Multi-level Acoustic Modeling for Automatic Speech Recognition

by

Hung-An Chang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author ....
Department of Electrical Engineering and Computer Science
April 27, 2012

Certified by ....
James R. Glass
Senior Research Scientist
Thesis Supervisor

Accepted by ....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Multi-level Acoustic Modeling for Automatic Speech Recognition

by

Hung-An Chang

Submitted to the Department of Electrical Engineering and Computer Science
on April 27, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Context-dependent acoustic modeling is commonly used in large-vocabulary Automatic Speech Recognition (ASR) systems as a way to model coarticulatory variations that occur during speech production. Typically, the local phoneme context is used as a means to define context-dependent units. Because the number of possible context-dependent units can grow exponentially with the length of the contexts, many units will not have enough training examples to train a robust model, resulting in a data sparsity problem.

For nearly two decades, this data sparsity problem has been dealt with by a clustering-based framework which systematically groups different context-dependent units into clusters such that each cluster can have enough data. Although dealing with the data sparsity issue, the clustering-based approach also makes all context-dependent units within a cluster have the same acoustic score, resulting in a quantization effect that can potentially limit the performance of the context-dependent model.

In this work, a multi-level acoustic modeling framework is proposed to address both the data sparsity problem and the quantization effect. Under the multi-level framework, each context-dependent unit is associated with classifiers that target multiple levels of contextual resolution, and the outputs of the classifiers are linearly combined for scoring during recognition. By choosing the classifiers judiciously, both the data sparsity problem and the quantization effect can be dealt with. The proposed multi-level framework can also be integrated into existing large-vocabulary ASR systems, such as FST-based ASR systems, and is compatible with state-of-the-art error reduction techniques for ASR systems, such as discriminative training methods.

Multiple sets of experiments have been conducted to compare the performance of the clustering-based acoustic model and the proposed multi-level model. In a phonetic recognition experiment on TIMIT, the multi-level model has about 8% relative improvement in terms of phone error rate, showing that the multi-level framework can help improve phonetic prediction accuracy. In a large-vocabulary transcription task, combining the proposed multi-level modeling framework with discriminative training can provide more than 20% relative improvement over a clustering baseline model in terms of Word Error Rate (WER), showing that the multi-level framework can be integrated into existing large-vocabulary decoding frameworks and that it combines well with discriminative training methods. In a speaker adaptive transcription task, the multi-level model has about 14% relative WER improvement, showing that the proposed framework can adapt better to new speakers, and potentially to new environments than the conventional clustering-based approach.

Thesis Supervisor: James R. Glass
Title: Senior Research Scientist

Acknowledgements

Ph. D study is a once in a lifetime journey: in the beginning, many adventures to expect; in between, many ups and downs to experience, but hopefully with enough grace to get through; and in the end, many memories to remember, and many thanks to give.

First of all, I would like to thank my advisor Jim Glass for providing constant support and guidance throughout the study, for offering the freedom of proposing and testing new ideas, and for giving many constructive encouragements and suggestions when some ideas failed. I would also like to give thanks to my thesis committee, Tommi Jaakkola and Victor Zue, for their helpful feedbacks to this thesis. Thanks to Tommi for providing many insightful inputs from machine-learning points of view. Thanks to Victor for giving many good suggestions of shaping the structure of the thesis. As no research would be possible without consistent financial support, I would like to give sincere thanks to Quanta Computer Inc. for funding the research of the thesis.

The research conducted in this thesis would not have been possible without the kind supports of many people. Specifically, I would like to give special thanks to Lee Hetherington for the assistance of FST-related issues, T. J. Hazen for the guidance of setting up the lecture corpus, Natasha Singh-Miller for providing the NCA transform for the experiments, Scott Cyphers for keeping the machines in good condition, Najim Dehak for bringing many new machines, Paul Hsu for teaching me many tips of using the SUMMIT recognizer, and Ken Schutte for teaching me how to use host-dispatch nearly in the beginning of my Ph. D study. Also, I would like to specially thank Steven Shum, David Harwarth, and Ekapol Chuangsuwanich for reviewing the thesis and providing many helpful feedbacks and suggestions.

Environment is an important factor that helps one progress in the Ph. D study. I would like to thank the entire SLS group for providing such an excellent and enjoyable environment. Thanks to Stephanie Seneff for many helpful suggestions to my conference presentations. Thanks to Najim for many warm encouragements. Thanks to Yushi and Ian for getting through the Ph. D study together with me. We came in the same year and graduated in the same year. I would like to thank them for many discussions of problem sets and research ideas for the past hmm...several years. Thanks to Jingjing for many fun and deep talks and for many afternoon snacks over the years. I would like to thank my excellent officemates, Yaodong, Jackie, and Ekapol. Thanks to Yaodong for teaching me how to use many convenient toolkits, and for many helpful suggestions on many aspects of life. Thanks to Jackie for bringing much fun and energy to the office, and for giving Zoe many cute gifts. Thanks to Ekapol for many insightful discussions, many interesting collaborations, and many fun board-games. (As a part of my thanks, I will remember your last name.) Also, I would like to thank our part-time officemates, Steven and David, for coming to our office from time to time and raising many interesting discussions over a variety of topics, and for being very good board-game players. Thanks to Ann for being a very good free-food scout, giving me advantage in the 4-floor vulture battle. Thanks to Carrie for teaching many tips that might help Zoe to be fluent in both Chinese and English. Thanks to William for helpful suggestions to my presentations. Thanks Xiao and Tuka for coming to my practice talk. Thanks to

Timo for many encouragements during my job search. (Look forward to cooperate with you, future colleague!) Thanks to Seastar for bringing Taiwanese desserts.

I would also like to thank many SLS emeritus. Thanks to Chao Wang for many kind handy tips when I just joined the SLS group. Thanks to Mitch Peabody for hosting me during the new student visit. Thanks to Ghinwa Choueiter and John Lee for many helpful tips for finding internship opportunities and for dealing with the logistics of CPT. Thanks to Tara Sainath and Larry Gillick for many interesting discussions on ASR related research. Thanks to Alex Gruenstein for lending a baby crib for Zoe during my internship at Google. Thanks to Alex Park for sharing many experience of Ph.D study. Thanks to Rabih Zbib for many nice squash games we have played. Thanks to Karen Livescu for being a very nice instructor in 6.345-2007. Thanks to Che-Kuang Lin for many warm encouragements and helpful discussions during the time I was writing my Master thesis. Thanks to Sean Liu for giving many cute gifts to Zoe. Thanks to Ibrahim Badr for many brain-storming discussions of pronunciation modeling. Thanks to Stanley Wang and Kevin Luu for letting me learn a lot from the teaching of 6.867. Thanks to Anna Goldie and Alice Li for giving me many opportunity to practice my English. Thanks to Bo Zhu, Greg Yu, and James Sun for being very nice classmates for several classes. Thanks to Luke Liu and Mike Hsu for sharing many industry experiences in Taiwan.

Thanks to many logistical helps from CSAIL support staffs. Specifically, I would like to thank Marcia Davidson for the helps in many conference reimbursements, and for many group events. More importantly, many thanks to Marcia for helping me decipher Jim's hand writings. (As a part of the thanks, I won't forget your birthday, as long as I still remember mine.) Thanks to Colleen Russell for many helps for scheduling meeting times with Victor and for event reimbursements. Thanks to Maria Rebelo for providing many afternoon snacks.

Thanks to my former officemates at 32G-360, Gabi Zaccak and Igor Malioutov, for many fun talks. Also, thanks to Gabi for many nice squash games we have played.

I would like to thanks the teaching staffs and students of 6.867-2008 for making me learn many valuable experiences from being one of the TAs. Thanks to the MIT professors whose classes I took or sat in, and thanks to all the classmates in the study groups of all the classes. I have learned a lot from the class materials and the discussions. Also, thanks the officers in the CSAIL student committee and EECS GSA for organizing many fun events to remember.

The place of learning during the Ph. D study was not limited to school. Thanks to Francoise Beaufays, Brian Strope, and Yun-Hsuan Sung for being very nice mentors during my internship at Google. It made me learned a lot of practical experiences of dealing with huge amount of data. Thanks to Mike Phillips for the short-term part-time opportunity at Vlingo on speaker adaptation. Thanks to Erik McDermott and Shinji Watanabe for many interesting discussions concerning discriminative acoustic model training for ASR.

While the Ph. D study being a valuable experience, I would like to give thanks to the professors who helped me begin my Ph. D study. Thanks to Prof. Lin-Shan Lee, Prof. Homer Chen, and Prof. Fan-Ren Chang for writing the recommendation letters for me.

For the rest of the acknowledgement, I would like to give thanks to my friends in different parts of my line, and to my family members. Because many thanks could not be expressed precisely by my limited capability in English, and because some of them barely know English, the rest of would be mainly written in Chinese.

感謝在 MIT 的朋友們. 感謝歷屆的 ROCSA 會長: 為穿和 Iris, 賢中, 禹志和 Emmy, 賴桑, 鴻文和 Christina, YiChang. 感謝會長們和他們的幹部們精心安排了許多很有趣又有台灣味的活動. (會長都是有結婚的和單身的輪替, 真的蠻有趣的.)

感謝在 MIT CSAIL 的學長姐. 感謝邦邦學長在我修 TQE 課程的時候, 常常和我一起討論問題, 也教會了我怎麼打 squash. 感謝趙之瑜學姐幫我 refer 履歷, 還分享了許多談 offer 的經驗. 感謝王思博學長帶我們去許多好吃好玩的地方, 也留了許多好的餐具給我們.

感謝陪我在舊 Ashdown 五樓的廚房留下許多回憶的朋友們. 感謝葛禹志兄妹激起我對廚藝的熱誠. 感謝 Andrew 教我烤箱的奧義. 感謝 Sidney 和 Scott 常常勇於試吃我燒的新菜, 吃完也常留下來玩個幾道遊戲(吃喝玩樂之間, 兩人也就慢慢的胡在一起了…). 感謝小 K 大大玩 Attika 以後還送了我一個非常特別的結婚禮物. (現在應該蠻多蓋好了.) 感謝 Tim Chen 燒了許多好菜, 玩了許多好 games, 也讓我見識到不論玩遊戲或是燒菜都是可以講理論建 model 的. 我的室友尤林常常陪我一起做菜, 一起打 squash, 還 host 了 Ashdown 五樓住戶的 part lot.

感謝 Westgate 的好鄰居們. 感謝余禎祥學長一家在琇珮懷孕期間, 幫了我們很多忙, 也分享了許多帶小孩的經驗. 也感謝學長 host 許多次歡樂的 Westgate 火鍋趴. 感謝禹志和 Emmy 帶我一起參加活動, 也留下很多美好的回憶. 感謝 David 和 Maggie 在琇珮生產的時候還特地送了 Trader Joe 的 Gift card 讓琇珮吃的補身子. 感謝士育和 Rita 在我們回台灣的時候幫我們收信, 換鎖, 還幫我寄 EAD 卡到台灣. 感謝 Rita 常陪琇珮和心澄出去走走, 讓她們母女倆都很開心. 感謝管軍毅和 Cynthia 常常會幫我們代買東西和幫忙照顧心澄. 感謝 Saiko 送了很多可愛的書給心澄. 感謝塞凡提斯和 Irene 一家以及 Chun-ming 學長一家留給我們不少家具. Thanks to WEC officers for hosting many interesting westgate-wide events. Thanks to 7th floor represents of different turns, Gloria, Jennifer, Li and Dasha, for hosting many fun floor activities. 感謝住 Tang Hall 李幼慈留給我們一些好用的補給.

感謝許多住在大西北宿舍區的朋友. 感謝(超發的)高宗常邀請我們去新 Ashdown 的火鍋趴, 也感謝 Jane Lee, Kay, vgod 和 Ingrid 幫忙準備好吃的食材, 讓我們一家每一次都吃的很開心. 感謝 JJ 和 Albert 讓我們吃到很好吃的 BBQ. 感謝 Samuel 在我 defense 的時候幫我載了很多東西. 感謝趙玲學姐和白佳靈學姊辦了幾場令人回味無窮的水餃大會. 感謝彥頡和政勳和我下了幾盤指導棋, 讓我圍棋進步不少. 感謝天雲大師給了我一張讓我用到畢業的書桌. 感謝政緯在各個活動中拍了許多值得留戀的照片. 感謝張梁益和我一起和日文一二拼鬥.

感謝 Tim 和 Kelly 請我們到他們家吃到不少好料. 感謝 Kelly 不時會找琇珮一起去買東西, 也推薦了許好用的東西. 感謝 Ted 和 Grace 幫心澄拍了好多漂亮的照片. 感謝 Nancy 和 Thomas 送了心澄一套超可愛的衣服. 感謝 podo 幫我選了一個很划算的 family plan. 感謝蔡琮涵和 Jessica 送了我們一組也是用到畢業的架子. 感謝李務熙學長介紹了許多有趣的 media lab project. 感謝莊志超大哥今年發起了值日生台便當的活動.

感謝那些年一起在 Sunday Rivier 滑雪的朋友們, 讓我們留下許多美好的回憶.

感謝強大的 NTUEE 網絡. 感謝金牌學長和寶萱學姊一家 讓我們有許多趟愉快的紐約之旅. 也感謝學長幫我 refer google 的 intern; 感謝寶萱學姐和琇珮分享許多懷孕和帶小孩的經驗. 感謝藍斯學長和旻文學姐帶我們去了許多灣區附近的地方玩, 也讓我們去參觀他們超發的家. (話說藍斯學長接我們機從來不會誤點, 完全沒有所謂的藍斯威能.) 感謝 KJ 大大屢次在灣區接送我們, 也給我們許多在灣區找房子和生活的建議. 感謝宋雲軒學長在我在 google intern 的時候幫了我很多忙. 感謝 bobbywin 學長帶我參觀傳說中的 S 大. 感謝黃表維夫婦讓我有一個愉快的香檳之旅. 感謝前前後後一起準備資料出國唸書的同學們, 讓各大校園都有駐點; 之後各大企業也大概也會有不少樁腳吧? (話說線在連內地也走的通了.) 感謝吳宗睿學長, 張國韋兄弟每次回台灣都讓我新學會一些可以鍛鍊思考的 borad-games.

感謝我的高中同學們. 感謝吳書, 漢霖, 陳毅, 游敏, 和國華讓我回台灣的時侯都聚的很歡樂 (聚會人數也要增加超過一倍了呢!). 感謝吳書在心澄生病的時候給了我們許多專業建議; 感謝國華免費幫我看牙齒; 感謝鴻安哥哥成了好醫生. 感謝林亞鋒讓我學到許多社會學的觀點, 也希望他能快點讓我那不願去簽美簽的老爸, 會願賭服輸的到美國.

感謝與我一同當兵的同梯們. 感謝建勳推薦了許多紐約好吃好玩的. (要好好照顧念慈和 Rocky 醬喔!) 感謝士洋, 開源, 泰儒和鐘銘讓我回台灣都聚得很歡樂. (這幾年來聚會人數也增加超過一倍了呢!)

感謝琇珮的朋友們, 讓她回台灣的時候都很開心. 也向和我老妹一起合謀, 介紹我和琇珮認識的李佳盈, 致上特別的感謝.

感謝各地的弟兄姊妹們和我們一起經歷何謂那擴, 長, 高, 深, 以及基督超越知識的愛; 讓琇珮, 心澄, 還有我在生活的各方面都很得供應和滋養

Thanks to the household of Brother Dave Bekker. Thanks to our dear Brother Dave for shepherding me over the years, even till his last few days on earth: giving me my first new testament Bible, coming to MIT for Bible study on a weekly basis while his physical condition was able, praying with me through the phone when his physical condition was no longer allowed him to come out, and trying to give Shiu-Pei and me the life study of Exodus even when his condition had degenerated to an extent that uttering each phoneme would require strenuous efforts. Brother Dave really demonstrated a strong pattern of shepherding under all circumstances. Thanks to Sister Debby for warmly inviting us to their home for several times, and for finishing the wish of Brother Dave of giving us the life study. It is really a nourishing gift, and we will treasure and enjoy it on a regular basis. Also, thanks to many Brothers for driving brother Dave to MIT, and for sharing many precious portions of Christ together.

感謝邦邦和 JuJu 一家. 感謝邦邦帶我到 Brother Dave 的 Bible study, 一起享受基督. 也感謝邦邦這一路上給我許多的勸勉和安慰. 感謝 JuJu 常常關懷問候琇佩. 感謝你們在我趕論文的時後幫忙照顧琇珮和心澄, 也感謝你們把心澄當親身女兒一般的疼愛. 感謝馥銘和 Ruth 一家. 感謝馥銘總是在我快要在學校被打趴的時候, 來找我喝個咖啡, 一起呼求主名, 一起禱告. 感謝 Ruth 常常和琇珮一起讀主的話, 一起帶小孩. (琇珮成長很多, 現在常用主話安慰人.) 感謝爲穿和 Iris 一家在琇珮剛到要 settle down 的時候幫了我們很多忙. 感謝 Yi Peng 常常和我一起晨興禱告. 感謝 Andrew 當了幾年的好鄰居, 也陪我一起坐車去了許多次的主日聚會. 感謝陽明在我們找工作的時候, 常常一起互相練習面試, 以及一起互相代禱. 感謝 Matt 麥和我一起讀創世紀和出埃及紀, 讓我很得供應. 感謝 Tom 葉弟兄在他還在 Cambridge 的時候常載邦邦和我去小排聚會和主日. 感謝梁妃常常關心琇珮和心澄. Thanks to Spancer and Matt from FTTA training for many nourishing Bible studies.

