameter estimation becomes feasible.

[4] The number of parameters used for transition probabilities in HMM is not counted, as it is the same for each model.

It needs to be pointed out that the above calculation is just to get an intuitive feeling about whether a model could be trained robustly. As the parameter roles are different in each model, using the computed number to directly guide the model construction is not recommended. For example, building a SHDDHMM whose number of free parameters is set to be the same as that in 4s-DHMM may not guarantee the optimal model performance.

Properties of Bases and Weights

Two interesting properties are found when analyzing the trained SHDDHMM:

- For each stream s, the learned bases in the pool, i.e., $B_1^{(s)}, B_2^{(s)}, \ldots, B_{M^{(s)}}^{(s)}$, are found to be linearly independent.

- The weights $c_{jm}^{(s)}$ are sparse. Specifically for our experiment, each stream s in state j is associated with 1024 weights $c_{jm}^{(s)}$, $m = 1, 2, \ldots, 1024$, with the summation of weight values to be 1. Many of the weights were found to be zero in the learned model. We computed the ratio of zero weights for each stream using the formula below:

  $$\frac{\text{number of zero weights}}{\text{total number of weights}}$$

  .

  The resulting ratios for the 4 streams are: 86%, 76%, 73%, 65%, In other words, on average, for each stream of a state, there will be $1024 \times 86\%, 1024 \times 76\%, 1024 \times 73\%, 1024 \times 65\%$ weights that are zero. However, this does not mean that the 1024 bases in the global pool is redundant, as the zero weight indices differ from one state to another.

4.8 Stream Weight Estimation by Iterative Linear Programming (ILP)

As can be seen in the last section, there is still a small loss of WER 0.45% in 4-stream SHDDHMM, comparing with 1-stream CDHMM, which in belief is caused by the incorrect stream independence assumption inherent in multiple-stream systems.

One way to compensate for the loss is to estimate the stream weights in a multiple-stream system. Specifically, in a S-stream system, the state output distribution b_j could be written as:

$$b_j(VQ(x_t)) = \prod_{s=1}^{S} (b_j^{(s)}(VQ^{(s)}(x_t^{(s)})))^{\eta_j^{(s)}},$$

or equivalently in the log domain as:

$$\log b_j(VQ(x_t)) = \sum_{s=1}^{S} \eta_j^{(s)} \log(b_j^{(s)}(VQ^{(s)}(x_t^{(s)}))),$$

where $b_j^{(s)}(\cdot)$ is the output distribution in the sth stream of state j, and $\eta_j^{(s)}$ is the corresponding stream weight with the constraint that $\sum_{s=1}^{S} \eta_j^{(s)} = S$. The weight $\eta_j^{(s)}$ is a positive value, which is used to specify the importance of stream s in state j. In conventional DHMM or SCHMM, the stream weights are all set to be one (i.e., the importance of different streams is treated as equal). However, it was shown that setting different weights for different streams in different states may improve the model performance [52, 59, 33, 16].

In this section, we would like to estimate the stream weights for SHDDHMM based on the method in [52, 59], which *discriminatively* trains the weights by *iterative linear programming (ILP)* optimization. The reasons to choose this ILP method for our SHDDHMM are as follows:

- There are very few system parameters to tune.
- The stream weight estimation is formulated into standard linear programming (LP) problem, which is well studied with established solutions, and efficient LP solvers, such as Mosek [57], are freely available.
- It was originally proposed by our research group. Thus, a lot of experiences of implementing and using this method are available, which are important for applying it in the new SHDDHMM task.

The ILP method has been applied successfully to estimate the stream weights of a 4-stream monophone CDHMM system, with the resulting 4-stream model showing comparable performance to 1-stream CDHMM (See [52, 59] for details). However,

directly applying it to our more complex triphone SHDDHMM did not show any improvement, which is probably due to that our system and task are more difficult and complex. As a result, some modifications to the original ILP method (both in theory and implementation) are done in the thesis to make it work for SHDDHMM. Nevertheless, the basic idea of the modified ILP method is the same as that of the original.

In the next section, we introduce our modified ILP method. The key differences compared with the original version (i.e., the one in [52, 59]) are also listed.

4.8.1 Modified ILP Method for Stream Weight Estimation

Basic Problem

Generally speaking, given an observation sequence of an utterance, the method would like to maximize the recognition score difference between the truth hypothesis (word sequence) and other hypotheses. Specifically, let R be the number of training utterances, $X^r = \{VQ(x_1^r), \ldots, VQ(x_{T^r}^r)\}$ be the discrete acoustic feature sequence with length T^r for the rth utterance, W^r and W^{rq} be the true word sequence and the qth competing word sequence (word sequence different from W^r) of utterance r, M_{W^r} and $M_{W^{rq}}$ are the word sequence HMMs constructed for W^r and W^{rq} respectively (see Section 2.2 for details of word sequence HMM). We would like to estimate a set of stream weights $\eta_j^{(s)}$ for M_{W^r} and $M_{W^{rq}}$, such that:

$$\begin{aligned} &\forall r, \forall q, \\ &(\log P(X^r|M_{W^r}) + \log P(M_{W^r})) \\ &- (\log P(X^r|M_{W^{rq}}) + \log P(M_{W^{rq}})) \\ &\geq C \end{aligned} \tag{4.8}$$

Basically, the above inequations require finding a set of stream weights, which could make the recognition score of the correct word sequence W^r larger than that of any of its competing word sequences W^{rq} by a non-negative margin C. The notations in the inequations could be described below:

- $\log P(X^r|M_{W^r})$: acoustic model score of X^r given M_{W^r}.
- $\log P(X^r|M_{W^{rq}})$: acoustic model score of X^r given $M_{W^{rq}}$.
- $\log P(W^r)$: language model score for word sequence W^r.

- $\log P(W^{rq})$: language model score for word sequence W^{rq}.
- $\log P(X^r|M_{W^r}) + \log P(M_{W^r})$: recognition score [5] of W^r.
- $\log P(X^r|M_{W^{rq}}) + \log P(M_{W^{rq}})$: recognition score of W^{rq}.
- C: a non-negative margin, with the value pre-set by a user. Similar to other large-margin classifiers (e.g., support vector machine [79]), the margin is introduced to improve the generalization ability of the model to unseen testing data.

In practice, there usually does not exist such stream weight values that can satisfy all the above inequations in (4.8). To relax the requirement, a non-negative slack variable $\xi^r \geq 0$ is introduced for each utterance r, and the original inequations become:

$$\begin{aligned} &\forall r, \forall q, \\ &(\log P(X^r|M_{W^r}) + \log P(M_{W^r})) \\ &- (\log P(X^r|M_{W^{rq}}) + \log P(M_{W^{rq}})) \\ &+ \xi^r \geq C \end{aligned} \tag{4.9}$$

Obviously, for each utterance r, we want the slack variable ξ^r to be as small as possible (the perfect case will be $\xi^r = 0$; that is, the recognition score for true hypothesis is larger than any of the competing ones by at least C). As a result, the problem for stream weight estimation could be expressed mathematically as:

$$\min_{\eta_j^{(s)}} \sum_{r=1}^{R} \xi^r \tag{4.10}$$

subject to the following constraints:

$$\forall r, \quad \xi_r \geq 0; \tag{4.11}$$

$$\forall r, \forall q, \quad \log P(X^r|M_{W^r}) - \log P(X^r|M_{W^{rq}}) + \mathbf{LD}(W^r, W^{rq}) + \xi^r \geq C; \tag{4.12}$$

$$\forall j, \forall s, \quad \eta_j^{(s)} \geq 0; \tag{4.13}$$

$$\forall j, \quad \sum_{s=1}^{S} \eta_j^{(s)} = S; \tag{4.14}$$

[5] Notice that, in practice, the language model score is usually scaled by a *grammar factor* α (see Section 2.1.1), with recognition score written as: $\log P(X^r|M_{W^r}) + \alpha \log P(M_{W^r})$. Here, for simplicity of the discussion, we assume that $\alpha = 1$.

Notice that stream weights affect only the acoustic model scores and not the language model scores. Thus, in constraint (4.12), we simply use $\text{LD}(W^r, W^{rq})$ to denote the language model score difference $\log P(M_{W^r}) - \log P(M_{W^{rq}})$, which does not depend on $\eta_j^{(s)}$.

LP Formulation

Now we show how to formulate the above stream estimation problem into linear programming optimization.

For the rth utterance, let $H^r = \{h_1^r, \ldots, h_{(T^r)}^r\}$ be the best "hidden" state sequence [6] in the model M_{W^r} that explains X^r; that is, H^r is the most likely state sequence given X^r and word sequence HMM M_{W^r}: $H^r = \text{argmax}_H P(H|X^r, M_{W^r})$; similarly, let $H^{rq} = \{h_1^{rq}, \ldots, h_{(T^r)}^{rq}\}$ be the state sequence in model W^{rq} that best explains X^r. A common practice in ASR is to approximate the likelihood based on the HMM (e.g., $P(X^r|M_{W^r})$) by the likelihood based on the best state sequence of the model (e.g., $P(X^r|H^r)$). As a result, the constraint in (4.12) for an S-stream HDDHMM (e.g., 4-stream SHDDHMM) can be written as:

$$
\begin{aligned}
&\forall r, \forall q, \\
&\log P(X^r|H^r) - \log P(X^r|H^{rq}) + \text{LD}(W^r, W^{rq}) + \xi^r \geq C \\
&\Rightarrow \\
&\sum_{t=1}^{T^r} \log b_{h_t^r}(VQ(x_t^r)) - \sum_{t=1}^{T^r} \log b_{h_t^{rq}}(VQ(x_t^r)) + \text{LD}(W^r, W^{rq}) + \xi^r \geq C \\
&\Rightarrow \\
&\sum_{t=1}^{T^r} \sum_{s=1}^{S} [\eta_{h_t^r}^{(s)} \log b_{h_t^r}^{(s)}(VQ^{(s)}((x_t^r)^{(s)})) - \eta_{h_t^{rq}}^{(s)} \log b_{h_t^{rq}}^{(s)}(VQ^{(s)}((x_t^r)^{(s)}))] + \text{LD}(W^r, W^{rq}) + \xi^r \geq C
\end{aligned}
\tag{4.15}
$$

where $\eta_{h_t^r}^{(s)}$ and $\eta_{h_t^{rq}}^{(s)}$ are the weights of the sth stream in state h_t^r and h_t^{rq} respectively; $VQ^{(s)}((x_t^r)^{(s)})$ is the discrete feature for the sth stream in frame t of utterance r; $b_{h_t^r}^{(s)}(\cdot)$ and $b_{h_t^{rq}}^{(s)}(\cdot)$ and are the output distribution in the sth stream given by state h_t^r and h_t^{rq} respectively.

[6] Given an HMM and an observation sequence, the most likely "hidden" state sequence can be found by the Viterbi algorithm, which is also known as "forced alignment" in speech community.

Replacing the constraint in (4.12) with (4.15), the problem in (4.10) becomes a standard linear programming problem, where both the objective function and constraints are a linear combination of the parameters $\eta_j^{(s)}$ and ξ^r.

Iterative LP (ILP)

To form the LP constraint in Inequations (4.15), competing hypotheses W^{rq} (i.e., word sequence different from W^r) for each utterance r are required. Unfortunately, the number of all possible competing hypotheses is basically infinite. In practice, we only use the set of hypotheses, that are confusable with W^r for the model, as the competing hypotheses. This set of confusable hypotheses are generated using the current model (thus with the current stream weights) in a similar way to standard speech decoding, which is called "Nbest decoding" in ASR community [7]. Given the competing hypotheses generated by the model, LP optimization will then find a globally optimal solution in the *feasible region* formed by constraints in Equations (4.11, 4.13, 4.14, 4.15). However, once we change the stream weights according to the LP solution, there will be a new set of models, which may give a different feasible region as they will probably generate a different set of hypotheses. In other words, the new stream weights found by LP optimization are only optimal with respect to the feasible region created based on the current model (with current stream weights), and are not guaranteed to be good for the feasible region based on the new model which use the new weights.