感謝在山景城召會的眾弟兄姊妹, 讓我們感受到我們不是外人或寄居的, 乃是神家裡的親人. 感謝 Tim, Katty, 和大 Zoe 一家對我們許多方面的照顧和供應. (大小 Zoe 真的好有姐妹相!) 感謝柯漢弟兄一家常常帶我們去買菜. 感謝蘇弟兄一家和 William 弟兄一家常常接待我們到家裡相調. 感謝關姊妹常常接送我們上主日, 也很照顧那時在加州的媽媽. 感謝 Wendy 姊妹教了我們許多照顧小 baby 的竅門. 感謝何弟兄在我們剛到加州的時後就聯絡我們, 把我們帶到召會. 感謝眾弟兄姊妹全備的供應和關愛, 讓不會開車又帶著小 baby 的我們, 沒有匱乏; 這也摸著到琇珮, 讓她決定信主受浸. 感謝讚美主!

感謝在劍橋召會的眾弟兄姊妹. 感謝崔弟兄和崔姊妹一家對我們的照顧, 常常關心我們家的狀況, 幫我們家情況代禱. 感謝崔弟兄常常來接送我們, 也常常帶我一起讀經, 讓我很得供應. 感謝崔姊妹常和 Ruth 與琇珮一起讀經. 感謝許多的弟兄姊妹讓小排聚會非常豐富, 滿了主的同在: 林舉賢弟兄一家, 吳興弟兄一家, 馬 Minling 弟兄一家, 當波弟兄, 彭芃, 子然, 文釗, 陳東, 郜婷, Iris 鄒, Rachel, 劉丹, 伯楊(準)弟兄... 感謝華卿弟兄和張彥姊妹讓我們見證了一場甜美的結婚聚會. 感謝許多接待我們到他們家的弟兄姊妹: 李大寧弟兄一家, 宋玉姊妹一家, Thomas 和 Cynthia 一家, Michael 和 Joyce 一家, Xintao 弟兄一家, John Chang 弟兄一家, 李雪弟兄一家, 戚繼發弟兄一家, 竇弟兄一家, Collin 弟兄一家… 讓我們有許多甜美的相調. 感謝許許多多的弟兄姊妹爲我們代禱, 以及許多弟兄姊妹在聚會上滿了供應的申言; 讓我們見證基督的身體如何藉著每一豐富供應的節, 以及每一肢體依其度量而有的功用, 在愛裡建造起來.

感謝在台灣的家人和親戚. 感謝外祖母, 大舅舅, 小舅舅, 和阿姨這些年在經濟上的支持. 感謝二姑丈幫我作菁英留學獎學金的擔保人. 感謝姑姑們和叔叔跟我爸媽都有很好的互動. 感謝所有表兄弟姐妹和堂弟們, 讓我們回台灣時的家族聚餐都很歡樂.

感謝我的哥哥宏印和我的妹妹君熒. 感謝哥哥這些年來一肩扛起家計; 感謝妹妹幫忙照顧在台灣的爸媽. 感謝他們的付出, 讓我在美國唸書的時候沒有後顧之憂; 感謝他們不時的關懷和鼓勵, 讓我更有衝刺的力量. 也感謝琇珮的姐妹們, 在我們回台灣的時候, 都很照顧琇珮和心澄. 手足之情, 真的很溫暖. (是不是也要快點讓心澄開始體驗手足之情了呢?) 感謝這些年陪伴哥哥的怡君姐, 還有才剛和妹妹結婚的吳孟儒. 也幫心澄感謝這些疼愛她的阿伯, 姑姑, 姑丈, 和阿姨們.

感謝從小把我拉拔長大的爸媽, 養育的恩情真的無法用言語描述. (有了心澄以後, 就更加體會到爸爸媽媽的辛苦.) 感謝媽媽在我們回台灣的時候, 總是準備了許多營養好吃的東西. 也很感謝媽媽在心澄出生的時候來美國幫忙照顧心澄和琇珮. 感謝爸爸在我們回台灣之前都會特地幫我們打掃房間和接送我們往來機場. 感謝琇珮的爸媽, 把這麼好的女兒許配給我. 也幫心澄感謝疼愛她的爺爺奶奶, 外公外婆. 看到她跟他們的互動, 真的天倫之樂何其美.

感謝並紀念已經過世爺爺奶奶和外公. 感謝他們從小就很疼我, 也很鼓勵我努力念書. 我也終於可以在唸書這件事上有所交代了.

感謝琇珮, 我的另一半, 我親愛的妻子. 感謝她這些年來一起和我經歷許多生活中的點點滴滴. 在我高興的時候陪我一起開心, 在我憂傷的時候安慰我, 在我受挫折的時候鼓勵我, 在我徬徨的時候拉我一起和主禱告. 感謝她在生活的各個方面對我的照顧和關懷; 感謝她和我一起在愛裡成長, 一起互相扶持; 感謝她讓我經歷婚姻生活是何等的幸福和喜樂.感謝我們的女兒,心澄. 感謝她帶給我們許多喜樂和甜美的記憶, 也讓我們一起經歷, 生命的長大是何等的奇妙, 可愛, 和可貴; 是何等的令人喜悅和期待.

最後, 感謝那施憐憫, 賜恩典, 獨一, 智慧的神. 感謝祂安排的一切; 也讓我經歷到, 信靠祂的人生中, 沒有所謂倒楣的事, 好運的事, 或是走不出的死胡同. 因爲萬有都互相效力, 爲要叫愛神的人得益處. 人生中會有起伏, 會有波折, 但祂總是會安排適當的人, 事, 物; 既給予應時的供應和幫助, 又讓人學到應學的功課; 藉著各樣的情境來變化和成全人. 缺乏之時豫備, 無倚之時扶持, 所有美時最美, 無論何時信實. 願往後的人生中, 能更加得著祂, 更加在生命上有所成長, 更加把愛和祝福帶給週遭的人; 更加摸著祂的美意, 走在祂經綸的路上. 讚美和榮耀歸與神!

張宏安

April, 2012

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automatic Speech Recognition (ASR) is the task of outputting a word sequence that corresponds to a given speech utterance. Over the years, ASR has been applied to many types of applications such as dictation [93, 78, 73], voiced-based information retrieval [31, 133, 28, 42], automatic processing and documentation of audio data [89, 36, 45], computer-aided language learning [124, 127, 98], speech-to-speech translation [72, 121], etc. While there are many ASR related applications, it is generally agreed upon that higher ASR accuracy results in better service [107, 61, 88]. In this thesis, we investigate a novel multi-level acoustic modeling framework that can improve the recognition accuracy of existing ASR systems.

1.1 Current Approach

Figure 1-1 illustrates a generic modeling framework of a state-of-the-art ASR system. Given a recorded speech waveform, the feature extraction module converts the waveform into a sequence of fixed-dimensional acoustic feature vectors (observations). The acoustic model scores the feature vectors, and provides scores for different phoneme sequences. The lexicon module applies lexical constraints and maps the phoneme sequences into possible word sequences. The word sequences are re-scored by the language model based on prior linguistic knowledge. Scores provided by the acoustic model, the lexicon, and the language model are incorporated by the search module to find the best-scoring word sequence. More

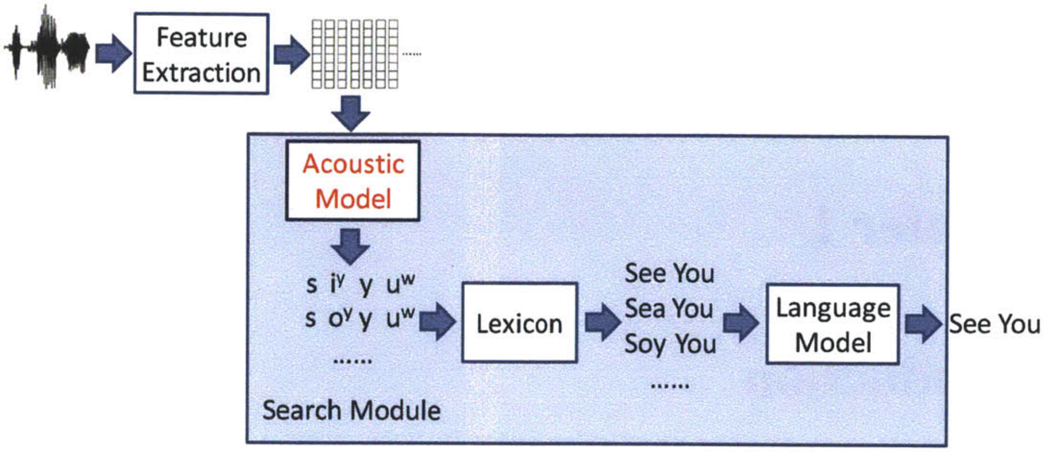


Figure 1-1: Modeling framework of a state-of-the-art ASR system.

details of the modeling framework of ASR systems can be found in Chapter 2. In this thesis, we work on developing a novel acoustic modeling framework that can further improve the ASR performance on large-vocabulary speech recognition tasks.

1.1.1 Context-Dependent Acoustic Modeling

In natural speech, a phonetic unit can be pronounced (realized) differently depending on its nearby phonetic units. Figure 1-2 shows several examples of how the realization of a phoneme "t" in American English can be affected by its nearby phonetic contexts. To model such coarticulatory variations, context-dependent acoustic modeling is commonly used in existing ASR systems. Given an acoustic observation (feature vector), $\mathbf{x}$, a context-dependent acoustic model seeks to score $\mathbf{x}$ with respect to both its phonetic identity and its nearby contexts; for example, how likely does $\mathbf{x}$ belong to the triphone unit "d-iy+n", an "iy" with left context "d" and right context "n", as in the word "dean"? As we increase the amount of contextual information being considered, the set of possible labels can grow exponentially with the length of the contexts. Take an ASR system with 60 basic phonetic units for example. A triphone acoustic model can have $60^3 = 216,000$ possible triphone units. Because the number of possible context-dependent labels can grow very large, many context-dependent units may have limited occurrences or even unseen in the training data,

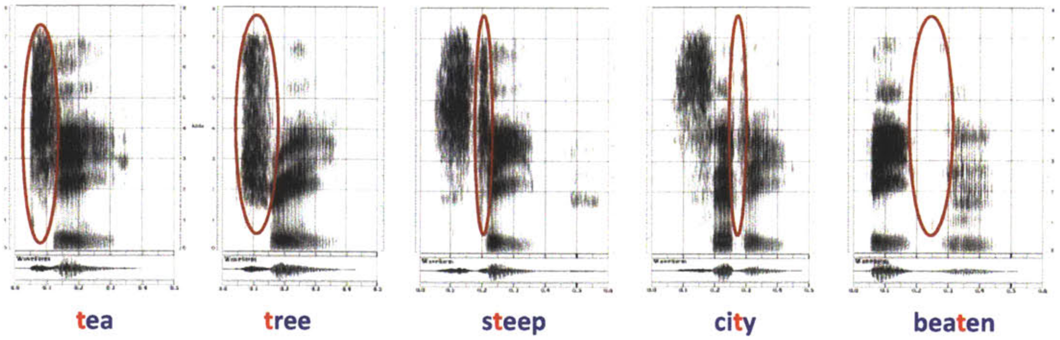


Figure 1-2: Acoustic realizations of the phoneme “t” with different nearby phonetic contexts. Each plot denotes the spectrogram of an utterance of the underlying word. For each spectrogram, the horizontal axis represents time, the vertical axis represents frequency, and the dark points represent the time-frequency locations with high energy level. The realization of “t” in the word “tea” is usually called the “canonical” realization. Compared with the “canonical” realization, the “t” in the word “tree” is retroflexed. In the word “steep”, the “t” is in a cluster with “s”, becoming unaspirated. The “t” in the word “city” is realized as a flap. In the word “beaten”, the “t” realized as a glottal stop.

resulting in a well-known data sparsity problem for training acoustic model parameters. To exploit the advantages of a context-dependent acoustic model, the data sparsity problem must be dealt with appropriately.

One commonly used approach to deal with the data sparsity problem is clustering [76]. By grouping a sufficient amount of context-dependent units into clusters, each cluster can have enough training examples to train a robust model (classifier) that provides scores for the cluster during recognition. When grouping the context-dependent units, a clustering algorithm typically encounters a trade-off between the context resolution and the number of training examples, as illustrated in Figure 1-3. For more than a decade, decision tree clustering [129, 58] has been used in almost every modern ASR system to select a good operating point in the trade-off.

Although decision tree clustering has been shown to be effective in dealing with the data sparsity problem and provides a reasonable context-dependent model, it still inevitably quantizes a certain number of contexts. The context-dependent units within a cluster will always have the same acoustic scores, and become acoustically indistinguishable to the recognizer. Such a quantization effect is not negligible. Take a conventional triphone-

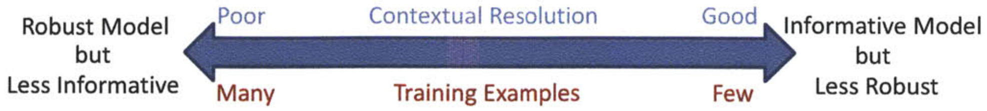


Figure 1-3: The trade-off between contextual resolution and the amount of training examples. If the clustering algorithm groups only a few context-dependent units into each cluster, the clusters can still preserve a variety of different contexts. However, some clusters might be less robust in response to new data due to a lack of training examples. On the other hand, if a cluster groups a lot of context-dependent units together, it can have more training examples to improve its generalization ability toward new data, but the resulting acoustic model cannot tell the recognizer much about the contexts.

based large-vocabulary ASR system for example, the number of clustered states is typically on the order of $10^3 \sim 10^4$. On the other hand, the number of valid triphone states can easily grow up to the order of 10^5 for large-vocabulary systems, which is a difference of about one or two orders of magnitude, compared with the number of clustered states. The quantization effect can potentially limit the discriminative power of the context-dependent model. If such an effect can be dealt with appropriately, the acoustic model performance can potentially be further improved.

1.1.2 Discriminative Training

Discriminative training is another commonly-used acoustic modeling technique which can refine model parameters and further reduce ASR errors. Instead of seeking model parameters that maximize the likelihood of the acoustic feature vectors given the labels, discriminative training methods seek parameters that minimize the degree of confusions occurring in the training data. Typically, the discriminative training procedure consists of two steps. First, construct a smooth and efficiently computable objective function that reflects the confusions. Second, update the model parameters to optimize the objective function.

Several types of objective functions have been proposed in the literature that have shown significant improvement on a variety of Large-Vocabulary Continuous Speech Recognition (LVCSR) tasks, including Minimum Classification Error (MCE) training [62, 86], Maximum Mutual Information (MMI) training [119], Minimum Phone Error (MPE) training [101], and their variants with margin-based [80, 130, 100, 87] modification. In terms of optimizing approaches, the gradient-based method [18, 86] and Baum-Welch-based method [7, 41, 8, 109, 101] are two commonly-used approach. More details about discriminative training methods can be found in Chapter 2.

Discriminative training has been shown to be effective in reducing ASR errors in a variety of tasks. Therefore, when designing a new modeling framework, it is desirable to make the new framework compatible with discriminative training to take advantage of its error reduction capability.

1.2 Proposed Approach

1.2.1 Motivation

The goal of this research is to develop an acoustic modeling technique that addresses the data sparsity problem while avoiding the quantization effect that potentially limits the performance of conventional context-dependent acoustic modeling frameworks. Not every context-dependent unit has enough training examples due to data sparsity, but can we still model different units differently? Can the new framework provide improvement over existing acoustic modeling frameworks? Will the new framework be compatible with commonly used error reduction techniques such as discriminative training methods to further improve ASR performance? To answer these questions, in this thesis, we propose a multi-level acoustic modeling framework that seeks to achieve the following goals:

- Address the data sparsity problem of context-dependent acoustic modeling while avoiding the quantization effect.

- Be compatible with state-of-the-art error reduction framework such as discriminative training methods

- Be able to be embedded into existing ASR systems, and improve the system performance on real word tasks.

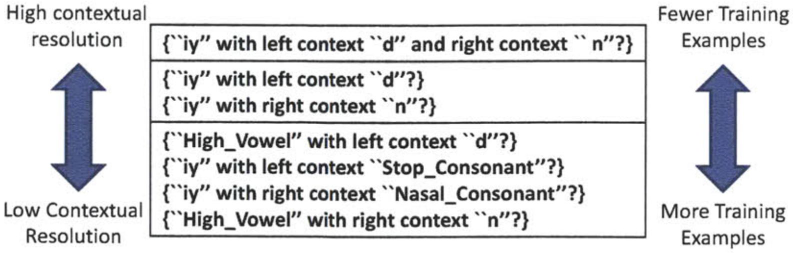

High contextual resolution
Low Contextual Resolution
{``iy" with left context ``d" and right context `` n"?}
{``iy" with left context ``d"?}
{``iy" with right context ``n"?}
{``High_Vowel" with left context ``d"?}
{``iy" with left context ``Stop_Consonant"?}
{``iy" with right context ``Nasal_Consonant"?}
{``High_Vowel" with right context ``n"?}
Fewer Training Examples
More Training Examples

Figure 1-4: Sets of questions that can identify the triphone "d-iy+n". Questions in a lower level set are less specific compared to questions in an upper level set, but the classifiers associated with questions in the lower sets can have more training examples. Except for the top set, each question in the lower level sets ignores certain contexts. However, when combined with other questions in the set, the fine-grained contexts can still be recovered.

1.2.2 Method Overview

To see how the proposed multi-level acoustic modeling works, we can think of scoring an acoustic feature vector, $\mathbf{x}$, with respect to a context-dependent unit as providing a soft decision to an identification question of the unit, e.g. {is $\mathbf{x}$ an "iy" with left context "d" and right context "n"?}. Although we can try to build a classifier that answers the question directly, sometimes such classifier might not be reliable due to the data sparsity problem. Instead of directly answering the specific question, we can try to answer more general questions such as: {is $\mathbf{x}$ an "iy" with left context "d"?} AND {is $\mathbf{x}$ an "iy" with right context "n"?}. If we answer "yes" to both questions, we can still identify the context-dependent unit as "d-iy+n". The two classifiers that correspond to this pair of questions can have more training examples, because each of them requires less contextual specification. We can arbitrarily tune the resolution of the contextual questions that can be asked, as illustrated in Figure 1-4.

Therefore, each context-dependent unit can be associated with classifiers that identify contexts at multiple levels of resolution. When given an input acoustic feature vector, the outputs of the classifiers can be combined for scoring. Since any pair of context-dependent units differ in some way, if the classifiers are judiciously selected based on the principle that the intersection of contexts can uniquely specify a particular context-dependent unit (as described in Figure 1-4), the pair of context-dependent units would have a least one different scoring classifier. In this way, each context-dependent unit can have a different score, and thus resolve the quantization effect. If certain context-dependent unit does not have enough training example, its corresponding bottom level classifiers can still have enough training examples and can be used for scoring. In this way, the data sparsity problem is dealt with.

Details of the multi-level modeling framework, including how it can be combined with discriminative training and be embedded in the existing ASR system, are discussed in Chapter 3. While a particular set of classifiers was used in this thesis, classifier selection is not limited to what was used in the experiments. The general principle that should be followed is to ensure that each context-dependent unit has a unique scoring mechanism to resolve quantization, and that each classifier used for scoring has enough training examples to avoid data sparsity. Thus, there are many possible classifier choices that could satisfy these criteria.

1.3 Contributions

- Proposed a novel context-dependent modeling framework that deals with both data sparsity and context quantization.

  - Designed a multi-level modeling framework that associates each context-dependent unit with classifiers identifying contexts at multiple levels of contextual resolution, and combine the classifier outputs for scoring. The proposed framework ensures any pair of context-dependent units have at least one differing scoring classifier to deal with quantization, and keeps classifiers with insufficient training data from being used for scoring to avoid data sparsity.

- Provided compatible implementation of the proposed multi-level framework for existing ASR systems and state-of-the-art error reduction frameworks.

  - Embedded the multi-level model in an existing Finite State Transducer (FST)[53] based recognizer, and provided general implementation guidelines for other types of ASR systems.

  - Integrated the multi-level modeling framework with discriminative training methods, and implemented training schemes such as computation of objective function and parameter update in a distributed computation paradigm.

- Improved accuracy of ASR system on large-vocabulary real word recognition tasks.

  - Provided overall more than 20% relative improvement in word error rate, compared with a clustering-based baseline, on a large-vocabulary speech recognition task of academic lectures, showing that the mutli-level framework integrated well with an existing large-vocabulary ASR system.

  - Provided about 15% relative improvement on a large-vocabulary speaker adaptation task, suggesting that the multi-level model has better adaptability to new data.

1.4 Thesis Outline

The organization of this thesis is as follows. Chapter 2 reviews mathematical formulation of modern ASR systems, context-dependent acoustic modeling, discriminative training methods, and finite state transducers. Chapter 3 introduces the formulation of the proposed multi-level model and its implementation, including how to integrate it with discriminative training, and merge it into an existing ASR decoding framework. Chapter 4 describes a phonetic recognition experiment on the TIMIT corpus. Chapter 5 reports the large-vocabulary speech recognition results on the MIT lecture corpus. Speaker adaptation experiment results are reported in Chapter 6, followed by the conclusion and future work in Chapter 7.

Chapter 2

Background:
Speech Recognition Techniques

In this chapter we review background materials related to this thesis. We start with the mathematical framework of modern Automatic Speech Recognition (ASR) systems, including model assumptions and decoding algorithms. Then we discuss the basic modeling components of an ASR system, including language models, lexicon/pronunication models, and acoustic models, and describe how to learn the model parameters. Mathematical methods that will be used or referred to in later parts of the thesis, including Maximum-Likelihood (ML) estimation, Lagrange multipliers, the Expectation-Maximization (EM) algorithm, and the Forward-Backward algorithm, will also be covered. After that, we discuss in more detail two commonly-used techniques for modern ASR systems, context-dependent acoustic modeling and discriminative training. Finally, we introduce Finite-State Transducers (FSTs) and show how they can be used to form a decoding module for ASR systems.

2.1 Automatic Speech Recognition

Given a speech utterance, the sequence of acoustic feature vectors $\mathbf{X}$ extracted from the waveform can be thought of as the observable data, and the word sequence $\mathbf{W}$ can be thought of as the underlying hypothesis to be inferred from the data. Following this notion, the decoding procedure of an ASR system can be formulated as finding the word sequence

$\mathbf{W}^*$ that maximizes the posterior probability $p(\mathbf{W}|\mathbf{X})$ under the Maximum A Posteriori (MAP) decision criterion [24]:

$$\begin{aligned} \mathbf{W}^* &= \arg\max_{\mathbf{W}} p(\mathbf{W}|\mathbf{X}) && (2.1)\\ &= \arg\max_{\mathbf{W}} \frac{p(\mathbf{X},\mathbf{W})}{p(\mathbf{X})} && (2.2)\\ &= \arg\max_{\mathbf{W}} p(\mathbf{X}|\mathbf{W})p(\mathbf{W}). && (2.3) \end{aligned}$$

The first term $p(\mathbf{X}|\mathbf{W})$ in Equation (2.3) is generally referred to as the likelihood of the data, and the second term $p(\mathbf{W})$ is referred to as the prior distribution by statisticians, or as the language model by researchers in speech related fields.

2.1.1 Model Assumptions and Approximations

Because both the speech utterance and the word sequence can be of variable length, modeling the full dependencies between the feature vectors and the words can be computationally intractable. For computational efficiency, certain model assumptions are typically applied to simplify computation.

For the prior distribution, it is typically assumed that the word sequence forms a generalized (higher order) finite-state Markov chain, where the joint probability of a word sequence can be decomposed as a product of the conditional probability of each word given a fixed length history. This approximation results in the well known n-gram language model [116], and the probability of a sequence of K words can be computed by

$$\begin{aligned} p(w_1, w_2, \dots, w_K) &= p(w_1)\prod_{i=2}^{K} p(w_i|w_{i-1}, \dots, w_1) && (2.4)\\ &= p(w_1)\prod_{i=2}^{n-1} p(w_i|w_{i-1}\dots w_1)\prod_{i=n}^{K} p(w_i|w_{i-1}, \dots, w_{i-n+1}), && (2.5) \end{aligned}$$

where the first equality is by chain rule and the second equality is based on the n-gram approximation. Note that the generalized Markov chain formed by the n-gram language model can be converted into a normal Markov chain by expanding the state space of the chain to incorporate the histories.

For the likelihood term, a Hidden Markov Model (HMM) [104] is typically used to provide a computationally efficient modeling framework. The HMM makes two main assumptions:

1. There exists a hidden state sequence, $\boldsymbol{S}$, that forms a Markov chain that generates the observation vectors.

2. Given the state s_t at any time point, the corresponding observation $\mathbf{x}_t$ is conditionally independent of other observations and states.

Under these assumptions, the decoding criterion becomes

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \sum_{\boldsymbol{S}} p(\mathbf{X}, \boldsymbol{S}|\mathbf{W})p(\mathbf{W}) \tag{2.6}$$

$$= \arg\max_{\mathbf{W}} \sum_{\boldsymbol{S}} \prod_{t} p(\mathbf{x}_t|s_t)p(\boldsymbol{S}|\mathbf{W})p(\mathbf{W}). \tag{2.7}$$

The hidden state sequence $\boldsymbol{S}$ is generally used to provide a sub-word/phonetic level representation of speech. In practice, each sub-word unit, e.g. a (context-dependent) phoneme, is modeled by a fixed number of states, and the lexical constraints provided by the lexicon can be embedded in the $p(\boldsymbol{S}|\mathbf{W})$ term. Under this formulation, the $p(\mathbf{x}_t|s_t)$ term provides a scoring mechanism of acoustic feature vectors with respect to the phonetic states and thus corresponds to the acoustic model.

In general, if there is a state sequence that dominates other sequences in a likelihood sense, the summation in Equation (2.7) can be replaced by a $\max$ and the decoding criterion can be approximated by jointly finding the best word and state sequence:

$$(\mathbf{W}^*, \boldsymbol{S}^*) = \arg\max_{\mathbf{W},\boldsymbol{S}} \prod_{t} p(\mathbf{x}_t|s_t)p(\boldsymbol{S}|\mathbf{W})p(\mathbf{W}). \tag{2.8}$$

The relation of the decoding criterion in Equation (2.8) with the components shown in Figure 1-1 can be interpreted as follows. For each acoustic feature $\mathbf{x}_t$, the Acoustic Model computes the likelihood of $\mathbf{x}_t$ being generated from each state s_t. The likelihoods from the Acoustic Model are jointly considered with the lexical constraints $p(\boldsymbol{S}|\mathbf{W})$ from the

Lexicon and the prior $p(\mathbf{W})$ from the Language Model by the Search Module to find the best scoring word sequence $\mathbf{W}^*$ and state alignment $\boldsymbol{S}^*$.

Since both the word sequence and the state sequence can be formulated as Markov chains under the model assumptions, we can further extend the state space such that every valid pair of $(\mathbf{W}, \boldsymbol{S})$ in the original space can be represented by a sequence S in the extended space. (This can be done with the composition operation that will be described in Section 2.4). As a result, the decoding process can be reduced to finding the best extended state sequence:

$$\begin{aligned} \mathrm{S}^* &= \arg\max_{\mathrm{S}} p(\mathbf{X}, \mathrm{S}) && (2.9) \\ &= \arg\max_{\mathrm{S}} \prod_t p(\mathrm{x}_t|\mathrm{s}_t) p(\mathrm{S}), && (2.10) \end{aligned}$$

which can be solved efficiently using a Viterbi algorithm [120] based on dynamic programming [11, 21].

2.1.2 Viterbi Decoding

The basic idea of the Viterbi decoding algorithm is that the optimal state sequence contains an optimal substructure under the HMM assumptions, and therefore can be found using a dynamic programming-based algorithm [104]. For any given time index t, the best state sequence that reaches a particular state s at time t has to pass one of the states s' at time $t-1$. Because of the Markov property, given the state at time $t-1$, how the chain behaves from time $t-1$ to time t is independent from how it behaved from time 1 to time $t-1$. Therefore, if the best sequence that reaches s at time t contains the state s' at time $t-1$, its prefix from time 1 to time $t-1$ must also be the best sequence that reaches s' at time $t-1$. Otherwise, a better path can be constructed by replacing the prefix with the best sequence that reaches s' at time $t-1$. Since the best sequence contains an optimal substructure, if at each time point we can remember the hub-state from the previous time point, when we reach the end of the utterance, we can trace back the hub-states to find the best state sequence.

Mathematically, given a sequence of T observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, for any time index t and state index s, we can define the best score of reaching s at time t

$$\delta(\mathbf{X}, t, s) = \begin{cases} p(\mathbf{x}_1, \mathrm{s}_1 = s), & \text{if } t = 1, \\ \max_{\{\mathrm{s}_1,\ldots,\mathrm{s}_{t-1}\}} p(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathrm{s}_1, \ldots, \mathrm{s}_{t-1}, \mathrm{s}_t = s), & \text{if } 2 \leq t \leq T. \end{cases} \tag{2.11}$$

With this definition, the joint probability of $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ and the best state sequence $\{\mathrm{s}_1^*, \ldots, \mathrm{s}_T^*\}$ in Equation (2.9) can be denoted by $\max_s \delta(\mathbf{X}, T, s)$. Using the conditional independence properties of HMM, for $2 \leq t \leq T$, we can rewrite $\delta(\mathbf{X}, t, s)$ in a recursive form:

$$\begin{aligned} \delta(\mathbf{X}, t, s) &= \max_{\{\mathrm{s}_1,\ldots,\mathrm{s}_{t-1}\}} p(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathrm{s}_1, \ldots, \mathrm{s}_{t-1}, \mathrm{s}_t = s) && (2.12) \\ &= \max_{\{\mathrm{s}_1,\ldots,\mathrm{s}_{t-1}\}} p(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathrm{s}_1, \ldots, \mathrm{s}_{t-1}) p(s|\mathrm{s}_{t-1}) p(\mathbf{x}_t|s) && (2.13) \\ &= \max_{s'} \max_{\{\mathrm{s}_1,\ldots,\mathrm{s}_{t-2}\}} p(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathrm{s}_1, \ldots, \mathrm{s}_{t-2}, \mathrm{s}_{t-1} = s') p(s|s') p(\mathbf{x}_t|s) && (2.14) \\ &= \max_{s'} \delta(\mathbf{X}, t-1, s') p(s|s') p(\mathbf{x}_t|s), && (2.15) \end{aligned}$$

where the second equality is from the conditional independence assumptions of HMM and the third equality holds from the optimal substructure argument.

In practice, to prevent underflow, instead of directly computing the joint probability, log-probability is computed. Let $\tilde{\delta}(\mathbf{X}, t, s) = \log(\delta(\mathbf{X}, t, s))$, and the recursive formula becomes

$$\tilde{\delta}(\mathbf{X}, t, s) = \begin{cases} \log(p(s)) + \log(p(\mathbf{x}_1|s)), & \text{if } t = 1, \\ \max_{s'} \tilde{\delta}(\mathbf{X}, t-1, s') + \log(p(s|s')) + \log(p(\mathbf{x}_t|s)), & \text{if } 2 \leq t \leq T. \end{cases} \tag{2.16}$$

To keep track of the best state sequence, we can use a set of back pointers

$$\tilde{\psi}(\mathbf{X}, t, s) = \arg\max_{s'} \tilde{\delta}(\mathbf{X}, t-1, s') + \log(p(s|s')) + \log(p(\mathbf{x}_t|s)), \tag{2.17}$$

for $2 \leq t \leq T$ and for each state s. Using $\{\tilde{\delta}(\mathbf{X}, t, s)\}$ and $\{\tilde{\psi}(\mathbf{X}, t, s)\}$ to keep track

Algorithm 2.1 Viterbi Decoding Algorithm

```
**Input**: Sequence of feature vectors $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$
**Output**: Best state sequence $\mathbf{S} = \{\mathrm{s}_1^*, \ldots, \mathrm{s}_T^*\}$

**for all** state $s$ **do**
  $\tilde{\delta}(\mathbf{X}, 1, s) \leftarrow \log(p(s)) + \log(p(\mathbf{x}_1|s))$
**end for**

**for** $t = 2 \rightarrow T$ **do**
  **for all** state $s$ **do**
    $\tilde{\delta}(\mathbf{X}, t, s) \leftarrow \max_{s'} \tilde{\delta}(\mathbf{X}, t-1, s') + \log(p(s|s')) + \log(p(\mathbf{x}_t|s))$
    $\tilde{\psi}(\mathbf{X}, t, s) \leftarrow \arg\max_{s'} \tilde{\delta}(\mathbf{X}, t-1, s') + \log(p(s|s')) + \log(p(\mathbf{x}_t|s))$
  **end for**
**end for**

$\mathrm{s}_T^* \leftarrow \arg\max_s \tilde{\delta}(\mathbf{X}, T, s)$
**for** $t = T - 1 \rightarrow 1$ **do**
  $\mathrm{s}_t^* \leftarrow \tilde{\psi}(\mathbf{X}, t+1, \mathrm{s}_{t+1}^*)$
**end for**
```

of the best current score and best previous state, the decoding can be done efficiently. A pseudo-code for how the decoding procedure works is described in Algorithm 2.1.

In practice, especially in real-time ASR applications, the decoder only keeps a subset of states as an active frontier at each time point, and only expands the search for the next time point from the active frontier. To decide the set of states in the active frontier, several pruning criteria, such as score differences from the current best score, are typically applied. The pruning criteria correspond to trade-off between accuracy and decoding speed, and thus may vary between applications.

2.1.3 Language Model

Parameters and Maximum-Likelihood Estimation

Given a vocabulary of size $\mathcal{V}$, the n-gram language model consists of a set of parameters $\boldsymbol{\theta} = \{\theta_v^h\}$ to represent the probability of seeing a word w_v given a word history h, where the parameters are subject to the constraint $\sum_{v=1}^{\mathcal{V}} \theta_v^h = 1$ for each history h. To estimate the model parameters, Maximum-Likelihood (ML) estimation [3] is typically used. For a given text corpus $\mathcal{D}$, the log probability of observing $\mathcal{D}$ under the set of parameters $\boldsymbol{\theta}$ can be computed by the following equation

$$\log(p_{\boldsymbol{\theta}}(\mathcal{D})) = \sum_{h \in \mathcal{H}^{\mathcal{D}}} \sum_{w_v} c(\mathcal{D}, hw_v) \log(\theta_v^h), \tag{2.18}$$

where $\mathcal{H}^{\mathcal{D}}$ is the set of n-gram histories used to express the word sequences in $\mathcal{D}$, and $c(\mathcal{D}, hw_v)$ is the number of times the word w_v occurred after h in the corpus. The Maximum-Likelihood (ML) estimation seeks to find the language model parameters $\boldsymbol{\theta}$ that maximizes Equation (2.18).