Iterative LP (ILP) is proposed to solve the above problem. In ILP, the LP optimization is run iteratively. In each iteration, the amount of change for stream weights is constrained as follows:

$$\forall j, \forall s, \quad \Delta\eta_j^{(s)} \leq \Delta\eta_{max}$$

Usually, $\Delta\eta_{max}$ is set to be small (e.g., 0.05). This small value ensures that the models in neighboring iterations do not differ too much, such that the optimal weights found using the old model should also be applied to the updated model. However, it also indicates that one iteration of LP will not make too much improvement on model performance. So, in practice, several iterations are run.

[7] While standard decoding only outputs the most likelihood word sequence $\hat{W}$ for speech utterance X^r using formula $\hat{W} = \arg\max_W P(W)P(X^r|W)$, the top N most likelihood word sequences are generated to form the competing hypotheses in "Nbest decoding". Notice that, if the true word sequence W^r appears in the decoded Nbest list, it should be removed, as the competing hypothesis must be different from the true one by definition.

The ILP procedure could be described as follows:

- STEP 1: Set $\Delta\eta_{max}$; Set the maximum number of iterations $MaxItr$; Set the stream weights for the initial model (usually use the default value one).
- STEP 2: Use the current model to generate N best competing hypotheses for each utterance.
- STEP 3: Get new stream weight values using LP optimization described in Section 4.8.1. The change of stream weights is constrained to be within the range $[-\Delta\eta_{max}, \Delta\eta_{max}]$.
- STEP 4: Update the current model using the weights found in the previous step.
- STEP 5: If the WER of the new model on the development set is worse than the old model, or the $MaxItr$ number of iterations is reached, stop. Otherwise, go to STEP 2.

Difference Compared with the Original ILP Method

The ILP method described above is a simple modification of the original version proposed in [52, 59], which is found to work well in practice. The major differences compared with the original method are as follows:

- The modified version works on the utterance level, while the original version works at the state or word level. More specifically, in the modified version, it wants the recognition score (a combination of acoustic and language model scores) for the true utterance to be better than that of the competing utterances; while in the original ILP, it wants the acoustic model score of a true state/word to be better than that of the competing states/words. The modified ILP is more related to the final speech decoding, which chooses the utterance with maximum recognition score as the recognition output.
- A margin C (in Inequations (4.15)) is used in the modified version, which is usually set to be a large value; while the original version does not use margin (i.e., C is set to be 0). The use of large margin was shown to have better generalization performance in [51], where the ILP technique was applied to find the decoding parameters (e.g., grammar factor).

4.8.2 Experiments

The modified ILP method is used to tune the stream weights of our best SHDDHMM (i.e., SHDDHMM S-c in Table 4.14 with WER 4.91%, and all stream weights set to be one). Below are the experimental settings, which were found to be good in practice:

- $\Delta\eta_{max}$: 0.05
- $MaxItr$: 6
- Training data for ILP: WSJ SI-84 training set with 7138 utterances, which is the same set used to train the SI-84 acoustic models
- Number of competing hypotheses for each utterance: 50
- Margin C: 10000
- Language model: to generate the competing hypotheses in Nbest decoding requires a language model. Rather than using the standard WSJ SI-84 trigram language (which is used for speech decoding), a weaker bigram language model is used, which is built using only the transcriptions of the training set. The use of a weakened language model for HMM discriminative training was shown to improve the system's generalization ability [81], as it focuses more on the discrimination provided by the acoustic model by loosening the language model constraints.

The results for the 6 iteration ILP are shown in Figure 4.6, where iteration 0 represents the initial SHDDHMM with all stream weights to be one. It can be seen:

- On both the development and test sets, the WER first decreases and then increases, which indicates that the estimated stream weights overfit the training data after the 4th iteration.
- The best result of ILP is at the 4th iteration (stopped using WER on dev set), with WER 4.65%, which cuts the WER of original SHDDHMM by a 0.26% absolute, and makes it very comparable with the WER 4.46% of the baseline CDHMM.

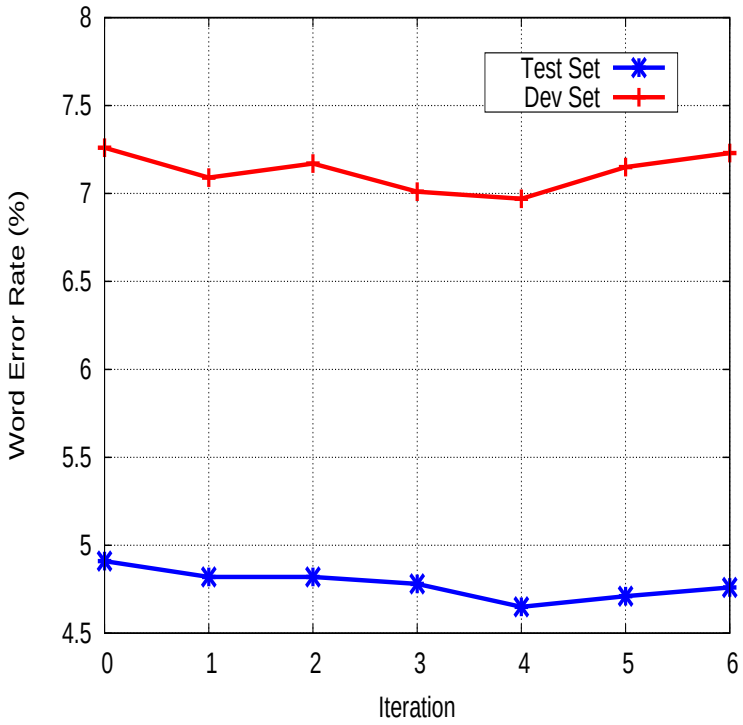


Figure 4.6: Stream weight estimation results for SHDDHMM with iterative linear programming

4.9 Summary on WSJ SI-84 Task

Up to now, we have investigated different techniques applied on HDDHMM for WSJ SI-84 task. In this section, we summarize the useful techniques, followed by a comparison of HDDHMM with other baseline systems.

4.9.1 Review of Various Techniques for HDDHMM

The techniques that were found useful for HDDHMM are summarized in Table 4.15, where the systems whose WER are not significantly different from that of baseline 1s-CDHMM (with WER 4.46%) are shown in boldface. The baseline system in this table represents the starting point of HDDHMM published in [53], and the relative WER reduction is the relative reduction compared with the baseline system with the following formula:

$$\frac{\text{baseline WER- current WER}}{\text{baseline WER}}$$

From the top to the bottom in Table 4.15, each system is built on top of the previous one, so it is hard to claim exactly which technique is more helpful than another. Basically, we could see that increasing the codebook size reduces the WER relatively by

Table 4.15: Effect of different techniques for HDDHMM on WSJ SI-84 Task

System	Technique	WER	WER Reduction (Relative)
SQ-HDDHMM (SQ-a)	baseline	6.71%	0%
SQ-HDDHMM (SQ-d)	increase codebook size	6.26%	7%
SVQ-HDDHMM (SVQ-b)	SVQ codebook with numerical integration	5.64%	16%
SHDDHMM (S-c)	parameter reduction by subspace modeling	4.91%	27%
SHDDHMM + tune stream weight	ILP stream weight estimation	4.65%	31%

7%, and changing from SQ to SVQ codebooks gives an additional 9% improvement, parameter reduction by subspace modeling on top of SVQ codebooks brings an additional 11% improvement, which makes the system comparable to baseline 1-stream CDHMM. The final stream weight tuning on top of SHDDHMM further cuts the WER by 4%. In terms of the WER reduction, the final best SHDDHMM cuts the WER by 2.06% absolute (31% relative) compared with the baseline system.

4.9.2 Recognition Time

Likelihood Computation Time Comparison between CDHMM and HDDHMM

One advantage of HDDHMM, compared with CDHMM, is the speed to get state output probability. We investigated the computation time for both CDHMM and HDDHMM, when doing the same number of likelihood evaluations. The computation time ratio between the two models was found to be 40:1; it indicates that 98% of GMM computation was cut by HDDHMM, as the latter only needs $O(1)$ table lookup time to get the likelihood. This ratio is larger than most other fast GMM computation techniques. For example, in [14], Chan implemented a set of fast GMM computation techniques and found that 80% cut-off was an empirical upper bound.

Operating Curves of Different Models

The WERs for different models reported until now are the asymptotic WERs by setting a very large beam width (see Section 2.2.2) during speech decoding. In other words, these WERs show the best recognition performance of each model, regardless of how much time it spends on decoding. However, in practice, fast recognition time is always desirable.

In Figure 4.7, different models are compared considering both recognition error and speed. Different points in an operating curve of a model were obtained by changing the beam width during speech decoding. The models in the figure, together with their asymptotic WERs, are shown below. Notice that, for fair comparison, the stream weights of all the 4-stream models, including SHDDHMM, are not tuned.

- Baseline 1-stream CDHMM with WER 4.46% (see Table 4.3)
- Baseline 4-stream CDHMM with WER 5.25% (see Table 4.3)
- Baseline 4-stream DHMM with WER 7.90% (see Table 4.3)
- Baseline 4-stream SCHMM with WER 4.82% (see Table 4.12)
- 4-stream SHDDHMM S-c with WER 4.91% (see Table 4.14)

The results in Figure 4.7 show that the 4-stream SHDDHMM performs better than the 4-stream SCHMM, the 4-stream CDHMM, and the conventional DHMM. Even compared with 1-stream CDHMM with the same WER, SHDDHMM cuts the decoding time by 25% in most part of the operating curve. The 25% speedup is quite substantial because in our CDHMM system, the Gaussian computation takes at most 27%[8] of the total recognition time.

4.9.3 Bandwidth Requirement

In distributed speech recognition systems (see Chapter 1 for details), the acoustic features in the client terminal need to be transmitted to the server, where the speech decoding is carried out. We show that discrete features used by HDDHMM require fewer bits to encode compared with continuous features in CDHMM, thus saving on bandwidth.

The conventional acoustic vectors in CDHMM consist of 39 MFCC coefficients, each of which is represented by a 4-byte float. Thus, $4 \times 39 = 156$ bytes are used to encode a frame of MFCC coefficients. In contrast, the codeword in each stream of our 4-stream HDDHMM is represented by a 2-byte unsigned short integer. As a consequence, only $2 \times 4 = 8$ bytes are used to encode the discrete features in a frame,

[8] In our experiments, HDecode (in HTK) was used for decoding with block caching mechanism, which caches the state output likelihoods and saves Gaussian computation time. As a result, in our WSJ CDHMM system, when computing the time ratio between GMM computation and the total decoding for different beam widths, the maximum ratio is found to be 27%.