Lagrange Multipliers

In addition to maximizing the log-likelihood expression in Equation (2.18), the language model parameters have to satisfy the sum-to-one constraint $\sum_{v=1}^{\mathcal{V}} \theta_v^h = 1$ for each history h appearing in the training corpus. To deal with the sum to one constraint for each history, the Lagrange Multiplier [12] method can be applied, and the optimization criterion becomes

$$\max_{\{\theta_v^h\}} \min_{\{\xi_h\}} \sum_{h \in \mathcal{H}^{\mathcal{D}}} \sum_{w_v} c(\mathcal{D}, hw_v) \log(\theta_v^h) + \sum_{h \in \mathcal{H}^{\mathcal{D}}} \xi_h (\sum_{v=1}^{\mathcal{V}} \theta_v^h - 1), \tag{2.19}$$

where ξ_h is the Lagrange multiplier for the word history h. Note that for a certain h, if $\sum_{v=1}^{\mathcal{V}} \theta_v^h < 1$, the minimizer can set ξ_h to $-\infty$ to make the objective function $-\infty$, and, similarly, it can set ξ_h to ∞ if $\sum_{v=1}^{\mathcal{V}} \theta_v^h > 1$. In this way, maximizer has to choose a set of parameters that satisfies all the sum to one constraints.

To find the optimal value, we can take the partial derivatives of Equation (2.19) with respect to $\{\theta_1^h, \theta_2^h \ldots, \theta_{\mathcal{V}}^h\}$ for each h and make the derivatives equal to zero:

$$\frac{c(\mathcal{D}, hw_v)}{\theta_v^h} + \xi_h = 0 \qquad \forall v \in \mathcal{V}. \tag{2.20}$$

Solving the system of equations in Equation (2.20) gives the optimal value under ML estimation:

$$\hat{\theta}_v^h = \frac{c(\mathcal{D}, hw_v)}{c(\mathcal{D}, h)}, \tag{2.21}$$

where $c(\mathcal{D}, h) = \sum_{v=1}^{\mathcal{V}} c(\mathcal{D}, hw_v)$ is the number of occurrences of h in $\mathcal{D}$.

From Equation (2.21), we can see that the ML estimation assigns zero to the probability $p(w_v|h)$ if the sequence hw_v never occurs in the training corpus $\mathcal{D}$. This fact makes the ML estimated model unable to generalize to new data. To ensure generalization, it is necessary to smooth the model parameters; that is, to assign some non-zero probabilities to the unseen n-grams. Many effective ways of assigning the probability mass have been proposed in the literature, including Good-Turing smoothing [64, 40], Kneser-Ney smoothing [66], and Modified Kneser-Ney [17] smoothing.

In general, if the language model is trained on text data related to the recognition task, the ASR performance is better than a model trained on generic texts [56]. If the recognition task covers a variety of possible topics, adapting the language model to the topic of the speech can provide some improvement [67, 111, 51, 54]. If an ASR system is to be deployed in a domain with limited amount of data, appropriate interpolation of language models from partially related domain may be required [55].

2.1.4 Lexicon/Pronunciation Model

Compared with other modeling components of an ASR system, the lexicon/pronunciation model is probably the component that requires the most knowledge from human experts. In a conventional ASR system, an expert-designed lexicon is typically used to decompose the words into sequences of sub-word units which are usually language-specific phonemes. Figure 2-1 shows some examples of mappings between words and sequences of phonemes. Although the canonical pronunciations provided by an expert-designed lexicon cover a majority of how the words are pronounced in actual speech, it still has limitations. Two main problems that an expert lexicon might have difficulties to deal with are pronunciation variation and Out-Of-Vocabulary (OOV) words.

Pronunciation variation can result from many causes [117]. For example, interaction between words in continuous speech can cause assimilation, reduction, deletion, and insertion of phonemes [131]. Also, a speaker may change speaking style and/or word pronunciations based on situations (formal/informal) and audience. In addition to intra-speaker

automatic	:	ao t ax m ae t ax k
speech	:	s p iy ch
recognition	:	r eh k ax g n ih sh ax n

Figure 2-1: Examples of lexical mappings from words to phonemes.

variation, regional differences such as dialects and accents can also cause a large degree of inter-speaker variation. To deal with coarticulatory variations, pronunciation rules designed based on acoustic-phonetic knowledge [131, 52] and context-dependent acoustic modeling can be used. For style and inter-speaker variation, learning a probabilistic pronunciation model from examples [5, 4] might be a potential solution.

Out-of-vocabulary (OOV) words can harm ASR performance in several ways. The recognizer might generate a sequence of short words in the lexicon to represent the OOV word, resulting in extra insertion errors. The recognizer can also modify nearby words around the OOV in the recognition output, resulting in extra substitution errors. One way to mitigate the problem is to detect OOVs and represent it with a special label in the recognition output [10, 48, 9]. Another idea is to have a systematic way of learning pronunciations for new words to increase the vocabulary coverage [19]. Building hybrid systems [13, 123] that are able to represent the OOVs with sequences of letter-to-sound, grapheme-based, sub-word units can also help reduce the effect of OOVs.

In addition to dealing with pronunciation variations and OOVs, another open question is whether we can exclude the expert knowledge when constructing the mapping between words and sub-word units. A solution to this question would be very helpful to develop ASR systems for languages with limited knowledge sources. While grapheme-based recognition systems have been shown to be effective for certain languages [65, 118], it remains open that is there a universal approach across all types of languages.

2.1.5 Acoustic Model

The acoustic model provides a scoring mechanism of the acoustic observations with respect to the set of states. Specifically, given an acoustic observation, $\mathbf{x}_t$, the acoustic model produces a conditional probability of seeing $\mathbf{x}_t$, $p(\mathbf{x}_t|s)$, for each state s. In most modern ASR systems, such scoring is done by a Gaussian Mixture Model (GMM). Given a feature vector $\mathbf{x}$ and a state label s, the log-likelihood of $\mathbf{x}$ with respect to s can be computed by

$$l_{\boldsymbol{\lambda}}(\mathbf{x}, s) = \log(\sum_{m=1}^{M_s} \omega_m^s \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m^s, \boldsymbol{\Sigma}_m^s)), \tag{2.22}$$

where the mixture weight, ω_m^s, is subject to the constraint $\sum_{m=1}^{M_s} \omega_m^s = 1$, and the term $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m^s, \boldsymbol{\Sigma}_m^s)$ is the multivariate Gaussian density function with mean $\boldsymbol{\mu}_m^s$, and covariance matrix $\boldsymbol{\Sigma}_m^s$, as described in Equation (A.1). In most ASR systems, the covariance matrices are assumed to be diagonal for computational efficiency and for ease of parameter estimation.

Before we show how to perform ML estimation for Gaussian Mixture Model (GMM) parameters for each Hidden Markov Model (HMM) state from a sequence of acoustic observations, we first consider a simpler case: given a set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}$ independently drawn from a GMM of M mixture components, how can we estimate the GMM parameters under the ML criterion? To see how to estimate the parameters, we have to first understand how a vector is drawn from a GMM.

Under the assumptions of the GMM, a vector is drawn as follows. First, pick a mixture component m with probability ω_m, and then draw a random vector from the Gaussian distribution with mean $\boldsymbol{\mu}_m$, and variance $\boldsymbol{\Sigma}_m$. If, in addition to the vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_K\}$, the indices of the mixture components $\{m_1, m_2, \ldots, m_K\}$ from which each vector was drawn were also recorded, then we can compute the log-likelihood of the complete data; that is, the log-likelihood of seeing both the vectors and the component indices by

$$\log(p_{\boldsymbol{\lambda}}(\mathbf{X}, \boldsymbol{\mathcal{M}})) = \sum_{k=1}^{K} \log(\omega_{m_k}) \log(\mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_{m_k}, \boldsymbol{\Sigma}_{m_k})). \tag{2.23}$$

Note that for each mixture component m, the mean $\boldsymbol{\mu}_m$ and variance $\boldsymbol{\Sigma}_m$ are only related to the set of vectors that are drawn from the mixture m. As a result, we can collect the set of vectors and use the ML estimation formula in Appendix A to find the optimal values of $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m$ that maximize Equation (2.23). To get the optimal mixture weight, we can count the number of occurrences of each component and divide the count by the total number of vectors K, just as we did for the language model.

However, in many applications, the indices of the mixture components are typically not available from the data. In such a incomplete data scenario, each vector can potentially be drawn from any of the mixture components. As a result, to compute the log-likelihood of seeing the vectors, we have to marginalize all possible assignments of mixture components, and the log-likelihood of the data $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_K\}$ can be computed by

$$\log(p_{\boldsymbol{\lambda}}(\mathbf{X})) = \sum_{k=1}^{K} \log(\sum_{m=1}^{M} \omega_m \mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)). \tag{2.24}$$

However, unlike the log-likelihood of the complete data, the log-likelihood of the form in Equation (2.24) does not have a closed-form optimal solution, so an iterative optimization procedure is required.

Expectation-Maximization (EM) Algorithm

The Expectation-Maximization (EM) algorithm [25] is an iterative procedure commonly used to perform Maximum-Likelihood (ML) estimation for models under which the computation of the log-likelihood of the observed data involves marginalization of hidden variables. For such models, the log-likelihood of seeing both the data and the true assignment of the hidden variables is much easier to optimize than the log-likelihood of seeing just the observable data. Since the true assignment of the hidden variables is unknown, the EM algorithm first uses the data and the current model parameters to compute the posterior probability of each particular assignment being the true assignment of the hidden variables. After the posterior distribution of the assignments is computed, the EM algorithm then uses the distribution to compute the expectation of the completed log-likelihood of the data and the assignment of the hidden variable. The expectation is then used as an auxiliary function for optimization. Since the auxiliary function is a linear combination of a set of functions that are relatively easy to optimize, the auxiliary function itself is also relatively easy to optimize. Also, as shown in Appendix B, the auxiliary function used by the EM algorithm is a strong sense auxiliary function [99], and therefore optimizing the auxiliary function guarantees an improvement of the log-likelihood until the iterative procedure converges.

To summarize, each iteration of the EM algorithm consists of two steps: an Expectation step (E-step) and a Maximization step (M-step). The procedures of the two steps can be summarized as follows:

- E-step: Compute the posterior probabilities of hidden variables and sufficient statistics that are required to optimize the auxiliary function, based on the data and the current model parameters.
- M-step: Update the model parameters to the optimal point of the current auxiliary function using the sufficient statistics from the E-step.

When estimating the GMM parameters from independently drawn vectors, the posterior probabilities that need to be computed in the E-step are

$$r_{km} = \frac{\omega_m \mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_m, \Sigma_m)}{\sum_{m'=1}^{M} \omega_{m'} \mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_{m'}, \Sigma_{m'})}, \quad \forall k, m, \tag{2.25}$$

where the numerator is the likelihood of seeing $\mathbf{x}_k$ drawn from the mixture component m, and the denominator is the likelihood of seeing $\mathbf{x}_k$. Note that although the posterior probability r_{km} depends on $\mathbf{x}_k$ and the current GMM parameters, it is treated as a constant during the M-step of the EM algorithm. Therefore, we do not attach notations related to $\mathbf{x}_k$ and the GMM parameters to the posterior probability to stress that they are treated as constants during maximization.

After the posterior probabilities are computed, the auxiliary function can be expressed by

$$\sum_{k=1}^{K} \sum_{m=1}^{M} r_{km}[\log(\omega_m) + \log(\mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_m, \Sigma_m))]. \tag{2.26}$$

Note that the terms related to $\boldsymbol{\mu}_m$ in Equation (2.26) can be expressed by

$$\sum_{k=1}^{K} r_{km} \log(\mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_m, \Sigma_m)), \tag{2.27}$$

which is similar to Equation (A.2) except for the scaling coefficients $\{r_{km}\}$. Since taking a partial derivative is a linear operation, we can first compute the partial derivative of $\log(\mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m))$ for each $\mathbf{x}_k$, as was done in Appendix A, and then multiply by $\{r_{km}\}$. As a result, the best $\boldsymbol{\mu}_m$ that maximizes Equation (2.26) satisfies the condition $\sum_{k=1}^{K} r_{km}(\mathbf{x}_k - \boldsymbol{\mu}_m) = \mathbf{0}$, and therefore can be computed by

$$\hat{\boldsymbol{\mu}}_m = \frac{\sum_{k=1}^{K} r_{km}\mathbf{x}_k}{\sum_{k=1}^{K} r_{km}}. \tag{2.28}$$

Similarly, the ML estimate of the covariance matrix $\boldsymbol{\Sigma}_m$ can be computed by

$$\hat{\boldsymbol{\Sigma}}_m = \frac{\sum_{k=1}^{K} r_{km}\mathbf{x}_k\mathbf{x}_k^{\mathrm{T}}}{\sum_{k=1}^{K} r_{km}} - \hat{\boldsymbol{\mu}}_m\hat{\boldsymbol{\mu}}_m^{\mathrm{T}}. \tag{2.29}$$

For the mixture weight, we can think of $\{r_{km}\}$ as partial counts and divide the sum of partial counts by the total number of vectors. As a result, the optimal weight that maximizes Equation (2.26) becomes

$$\hat{\omega}_m = \frac{\sum_{k=1}^{K} r_{km}}{K}. \tag{2.30}$$

To compute the quantities required in Equations (2.28-2.30), we can accumulate a set of sufficient statistics for each mixture component m during the E-step:

$$\vartheta_m = \sum_{k=1}^{K} r_{km}, \tag{2.31}$$

$$\mathcal{O}_m(\mathbf{X}) = \sum_{k=1}^{K} r_{km}\mathbf{x}_k, \tag{2.32}$$

$$\mathcal{O}_m(\mathbf{X}^2) = \sum_{k=1}^{K} r_{km}\mathbf{x}_k\mathbf{x}_k^{\mathrm{T}}. \tag{2.33}$$

During the M-step, the sufficient statistics can be used to update the model parameters for the next EM iteration. While the EM algorithm provides an iterative procedure of improving the model parameters in the Maximum-Likelihood sense, it does not tell how to initialize the GMM parameters. To initialize the GMM parameters, the K-means algorithm [83] is a commonly-used method. Another common approach is to start with one component and split one component[1] at a time after several EM iterations, until the desired number of components is reached, as was done in [128].

[1] Typically, the one with largest variance, but there are some implementations that split the component with largest mixture weight.

Forward-Backward Algorithm

So far we have shown how to do ML estimation for GMM parameters from a set of independently drawn vectors. Estimating the GMM parameters for each state under the HMM framework involves a few more steps, for the following reasons:

1. Under HMM assumptions, the observed vectors are not independent unless the underlying state sequence is given.

2. The states are not independent but are governed by a Markov chain.

Although it is more complex than the example we have gone through, we can still apply the EM algorithm to estimate the parameters. Because a feature vector is conditionally independent from everything else if given its underlying state, the key is finding the posterior distribution of the states at any given time point.

The Forward-Backward algorithm [104, 6] is a commonly-used algorithm to find the posterior probability of each state at any time point, given a sequence of observation vectors under a HMM framework. The Forward-Backward algorithm consists of a forward procedure and a backward procedure. Given a sequence of observation vectors $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ and the current model parameter set $\boldsymbol{\lambda}$, the forward procedure computes and records the forward probability

$$\alpha_{\boldsymbol{\lambda}}(\mathbf{X}, t, s) = p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathrm{s}_t = s), \tag{2.34}$$

the probability of seeing the observation up to the time point t and reaching state s, for each time point t and state s. Under the HMM assumptions, the forward probabilities can be computed efficiently using the following recursive formula

$$\begin{aligned}
\alpha_{\boldsymbol{\lambda}}(\mathbf{X}, t+1, s) &= \sum_{s'} p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{x}_{t+1}, \mathrm{s}_t = s', \mathrm{s}_{t+1} = s) \\
&= \sum_{s'} p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathrm{s}_t = s') p_{\boldsymbol{\lambda}}(\mathrm{s}_{t+1} = s | \mathrm{s}_t = s') p_{\boldsymbol{\lambda}}(\mathbf{x}_{t+1} | \mathrm{s}_{t+1} = s) \\
&= \sum_{s'} \alpha_{\boldsymbol{\lambda}}(\mathbf{X}, t, s') p_{\boldsymbol{\lambda}}(\mathrm{s}_{t+1} = s | \mathrm{s}_t = s') p_{\boldsymbol{\lambda}}(\mathbf{x}_{t+1} | \mathrm{s}_{t+1} = s). \qquad (2.35)
\end{aligned}$$

The backward procedure computes and records the backward probability

$$\beta_{\boldsymbol{\lambda}}(\mathbf{X}, t, s) = p_{\boldsymbol{\lambda}}(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T | \mathrm{s}_t = s), \tag{2.36}$$

the probability of seeing the remaining observations given that the chain reaches state s at time t. Similar to the forward probabilities, the backward probabilities can also be computed recursively. To initialize the recursion, we set $\beta_{\boldsymbol{\lambda}}(\mathbf{X}, T, s) = 1, \forall s$. Given the backward probabilities at time t, the backward probabilities at time t-1 can be computed by

$$\begin{aligned}\beta_{\boldsymbol{\lambda}}(\mathbf{X}, t-1, s) &= \sum_{s'} p_{\boldsymbol{\lambda}}(\mathbf{x}_t, \mathrm{s}_t = s', \mathbf{x}_{t+1}, \ldots, \mathbf{x}_T | \mathrm{s}_{t-1} = s) \\ &= \sum_{s'} p_{\boldsymbol{\lambda}}(\mathrm{s}_t = s' | \mathrm{s}_{t-1} = s) p_{\boldsymbol{\lambda}}(\mathbf{x}_t | \mathrm{s}_t = s') p_{\boldsymbol{\lambda}}(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T | \mathrm{s}_t = s') \\ &= \sum_{s'} p_{\boldsymbol{\lambda}}(\mathrm{s}_t = s' | \mathrm{s}_{t-1} = s) p_{\boldsymbol{\lambda}}(\mathbf{x}_t | \mathrm{s}_t = s') \beta_{\boldsymbol{\lambda}}(\mathbf{X}, t, s'), \end{aligned} \tag{2.37}$$

where the second equality holds because given s_t, $\{\mathbf{x}_t, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_T\}$ are conditionally independent of s_{t-1}.

Using the forward and backward probabilities, we can compute the posterior probability for each state s at time t as

$$\begin{aligned}\gamma_{ts} = p_{\boldsymbol{\lambda}}(\mathrm{s}_t = s | \mathbf{X}) &= \frac{p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathrm{s}_t = s, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_T)}{p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_T)} \\ &= \frac{p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathrm{s}_t = s) p_{\boldsymbol{\lambda}}(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T | \mathrm{s}_t = s)}{p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_T)} \\ &= \frac{\alpha_{\boldsymbol{\lambda}}(\mathbf{X}, t, s) \beta_{\boldsymbol{\lambda}}(\mathbf{X}, t, s)}{\sum_{s'} \alpha_{\boldsymbol{\lambda}}(\mathbf{X}, t, s') \beta_{\boldsymbol{\lambda}}(\mathbf{X}, t, s')}, \end{aligned} \tag{2.38}$$

where the second equality is from the fact that given s_t, $\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$ and $\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T\}$ are conditionally independent, and the third equality is from the definitions of the forward and backward probabilities, and from the law of total probability (for the denominator). After the posterior probabilities are computed, we can collect the sufficient statistics for each mixture component m of each state s

$$\vartheta_{sm} = \sum_{t=1}^{T} \gamma_{ts} r_{tm}^{s}, \tag{2.39}$$

$$\mathcal{O}_{sm}(\mathbf{X}) = \sum_{t=1}^{T} \gamma_{ts} r_{tm}^{s} \mathbf{x}_t, \tag{2.40}$$

$$\mathcal{O}_{sm}(\mathbf{X}^2) = \sum_{t=1}^{T} \gamma_{ts} r_{tm}^{s} \mathbf{x}_t \mathbf{x}_t^{\mathrm{T}}, \tag{2.41}$$

where r_{tm}^{s} can be computed using Equation (2.25).

For each utterance in the training set, we can compute the posterior probabilities using the Forward-Backward algorithm, and accumulate statistics in Equation (2.39-2.41) per utterance. After the statistics are summed up, we can update the GMM parameters for each mixture m of each state s by

$$\omega_m^s = \frac{\vartheta_{sm}}{T}, \tag{2.42}$$

$$\hat{\boldsymbol{\mu}}_m^s = \frac{\mathcal{O}_{sm}(\mathbf{X})}{\vartheta_{sm}}, \tag{2.43}$$

$$\hat{\boldsymbol{\Sigma}}_m^s = \frac{\mathcal{O}_{sm}(\mathbf{X}^2)}{\vartheta_{sm}} - (\hat{\boldsymbol{\mu}}_m^s)(\hat{\boldsymbol{\mu}}_m^s)^{\mathrm{T}}, \tag{2.44}$$

where T is the total number of observation vectors in the training set. The update formulas in Equations (2.42-2.44) are also known as Baum-Welch updates [8, 99].

In practice, the reference word sequence $\mathbf{Y}$ corresponding to each training utterance is also given, and therefore the objective function to be optimized should be $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}, \mathbf{Y}))$, the log-likelihood of seeing both the acoustic observations and the word sequence (rather than $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}))$, the log-likelihood of seeing just the acoustic observations). To optimize such an objective function, the EM algorithm and the Forward-Backward algorithm can be still applied, but the state space should be constrained such that it only allows state sequences that correspond to the word sequence. Typically, the set of allowable states and transitions at each time point can be represent efficiently by a lattice as shown in Figure 2-2. The Forward-Backward algorithm can still be used, but instead of considering all states and transitions, only the valid states and transitions provided by the lattice are considered.

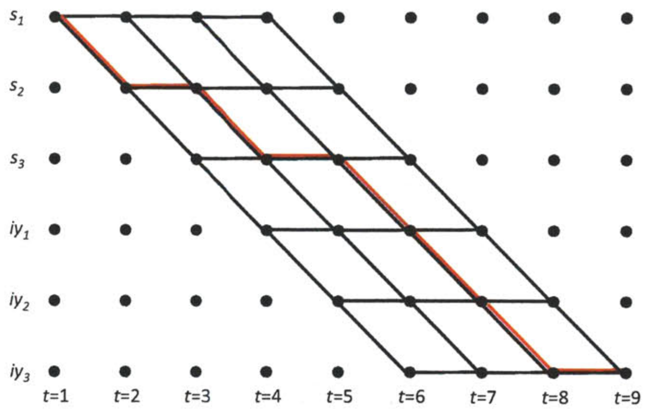


Figure 2-2: An example lattice of an utterance of "see" with 9 acoustic observations. The word "see" is represented by a sequence of two phonemes "s" and "iy", where each phoneme is modeled by a 3-state sequential HMM. The dots represent the states at each time point, and the black line segments represent the allowable transitions in the lattice. Given the word constraint, the allowable state at $t = 1$ should be the first state of "s", and the allowable state at $t = 9$ should be the last state of "iy". The red line segments form a valid state sequence under the word level constraint.

One final remark concerning parameter learning is that while the transitions between basic sub-word units are mainly governed by the lexicon and language model, the state transition probabilities within a sub-word unit can also be estimated from the observation vectors using the Forward-Backward algorithm. Let $a_{s's}$ be the transition probability from state s' to s. At each time point $t > 1$, we can compute the posterior probability of being at state s' at time t-1 and state s at time t by

$$\varsigma_{ts's} = \frac{p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathrm{s}_{t-1} = s', \mathrm{s}_t = s, \mathbf{x}_t, \ldots, \mathbf{x}_T)}{p_{\boldsymbol{\lambda}}(\mathbf{x}_1, \ldots, \mathbf{x}_T)} \tag{2.45}$$

$$= \frac{\alpha_{\boldsymbol{\lambda}}(\mathbf{X}, t-1, s') a_{s's} p_{\boldsymbol{\lambda}}(\mathbf{x}_t|s) \beta_{\boldsymbol{\lambda}}(\mathbf{X}, t, s)}{\sum_z \alpha_{\boldsymbol{\lambda}}(\mathbf{X}, t, z) \beta_{\boldsymbol{\lambda}}(\mathbf{X}, t, z)}. \tag{2.46}$$

After the posterior probability at each time point is computed, the transition probability can be updated similarly to the mixture weights by

$$\hat{a}_{s's} = \frac{\sum_{t=1}^{T} \varsigma_{ts's}}{T}. \tag{2.47}$$

2.2 Context-Dependent Acoustic Models

2.2.1 Overview

Given an acoustic feature vector $\mathbf{x}$, the acoustic model predicts how likely $\mathbf{x}$ belongs to a phonetic state, s, and provides scores to identity questions such as whether $\mathbf{x}$ belongs to the first state of the phoneme "iy". A context-dependent acoustic model seeks to extend the prediction and jointly predicts the current phonetic identity of $\mathbf{x}$ and its nearby phonetic contexts; for example, if $\mathbf{x}$ belongs to the first state of an "iy" with left context "d" and right context "n", as in the word "dean".

Considering acoustic-phonetic contexts can provide potential advantages from several perspectives. First, it can provide a modeling framework to model coarticulatory variations as shown in Figure 1-2 and phonological effects such as the gemination of consonants, as in the phrase "gas station". Second, the acoustic feature vector extracted at a given time point typically incorporates temporal information from nearby regions. Considering context can potentially better utilize temporal information captured by the feature vectors and improve ASR performance. Third, because the nearby predictions made by the context-dependent model must match in context, it can provide acoustic-level constraints that can potentially help refine the search. Context-dependent acoustic modeling has been successful in reducing ASR errors since it was proposed [110, 20, 75, 76], and it has now become a standard modeling component for almost all modern ASR systems.

2.2.2 Notation

To model the acoustic-phonetic contexts, the basic sub-word units used by a recognizer are expanded such that the contextual information can be incorporated. For example, a diphone unit incorporates the identities of the current phonetic label and the previous phonetic label (left context); a triphone unit incorporates the current label, the previous label, and the next label (right context); a quinphone unit incorporates the current label, the previous two labels, and the next two labels. Note that an "iy" with left context "s" in the word "see" is different from an "s" with right context "iy" (as in "see"). To show the differences, the left

word sequence	see you
phoneme sequence	s iy y uw
diphone sequence	sil-s s-iy iy-y y-uw
triphone sequence	sil-s+iy s-iy+y iy-y+uw y-uw+sil
quinphone sequence	<>-sil-s+iy+y sil-s-iy+y+uw s-iy-y+uw+sil iy-y-uw+sil+<>

Figure 2-3: Example of representing word sequence with context-dependent phonetic units. The label "sil" denotes silence and the label "<>" denotes the sentence boundary.

context is typically attached with a minus "-" symbol, and right context is attached with a "+" symbol in the representation of context-dependent units [129]. Figure 2-3 shows an example of representing a word sequence using context-dependent units. Note that the sequential context-dependent units for a word sequence must match in the context. For a HMM based ASR system, such constraint can be maintained by allowing transitions from one context-dependent unit to another only when the corresponding contexts agree.

2.2.3 Data Sparsity Problem

Although modeling the phonological contexts has many potential advantages, exploits such advantages requires the context-dependent acoustic model must be able to provide a reliable score for each context-dependent state. Generating reliable scores for the states, however, is not trivial due to data sparsity. Take an ASR system with 60 basic phonetic units for example, it can have $60^2 = 3,600$ possible diphones and $60^3 = 216,000$ triphones, and each context-dependent phonetic unit can have multiple HMM states. While lexical constraints can rule out a significant number of units, it is still very common that a large-vocabulary triphone-based ASR system can have about 10^5 triphone states. Because the number of context-dependent states can grow very large, many context-dependent states may have a limited amount of training data to train a reliable model (classifier) for scoring. To leverage the advantages of context-dependent modeling, the data sparsity problem has to be dealt with. In the following, we review several approaches proposed in the literature to deal with the data sparsity problem.

2.2.4 Addressing Data Sparsity

Clustering-Based Approach

The basic idea of using clustering to deal with data sparsity is that by grouping sufficient numbers of context-dependent units into clusters, each cluster can have enough training examples to train a reliable scoring model. In terms of clustering method, hierarchical clustering algorithms [46] that can keep track of the history of how the clusters are formed are typically used for ease of model analysis and parameter tuning. Among hierarchical clustering algorithms, the (top-down) decision tree clustering [129, 58] is commonly used in that the algorithm has a closed-form solution at each step and that the cluster of an unseen unit can be found easily by walking down the decision tree. More details of the decision tree clustering algorithm are described in Section 2.2.5.

Reduction-Based Approach

The basic idea of the reduction-based approach is to reduce the problem of modeling a longer context-dependent unit into a problem of modeling the composition of a set of shorter units. Shorter units can have more training examples, and thus the data sparsity problem can be mitigated. One example of the reduction based approach is the quasi-triphone modeling proposed in [82]. In the quasi-triphone approach, the HMM states of a triphone are decomposed into a left context sensitive diphone state at the beginning, several context independent states in the middle, and a right context sensitive diphone state at the end. Another way of decomposing a triphone is to use the Bayesian approach proposed in [90]. Under a Bayesian assumption, a triphone can be represented by a product of two diphone probabilities divided by the monophone probability of the center unit. Using reduction and composition, it can mitigate the sparsity problem. However, if some reduced units still do not have enough training examples, and certain degree of clustering might still needed.

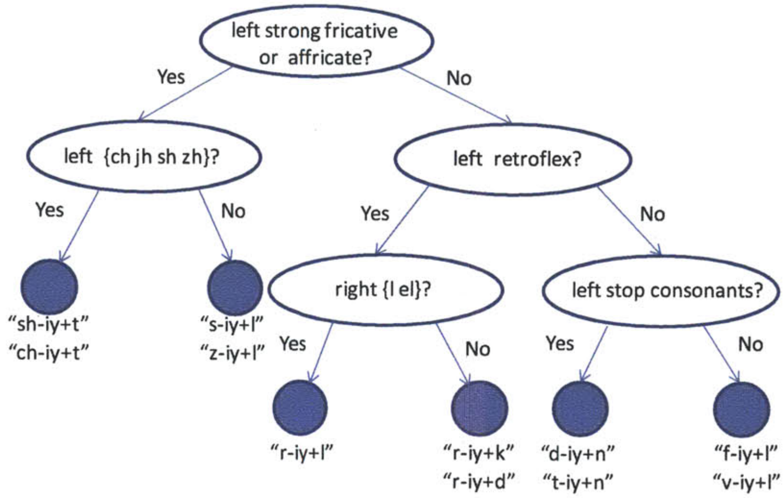


Figure 2-4: Example of clustering the contexts of the high vowel "iy" using the decision tree clustering algorithm.

2.2.5 Decision Tree Clustering

The decision tree clustering algorithm utilizes a set of manually constructed binary questions based on acoustic-phonetic knowledge as potential ways to split the data. The algorithm starts with a single root node, meaning all the units are grouped into one cluster. At each step of the algorithm, a leaf node of the tree is picked, and a binary question is applied to split the node into two. The node and the question are selected such that the objective function of the clustering can be maximized. The algorithm continues until a stopping criterion is reached. Appendix D describes the mathematical details of how to select the best splitting question at each step.

Figure 2-4 is an example of clustering the contexts of the phoneme "iy" using the decision tree clustering. After the clustering finishes, the data corresponding to each leaf node are collected to train a GMM for the scoring of the context-dependent units within the leaf node during decoding. Since each context-dependent phonetic unit can have multiple states, the clustering can be done at the state level (e.g, growing a decision tree for each state of "iy"). Experiment results in [129] showed that doing state-level clustering can yield slightly better performance.

Related Methods

Decision tree clustering has been shown to be effective in producing reasonable clusters for context-dependent acoustic models on various tasks, and therefore is commonly used as a standard procedure when building an ASR system. In the literature, several methods have been proposed for decision tree-based modeling. We review some of them as follows.

Instead of splitting nodes using the best splitting question at each step, the Random tree method [115] picks a random question from the set of top ranking questions. The randomization can produce multiple trees to create different context-dependent acoustic models on which multiple ASR systems can be built. During recognition time, the outputs from different systems can be combined using methods such as ROVER [27] to further reduce the recognition errors. However, because of the randomness, the improvement of the Random tree method is not always stable.

Instead of trying to create multiple trees, another idea is how can we better utilize an existing tree. When growing the decision tree, if the splitting stops earlier, each cluster will have more training examples, and the classifiers trained from the clusters may be more robust and generalizable to new data. On the other hand, if the tree grows deeper, the clusters can maintain more context information for the recognizer. For each context-dependent unit, if it can be associated with both types of classifiers for scoring, the recognition accuracy can be improved, as shown in the hierarchical acoustic modeling framework proposed in [16].

2.2.6 Limitations of Current Approaches

While the methods described above deal with the data sparsity problem to some degree, each of them has certain limitations. In this section, we first summarize the limitations of the existing methods, and then we discuss the properties we would like to see in the proposed context-dependent acoustic modeling framework.

The main limitation of the clustering-based approach is the quantization effect. The context-dependent states clustered together in a leaf node will always have the same acoustic scores, making them acoustically indistinguishable to the recognizer. The quantization effect is not negligible. In a conventional triphone based ASR system, the number of clustered states are in the range of $10^3 \sim 10^4$, and compared with the number of triphone states, the difference is of one or two orders of magnitude. To exploit the advantages of context-dependent acoustic modeling, such quantization effect should be appropriately addressed.

Another limitation of the clustering-based approach is that the sharing of data is constrained by the clustering result. In the decision tree example of Figure 2-4, the data for "r-iy+l" might be helpful in building a model for "z-iy+l", but the decision tree does not allow such kind of data sharing. The improvement shown in the Random tree method [115] suggests that having different ways of sharing the data might be beneficial. The improvement shown in the hierarchical acoustic modeling method [16] suggests that grouping data with multiple levels of granularity might be helpful.

For reduction-based approaches, although it is possible to address the data sparsity problem while keeping the context-dependent states distinguishable from each other, a pure reduction-based approach does not consider the fact that if some context-dependent units have many occurrences in the training data, it may be beneficial to incorporate a classifier that directly models the occurrences of the context-dependent state as a part of the modeling framework.

In sum, the main desired properties of a context-dependent acoustic model can be described as follows:

- Address both the data sparsity problem and the quantization effect.
- Allow flexible sharing of data.
- When enough data are available, enable models to target the finest-level of context resolution.

Also, it would be beneficial if the modeling framework can be compatible with state-of-the-art error reduction methods such as discriminative training to exploit their error reduction capability.

2.3 Discriminative Training

2.3.1 Goal

In Section 2.1.5, we introduced how to perform ML estimation for acoustic model parameters. However, on various tasks [62, 119, 125, 101, 86, 100], the ML trained acoustic model did not yield best performance. This is because the optimality criterion of the ML training in general does not hold for speech data [125]. To further improve the acoustic model performance, discriminative training methods have been proposed and have shown significant improvements over ML-trained models on many large-vocabulary speech recognition tasks. Therefore, when proposing a new acoustic modeling framework, it would be beneficial to make it compatible with discriminative training methods to take advantage of its error reduction capability.