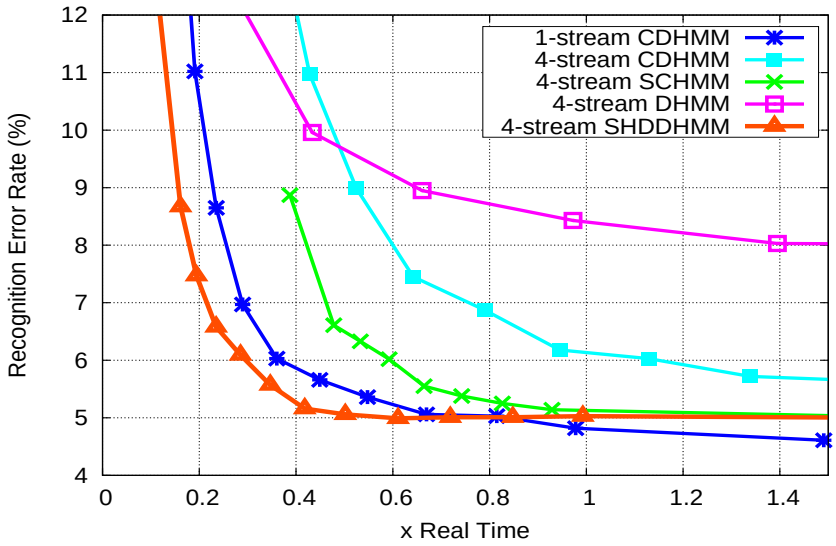


Figure 4.7: Operating characteristics of various SI-84 models (finding codeword time is included for discrete models.)

resulting in a 20-fold saving of bandwidth in distributed systems. Even for systems which only transfer the 13 static MFCCs and construct the full 39-dimensional MFCC vectors at the server side, the HDDHMM feature could still result in a 6.5-fold saving of bandwidth.

4.10 Comparison with Other Researches

This section compares HDDHMM with two other recent researches on DHMM.

4.10.1 Decision Tree Based Acoustic Model

The decision tree based acoustic model, proposed in [77, 1, 2, 3], is compared with our work. The reasons we choose this system for comparison are:

- It is a very recent research, with various advanced techniques applied on the system with conventional codebook size. Thus, it may represent the state-of-the-art performance of discrete system with conventional codebook size.
- It was evaluated on the same WSJ SI-84 task, which allows us to compare the results directly.

The decision tree in this system is used differently compared with that in conventional DHMM with a tree-based codebook. In a conventional DHMM, a single tree is

constructed, which is used only to quantize (or partition) the feature space. After the continuous features are quantized, an independent maximum likelihood parameter estimation step is carried out to estimate the state output distributions. In this decision tree based system, however, the feature space quantization and state output distribution estimation are integrated together during tree construction process. Besides partitioning the feature space, the decision tree is also used as a discriminative classifier to estimate the posterior probability of a state given the codeword, which is then converted into state output probability.

Specifically, in this system, rather than building only one tree as in conventional DHMM, a tree is built for each HMM state. To construct the tree for state j, all the training samples, each associated with a state label, are first pooled at the root node. A binary tree is then constructed by repeatedly splitting a node into 2 children until some stopping criterion is reached, where the splitting objective is to discriminative between the samples that belong to state j and all other samples. After the splitting is stopped, for each leaf node (i.e., codeword) v , the posterior probability $P(j|v)$ is computed by dividing the number of samples with state label j in leaf v by the total number of samples in the leaf. To incorporate into HMM framework, the posterior probability is further converted into the state output probability by:

$$\frac{P(j|v)}{P(j)},$$

where $P(j)$ is the prior probability of state j estimated using all the training samples. The procedure to first estimate posterior probability, and then convert it to state likelihood, was also adopted in hybrid artificial neural network (ANN)/HMM systems [11], which was found to improve the system's local discrimination compared with direct estimation of state output distribution in conventional HMM.

Several advanced techniques are further applied on the above system:

- Incorporation of high level information: for the questions asked in each non-leaf node, rather than only asking simple acoustic questions, e.g., which side of hyperplane a sample lies in, some higher level questions, such as gender and channel, are used. As a result, more information is utilized in this system, compared with a conventional DHMM.

- Forest models: in conventional DHMM, the acoustic space is hard partitioned, and the state output probability is piecewise constant with discontinuities at the

VQ boundaries. To smooth the output distribution, forest models are used in this system. For an HMM state, rather than building one decision tree, a forest (mixture) of trees are used, which makes use of the smoothing properties of ensemble methods. The state output probability in the forest model is a weighted average of the output probabilities from each tree.

The resulting system showed comparable and even better performance than GMM based system in a simple connected digit string recognition task, called Aurora 2 [34]. However, on the large vocabulary WSJ-SI84 task, where our HDDHMM has comparable WER with CDHMM, this system was still a 3.5% absolute worse in WER compared with 1-stream CDHMM [9]. Considering the fact that a respective amount of techniques have been applied on this conventional-codebook system, and yet the performance was still incomparable with that of CDHMM, it makes us believe that it is important to build DHMM with a very large codebook.

4.10.2 DHMM with Maximum Mutual Information (MMI) Tree Based Codebook

Besides the unsupervised linear codebook, MMI-tree based codebook is another commonly used codebook for DHMM (see Section 3.1.2 for details). In this section, we will compare HDDHMM to DHMM with MMI-tree based codebook. Specifically, we will compare with the recent work in [24], which is featured by:

- There is no restrictions on the hyperplane at each non-leaf node of the MMI-tree. The hyperplane in a non-leaf node partitions the samples of the node into 2 groups, each forms a child. In some old works [26], due to the limitations of computation power, the hyperplane is forced to be perpendicular to the axis of a feature dimension, and the hyperplane can be determined by a simple grid search.

- Their system also used very large codebook and showed better performance than maximum-likelihood estimated GMM-based system.

[9] One difference in experimental settings is that our system uses a trigram language model for both CDHMM and HDDHMM; while this system uses a bigram language model. However, we believe it should not affect too much the discussions here, as long as the same language model is used for the continuous and discrete model in comparison.

Unfortunately, their system was evaluated on a simple frame classification task, whose results cannot be compared directly to us. So we implemented their system, and evaluated it on the WSJ-SI84 task.

The implementation details are given in the next section.

Construction of DHMM with MMI-tree Based Codebook

The following procedure describes how to build a 1-stream DHMM with MMI-tree based codebook in [24]. To build a multiple-stream system, this procedure should be carried out independently for each stream.

- STEP 1: Get the tied-state label for each training sample x_t by Viterbi algorithm, using the baseline 1-stream CDHMM and the correct utterance transcription. This is known as *forced alignment* in ASR. Set the tree height L. (the codebook size is equal to 2^L, assuming the tree is a full binary tree.)

- STEP 2: Pool all the training samples at the root node. Each sample is associated with a state label. Set the current tree level to be 0.

- STEP 3: Partition each node in the current tree level by a hyperplane into 2 children. After all the nodes in the current level are partitioned, increase the tree level by 1. The hyperplane for a node partition is estimated to maximize the mutual information between its child nodes and the state labels. Since the hyperplane estimation problem does not have a closed form solution, the limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method [50] is used, which is a quasi-Newton gradient-based optimization routine with details in Appendix A.

- STEP 4: Stop if the tree level is equal to the desired tree height. Otherwise, go to STEP 3. Once the tree construction is complete, each leaf node represents a codeword and the number of leaves is equal to the codebook size.

- STEP 5: Quantize continuous feature into discrete one using the constructed tree as follows. Each continuous feature x_t is fed into the tree root, and traverses through the tree by checking which side of hyperplane it lies in at each node, until it reaches a leaf (i.e., codeword). The leaf node servers as the corresponding discrete feature of x_t.

- STEP 6: Estimate the state output distribution by a simple bin counting as in conventional DHMM, and apply the flooring technique to eliminate very small probability values.

"Smoothing by Adding 1" Technique

In the above system, only the simple flooring technique is applied for robust parameter estimation. It showed good performance in the simple frame classification task [24]. However, on the more complex WSJ SI-84 task, the flooring technique does not work well as shown in Section 4.6.1.

In our implementation of the above system, a robust parameter estimation technique, called "smoothing by adding 1", is applied, which is specifically designed for DHMM with tree-based codebook [5] [10].

Assume we have already constructed a MMI-tree as in Figure 4.8. Each non-leaf node in the tree is associated with a hyperplane. The following procedure describe the use of "smoothing by adding 1" to estimate the output distribution of state j. We use r to denote the root node, v to denote a leaf node, and q to denote a general node which could be any node in the tree.

- STEP 1: Pool all the continuous training samples of states j at the tree root. Denote the number of samples by C.
- STEP 2: Let each sample traverse through the tree by checking which side of the hyperplane it lies in at each node, until it arrives at a leaf node. Set a counter $C(q)$ for each node q, which records the number of samples passing through node q. Notice that the current counter $C(r)$ for root r is equal to C.
- STEP 3: Find all the nodes in the tree whose counter is 0 and its parent node's counter is larger than 0 (in Figure 4.8, we use *black* and *white* to distinguish the nodes of $C(q) > 0$ with those of $C(q) = 0$). There are three nodes in Figure 4.8 satisfying the above property, which are labeled by number 1, 2 and 3 respectively. Each of the three nodes is a root of a subtree, and every nodes in the subtree has a counter 0. For each of the three roots, do the followings:

[10] In [5], three smoothing methods for DHMM with tree-based codebook were developed: mixture smoothing, smoothing by adding 1, and Gaussian smoothing, which basically yield comparable results. "Smoothing by adding 1" is chosen in our implementation for its simplicity

- Add 1 pseudo sample to it, and set its counter $C(q) = 1$ (hence called "smoothing by adding 1"). To make the data statistics consistent, the counter for each of its ancestor nodes is increased by 1.
- Pass down the 1 sample evenly in the subtree, and set the counter of the nodes in the subtree accordingly. For example, in the 3rd subtree in Figure 4.8, the counter is 1 for the root (i.e., the node labeled by number 3), 0.5 for each of its 2 children, and 0.25 for each of its 4 grandchildren (the leaf nodes in this example). To generalze, assume K leaves are in a subtree, each will have a counter $C(q) = 1/K$.

- STEP 4: The output probability of a leaf node v (i.e., a codeword) from state j is given by

$$P(v|j) = \frac{C(v)}{C(r)}$$

  Notice that after STEP 3, the $C(r)$ value may be different from that in STEP 2 (in our example, $C(r)$ value is increased by 3).

- STEP 5: Despite the state output probability values estimated in STEP 4 are guaranteed to be positive, they can be very small. A flooring is applied to them as in conventional DHMM.

The key component in the above procedure is STEP 3, which makes sure each leaf has non-zero number of samples. Without STEP 3, the above procedure is the same as the parameter estimation in conventional DHMM.

Experiments

MMI-tree-codebook based DHMMs with different codebook sizes are constructed. The results are shown in Table 4.16. For the column with "smoothing by adding 1?", the value "N" means this smoothing technique is not applied (i.e., the system is implemented exactly based on [24] with only simple flooring applied), where the value "Y" means the smoothing technique is applied to the system. The WER in boldface for each type (i.e., "Y" or "N") represents the best result among systems with different codebook sizes under that type. Finally, the flooring values are determined based on the codebook size. When the codebook is not larger than "2048, 2048, 2048, 1024", $1e{-}6$ is used; otherwise, $1e{-}12$ is used.