Instead of seeking model parameters that can maximize the likelihood of the data, discriminative training methods seek parameters that can minimize the confusions that occur in the training data. In general, discriminative training methods consist of two steps: First, construct a smooth and efficiently computable objective function that reflects the degree of confusions; second, adjust the model parameters such that the objective function can be optimized. In the following sections, we review several commonly used discriminative training criteria and optimization methods proposed in the literature.

2.3.2 Training Criteria and Objective Functions

Minimum Classification Error Training

The Minimum Classification Error (MCE) training method [62, 86] seeks to minimize the number of incorrectly recognized (classified) utterances in the training data. Let $\mathbf{X}_n$ be the sequence of acoustic observations of the n^{th} training utterance, and let $\mathbf{Y}_n$ be the corresponding reference words. Given a set of model parameters, $\boldsymbol{\lambda}$, the recognizer can compute the score (log-likelihood) of the reference, $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n))$, and the score of any hypothesized word string, $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W}))$. An error occurs if the best scoring hypothesis has a better score than the reference string, and therefore the number of incorrectly recognized utterances can be computed by

$$\mathcal{N}_{err} = \sum_{n=1}^{N} \text{sign}[\max_{\mathbf{W}\neq\mathbf{Y}_n} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W})) - \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n))], \tag{2.48}$$

where N is the number of training utterances, the function sign$[d]$ returns 1 if d is positive, and 0 otherwise. Although $\mathcal{N}_{err}$ reflects the degree of confusions made by the recognizer and is efficiently computable using the Viterbi algorithm, it is difficult to optimize because both the sign function and the max function are not continuous, nor differentiable with respect to the model parameters.

To construct a smooth objective function for optimization, certain relaxations have to be applied to $\mathcal{N}_{err}$. The sign function in Equation (2.48) is typically relaxed to a continuously differentiable sigmoid function $\ell(d) = \frac{1}{1+\exp(-\zeta d)}$, where the value of ζ determines the sharpness[2] of the sigmoid function around $d = 0$. The max function can be relaxed by a scaled log-sum of the scores of the $\mathcal{K}$ best hypothesis strings. Using the two relaxations, the MCE objective function can be expressed by

$$\mathcal{L}_{MCE}(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \ell(-\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n)) + \log(\left[\frac{1}{\mathcal{K}} \sum_{\mathbf{W}\in\mathcal{W}_n^{\mathcal{K}}} \exp(\eta \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W})))\right]^{1/\eta})), \tag{2.49}$$

where $\mathcal{W}_n^{\mathcal{K}}$ is the set of $\mathcal{K}$ best hypotheses for the n^{th} utterance, and η is a scaling coefficient that determines how important the best hypothesis is compared with other hypotheses[3]. One advantage of using the log-sum for relaxing the max is that when taking the partial derivative of the objective function with respect to the model parameters, the form of the partial derivative is similar to posterior probability, and so the Forward-Backward algorithm can be used. The MCE loss function in Equation (2.49) is continuously differentiable, and can be optimized using methods described in Section 2.3.3.

[2] Typically, ζ is set such that ζd can be of the range $[-5, 5]$ for most of the utterances. This can generally be done by setting ζ inversely proportional to the number acoustic observations of the utterance for each utterance.

[3] In the experiments conducted in this work, the η was always set to 1.0 for convenience.

Maximum Mutual Information Training

Given the acoustic observation sequence $\mathbf{X}_n$ and the reference label $\mathbf{Y}_n$ of an utterance, the mutual information between observation and the label can be computed by

$$I(\mathbf{X}_n;\mathbf{Y}_n) = \log(\frac{p(\mathbf{X}_n,\mathbf{Y}_n)}{p(\mathbf{X}_n)p(\mathbf{Y}_n)}). \tag{2.50}$$

The Maximum Mutual Information (MMI) training method [119, 125] seeks to maximize the sum of the mutual information between the acoustic observations and the reference labels of all the training utterances. Because the probability of seeing the reference label, $\mathbf{Y}_n$, is determined by the language model (and lexicon), changing the acoustic model parameters $\boldsymbol{\lambda}$ does not affect the probability. Therefore, we can drop the $p(\mathbf{Y}_n)$ term in Equation (2.50) when constructing the objective function for MMI training, so the objective function can be expressed by

$$\begin{aligned}\mathcal{F}_{MMI}(\boldsymbol{\lambda}) &= \sum_{n=1}^{N} \log(\frac{p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{Y}_n)}{p_{\boldsymbol{\lambda}}(\mathbf{X}_n)}) \\ &= \sum_{n=1}^{N} [\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{Y}_n)) - \log(\sum_{\mathbf{W}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{W}))]. \end{aligned} \tag{2.51}$$

Note that under the HMM, the two terms in the MMI objective function are already continuously differentiable with respect to the acoustic model parameters, thus no relaxation is needed. Also, in some research, the MMI training is referred to as Conditional Maximum Likelihood (CML) [126, 94] in the sense that although both $\mathbf{X}_n$ and $\mathbf{Y}_n$ are given, the algorithm seeks to maximize $p_{\boldsymbol{\lambda}}(\mathbf{Y}_n|\mathbf{X}_n)$ rather than $p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{Y}_n)$.

Minimum Phone Error Training

The Minimum Phone Error (MPE) training method [101] seeks to minimize the amount of phone errors that occur during recognition. Given the reference phone sequence and the phone sequence of the recognition output, the number of phone errors can be computed by summing the number of phone substitutions (Sub_p), the number of phone deletions (Del_p), and the number of phone insertions (Ins_p) by aligning the two hypothesis and reference phone strings using dynamic programming [21]. Minimizing the phone errors is equivalent to maximizing the unnormalized phone accuracy:

$$\begin{aligned} \text{PhoneAcc} &= \text{Num}_p - \text{Sub}_p - \text{Del}_p - \text{Ins}_p \\ &= \text{Corr}_p - \text{Ins}_p, \end{aligned} \tag{2.52}$$

where Num_p is the total number of phones in the reference and Corr_p is the number of phones in the reference that are correctly recognized. Although the phone accuracy can be efficiently computed, it requires the alignment of the best hypothesis (which requires a max operation similar to the case of the MCE training). Therefore, the exact phone accuracy is not a continuously differentiable function with respect to the acoustic model parameters. As a result, relaxation is required. A commonly used relaxation method was proposed in [99], whereby an approximated (per arc) accuracy is computed for each phone arc considered by the recognizer, and the relaxed utterance phone accuracy can be computed by a weighted sum of the per arc accuracies.

The set of possible phone sequences considered by the recognizer and the corresponding time alignments can be represented efficiently by a phone lattice. Given a phone arc z of the reference phone string and a phone arc q in the recognition lattice, the approximated accuracy of q with respect to z is defined in [99] by

$$\text{Acc}(q, z) = \left\{ \begin{array}{ll} -1 + 2e(q, z), & \text{if } z \text{ and } q \text{ are the same phone} \\ -1 + e(q, z), & \text{if different phones} \end{array} \right\}, \tag{2.53}$$

where $e(q, z)$ is the proportion of the length of z that overlaps with q. The intuition behind the above definition is that a phone arc q can potentially be an insertion (-1 to phone accuracy), but if it overlaps with a phone arc z in the reference alignment, it can contribute partially to either substitution (0 to phone accuracy) or correctness (+1 to phone accuracy). Using this notion, the approximated accuracy of q with respect to the reference $\mathbf{Y}_n$ can be represented by

$$\text{Acc}(q, \mathbf{Y}_n) = \max_{z \in \mathbf{Y}_n} \text{Acc}(q, z). \tag{2.54}$$

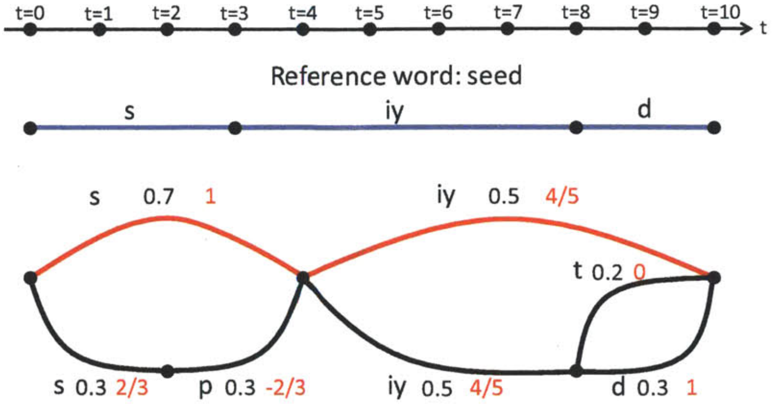


Figure 2-5: Computing approximated phone accuracy on a phone lattice. The blue line segments represent the phone alignment of the reference word "seed". The red arcs are the phone arcs corresponding to recognition output. The first number corresponding to each phone arc is its posterior probability, and the second number is the approximated phone accuracy for the arc. For example, the red "s" arc covers entire reference arc for "s", and thus has an approximated accuracy of 1. The arc corresponds to "p" overlaps with $\frac{1}{3}$ of the reference phone arc for "s" (but only overlaps with $\frac{1}{5}$ of the reference "iy" arc), and thus has an approximated accuracy of $-\frac{2}{3}$. Summing the approximated per arc accuracy with posterior probabilities as the weights results in 1.8, while the actual phone accuracy of the recognition output is 2.0. Note that the phone lattice also implies there are only 3 phone sequences considered by the recognizer: "s iy"(as in the word "see") with posterior probability 0.5, "s p iy d"("speed") with probability 0.3, and "s iy t"("seat") with probability 0.2.

Figure 2-5 illustrates an example of how to compute the approximated phone accuracy on a phone lattice. The posterior probabilities of the phone arcs can be computed by the Forward-Backward algorithm.

Note that given a lattice, changing the acoustic model parameters only affects the posterior probabilities of seeing the phone arcs, but not the approximated accuracies of the arcs. The approximated accuracy of each utterance can be thought of as a linear combination of the posterior probabilities with combination coefficients as the per arc approximated accuracy. Also, the posterior probabilities are continuously differentiable with respect to the model parameters. Therefore, the per utterance approximated accuracy is also continuously differentiable given the recognition lattices. As a result, the MPE objective function can be computed by summing over the approximated accuracy of the training utterances

$$\mathcal{F}_{MPE}(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \sum_{q \in \mathcal{A}_n} \frac{\sum_{\mathbf{W}: q \in \mathbf{W}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W})}{\sum_{\mathbf{W}'} p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W}')} \mathrm{Acc}(q, \mathbf{Y}_n), \tag{2.55}$$

where $\mathcal{A}_n$ is the set of phone arcs in the recognition lattice of the n^{th} utterance, and the term before Acc is the posterior probability of the arc q being traversed. In general, the recognition lattices are recomputed after several MPE updates.

Margin-Based Training Methods

Given an acoustic observation $\mathbf{x}$, the acoustic model provides the score of $\mathbf{x}$ with respect to each state, and effectively is doing a classification on $\mathbf{x}$. In recent years, encouraged by the success of Support Vector Machines (SVMs) [14], which are maximum margin classifiers, several researchers have proposed incorporating classification margin in the discriminative training of acoustic model parameters in order to increase the generalization ability of the model. In [80], it was proposed that when computing the MCE training loss function, the scores of the competing hypotheses should be increased by a positive constant margin. In [112], it was suggested that the margin should be proportional to the edit distance between the reference and the competing hypothesis. Inspired by the idea proposed in [112], the notion of the string distance-based margin was incorporated in the MMI training, and a Boosted-MMI training criterion [100] was proposed:

$$\mathcal{F}_{BMMI}(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \log(\frac{p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n)}{\sum_{\mathbf{W}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W}) \exp(\rho \mathcal{E}(\mathbf{W}, \mathbf{Y}_n))}), \tag{2.56}$$

where $\mathcal{E}(\mathbf{W}, \mathbf{Y}_n)$ denotes the edit distance between $\mathbf{W}$ and $\mathbf{Y}_n$, which can be either at word-level, phone-level, or state-level. The relations between margin-based variants of discriminative training methods were further studied in [87]. In general, when the training error rate is not too high, considering the margin can help reduce recognition error. However, the experimental results in [15] show that if the training error rate is high, then considering the margin does not provide much gain.

2.3.3 Optimizing Parameters

In general, the optimization of acoustic model parameters under a discriminative training criterion can be done iteratively via the following procedures.

1. Forced-Alignment Step: For each training utterance, compute and store the recognition score and the state/phone alignment in the constrained search space where only the reference words $\mathbf{Y}_n$ can be generated. (Lattices can be used to store multiple alignments.)

2. Recognition Step: Allow the recognizer to search through the unconstrained space, compute the scores, alignments, and posterior probabilities of the recognition hypotheses, and store the results in recognition lattices.

3. Accumulation Step: Scan through the outputs of the previous steps, and accumulate the statistics of model parameters needed in the update step under the guidance of the discriminative objective function.

4. Update Step: Use the accumulated statistics to update the model parameters.

In general, during the recognition step, a weaker language model (typically a unigram language model) is used to force the acoustic model to fix more training errors, and to reduce the amount of time to generate the recognition outputs. Some pruning might also be required to restrict the size of recognition lattices. In terms of the update step, two commonly-used methods are the Gradient-Based method [18, 86] and the Extended Baum-Welch (EBW) method [99]. In the following, we use several discriminative training criteria as examples and show how to apply the two types of update methods. More specifically, we will show how to update the GMM parameters. The model parameters $\boldsymbol{\lambda}$ to be learned consist of three parts, the mixture weights $\{\omega_m^s\}$, the mean vectors $\{\boldsymbol{\mu}_m^s\}$, and the covariance matrices $\{\Sigma_m^s\}$.

Extended Baum-Welch Method

We use Maximum Mutual Information (MMI) training as an example to show how to use the Extended Baum-Welch (EBW) method as an iterative procedure to update the acoustic model parameters under a discriminative training criterion. The key to applying the EBW method is to construct a weak-sense auxiliary function [99] for the training criterion. Given an objective function $F(\boldsymbol{\lambda})$ and the current model parameters $\bar{\boldsymbol{\lambda}}$, a weak-sense auxiliary function $Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ satisfies

$$\frac{\partial F(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}\Big|_{\boldsymbol{\lambda}=\bar{\boldsymbol{\lambda}}} = \frac{\partial Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})}{\partial \boldsymbol{\lambda}}\Big|_{\boldsymbol{\lambda}=\bar{\boldsymbol{\lambda}}}. \tag{2.57}$$

More details about the auxiliary function can be found in Appendix B.

To construct a weak-sense auxiliary function, note that the MMI objective function can be represented by the difference of two terms: the numerator term $\sum_{n=1}^{N} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n))$ and the denominator term $\sum_{n=1}^{N} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n))$. The numerator term is the objective function of the ML training of $\boldsymbol{\lambda}$ with word references, and the denominator term is the objective function of the ML training without references. Under the principle of the EM algorithm, both terms can have a strong-sense auxiliary function for optimization. Let $Q^{num}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ be the auxiliary function for the numerator term, and $Q^{den}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ be that of the denominator term. Because a continuously differentiable strong-sense auxiliary function is also a weak-sense auxiliary function (as shown in Appendix B), we can construct a weak-sense auxiliary function for MMI by

$$Q^{MMI}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}}) = Q^{num}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}}) - Q^{den}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}}) + Q^{sm}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}}), \tag{2.58}$$

where $Q^{sm}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ is a smoothing term with maximum at $\boldsymbol{\lambda} = \bar{\boldsymbol{\lambda}}$. The reason to have the smoothing term is to ensure that the optimum for $Q^{MMI}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ would not be too far away from $\bar{\boldsymbol{\lambda}}$ such that optimizing the weak-sense auxiliary function can ensure improvement on the MMI objective function.

To optimize the auxiliary function of MMI training, we can first compute the sufficient statistics for each term in Equation (2.58). Given the lattices from the Forced-Alignment step (numerator lattices), statistics of $Q^{num}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ for each state s and mixture component m, including ϑ_{sm}^{num}, $\mathcal{O}_{sm}^{num}(\mathbf{X})$, and $\mathcal{O}_{sm}^{num}(\mathbf{X}^2)$, can be computed using Equations (2.39-2.41). Similarly, ϑ_{sm}^{den}, $\mathcal{O}_{sm}^{den}(\mathbf{X})$, and $\mathcal{O}_{sm}^{den}(\mathbf{X}^2)$ can be computed using recognition lattices (denominator lattices).

To update the GMM means, if we think of posterior probabilities as partial counts, a similar formulation as in Appendix A can be applied, and we can have $\frac{\partial Q^{num}(\boldsymbol{\lambda},\bar{\boldsymbol{\lambda}})}{\partial \boldsymbol{\mu}_m^s} = (\bar{\boldsymbol{\Sigma}}_m^s)^{-1}(\mathcal{O}_{sm}^{num}(\mathbf{X}) - \vartheta_{sm}^{num}\boldsymbol{\mu}_m^s)$ for the mean of the m^{th} mixture component of state s. (This is how we obtained Equation (2.43).) Similarly, the partial derivative of the denominator term with respect to the mean can also be computed using the denominator statistics. For the smoothing term, the partial derivative is equal to $(\bar{\boldsymbol{\Sigma}}_m^s)^{-1}\nu_{sm}(\bar{\boldsymbol{\mu}}_m^s - \boldsymbol{\mu}_m^s)$, where ν_{sm} is a scalar that controls how close the optimal $\hat{\boldsymbol{\mu}}_m^s$ of the auxiliary function is to the current mean $\bar{\boldsymbol{\mu}}_m^s$. Adding the contribution of the three partial derivatives, and setting it equal to zero, the EBW update equation for the mean is

$$\hat{\boldsymbol{\mu}}_m^s = \frac{\mathcal{O}_{sm}^{num}(\mathbf{X}) - \mathcal{O}_{sm}^{den}(\mathbf{X}) + \nu_{sm}\bar{\boldsymbol{\mu}}_m^s}{\vartheta_{sm}^{num} - \vartheta_{sm}^{den} + \nu_{sm}}. \tag{2.59}$$

Typically, a large enough ν_{sm} is chosen such that the denominator of Equation (2.59) is strictly positive.