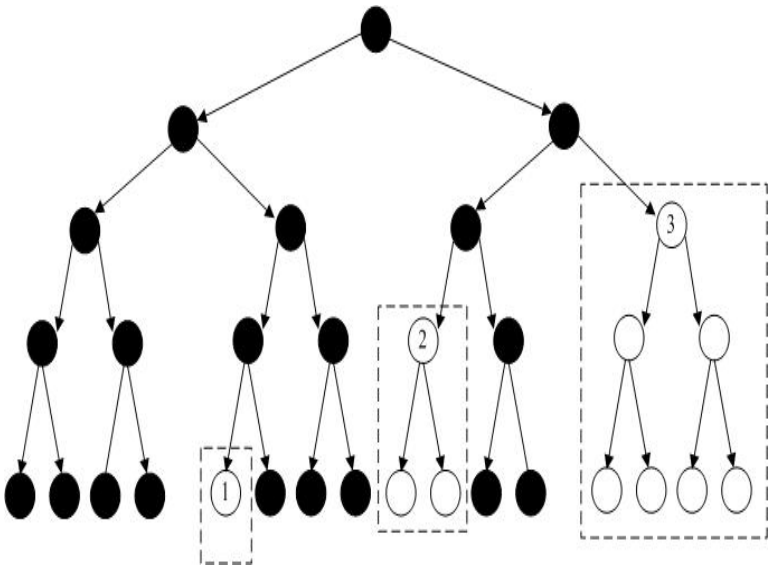


Figure 4.8: An example of "smoothing by adding 1" technique

Table 4.16: Performance of DHMM with MMI-tree codebook on WSJ SI-84 task

Smoothing By Adding 1?	Floor	Codebook (4 streams)	WER
N	1e-6	256,256,256,128	6.15%
N	1e-6	512,512,512,256	**6.15%**
N	1e-6	1024,1024,1024,512	7.15%
N	1e-6	2048,2048,2048,1024	7.53%
Y	1e-6	512,512,512,256	6.02%
Y	1e-6	1024,1024,1024,512	**5.74%**
Y	1e-6	2048,2048,2048,1024	6%
Y	1e-12	4096,4096,4096,2048	5.81%
Y	1e-12	8192,8192,8192,4096	5.96%
Y	1e-12	16384,16384,16384,8192	6.41%
Y	1e-12	32768,32768,32768,16384	6.56%
Y	1e-12	65536,65536,65536,32768	6.15%

As can be seen from the table:

- For the original systems without "smoothing by adding 1" technique, the best WER 6.15% is obtained at codebook size "512, 512, 512, 256". This result is better than the conventional linear codebook based DHMM, whose WER is 7.90% with the same codebook size (see Section 4.2.2). Further increasing the codebook size hurts the recognition performance due to the lack of training data, so we stop at the codebook size "2048, 2048, 2048, 1024". This finding is different from the original paper [24], where codebook size 16384$\sim$65536 was found to be the best. The possible reason is that they worked on a simple frame classification task with a lot of training data, such that the model could be trained robustly.

- For the systems with "smoothing by adding 1" technique, the best WER 5.74% is achieved at codebook size "1024, 1024, 1024, 512". This result is better than the WER 6.15% of system without "smoothing by adding 1" technique. Due to the smoothing effect, the recognition performance is still reasonably good even with a very large codebook "65536, 65536, 65536, 32768".

- None of the results are better than our best SHDDHMM with WER 4.91% (for fair comparison, the result of SHDDHMM without stream weight tuning is used).

4.11 Experiments on RM and WSJ SI-284 Tasks

In this section, we evaluate HDDHMM on two other ASR tasks: RM and WSJ SI-284 (see Section 4.1.1 for detailed experimental settings). For each task, the performances of the following models are compared:

- 1-stream CDHMM

- 4-stream SCHMM, which is used to initialize SHDDHMM

- 4-stream SHDDHMM

The basic architecture for each model, e.g., the number of triphones and tied states, can be found in Table 4.1. The optimal model configurations are described below.

Table 4.17: Subvector-partition and bit-allocation schemes for SHDDHMM in RM and WSJ SI-284

Systems	Codebook Size	Bit-alloc (Stream 1)	Bit-alloc (Stream 2)	Bit-alloc (Stream 3)	Bit-alloc (Stream 4)
RM	262144, 262144, 262144, 32768	$6(10)6(8)$	$6(10)6(8)$	$6(10)6(8)$	$[1(5)]^3$
WSJ SI-284	65536, 65536, 65536, 32768	$6(9)6(7)$	$6(9)6(7)$	$6(9)6(7)$	$[1(5)]^3$

4.11.1 Optimal Model Configurations

RM

Compared with WSJ, RM corpus is relatively small and the models are simpler. Specifically, the CDHMM has 6 mixtures per state; the 4-stream SCHMM has 512 Gaussians per pool. The codebook size and subvector-partition bit-allocation scheme of the 4-stream SHDDHMM are shown in Table 4.17. As can be seen in the table, RM use a larger codebook than WSJ due to that the RM systems have fewer tied states.

WSJ SI-284

Compared with WSJ SI-84, the models in WSJ SI-284 are built using a 5 times larger corpus. The CDHMM in WSJ SI-284 has 32 mixtures per state. The 4-stream SCHMM has 4096 Gaussians per pool. Since the experiments on the *large* WSJ SI-284 corpus are costly, the SHDDHMM codebook size and subvector-partition bit-allocation scheme are borrowed directly from that of the optimal WSJ SI-84 SHDDHMM, as shown in Table 4.17.

4.11.2 Results

The performances of different models on both RM and WSJ SI-284 tasks are shown in Table 4.18 (WSJ SI-84 results are also included as a reference). The RM WER is an average over the four test sets: feb89, oct89, feb91 and sep92. The stream weights of all the 4-stream models are not tuned. For each task, the models whose WERs are significantly different (either better or worse) from SHDDHMM are shown in boldface.

As can be seen from the table:

- For the three tasks, the 4-stream SHDDHMM is comparable to its 4-stream

Table 4.18: Model performance (in WER) on different ASR tasks

Task	1-stream CDHMM	4-stream SCHMM	4-stream SHDDHMM
RM	3.84%	4.02%	4.05%
WSJ SI-284	**3.47%**	4.43%	4.15%
WSJ SI-84	4.46%	4.82%	4.91%

SCHMM counterpart. Since SCHMM could be viewed as a continuous version of SHDDHMM, it shows that the discrete SHDDHMM can perform as good as the continuous model given that they have the same number of streams.

- On both RM and WSJ SI-84 task, the WERs of SHDDHMM are comparable to that of 1-stream CDHMM.

- On the WSJ SI-284 task, the SHDDHMM leads to a significant yet small (0.68% absolute) degradation compared with 1-stream CDHMM. Since the codebook size and subvector-partition bit-allocation scheme of SHDDHMM in WSJ SI-284 are directly copied from the WSJ SI-84 SHDDHMM, they are not optimal for the SI-284 task. Further increasing the codebook size and systematically finding the optimal scheme need to be done for WSJ SI-284 task in the future.

4.12 Summary

This chapter extensively investigates various issues of very large codebook discrete systems in ASR. The best HDDHMM (i.e., SHDDHMM) is based on subspace modeling technique, together with SVQ codebooks. Stream weight tuning by the modified ILP method can further improve the model performance.

We show that the SHDDHMM is significantly better than the conventional DHMM, and is comparable to SCHMM. Since SCHMM is a continuous version of SHDDHMM, it indicates that the quantization error inherent in a discrete model is largely alleviated in SHDDHMM.

The SHDDHMM has computational and bandwidth advantages over 1-stream CDHMM. Furthermore, the recognition accuracy of the SHDDHMM is very close to CDHMM. On RM and WSJ SI-84 task, they are comparable. On WSJ SI-284 task, the WER of a not-well-tuned SHDDHMM is only 0.68% absolute worse than that of CDHMM.

Finally, compared with other recent proposed discrete systems, our HDDHMM shows better recognition performance.

CHAPTER 5

VERY LARGE CODEBOOK DISCRETE MODEL FOR SPEAKER VERIFICATION (SV)

This chapter investigates very large codebook systems for speaker verification (SV) tasks. Specifically, we want to use a high-density discrete distribution to replace the GMM in a SV system, which is built based on the Gaussian Mixture Model Universal Background Model (GMM-UBM). No new techniques are developed in this chapter; rather, the techniques learned from ASR are applied directly. There are a few reasons we want to try on SV task:

- One advantage of a discrete system is its fast decoding speed. However, this is not fully exhibited in an ASR task. For example, as seen in Section 4.9.2, despite almost all the GMM evaluation time in CDHMM is saved by HDDHMM, the total recognition time is only cut by 25%, as the GMM computation only takes 27% of the total recognition time in that experimental task. Unlike ASR, almost all the decoding time is spent for GMM evaluation in GMM-UBM based SV task. Thus, SV could serve as a perfect example to show the superior speed of very large codebook discrete systems.

- In practice, the memory cannot be infinitely large, so there is a limitation for the codebook size in very large codebook systems. In ASR systems, the model size scales linearly with both the number of tied states and the codebook size. The number of tied states is usually large (e.g., 3132 in our WSJ experiments) in state-of-the-art ASR systems, to get an overall reasonable model size, the codebook size has to be limited (in our 4-stream WSJ system, the maximum codebook sizes we try are 65536, 65536, 65536 and 32768 for each of the 4 streams). In a standard GMM-UBM based SV system, there are only 2 GMMs, one for the background model, and the other for the target speaker model, whose model complexity is equivalent to an ASR system with only 2 tied states. As a result, we could try to investigate an even larger codebook.

Figure 5.1: An example of computer access control using speaker verification technology

- It is novel. Very few works on SV, or in general, speaker recognition, used discrete systems. The few we are aware of (e.g., [55]), showed much worse performance compared with continuous models.

A brief introduction of speaker verification is given in Section 5.1. In Section 5.2, we summarize the related speedup research for GMM-UBM based SV system. The description of very large codebook discrete models and some experimental results are shown in Section 5.3 and 5.4. Finally, we conclude the chapter in Section 5.5.

5.1 Introduction to SV

This section introduces the background knowledge for speaker verification task. For a detailed review, the readers may refer to [44, 8].

5.1.1 The Problem

The SV task is to decide whether the unknown input speech is from a claimed speaker. It is commonly applied in person authentication systems, such as an identity check of a user to grant access to a computer system as shown in Figure 5.1.

Depending on whether the utterances of a user are fixed or not, SV could be divided into text dependent and text independent ones. In the text-dependent SV, for example, the user is promoted to read an utterance given by the system. In text-independent SV, however, the user could speaker anything as he likes, which is thus a more difficult task compared with text-dependent one. The SV task evaluated in the thesis is text-independent.

It is worth mentioning that speaker verification actually belongs to a more general task called *speaker recognition*, which by definition is to recognize people from their voices. Usually, speaker recognition refers to two different tasks: speaker verification and speaker identification, where the latter decides which person (from a set of known speakers) gave the input speech. Regardless of the fact that the tasks are different, many techniques developed for one task could be applied to another.

5.1.2 Feature

The common acoustic features used in SV are similar to that in ASR. For example, in our later experiment, the MFCC feature is used. To improve the performance, feature normalization techniques, e.g., *cepstral mean normalization*, or *feature warping*, are usually applied. In cepstral mean normalization, the mean value estimated over the entire utterance is subtracted from each feature, which effectively reduces the variations introduced by different channels. In feature warping, the cumulative distribution function of the feature is warped, so that it matches some reference distribution function, e.g., Gaussian, which is much easier to be modeled by a statistical system.

Like ASR, an utterance is denoted by X, which consists of a sequence of feature vectors: $X = x_1, x_2, \ldots, x_T$.

5.1.3 Gaussian Mixture Model Universal Background Model (GMM-UBM) for SV

Gaussian Mixture Model Universal Background Model (GMM-UBM) [69], is a classical method for an SV task. In GMM-UBM, a speaker is modeled by a single GMM (similar as a state in ASR). To decide whether an utterance X is from the target speaker, the posterior probability of the target speaker model M_{target} given X, denoted as $P(M_{\text{target}}|X)$, is evaluated. The computed probability value is then compared with a pre-defined threshold θ to make the decision. Since the speaker GMM could only give $P(X|M_{\text{target}})$, to compute the posterior probability, Bayes's rule needs to be applied:

$$P(M_{\text{target}}|X) = \frac{P(X|M_{\text{target}})P(M_{\text{target}})}{P(X)}$$

By ignoring the term $P(M_{\text{target}})$, which is a constant irrespective of X, and approximating $P(X)$ with $P(X|M_{\text{UBM}})$, where M_{UBM} is also a GMM trained using data

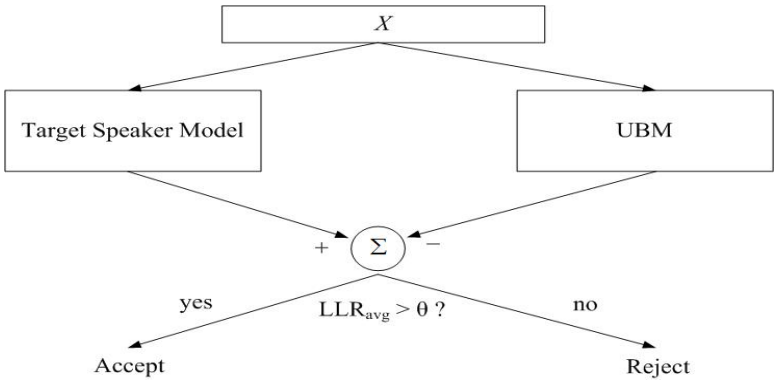


Figure 5.2: GMM-UBM based speaker verification system

from various speakers (called universal background model (UBM)), and also normalizing out the utterance duration effects by dividing the probability with the utterance length T, the posterior probability could be transformed into the following average *log likelihood ratio* (LLR):

$$\begin{aligned} \text{LLR}_{\text{avg}}(X, M_{\text{target}}, M_{\text{UBM}}) &= \frac{1}{T}(\log P(X|M_{\text{target}}) - \log P(X|M_{\text{UBM}})) \\ &= \frac{1}{T}\sum_{t=1}^{T}(\log P(x_t|M_{\text{target}}) - \log P(x_t|M_{\text{UBM}})) \end{aligned} \quad (5.1)$$

The use of a log ratio in the above equation could compensate for the mismatch between the training and test set. For example, if the speech in the test utterance is recorded using a microphone different from the one in the training set, a bias will be introduced to the term $\log P(X|M_{\text{target}})$. Fortunately, a similar bias will also appear in the term $\log P(X|M_{\text{UBM}})$, which is eliminated after the substraction.

Figure 5.2 shows an example of a GMM-UBM based SV system. An utterance X is fed into both the UBM and speaker model, and the average LLR is computed based on Equation 5.1. Depending on whether the ratio is larger than a predefined threshold θ, the utterance is accepted (X is from the target speaker), or rejected (X is from some impostor).

The two key components in Figure 5.2 are the target speaker model M_{target} and the UBM M_{UBM}, each is modeled by a GMM. Historically people also tried to use HMM instead of a single GMM. However, it did not seem to perform as well as GMM in text independent SV task [8].

Construction of UBM

The GMM in UBM represents a speaker-independent distribution of the feature vectors, which is trained by a data set generated from different speakers. As in ASR, diagonal-covariance Gaussians are used in GMM, with the model parameters estimated by the iterative E-M algorithm. Since only a single GMM is trained, the number of components is usually set to be large (e.g., $512 \sim 2048$).

Construction of Target Speaker Model

The GMM in a target speaker model represents the feature distribution generated by this speaker. Since the speaker dependent training data is usually very limited (e.g., 1 or 2 minutes only), this model cannot be trained directly as UBM. Usually, the speaker model is derived by adapting the parameters of a well-trained UBM using the speaker dependent training data, by *maximum a posteriori* (MAP) estimation [70].

The MAP method could be used to adapt all the parameters (i.e., mean, variance, and mixture weights) of UBM to a speaker model. In practice, as the speaker dependent data is small, adapting the mean only has found to work well [8, 44]. In the following we show how to construct the target speaker model by mean-only adaptation.

- STEP 1: Based on the well-trained UBM, run one iteration of E-M algorithm, using the train utterance X from the target speaker. Denote the means of the mth Gaussian component in UBM and the new estimated GMM as μ_m^0 and the μ_m^1.

- STEP 2: Let the target speaker GMM have the same number of components as UBM. For the mth component, the variance vector and weight are copied directly from the corresponding ones in UBM, and the mean μ_m is computed as a weight sum of μ_m^0 and the μ_m^1:

$$\mu_m = \alpha_m \mu_m^1 + (1 - \alpha_m)\mu_m^0 \tag{5.2}$$

  where α_m is the mixture dependent adaptation coefficient, which controls the balance between the old and new estimates, and is defined as:

$$\alpha_m = \frac{n_m}{n_m + r}, \tag{5.3}$$

where r is a fixed "relevance" parameter, n_m is the number of feature vectors from utterance X that belongs to the mth mixture, which could be obtained after the E-step in STEP 1.

The above parameter adaptation can be derived from the general MAP estimation equations for a GMM with constrains on the prior distribution [29]. Basically, the mean μ_m^0 from UBM represents the prior information, and μ_m^1 represents the information from the speaker data. From Equation (5.2) and (5.3), we can see that: when the number of samples n_m is large for the mth mixture, α_m is close to 1, the mean μ_m in the speaker model is then estimated mainly based on μ_m^1, which represents the speaker data. On the contrary, for the mixture with a small number of samples, the prior information μ_m^0 from UBM plays a more important role to estimate μ_m. The relevance parameter r specifies the importance of speaker data, with respect to prior information. The larger the r, the less important the data, which indicates more samples are needed in order for the estimation to be biased towards μ_m^1.

The speaker model trained by MAP is tightly coupled with UBM: they have the same number of mixture components; also the components have a one-one correspondence. That is, for each component in the speaker model, there is a corresponding one in the UBM which the speaker component is adapted from. This one-one correspondence allows for a fast computation technique of average LLR, which is described in the next paragraph.

Computation of Average LLR

To compute the average LLR in Equation (5.1), the likelihood of each feature x_t, given the UBM or speaker model, needs to be evaluated. Since the number of components in both models are large (e.g., 2048), the evaluation could be computationally expensive. In practice, a fast scoring procedure is applied. For each feature vector, the top C most likely Gaussians from the UBM are first found out, and the UBM likelihood is computed using only these C mixtures. And then, the speaker's likelihood for this feature vector is computed based on the corresponding C components in the target speaker model (thanks to the one-one component correspondence between UBM and speaker model). Assuming that the number of components is M, fast computation requires $M + C$ Gaussian evaluations for a feature vector, which is faster than evaluating all

the components in both UBM and speaker model with $2M$ Gaussian computations. Usually, $C = 5$ is set, which was found to be good in practice [44, 8].

GMM-UBM had once dominated the SV field. In recent years, new models , such as support vector machine (SVM) and joint factor analysis (JFA), are applied to SV, which showed better performance than GMM-UBM. However, these new methods also make use of the GMMs. For SVM [13], each sample represents a conversation of a speaker, which is a supervector formed by concatenating all the Gaussian means in a speaker GMM. In JFA [41], to remove the channel effect, the mean supervector in a speaker GMM is decomposed into two statistically independent components: a speaker supervector and a channel supervector, and only the speaker supervector is used to represent the speaker. Furthermore, the best SV system is usually obtained by a fusion of multiple models (e.g., GMM and SVM were combined in [42]). In any case, it is fair to say that GMM-UBM still serves as de-facto reference method for speaker verification task. This thesis only investigates the GMM-UBM based systems.

5.1.4 Evaluation Criterion

There are two types of errors in SV systems:

- False rejection error (miss): a genuine speaker gets rejected.
- False acceptance error (false alarm): an imposter gets accepted.

As could be seen in Figure 5.2, the probabilities of miss (miss rate) and false alarm (false alarm rate) depend on the threshold θ. When θ is large, the system tends to reject every test utterance, thus with a high miss rate and a low false alarm rate. On the contrary, if θ is small, most utterances are accepted, thus the miss rate is low but the false alarm rate is high.

One popular evaluation criterion is called *equal error rate* (EER), which gives the error rate at the decision threshold for which the false alarm rate and miss rate are equal. EER measures the ability of a system to separate impostors from a genuine speaker.

Another commonly used evaluation criterion, defined by the National Institute of Standards and Technology (NIST) [66], is called *Minimum detection cost function*

(MinDCF). A detection cost function is a function of threshold θ, which is defined as:

$$DCF(\theta) = 0.01 \times 10 \times P_{\text{miss}}(\theta) + 0.99 \times 1 \times P_{\text{fa}}(\theta)$$

where $P_{\text{miss}}(\theta)$ and $P_{\text{fa}}(\theta)$ are the miss and forced alarm rate given θ respectively, 0.01 and 0.99 are the assumed prior probabilities that the test utterance is from a genuine speaker or an imposter respectively, 10 and 1 a are the costs for a miss detection and a false alarm. The total effect of the DCF is to put more emphasis on the low forced alarm rate. MinDCF is the DCF value at the threshold of which $DCF(\theta)$ is smallest.

5.2 Related Speedup Methods for GMM-UBM Based Systems

One motivation to apply very large codebook systems in SV is to increase speed. In this section, we review the related computational speedup methods for GMM-UBM based systems.

The most commonly used speedup method is to compute the likelihood using only the top C (usually $C = 5$) Gaussians as described in Section 5.1.3, which could cut the computation time by almost half. This technique has been a standard practice in state-of-the-art GMM-UBM based systems.

To identify the top C Gaussians in the above fast scoring method, one still needs to evaluate all the Gaussian components in the UBM, which is still computationally expensive. In [4], a smaller GMM with much less number of components (e.g., 32 components) was created using the same training data as for UBM; For each component in this small GMM, a short list of Gaussian indices are stored, which represent the expected most likely components in the big UBM. A test feature vector is first scored against each component in the small GMM to figure out the one that gives the highest likelihood. The UBM and speaker model likelihoods for this feature vector are then computed by evaluating only the Gaussians whose indices are stored in this best-scoring component. Compared with the baseline UBM-GMM (with all the Gaussian components evaluated for both the UBM and speaker models), a 10-fold speed up was achieved, with only slight degradation in verification performance.

In [82], a tree-structured UBM was constructed by top-down hierarchically clus-

tering all the Gaussian components in the conventional UBM. Each non-leaf node in the tree represents a cluster of Gaussian components in conventional UBM, and each leaf corresponds to a single Gaussian component. In this way, the acoustic space is partitioned into multiple regions at different levels of resolution. For the target speaker model, the same tree structure as in the new UBM is used, with each layer adapted from the corresponding layer in UBM. Based on the tree structure, for each test feature vector, only a small set of Gaussians are scored, and the scores at different level are combined by a neural network for the final decision. Compared with the baseline GMM-UBM system (with top 5 Gaussian fast scoring), this new model was found to cut the computation time by 17 fold, with a slightly better EER.

In [56], McLaughlin $et.al$ identified two orthogonal factors in GMM computation: the acoustic resolution in terms of the number of Gaussian components in a GMM, and the temporal resolution in terms of various forms of decimation (or down-sampling). It has been shown that reduction of the number of Gaussian components by a factor of 4 and decimation factors as high as 20 will not affect the SV performance too much.

5.3 High-density Discrete Model (HDDM) for SV

In this section, we describe how to extend the very large codebook discrete models in ASR to SV. Specifically, we replace the GMMs in UBM and target speaker model by discrete distributions with a very large codebook. Following the terminology for ASR in the last chapter, this very large codebook discrete model is also called *high-density discrete model* (HDDM) [1].

For the relatively simple SV task, we did not try all the techniques developed in ASR. Rather, we only evaluated the scalar/subvector quantized HDDMs, which are directly converted from GMMs by integration. The choice is made mainly for the sake of simplicity: given the original well-trained GMM-UBM systems, only simple conversions are needed to generate HDDMs, without requiring the use of any training data.

The procedure to construct HDDMs is given below:

- STEP 1: Build 1-stream GMM-UBM systems. Decide the number of streams S for HDDMs.

[1] in ASR, since HMM is used, the model there is called *high-density discrete HMM* (HDDHMM).