The update equation can be interpreted as follows. The mean $\boldsymbol{\mu}_m^s$ can be thought of as a weighted average of the training acoustic feature vectors aligned with state s and mixture component m. If state s appears in a numerator lattice at a certain time point, the feature vector at that time point would contribute a positive example to the mean. Adjusting the mean based on the positive examples can increase the auxiliary function $Q^{num}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ as we have done in ML training. Conversely, if the state s appears in the denominator lattice at a certain time point, the feature vector at that time point would be a negative example, or one that we want the mean to be dissimilar to. Making the mean dissimilar to the negative examples decreases $Q^{den}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$, and thus also improves $Q^{MMI}(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$. The ν_{sm} term reflects how close we believe $\boldsymbol{\mu}_m^s$ is to $\bar{\boldsymbol{\mu}}_m^s$, and thus can be seen as a prior count. As a result, to compute the update of the mean, we can sum up first order statistics of the positive, negative, and prior examples, and divide the sum by the total number of counts (including positive, negative, and prior), as shown in Equation (2.59).

To update the covariance matrix $\boldsymbol{\Sigma}_m^s$, similar concepts to the ones described above can still be applied, except that we need to consider the second order statistics of the three types of examples. As shown in the ML estimation example in Equation (2.44), the covariance matrix is the average of the second order statistics minus $\boldsymbol{\mu}^s_m(\boldsymbol{\mu}^s_m)^{\mathrm{T}}$. Similarly, the EBW update for the covariance matrix can be expressed by

$$\hat{\boldsymbol{\Sigma}}^s_m = \frac{\mathcal{O}^{num}_{sm}(\mathbf{X}^2) - \mathcal{O}^{den}_{sm}(\mathbf{X}^2) + \nu_{sm}(\bar{\boldsymbol{\Sigma}}^s_m + \bar{\boldsymbol{\mu}}^s_m(\bar{\boldsymbol{\mu}}^s_m)^{\mathrm{T}})}{\vartheta^{num}_{sm} - \vartheta^{den}_{sm} + \nu_{sm}} - \hat{\boldsymbol{\mu}}^s_m(\hat{\boldsymbol{\mu}}^s_m)^{\mathrm{T}}, \tag{2.60}$$

where $\nu_{sm}(\bar{\boldsymbol{\Sigma}}^s_m + \bar{\boldsymbol{\mu}}^s_m(\bar{\boldsymbol{\mu}}^s_m)^{\mathrm{T}})$ is the second order statistics of the prior examples, and $\hat{\boldsymbol{\mu}}^s_m$ can be computed from Equation (2.59).

To update the mixture weights, using $-\frac{\vartheta^{den}_{sm}}{\bar{\omega}^s_m}\omega^s_m$ instead of $-\vartheta^{den}_{sm}\log(\omega^s_m)$ as part of the auxiliary function for the denominator term is easier to optimize [99]. Note that the latter expression has the same partial derivative as the first one, but is linear with respect to the parameter to be optimized. We can also add a smoothing term, which results in the auxiliary function for the mixture weights of state s to be computed by

$$\sum_{m=1}^{M_s} \vartheta^{num}_{sm}\log(\omega^s_m) - \frac{\vartheta^{den}_{sm}}{\bar{\omega}^s_m}\omega^s_m + c_{sm}(\bar{\omega}^s_m\log(\omega^s_m) - \omega^s_m), \tag{2.61}$$

where c_{sm} is a non-negative constant and the smoothing term has a maximum at $\omega^s_m = \bar{\omega}^s_m$. As shown in [99], if we set $c_{sm} = \frac{-\vartheta^{den}_{sm}}{\bar{\omega}^s_m} + \max_m \frac{\vartheta^{den}_{sm}}{\bar{\omega}^s_m}$, the linear terms related to ω^s_m can cancel out, and the auxiliary function becomes

$$\sum_{m=1}^{M_s}[\vartheta^{num}_{sm}\log(\omega^s_m) + c_{sm}\bar{\omega}^s_m\log(\omega^s_m)] + \max_m \frac{\vartheta^{den}_{sm}}{\bar{\omega}^s_m}, \tag{2.62}$$

where the last term is just a function of parameters from the previous step and can be treated as a constant in the current iteration. Using Lagrangian multipliers [12] on Equation (2.62), we can find the update equation for the mixture weights

$$\hat{\omega}^s_m = \frac{\vartheta^{num}_{sm} + c_{sm}\bar{\omega}^s_m}{\sum_{m'=1}^{M_s}\vartheta^{num}_{sm'} + c_{sm'}\bar{\omega}^s_{m'}}. \tag{2.63}$$

As described in [99], typically the mixture weights are updated for several iterations before recomputing the numerator and denominator statistics.

Note that although we use the weak-sense auxiliary function and its partial derivatives to explain the update equations for the EBW method, the auxiliary function and the partial derivatives need not be computed. During the update, only the statistics appearing in the update Equations (2.59-2.63) need to be computed. In practice, the statistics from ML training are also used in the update to further smooth the model in order to prevent over-training. The use of ML statistics in the update is called I-smoothing in [99]. For the MMI training, I-smoothing is equivalent to scaling the numerator statistics by a constant larger than one, but for other training criterion such as MPE training, the statistics for I-smoothing need to be computed separately. In general, the EBW method provides closed-form update equations for the parameters, but a good heuristic is typically needed to adjust the smoothing constants.

Gradient-Based Method

For the gradient-based method, the key is to compute the gradient of the objective function with respect to the model parameters. If the gradient can be computed, the objective function can be improved by moving the parameters along an appropriate direction indicated by the gradient. (The direction of the movement is of the same sign as the gradient for maximization problems, and of opposite sign for minimization problems.) To compute the gradient, we can utilize the chain rule and the linearity of the differential operation.

For an acoustic feature vector $\mathbf{x}$ appearing in the training data, and a HMM state s, let $a_{\boldsymbol{\lambda}}(\mathbf{x}, s) = \log(p_{\boldsymbol{\lambda}}(\mathbf{x}|s))$ be the acoustic score of $\mathbf{x}$ with respect to the state s. Because the acoustic model parameters affect the discriminative training objective function via the acoustic scores, and because the derivative of the acoustic score with respect to the model parameters is of closed form, we can compute the gradient as follows. First, compute the partial derivative of the objective function with respect to each acoustic score. Second, compute the partial derivative of the acoustic score with respect to the parameters. Then, the gradient can be computed by summing the product of the two partial derivatives over all acoustic scores:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\lambda}} = \sum_{n=1}^{N} \sum_{\mathbf{x}_{nt}} \sum_{s} \frac{\partial \mathcal{L}}{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}_{nt}, s)} \frac{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}_{nt}, s)}{\partial \boldsymbol{\lambda}}, \tag{2.64}$$

where $\mathbf{x}_{nt}$ denotes the vector at time t in the n^{th} utterance. In the following, we use Minimum Classification Error (MCE) training as an example, to illustrate how to compute the gradient in Equation (2.64).

For each utterance, the Forced-Alignment step can provide the score $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n))$, and the Recognition step can provide the scores $\{\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W})) : \mathbf{W} \in \mathcal{W}^{\mathcal{K}}\}$ for the top $\mathcal{K}$ competing hypotheses. The scores can be used to compute the per utterance loss function in Equation (2.49). Let ε_n denote the per utterance loss, and let d_n denote the score difference within the sigmoid function

$$d_n = -\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n)) + \log(\left[\frac{1}{\mathcal{K}} \sum_{\mathbf{W} \in \mathcal{W}_n^{\mathcal{K}}} \exp(\eta \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W})))\right]^{1/\eta}) \qquad (2.65)$$

To compute the gradient, we can first take the partial derivative of the MCE loss function with respect to d_n

$$\frac{\partial \mathcal{L}}{\partial d_n} = \frac{\partial}{\partial d_n} \frac{1}{1 + \exp(-\zeta d_n)} = \zeta \frac{-\zeta \exp(-\zeta d_n)}{[1 + \exp(-\zeta d_n)]^2} = \zeta \varepsilon_n (1 - \varepsilon_n), \qquad (2.66)$$

where the partial derivative is of maximum 0.25ζ when $\varepsilon_n = 0.5$ and the ratio $\frac{\zeta \varepsilon_n (1-\varepsilon_n)}{0.25\zeta}$ decreases as ζ increases. This property shows that MCE training gives more weight to the utterances near the decision boundary, and becomes more focused as ζ increases. Experiments in [85] show that choosing an appropriately large value of ζ for training results in better model performance.

After the partial derivative with respect to d_n is computed, the next step is to take the partial derivatives of d_n with respect to the acoustic scores. Given the n^{th} training utterance, let $\boldsymbol{S}$ be a valid state alignment that correspond to $\mathbf{Y}_n$, and let $\mathbf{x}_{nt}$ be the acoustic feature vector at time t. For each state s, if the state sequence $\boldsymbol{S}$ contains s at time t, then the joint probability $p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \boldsymbol{S}, \mathbf{Y}_n)$ will contain $\exp(a_{\boldsymbol{\lambda}}(\mathbf{x}_{nt}, s))$ under the HMM formulation. On the other hand, if $\boldsymbol{S}$ does not contain s at time t, $p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \boldsymbol{S}, \mathbf{Y}_n)$ will be a constant with respect to the score $a_{\boldsymbol{\lambda}}(\mathbf{x}_{nt}, s)$. As a result, we can have the following

$$\gamma_{ts}^{ref_n} = \gamma_{ts}^{\mathbf{Y}_n} = \frac{\partial}{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}_{nt}, s)} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n)) = \frac{\sum_{\boldsymbol{S}:s_t=s} p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \boldsymbol{S}, \mathbf{Y}_n)}{\sum_{\boldsymbol{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \boldsymbol{S}, \mathbf{Y}_n)}, \qquad (2.67)$$

which can be computed efficiently using the Forward-Backward algorithm on the forced-alignment lattice of $\mathbf{Y}_n$. Similarly, for each competing hypothesis $\mathbf{W}$, we can compute $\gamma_{ts}^{\mathbf{W}}$. As a result, the partial derivative of the second (big) term in Equation (2.65) with respect to $a_{\boldsymbol{\lambda}}(\mathbf{x}_{nt}, s)$ can be computed by summing the contributions of each competing word sequence:

$$\gamma_{ts}^{comp_n} = \sum_{\mathbf{W}\in\mathcal{W}^{\mathcal{K}}} \frac{\exp(\eta\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W})))}{\sum_{\mathbf{W}'\in\mathcal{W}^{\mathcal{K}}}\exp(\eta\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{W})))}\gamma_{ts}^{\mathbf{W}}, \tag{2.68}$$

where the term in the middle can be thought of as the adjusted relative importance of each competing hypothesis. As a result, the gradient of the MCE objective function can be rewritten as

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\lambda}} = \sum_{n=1}^{N}\sum_{\mathbf{x}_{nt}\in\mathbf{X}_n}\sum_{s} -[\zeta\varepsilon_n(1-\varepsilon_n)(\gamma_{ts}^{ref_n} - \gamma_{ts}^{comp_n})]\frac{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}_{nt}, s)}{\partial\boldsymbol{\lambda}}. \tag{2.69}$$

Computing the partial derivative of the acoustic score with respect to model parameters in Equation (2.69) consists of two steps. The posterior probability of each mixture component can be computed using Equation (2.25), and the partial derivatives with respect to each mean and covariance can be computed as shown in Appendix A. Since keeping the covariances positive definite after each update requires extra work, for the gradient-based method, a diagonal GMM is typically used. Even though each dimension can be treated independently, the variance at each dimension should still be positive. To avoid checking the constraint on the variance during the optimization, instead of computing the gradient with respect to the variance and updating the variance, the gradient with respect to the square root of the variance is computed and the square root is updated at each iteration. In this way, even though the square root of the variance can go negative, the variance is still kept positive. After the gradient is computed, the model parameters can be updated by Quasi-Netwon methods such as the Quickprop algorithm [85]. Details of the Quickprop algorithm can be found in Appendix G.

Remarks

In the previous paragraphs we have reviewed two update methods for discriminative training. Both of which have their advantages and disadvantages. For the gradient descent method, the only thing that needs to be computed is the gradient. However, the step size for the gradient-based method needs to be selected appropriately; it might overshoot if set too large, or it might converge very slowly if set too small. The EBW method provides closed-form solution for each update, but the smoothing constants need to be set heuristically. Although we used MMI training as an example for the EBW update and MCE training for the gradient-based update, the training methods are not limited to a particular update method. In subsequent experiments discussed in Chapter 4-6, we use gradient-based MCE training to refine the ML trained GMM parameters.

2.4 Finite-State Transducers for ASR

A Finite-State Transducer (FST) is a finite-state machine that can map strings of input symbols to strings of output symbols. Basic FST operations such as composition, determinization, and minimization make it easy to modularize each block when designing a complex system using FSTs. Generic algorithms of FST operations can be implemented once and used for many applications. Over the years, FSTs have been successfully used for many speech and natural language processing related applications [92, 53, 113].

When each arc between two FST states is associated with a weight or score, it is commonly referred to as a weighted FST. However, in this thesis, we ignore the distinction and just use FST to refer to a weighted FST by default. Given an input string to an FST, the best weighted output string can be found using a dynamic programming-based (Viterbi) search algorithm. The algebra operations of FST weights are defined by a semiring. A more formal definition of FST and semiring based on [113] is described in Section 2.4.1.

Note that a Markov chain can be represented using an FST by choosing a semiring such that the arc weight reflects the transition probability from one state to another. If we think of the acoustic observation sequence as the input string and the word sequence as the output string, then we can build a HMM-based ASR system using FSTs. Section 2.4.2 shows an example of this.

2.4.1 Definitions

Weight Semirings

A semiring $\mathfrak{K} = (\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ is defined on a set $\mathbb{K}$ with addition operator $\oplus$, multiplication operator $\otimes$, identity element for addition $\bar{0}$, and identity element for multiplication $\bar{1}$.

The addition and multiplication operations on elements of a semiring $\mathfrak{K}$ satisfy all required conditions of a Field in algebra theories [29], including closure, associativity, commutativity, and distributivity, except that some element in $\mathbb{K}$ may not have an additive inverse. For an FST, the $\otimes$ operation defines how to combine the weights of a series of transitions, and the $\oplus$ operation defines how to combine the weights of parallel paths.

There are several commonly used semirings for speech and language applications [113]. The real semiring $(\mathbb{R}, +, \times, 0, 1)$, is a semiring on the set of real numbers with conventional addition and multiplication operations. The FST weights under a real semiring can be directly used to represent the model probabilities. To avoid underflow errors, the log version of the real semiring $(-\infty \cup \mathbb{R}, \text{log\_sum}, +, -\infty, 0)$ is often used to compute the (log) posterior probabilities in the Forward-Backward algorithm. Note that log_sum of two weights w_1 and w_2 can be computed by

$$\begin{aligned} \log\_\text{sum}(w_1, w_2) &= \log(\exp(w_1) + \exp(w_2)) \\ &= \max(w_1, w_2) + \log(1 + \exp(\min(w_1, w_2) - \max(w_1, w_2))), \quad (2.70) \end{aligned}$$

where the second equation provides better numerical stability. Another commonly used semiring is the tropical semiring $(\mathbb{R} \cup \infty, \min, +, \infty, 0)$. The tropical semiring can be used to represent $-\log$ probabilities of paths during Viterbi decoding described in Section 2.1.2, where at each time point the $-\log$ of the state transition and observation probabilities are added to a path, and only the most probable path among the parallel paths reaching a particular state is kept.

Finite-State Transducer

A weighted Finite-State Transducer $\mathfrak{T}$ over a semiring $\mathfrak{K}$ is defined by a tuple $(\mathbb{I}, \mathbb{O}, \mathbb{S}, \mathbb{T}, i, \mathbb{F}, \kappa, \varrho)$ [113], where $\mathbb{I}$ is the set of input symbols, $\mathbb{O}$ is the set of output symbols, $\mathbb{S}$ is the set of states, $\mathbb{T}$ is the set of transitions, i is the initial state, $\mathbb{F}$ is the set of final states, κ is the initial weight, and ϱ is the final weights function $\varrho : \mathfrak{f} \in \mathbb{F} \rightarrow \mathbb{K}$.

A transition of $\mathfrak{T}$ is defined by a tuple $\mathfrak{t} = (\mathfrak{p}[\mathfrak{t}], \mathfrak{n}[\mathfrak{t}], \mathfrak{l}_i[\mathfrak{t}], \mathfrak{l}_o[\mathfrak{t}], \mathfrak{u}[\mathfrak{t}])$. The transition $\mathfrak{t}$ is an arc from the source state $\mathfrak{p}[\mathfrak{t}]$ to the destination state $\mathfrak{n}[\mathfrak{t}]$ that takes an input symbol

$\mathfrak{l}_\mathfrak{i}[\mathfrak{t}]$ and produces an output symbol $\mathfrak{l}_\mathfrak{o}[\mathfrak{t}]$ with weight $\mathfrak{u}[\mathfrak{t}]$. Note that the input symbol of a transition can be empty (meaning that the transition does require an input symbol), and so can the output symbol. A special symbol ε is used to handle such empty cases.