- STEP 2: Convert each 1-stream GMM in GMM-UBM systems into an S-stream GMM. Specifically, assume the 1-stream GMM has M components, and the mean vector, variance vector, and weight for the mth component are denoted as μ_m, σ_m^2 and c_m respectively. Then, an S-stream GMM is converted from the 1-stream GMM, with each stream containing M components. The mean, variance and weight for the mth component in the sth stream of the multi-stream GMM, denoted as $\mu_m^{(s)}$, $(\sigma^2)^{(s)}$, and $c_m^{(s)}$, are set as below: $\mu_m^{(s)}$ and $(\sigma^2)^{(s)}$ are copied from the subvectors corresponding to the sth stream of μ_m and σ^2 respectively, and $c_m^{(s)}$ is copied directly from its one stream counterpart with $c_m^{(s)} = c_m$.

- STEP 3: Construct SQ/SVQ codebooks for each stream.

- STEP 4: Each S-stream GMM is converted to an S-stream SQ/SVQ-HDDM by integration, with the same procedure as in ASR task (see Section 4.2.1 and 4.4.1 for details of conversion).

5.4 Experiments

NIST text-independent speaker recognition evaluation (SRE) 2001 and 2002 are used in the experiments. Specifically, all of the 1006 male utterances (from > 120 speakers) in NIST 2001 are used to create the UBM, and the speaker models are created by MAP adaptation on the UBM for each of the 139 male speakers in NIST 2002. Each adaptation utterance is about 2 minutes long but half of the content is silence. Each of the 1442 male verification utterances in NIST 2002 is scored against 11 hypothesized speakers. This amounts to 1,232 speaker trials and 14,630 impostor attempts.

The features used are 12 MFCCs plus their first derivatives, leading to 24-dimensional acoustic vectors. Cepstral mean normalization is applied to the MFCCs, followed by feature warping. Speaker verification is based on the log-likelihood ratio between a speaker model and the UBM.

The computational environment is the same as for ASR task: a Linux machine with 3GHz CPU and 4GB RAM.

5.4.1 SV Performance of Baseline GMM-UBM Systems

1-stream GMMs with 64$\sim$1024 Gaussian components are trained and converted to 2-stream GMMs (the 2 streams are defined as: the first stream with 12 MFCCs, and the second stream with 12 ΔMFCCs); their SV performance is shown in Table 5.1. The GMM likelihoods are computed using either only the top-5 Gaussian components, or all Gaussian components.

The effect of loss of correlation in multi-stream modeling can readily be seen in the performance of 1-stream GMMs and their 2-stream counterparts: as expected, 1-stream GMMs have better EER performance than their 2-stream counterparts (10.66% vs. 11.68%). The use of the top-5 Gaussian components in GMM likelihood computation actually results in some performance degradation (10.66% vs. 11.43%); but since it speeds up the calculation by almost 50% and is the common practice, we continue to use it for all SV experiments using GMMs. On the other hand that the performance degradation due to the heuristic use of top-5 Gaussians in GMM likelihood computation is comparable to that due to the use of two independent feature streams (11.68% vs. 11.43%) can be good news for the proposed high-density discrete model (HDDM) because HDDM *always uses* all Gaussian components in a GMM during model conversion.

Table 5.1: EER performance of 1-stream or 2-stream GMMs of varied number of mixtures, M. (S = #streams; all/top5 = GMM likelihood computation using all/top 5 Gaussian components.)

M	EER(S=1, all)	EER(S=2, all)	EER(S=1, top5)
1024	11.11%	**11.68%**	11.84%
512	**10.66%**	12.08%	**11.43%**
256	11.45%	12.32%	11.88%
128	11.72%	12.50%	12.37%
64	12.52%	13.30%	13.20%

5.4.2 SQ-HDDM vs. SVQ-HDDM

For a given codebook size, the model size of an HDDM is the same regardless of the number of mixtures in the parent GMM. Thus, one is free to choose the best GMM to convert to HDDM. Based on the results shown in Table 5.1, the 2-stream GMM with 1024 components has the lowest EER, and thus is chosen and converted to both SQ-

Table 5.2: 2-stream HDDMs of varied resolutions. (w = subspace dimension. Probabilities are assumed 4-byte floating point numbers. The model size is in megabytes.)

Model	Codebook Size (2 streams)	Bit-alloc (Stream 1)	Bit-alloc (Stream 2)	Memory (MB)
SQ-HDDM-a	1048576, 4096	$[1(2)]^8[1(1)]^4$	$[1(1)]^{12}$	4
SQ-HDDM-b	1048576, 1048576	$[1(2)]^8[1(1)]^4$	$[1(2)]^8[1(1)]^4$	8
SQ-HDDM-c	2097152, 2097152	$[1(2)]^9[1(1)]^3$	$[1(2)]^9[1(1)]^3$	17
SQ-HDDM-d	4194304, 4194304	$[1(2)]^{10}[1(1)]^2$	$[1(2)]^{10}[1(1)]^2$	34
SQ-HDDM-e	8388608, 8388608	$[1(2)]^{11}1(1)$	$[1(2)]^{11}1(1)$	67
SQ-HDDM-f	16777216, 16777216	$[1(2)]^{12}$	$[1(2)]^{12}$	134
SVQ-HDDM-a	1048576, 4096	$[2(4)]^4[2(2)]^2$	$[2(2)]^6$	4
SVQ-HDDM-b	1048576, 1048576	$[2(4)]^4[2(2)]^2$	$[2(4)]^4[2(2)]^2$	8
SVQ-HDDM-c	2097152, 2097152	$[2(4)]^42(3)2(2)$	$[2(4)]^42(3)2(2)$	17
SVQ-HDDM-d	4194304, 4194304	$[2(4)]^52(2)$	$[2(4)]^52(2)$	34
SVQ-HDDM-e	8388608, 8388608	$[2(4)]^52(3)$	$[2(4)]^52(3)$	67
SVQ-HDDM-f	16777216, 16777216	$[2(4)]^6$	$[2(4)]^6$	134
SVQ-HDDM-g	16777216, 16777216	$[3(6)]^4$	$[3(6)]^4$	134

Table 5.3: SV performance of various SQ-HDDMs and SVQ-HDDMs converted from 2-stream GMMs with 1024 components

Model Index	SQ-HDDM		SVQ-HDDM	
	EER	minDCF	EER	minDCF
a	12.97%	0.0671	12.60%	0.0661
b	12.93%	0.0662	12.43%	0.0662
c	12.47%	0.0654	12.20%	0.0651
d	12.39%	0.0649	12.14%	0.0649
e	12.16%	0.0644	11.79%	0.0640
f	12.11%	0.0635	**11.60%**	**0.0631**
g	N/A	N/A	**11.48%**	**0.0618**

HDDMs and SVQ-HDDMs of varied configurations indexed from 'a' to 'g' as shown in Table 5.2. In the table, all the SQ models have subvector dimension 1, and all the SVQ models have subvector dimension 2, except SVQ-HDDM-g with subvector dimension 3. The codebook size used in a stream ranges from 1048576 to 16777216, which is much larger than that in ASR, e.g., the maximum codebook size is 65536 in WSJ task. The SV task could afford larger codebook as only two models (UBM and target speaker model) are loaded into the memory for each test.

Table 5.3 summarizes the SV performance of the various SQ-HDDMs and SVQ-HDDMs. The findings are basically consistent with that in ASR task:

- SV performance improves with increasing codebook size.
- For the same model size, an SVQ-HDDM performs better than an SQ-HDDM.
- SVQ-HDDMs with larger subvector dimension are preferred.
- Here we used at most 24 bits for each stream to limit the size of the generated HDDMs. Within the limit, none of the generated SQ-HDDMs perform as well as the parent 2-stream GMM from which they are converted. On the contrary, SVQ-HDDM-f and SVQ-HDDM-g (in boldface in the table) both perform better than their parent 2-stream GMM (11.60% and 11.48% vs. 11.68%). This shows the superiority of SVQ over SQ in reducing the quantization error.
- The performance of SVQ-HDDM-g actually matches that of the parent 1-stream GMM if the top-5 Gaussian speedup heuristic is used.

5.4.3 Operating Characteristics of Various Models

There can be two operation modes for an SV application depending on whether the models are pre-loaded into the memory. Model pre-loading is preferred if there is sufficient memory or when the number of speaker models is small. Thus, we measure and report the operating characteristics of various GMMs, SQ-HDDMs, and SVQ-HDDMs on the NIST 2002 SV task in two figures, depending on whether models are preloaded in Figure 5.3 and Figure 5.4. The computation time is collected over all 1442×11 trials. In both cases, the performance of the following models is compared:

- GMMs with varied number of Gaussian components.

- SQ-HDDMs and SVQ-HDDMs of varied resolutions (indexed from a to f) as shown in Table 1 and they are converted from the GMMs with 1024 Gaussian components.

From Figure 5.3, we find that if all models are pre-loaded into the memory, for the same EER

- SQ-HDDM is faster than SVQ-HDDM which is in turn faster than GMM. Since the speed of finding the HDDM likelihood is the same for both SQ-HDDM and SVQ-HDDM which is simply a table-lookup, SQ-HDDM is faster because the search for SQ codewords is faster. However, SQ-HDDM cannot achieve the best performance of its parent 1-stream or 2-stream GMM.
- On the other hand, SVQ-HDDM can achieve almost the best performance of its parent 1-stream GMM at a much faster speed.
- In general, for the same EER, SVQ-HDDM is 8$\sim$25 times faster than GMM. The speedup performance compares favorably with other fast GMM-UBM computation methods as reviewed in Section 5.2.

On the other hand, if speaker models are loaded on-the-fly, Fig 5.4 shows that the computational advantage of HDDM (both SQ-HDDM and SVQ-HDDM) over GMM is largely offset by the memory loading time of its much bigger model. However, the memory loading time is affected greatly by the hardware. We believe that using better server-grade hardware may improve the operating characteristics of HDDMs in Figure 5.4.

5.5 Summary

In this chapter, the very large codebook discrete distributions have been applied to replace the GMMs in GMM-UBM systems. The SQ/SVQ discrete models, which were originally developed in ASR, are evaluated here. Given the original well-trained GMM-UBM systems, only simple conversions are needed to generate the discrete systems, without requiring the use of any training data.

When the memory is large enough, and the models can be pre-loaded into it, the discrete model shows a speedup of 8$\sim$25 folds compared with GMM-UBM systems. This speedup ratio compares favorably with other speedup methods in the literature.

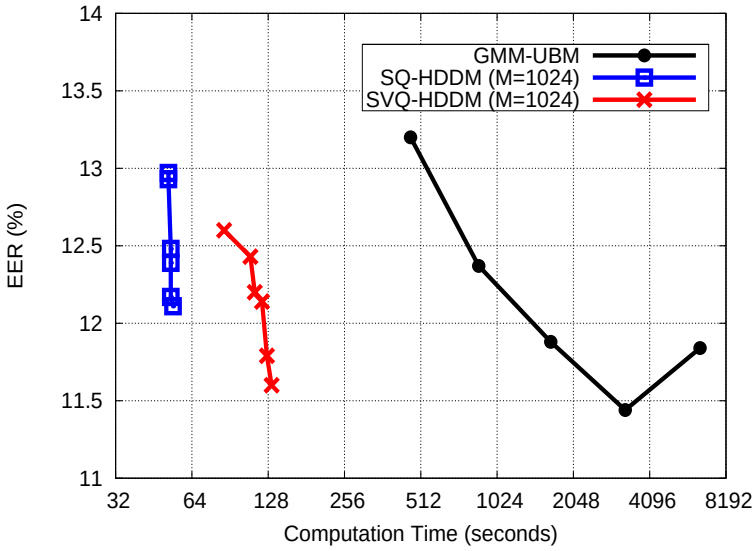


Figure 5.3: SV operating characteristics of various models, when the UBM and speaker models are pre-loaded to the memory.

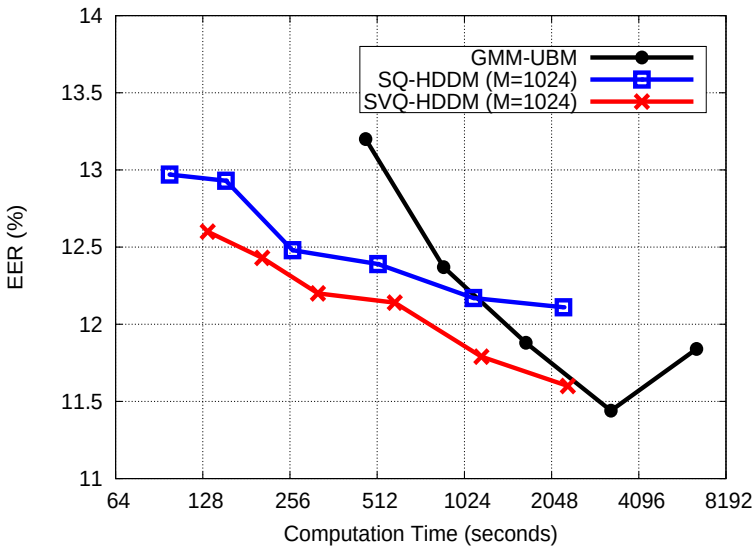


Figure 5.4: SV operating characteristics of various models, when the UBM and speaker models are loaded on-the-fly.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

With the advance of semiconductor technology, and the popularity of cloud computing paradigm, this thesis revisits the use of discrete models in automatic speech recognition (ASR) and speaker verification (SV) tasks. A very large codebook is proposed to reduce the quantization error in discrete models. Various issues of the very large codebook systems are investigated in the ASR task: what is the effect of codebook size on recognition performance, which type of codebook to use, how many streams should be used, how to train such a very large codebook DHMM (or called high-density DHMM, HDDHMM) with hundreds of times more parameters, and how to compensate the loss due to the incorrect stream independence assumption in multiple-stream systems. We also implement the very large codebook system for SV tasks. The major results/findings are:

- Increasing the codebook size helps the recognition performance, but not in a very efficient way.
- The use of SVQ codebooks gives better performance than SQ codebooks for very large codebook systems.
- Even with very large codebooks, it is still necessary to split the features into multiple independent streams.
- The subspace modeling technique with constrained bases could effectively reduce the number of free parameters by ten and even hundred folds, which makes robust parameter estimation of such a high-density system feasible.
- A modified iterative linear programming discriminative training technique could be applied to tune the stream weights of the multiple-stream system, which further improves the system performance.

- The recognition accuracy of HDDHMM outperforms SCHMM and conventional DHMM, and is very close to 1-stream CDHMM, with additional decoding speed and bandwidth advantages.

- Compared with other recently proposed discrete systems in ASR, HDDHMM shows better recognition accuracy.

- When the models are pre-loaded into the memory, the very large codebook systems give a speedup of 8$\sim$25 fold in SV, compared with classical GMM-UBM based systems. This speedup ratio also compares favorably with other speedup methods in the literature.

6.2 Future Work

The following directions are worth exploring in the future:

- A combination of MMI tree based codebook and subspace modeling technique. Despite the discrete system based on MMI codebook was 0.83% absolute worse compared to our HDDHMM with subspace modeling and SVQ codebooks (see Section 4.10.2), it uses a much smaller codebook size (1024 vs. 65536), which indicates that the MMI codebook may be more efficient in quantizing the space. However, as seen in Section 4.10.2, under its current smoothing technique (called "smoothing by adding 1"), using larger codebook did not show better performance for the MMI system, which we believe is due to the fact that the smoothing technique used is not very helpful. Thus, combining MMI tree based codebook with our proposed smoothing technique by subspace modeling could be a promising direction, which may effectively reduce the model size of our current SHDDHMM, due to the more efficient MMI partition.

- For SHDDHMM, currently, each stream stores a global pool of bases, which may not have enough discriminative power for more complex tasks with a large amount of training data; Thus, finer level pools, e.g., phone/state dependent pools, may be considered in the future. The extreme case will be to store a pool for each tied state, and the model would actually degenerate to use the same structure as CDHMM.

- Apply subspace modeling technique in SV task. Currently, we only investigated the simple model conversion by integration technique in SV systems. It should be interesting to also apply the subspace modeling technique to SV task. Specifically, we could treat each speaker model as a tied state in the ASR task, and let them share a global pool of bases.

APPENDIX A

L-BFGS METHOD FOR NODE SPLITTING IN MMI-TREE BASED CODEBOOK CONSTRUCTION

Limited memory Broyden-Fletcher-Goldfarb-Shanno (BFGS) (L-BFGS) method is a member of the broad family of quasi-Newton gradient-based optimization algorithm. The algorithm stores the last M function value/gradient pairs and uses them to build positive definite approximate Hessian matrix, which is then used to make quasi-Newton step. One attractive property of LBFGS algorithm is its moderate memory requirement: only M (e.g., $3 \sim 10$) function/gradient pairs are used, which is usually smaller than the problem size.

There are many well developed numerical optimization tools that support L-BFGS method (e.g., [10]). Basically, these tools require to specify the objective function, as well as its gradient. In this Appendix, we will show how to apply L-BFGS for node splitting in MMI-tree based codebook construction in Section 4.10.2. Specifically, we will focus on how to define the objective function and derive the gradient for the node splitting problem.

To facilitate the discussion, the following notations are first introduced:

- s: state label.
- y: a vector with $y = [x^T, -1]^T$, where x is a feature vector (e.g., 39-dimensional MFCC vector in our experiment), and x^T is the transpose of x.
- θ: a vector of parameters that defines the hyperplane, with the number of elements equal to that in y. The point y lies in the hyperplane if and only if $y^T \cdot \theta = 0$.
- b: child node label, with $b = 0$ to denote left child, and $b = 1$ to denote right child. Given a point y, and the hyperplane θ, the child node that y belongs to will be computed as:

$$b = \begin{cases} 0, & y^T \cdot \theta < 0 \\ 1, & y^T \cdot \theta \geq 0 \end{cases} \tag{A.1}$$

Basically, the task is to find a hyperplane to partition the samples y (each associated with a state label s) in a node into 2 child nodes, such that the mutual information between state label s and child node b, denoted as $I(s,b)$, is maximized. Based on t entropy theory, we have:

$$I(s,b) = H(s) - H(s|b),$$

where $H(\cdot)$ means the entropy. Because the entropy of s is independent of the hyperplane parameters, maximize the mutual information between s and b is the same as minimizing the conational entropy $H(s|b)$. Thus, the objective function of the problem could be formally defined as:

$$\arg\min_{\theta} H_{\theta}(s|b) \tag{A.2}$$

The gradient of objective function is derived below:

$$\begin{aligned}
\frac{\delta}{\delta\theta}H(s|b) &= -\frac{\delta}{\delta\theta}\sum_{b,s} P(s,b)\ln P(s|b) \\
&= -\sum_{b,s}\ln P(s|b)\frac{\delta}{\delta\theta}P(s,b) + P(s,b)\frac{\delta}{\delta\theta}\ln P(s|b) \\
&= -\sum_{b,s}\ln P(s|b)\frac{\delta}{\delta\theta}P(s,b) + \frac{P(s,b)}{P(s|b)}\frac{\delta}{\delta\theta}P(s|b) \\
&= -\sum_{b,s}\ln P(s|b)\frac{\delta}{\delta\theta}P(s,b) + P(b)\frac{\delta}{\delta\theta}P(s|b) \\
&= -\sum_{b,s}\ln P(s|b)\frac{\delta}{\delta\theta}P(s,b) + \frac{\delta}{\delta\theta}P(s,b) - P(s|b)\frac{\delta}{\delta\theta}P(b) \\
&= -\sum_{b,s}(\ln P(s|b)+1)\frac{\delta}{\delta\theta}P(s,b) - P(s|b)\frac{\delta}{\delta\theta}P(b) \\
&= \sum_{b,s} P(s|b)\frac{\delta}{\delta\theta}P(b) - (\ln P(s|b)+1)\frac{\delta}{\delta\theta}P(s,b),
\end{aligned} \tag{A.3}$$

where

$$\frac{\delta}{\delta\theta}P(b) = \sum_{s}\frac{\delta}{\delta\theta}P(s,b) \tag{A.4}$$

$$\frac{\delta}{\delta\theta}P(s,b) = \sum_{y} P(s|y)P(b|y)P(y)\frac{\delta}{\delta\theta}\ln P(b|y) = \sum_{y} P(s,b,y)\frac{\delta}{\delta\theta}\ln P(b|y) \tag{A.5}$$

Finally, to compute the derivative of $\ln P(b|y)$ with respective to θ in Equation (A.5), the original hard decision $P(b|y)$ in Formula (A.1) is replaced with the following differentiable form:

$$P(b|y) = \frac{exp(by^T \cdot \theta)}{1 + exp(y^T \cdot \theta)}$$

such that,

$$\begin{aligned} \frac{\delta}{\delta\theta} \ln P(b|y) &= \frac{\delta}{\delta\theta}(by^T \cdot \theta) - \frac{\delta}{\delta\theta} \ln(1 + exp(y^T \cdot \theta)) \\ &= by - \frac{1}{1 + exp(y^T \cdot \theta)} exp(y^T \cdot \theta) y \\ &= (b - \frac{exp(y^T \cdot \theta)}{1 + exp(y^T \cdot \theta)}) y \end{aligned} \tag{A.6}$$

REFERENCES

[1] J. Ajmera and M. Akamine. Speech recognition using soft decision trees. In *the Ninth Annual Conference of the International Speech Communication Association*, 2008.

[2] J. Ajmera and M. Akamine. Decision tree acoustic models for ASR. In *the Tenth Annual Conference of the International Speech Communication Association*, 2009.

[3] M. Akamine and J. Ajmera. Decision tree based acoustic models for speech recognition. *EURASIP Journal on Audio, Speech, and Music Processing*, 2012:10, 2012.

[4] R. Auckenthaler and J.S. Mason. Gaussian selection applied to text-independent speaker verification. In *2001: A Speaker Odyssey-The Speaker Recognition Workshop*, 2001.

[5] M. Barszcz, W. Chen, G. Boulianne, and P. Kenny. Tree-structured vector quantization for speech recognition. *Computer Speech & Language*, 14(3):227–239, 2000.

[6] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[7] J.R. Bellegarda and D. Nahamoo. Tied mixture continuous parameter modeling for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(12):2033–2045, 1990.

[8] F. Bimbot, J.F. Bonastre, C. Fredouille, G. Gravier, I. Magrin-Chagnolleau, S. Meignier, T. Merlin, J. Ortega-García, D. Petrovska-Delacrétaz, and D.A. Reynolds. A tutorial on text-independent speaker verification. *EURASIP Journal on Advances in Signal Processing*, 2004(4):430– 451, 2004.

[9] E. Bocchieri and B.K.W. Mak. Subspace distribution clustering hidden Markov model. *IEEE Transactions on Speech and Audio Processing*, 9(3):264–275, 2001.

[10] S. Bochkanov and V. Bystritsky. Alglib, 2007.

[11] H. Bourlard and N. Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer, 1994.

[12] A. Buzo, A. Gray Jr, R. Gray, and J. Markel. Speech coding based upon vector quantization. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(5):562–574, 1980.

[13] W.M. Campbell, D.E. Sturim, and D.A. Reynolds. Support vector machines using GMM supervectors for speaker verification. *Signal Processing Letters, IEEE*, 13(5):308–311, 2006.

[14] A. Chan, R. Mosur, A. Rudnicky, and J. Sherwani. Four-layer categorization scheme of fast GMM computation techniques in large vocabulary continuous speech recognition systems. In *Eighth International Conference on Spoken Language Processing*, 2004.

[15] I.T. Chapter. Getting to know SIRI.

[16] Y.L. Chow. Maximum mutual information estimation of HMM parameters for continuous speech recognition using the N-best algorithm. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 701–704. IEEE, 1990.

[17] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366, 1980.

[18] J.A. De Loera, B. Dutra, M. Koeppe, S. Moreinis, G. Pinto, and J. Wu. Software for exact integration of polynomials over polyhedra. *Computational Geometry*, 2012.

[19] M. De Wachter, M. Matton, K. Demuynck, P. Wambacq, R. Cools, and D. Van Compernolle. Template-based continuous speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1377–1390, 2007.

[20] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[21] K. Demuynck, D. Seppi, D. Van Compernolle, et al. Progress in example based automatic speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4692–4695. IEEE, 2011.

[22] V. Digalakis, S. Tsakalidis, and L. Neumeyer. Reviving discrete HMMs: the myth about the superiority of continuous HMMs. In *the Sixth European Conference on Speech Communication and Technology*, 1999.

[23] V.V. Digalakis, L.G. Neumeyer, and M. Perakakis. Quantization of cepstral parameters for speech recognition over the world wide web. *IEEE Journal on Selected Areas in Communications*, 17(1):82–90, 1999.

[24] J. Droppo, M.L. Seltzer, A. Acero, and Y.H.B. Chiu. Towards a non-parametric acoustic model: an acoustic decision tree for observation probability calculation. In *the Ninth Annual Conference of the International Speech Communication Association*, 2008.

[25] R.O. Duda, P.E. Hart, and D.G. Stork. Pattern classification. *New York: John Wiley, Section*, 10:l, 2001.

[26] J.T. Foote. *Decision-tree probability modeling for HMM speech recognition*. PhD thesis, Brown University, 1994.

[27] S. Furui. Speaker-independent isolated word recognition using dynamic features of speech spectrum. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(1):52–59, 1986.

[28] M.J.F. Gales, K.M. Knill, and S.J. Young. State-based Gaussian selection in large vocabulary continuous speech recognition using HMMs. *IEEE Transactions on Speech and Audio Processing*, 7(2):152–161, 1999.

[29] J.L. Gauvain and C.H. Lee. Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains. *IEEE Transactions on Speech and Audio Processing*, 2(2):291–298, 1994.

[30] L. Gillick and S.J. Cox. Some statistical issues in the comparison of speech recognition algorithms. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 532–535. IEEE, 1989.

[31] R. Gray. Vector quantization. *ASSP Magazine, IEEE*, 1(2):4–29, 1984.

[32] H. Hermansky. Perceptual linear predictive (PLP) analysis of speech. *The Journal of the Acoustical Society of America*, 87:1738, 1990.

[33] J. Hernando. Maximum likelihood weighting of dynamic speech features for CDHMM speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 1267–1270. IEEE, 1997.

[34] H.G. Hirsch and D. Pearce. The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In *ASR2000-Automatic Speech Recognition: Challenges for the New Millenium ISCA Tutorial and Research Workshop (ITRW)*, 2000.

[35] X. Huang, A. Acero, H.W. Hon, et al. *Spoken language processing*, volume 15. Prentice Hall PTR New Jersey, 2001.

[36] X. Huang and K.F. Lee. On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition. *IEEE Transactions on Speech and Audio Processing*, 1(2):150–157, 1993.

[37] X.D. Huang, M.Y. Hwang, L. Jiang, and M. Mahajan. Deleted interpolation and density sharing for continuous hidden Markov models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 885–888. IEEE, 1996.

[38] X.D. Huang and M.A. Jack. Semi-continuous hidden Markov models for speech signals. *Computer Speech & Language*, 3(3):239–251, 1989.

[39] M.Y. Hwang and X. Huang. Shared-distribution hidden Markov models for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 1(4):414–420, 1993.

[40] F. Jelinek. Interpolated estimation of Markov source parameters from sparse data. *Pattern Recognition in Practice*, pages 381–397, 1980.

[41] P. Kenny, G. Boulianne, P. Ouellet, and P. Dumouchel. Joint factor analysis versus eigenchannels in speaker recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1435–1447, 2007.

[42] J. Kharroubi, D. Petrovska-Delacrétaz, and G. Chollet. Combining GMM's with support vector machines for text independent speaker verification. In *Proc. European Conference on Speech Communication and Technology*, pages 1757–1760, 2001.

[43] N.S. Kim and C.K. Un. On estimating robust probability distribution in HMM-based speech recognition. *IEEE Transactions on Speech and Audio Processing*, 3(4):279–285, 1995.

[44] T. Kinnunen and H. Li. An overview of text-independent speaker recognition: from features to supervectors. *Speech Communication*, 52(1):12–40, 2010.

[45] K.F. Lee. Context-dependent phonetic hidden Markov models for speaker-independent continuous speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(4):599–609, 1990.

[46] K.F. Lee and H.W. Hon. Speaker-independent phone recognition using hidden Markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(11):1641–1648, 1989.

[47] S.E. Levinson, L.R. Rabiner, and M.M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *Bell Syst. Tech. J*, 62(4):1035–1074, 1983.

[48] J. Li, Y. Tsao, and C.H. Lee. Shrinkage model adaptation in automatic speech recognition. In *the Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[49] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.

[50] D.C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[51] B. Mak and T. Ko. Automatic estimation of decoding parameters using large-margin iterative linear programming. In *Proc. of Interspeech*, pages 1219–1222, 2009.

[52] B. Mak and B. Ng. Discriminative training by iterative linear programming optimization. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4061–4064. IEEE, 2008.

[53] B.K.W. Mak, S.K. Au Yeung, Y.P. Lai, and M.H. Siu. High-density discrete HMM with the use of scalar quantization indexing. In *Proc. of Interspeech*, pages 2121–2124, 2005.

[54] J. Makhoul. Spectral analysis of speech by linear prediction. *IEEE Transactions on Audio and Electroacoustics*, 21(3):140–148, 1973.

[55] T. Matsui and S. Furui. Comparison of text-independent speaker recognition methods using VQ-distortion and discrete/continuous HMMs. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 157–160. IEEE, 1992.

[56] J. McLaughlin, D.A. Reynolds, and T. Gleason. A study of computation speed-ups of the GMM-UBM speaker recognition system. In *Proc. Eurospeech*, volume 99, pages 1215–1218, 1999.

[57] A.S. MOSEK. The MOSEK optimization software. *Online at http://www. mosek. com*, 2010.

[58] C. Neukirchen, J. Rottland, D. Willett, and G. Rigoll. A continuous density interpretation of discrete HMM systems and MMI-neural networks. *IEEE Transactions on Speech and Audio Processing*, 9(4):367–377, 2001.

[59] Y.L. Ng. *Discriminative training of stream weights in a multi-stream HMM as a linear programming problem*. PhD thesis, The Hong Kong University of Science and Technology, 2008.

[60] M. Ostendorf and J.R. Rohlicek. Joint quantizer design and parameter estimation for discrete hidden Markov models. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 705–708. IEEE, 1990.

[61] K.K. Paliwal and B.S. Atal. Efficient vector quantization of LPC parameters at 24 bits/frame. *IEEE Transactions on Speech and Audio Processing*, 1(1):3–14, 1993.

[62] D.S. Pallet, W.M. Fisher, and J.G. Fiscus. Tools for the analysis of benchmark speech recognition tests. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 97–100. IEEE, 1990.

[63] D. Pearce. Enabling new speech driven services for mobile devices: an overview of the ETSI standards activities for distributed speech recognition front-ends. In *AVIOS 2000: The Speech Applications Conference*, 2000.

[64] D. Povey, L. Burget, M. Agarwal, P. Akyazi, K. Feng, A. Ghoshal, O. Glembek, N.K. Goel, M. Karafiát, A. Rastrow, et al. Subspace Gaussian mixture models for speech recognition. In *IEEE International Conference on Acoustics Speech and Signal Processing*, pages 4330–4333. IEEE, 2010.

[65] D. Povey, L. Burget, M. Agarwal, P. Akyazi, F. Kai, A. Ghoshal, O. Glembek, N. Goel, M. Karafiát, and A. Rastrow. The subspace Gaussian mixture model–a structured model for speech recognition. *Computer Speech & Language*, 25(2):404–439, 2011.

[66] M. Przybocki and A.F. Martin. NIST speaker recognition evaluation chronicles. In *ODYSSEY04-The Speaker and Language Recognition Workshop*, 2004.

[67] L. Rabiner and B. Juang. An introduction to hidden Markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[68] W. Reichl and W. Chou. Robust decision tree state tying for continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 8(5):555–566, 2000.

[69] D.A. Reynolds. Comparison of background normalization methods for text-independent speaker verification. In *the Fifth European Conference on Speech Communication and Technology*, 1997.

[70] D.A. Reynolds, T.F. Quatieri, and R.B. Dunn. Speaker verification using adapted Gaussian mixture models. *Digital Signal Processing*, 10(1):19–41, 2000.

[71] T.N. Sainath, B. Ramabhadran, M. Picheny, D. Nahamoo, and D. Kanevsky. Exemplar-based sparse representation features: from TIMIT to LVCSR. *IEEE Transactions on Audio, Speech, and Language Processing*, (99):1–1, 2011.

[72] R. Schwartz, O. Kimball, F. Kubala, M.W. Feng, Y.L. Chow, C. Barry, and J. Makhoul. Robust smoothing methods for discrete hidden Markov models. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 548–551. IEEE, 1989.

[73] C.H. Sit, M.W. Mak, and S.Y. Kung. Maximum likelihood and maximum a posteriori adaptation for distributed speaker recognition systems. *Biometric Authentication*, pages 15–31, 2004.

[74] G.K. Smyth. Numerical integration. *Encyclopedia of Biostatistics*, 2005.

[75] X. Sun and Y. Zhao. Integrate template matching and statistical modeling for speech recognition. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[76] S. Takahashi, K. Aikawa, and S. Sagayama. Discrete mixture HMM. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 971–974. IEEE, 1997.

[77] R. Teunen and M. Akamine. HMM-Based speech recognition using decision trees instead of GMMs. In *the Eighth Annual Conference of the International Speech Communication Association*, 2007.

[78] S. Tsakalidis, V. Digalakis, and L. Neumeyer. Efficient speech recognition using subvector quantization and discrete-mixture HMMs. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 569–572. IEEE, 1999.

[79] V. Vapnik. *The nature of statistical learning theory*. Springer, 1999.

[80] C. Weinstein, S.S. McCandless, L. Mondshein, and V. Zue. A system for acoustic-phonetic analysis of continuous speech. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(1):54–67, 1975.

[81] P.C. Woodland and D. Povey. Large scale discriminative training of hidden Markov models for speech recognition. *Computer Speech & Language*, 16(1):25–47, 2002.

[82] B. Xiang and T. Berger. Efficient text-independent speaker verification with structural Gaussian mixture models and neural network. *IEEE Transactions on Speech and Audio Processing*, 11(5):447–456, 2003.

[83] G. Ye and B. Mak. Subvector-quantized high-density discrete hidden Markov model and its re-estimation. In *the 7th International Symposium on Chinese Spoken Language Processing*, pages 109–113. IEEE, 2010.

[84] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X.A. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, et al. The HTK book (for HTK version 3.4). 2006.

[85] S.J. Young, J.J. Odell, and P.C. Woodland. Tree-based state tying for high accuracy acoustic modelling. In *Proceedings of the Workshop on Human Language Technology*, pages 307–312. Association for Computational Linguistics, 1994.