A sequence of N transitions connecting the initial state and a final state forms a permissible path $\pi = \mathfrak{t}_1\mathfrak{t}_2 \ldots \mathfrak{t}_N$, with $\mathfrak{p}[\mathfrak{t}_1] = i$, $\mathfrak{n}[\mathfrak{t}_N] \in \mathbb{F}$, and $\forall j = 1, 2, \ldots, N-1$, $\mathfrak{n}[\mathfrak{t}_j] = \mathfrak{p}[\mathfrak{t}_{j+1}]$. The input (label) sequence associated with the path π is $\mathfrak{l}_\mathfrak{i}[\pi] = \mathfrak{l}_\mathfrak{i}[\mathfrak{t}_1]\mathfrak{l}_\mathfrak{i}[\mathfrak{t}_2]\ldots\mathfrak{l}_\mathfrak{i}[\mathfrak{t}_N]$. Similarly, the output (label) sequence associated with π is $\mathfrak{l}_\mathfrak{o}[\pi] = \mathfrak{l}_\mathfrak{o}[\mathfrak{t}_1]\mathfrak{l}_\mathfrak{o}[\mathfrak{t}_2]\ldots\mathfrak{l}_\mathfrak{o}[\mathfrak{t}_N]$.

The semiring $\mathfrak{K}$ specifies how the weights can be combined for an FST. The path weight of a permissible path can be computed by $\mathfrak{u}[\pi] = \kappa \otimes \mathfrak{u}[\mathfrak{t}_1] \otimes \mathfrak{u}[\mathfrak{t}_2] \otimes \cdots \otimes \mathfrak{u}[\mathfrak{t}_N] \otimes \varrho(\mathfrak{n}[\mathfrak{t}_N])$. The combined weight for a set of K paths can be computed by $\mathfrak{u}[\pi_1, \pi_2, \ldots, \pi_K] = \mathfrak{u}[\pi_1] \oplus \mathfrak{u}[\pi_2] \oplus \cdots \oplus \mathfrak{u}[\pi_K]$

Given a sequence of input symbols, an FST accepts the input sequence if it results in a permissible path. If the FST accepts an input sequence, it produces a set of output sequences and their corresponding weights. If the input sequence is rejected, the FST may or may not generate output sequences, depending on the implementation.

Basic FST Operations

There are several commonly used FST operations for speech and language processing applications. The definitions of the operations are provided below, and Section 2.4.2 will shows some example usages of these operations. There have been several existing FST Libraries such as [53] that have the basic operations implemented and can be applied to various applications.

- Composition: The operation $\mathfrak{F}_X \circ \mathfrak{F}_Y$ takes in two FSTs and produces an FST that has the input symbols of $\mathfrak{F}_X$ and the output symbols of $\mathfrak{F}_Y$. Given an input sequence, the composed FST functions as though $\mathfrak{F}_X$ took the input sequence and then $\mathfrak{F}_Y$ took the corresponding outputs of $\mathfrak{F}_X$ to generate the final output sequences.

- Projection: The operator $\mathrm{project}_{\mathfrak{ii}}(\cdot)$ replaces the output symbol of each transition with the input symbol of the transition. (The source state, destination state, and the weight are kept the same.) Similarly, $\mathrm{project}_{\mathfrak{oo}}(\cdot)$ replaces the input symbol with output symbol, and $\text{project}_{oi}(\cdot)$ swaps the input and output symbols.

- Determinization: the operator $\det(\cdot)$ determinizes the FST such that there is only one path for each distinct pair of input and output sequences.

2.4.2 Building a HMM based ASR System with FSTs

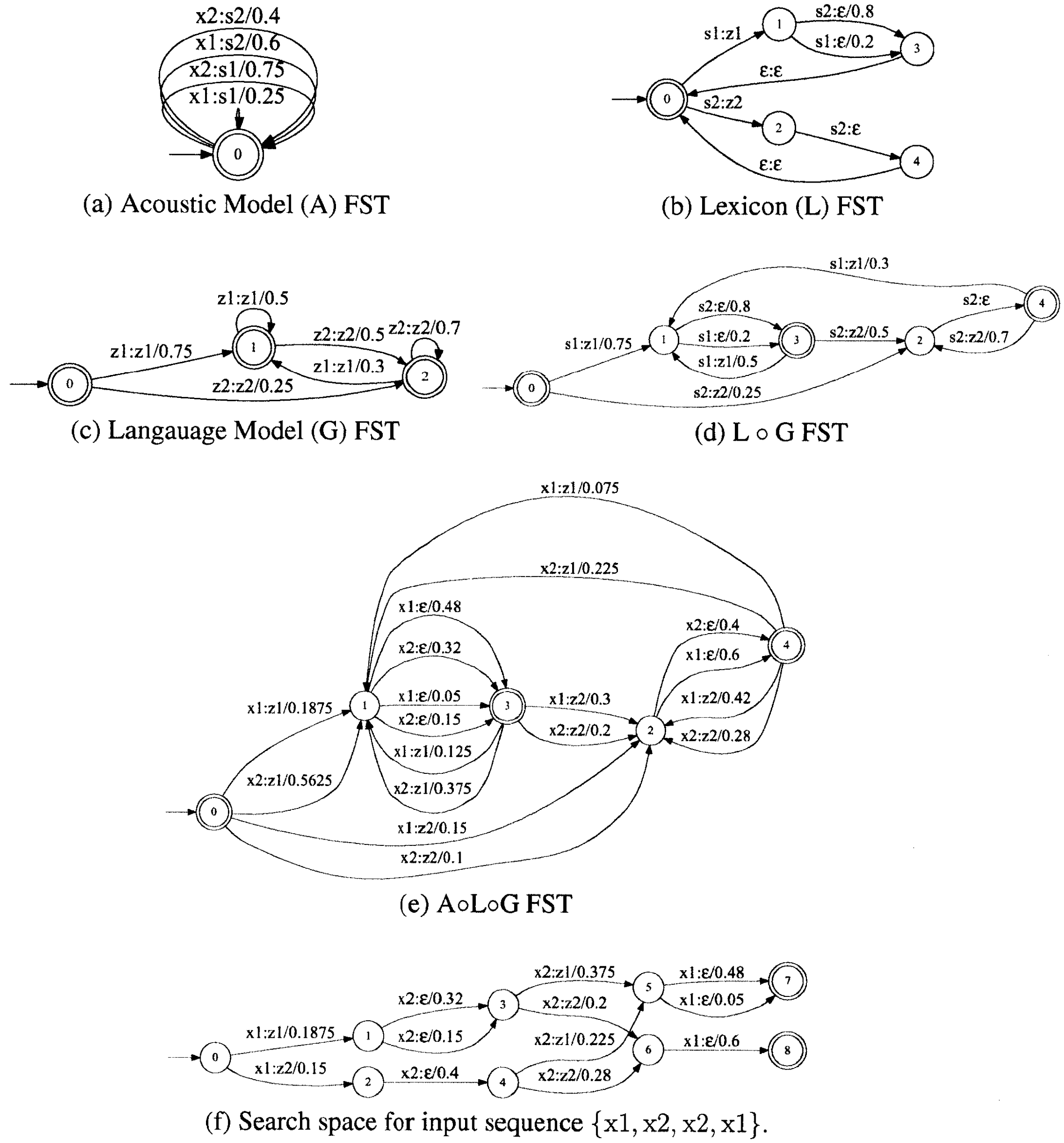
A Simplified Example

Figure 2-6 illustrates a toy example of how to use FSTs with a real semiring to construct a HMM- based recognizer. For each FST in the figure, each circle represents a state. The state pointed by the left-most arrow is the initial state (typically state 0), and the states with double circles are the final states. For each arc, the first label denotes the input symbol of the transition, the second label denotes the output symbol, and the final number denotes the weight. If the input or output symbol is empty, the ε symbol is used. If the weight of an arc is omitted, it means that the weight is equal to the multiplicative inverse, and is 1 for the real semiring.

The conditional distribution of seeing an observation given a particular state can be represented by the Acoustic Model (A) FST in Figure 2-6(a). In this example, there are two types of observations, x1 and x2, and two phonetic states, s1 and s2. Under the model, the conditional probability of seeing x1 given state s1 is 0.25, and the conditional probability of seeing x2 is 0.75. Because the Acoustic Model independently scores each observation, the A FST only has one single state.

There are two words z1 and z2 considered by the recognizer. For the word z1, there are two possible pronunciations, "s1 s1" with probability 0.2 and "s1 s2" with probability 0.8. The word z2 only has one pronunciation "s2 s2". The lexical constraints specify the conditional probability of seeing a particular state sequence given a word sequence, and can be represented by the Lexicon (L) FST in Figure 2-6(b). Also, the L FST has a closure arc from the last state of each word back to the initial state; these closure arcs allow the FST to accept multiple numbers of words.

The marginal (prior) distribution of the word sequences can be represented by a Language Model (G) FST. Figure 2-6(c) illustrates how to represent a bigram language model



(a) Acoustic Model (A) FST

(b) Lexicon (L) FST

(c) Langauage Model (G) FST

(d) L ∘ G FST

(e) A∘L∘G FST

(f) Search space for input sequence $\{x1, x2, x2, x1\}$.

Figure 2-6: A simplified Example of constructing HMM based recognizer using FSTs with real semiring.

using an FST. The probability of seeing $\mathrm{z1}$ as the first word of the utterance is 0.75, and the probability of seeing $\mathrm{z2}$ as the first word is 0.25. Given the previous word is $\mathrm{z1}$, it has probability 0.5 of seeing either z1 or z2; given the previous word is z2, it has probability 0.7 of seeing $\mathrm{z2}$ and 0.3 of seeing $\mathrm{z1}$, as represented by the transitions of the state 2 in Figure 2-6(c). Note that the G FST can end in any state.

The sequential constraints specified by the Lexicon and the Language Model can be combined using the composition operation. Figure 2-6(d) shows the composed FST. Under the real semiring, since the L FST represents the conditional distribution of state sequences given a word sequence, and since the G FST represents the marginal distribution of word sequences, the composed L $\circ$ G FST also describes the joint distribution of the state and word sequences.

The FST used for decoding can be constructed explicitly by composing the Acoustic Model (A) FST with the L $\circ$ G FST, as shown in Figure 2-6(e). Given a sequence of observations, the search space can be expanded by traversing the decoding FST according to the observations, and the Viterbi algorithm can be used to find the best path. Figure 2-6(f) represents the search space for the observation sequence $\{\mathrm{x1}, \mathrm{x2}, \mathrm{x2}, \mathrm{x1}\}$. In this example, the best permissible path has probability of 0.0108 and consists of traversing the state sequence $0 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 7$ using the upper arcs. The corresponding output word sequence is "z1 z1". The marginal probability of seeing the observation sequence can also be computed by performing $\mathrm{det}(\mathrm{project}_{\mathrm{ii}}(\cdot))$ operation on the FST in Figure 2-6(f), where the projection operation will make the output symbol of each arc the same as the input symbol, and the determinization operation will sum up the probabilities of parallel paths that correspond to each distinct input sequence.

Large-Vocabulary ASR

To build a large-vocabulary ASR system using FSTs requires several additional components and practical considerations. To enable context-dependent acoustic modeling, a Context (C) FST is required to translate the sequences of context-dependent phonetic units to sequences of basic phonetic units. Because a context-dependent phonetic unit can consist of multiple HMM states, a Model Topology (M) FST is used to represent the state topologies within a phonetic unit. Pronunciation rules that seek to capture pronunciation variation can also be implemented as a Pronunciation Rule (P) FST [52]. To prevent underflow when computing the probabilities, the tropical semiring is used. Theoretically, the decoding FST can be constructed explicitly by sequentially composing the FSTs, as shown in Figure 2-7.

$$\begin{array}{cccccc} -\log(p(\mathbf{X},\mathbf{S},\mathbf{W})) = & -\log(p(\mathbf{X}|\mathbf{S})) & - & \log(p(\mathbf{S}|\mathbf{W})) & - & \log(p(\mathbf{W})) \\ & \mathsf{A} & \circ & \overbrace{\mathsf{M}\circ\mathsf{C}\circ\mathsf{P}\circ\mathsf{L}} & \circ & \mathsf{G} \end{array}$$

Figure 2-7: Decoding FST for a large-vocabulary HMM based ASR system.

In practice, such an FST is very large and is never physically composed. Instead, the acoustic scores are dynamically computed as needed and are added to arc weights. For a large-vocabulary triphone based system, statically composing the M FST with subsequent FSTs can still require a huge amount of memory during the composition operation, and can result in a huge FST. Appendix E describes the reason for such a memory/space explosion, and briefly illustrates an implementation of dynamic state expansion that addresses the problem. Typically, only the C ∘ P ∘ L ∘ G FST is statically composed for a triphone or quinphone based system.

Remarks

An FST representation can make it convenient to perform many operations during model training and to re-score recognition results. For example, performing recognition in the forced-alignment mode can be done by composing the C ∘ P ∘ L ∘ G FST with a word reference FST. Figure 2-8 is an example of the word string FST for "automatic speech recognition". By composing C ∘ P ∘ L ∘ G with the FST in Figure 2-8, the recognizer will align the utterance with word string "automatic speech recognition". In addition, the recognition lattices can also be represented by FSTs, and by just changing the semiring, one can either compute the best sequence or compute the posterior probabilities on the recognition lattices. To re-score the recognition result using a higher order language model, the differences between the higher and lower order language models can be computed beforehand.

During re-scoring, the recognition lattice generated with a lower order language model can be composed with the FST that stores the differences between the language models. The new best path can then be found by performing a Viterbi-based search on the newly composed FST. Also, FSTs make it easier to modularize each modeling component. Because of these conveniences and advantages, more and more ASR systems are now implemented using FSTs.

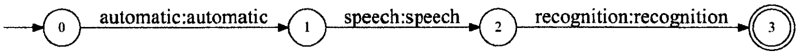


Figure 2-8: An FST that only accepts the word string "automatic speech recognition" and outputs the same word string.

2.5 Chapter Summary

In this chapter, we reviewed the mathematical framework of Hidden Markov Model (HMM) based ASR system, and described how to learn the model parameters for each modeling component. Based on the HMM assumptions, the decoding mechanism can be summarized as follows. The Acoustic Model provides instant, independent scoring for each acoustic observation with respect to phonetic states, the Lexicon and Language Model provide long-term, linguistic, sequential constraints, and the Viterbi decoding algorithm jointly considers the instant scores and the sequential constraints and utilizes dynamic programming to find the best word hypothesis. We also showed that such a decoding mechanism can be implemented efficiently using Finite-State Transducers (FSTs).

In terms of acoustic modeling techniques, we first reviewed the context-dependent acoustic modeling that seeks to incorporate the effect of nearby acoustic-phonetic contexts during scoring, and we pointed out the limitations of current modeling frameworks. We then introduced the concept of discriminative training that seeks to reduce ASR errors by adjusting acoustic model parameters to optimize an objective function that reflects the confusions of the recognizer on the training data. We reviewed multiple commonly used training criteria and explained how to update the model parameters under discriminative training.

Chapter 3

Multi-level Acoustic Modeling

3.1 Motivation

In Chapter 2, we described how an HMM-based ASR system utilizes an acoustic model in its decoding process to score each acoustic observation (feature vector) $\mathbf{x}$ with respect to each phonetic state s. For the rest of this thesis, we use $a_{\boldsymbol{\lambda}}(\mathbf{x}, s)$ to denote the acoustic score of $\mathbf{x}$ with respect to the state s. In a conventional setting, a GMM is associated with each state, and the log-likelihood of $\mathbf{x}$ with respect to the GMM is used as its acoustic score. Let $l_{\boldsymbol{\lambda}}(\mathbf{x}, s)$ denote the log-likelihood of $\mathbf{x}$ with respect to the GMM associated with state s.

For modern ASR systems, context-dependent acoustic models are typically used to model coarticulatory variations during the speech production process. Under the context-dependent modeling framework, the set of states is expanded such that the scoring of an acoustic observation can depend on its nearby phonetic contexts. Because the number of states can become very large when contexts are considered, many states might not have enough training examples to train robust GMM parameters, resulting in a data sparsity problem.

To handle the data sparsity problem, a clustering algorithm is commonly used to group the states into clusters such that each cluster can have enough training examples. Although clustering deals with data sparsity, it also introduces a quantization effect; that is, the states within a cluster share the same GMM parameters, and therefore always have the same acoustic scores, becoming acoustically indistinguishable from each other. For a conventional large-vocabulary ASR system, the number of clusters is generally in the range between $10^3 \sim 10^4$, while the number of context-dependent states can be larger than 10^5. Therefore, the effect of quantization is significant and can potentially limit the power of a context-dependent acoustic model.

To exploit the advantages of context-dependent acoustic modeling, both the data sparsity and the quantization effect have to be dealt with appropriately. In the following section, we use an output code concept to illustrate the problem of the conventional modeling framework, and provide some insight on how to address the problems of data sparsity and quantization.

3.1.1 An Output Code Illustration

An output code [26, 108] is an effective technique for multi-class classification problems. Under the output code framework, a complex multi-class classification problem is broken down into a set of simpler, and more basic classification problems. When doing classification, the outputs of the basic classifiers are combined via an output code matrix to provide classification scores for the classes in the original multi-class problem.

If we think of each context-dependent state as a class, and the GMMs as basic classifiers, we can use an output code matrix to describe the relations between the acoustic scores and the GMM log-likelihoods in the conventional acoustic modeling framework. Figure 3-1 illustrates a simplified 3-state example. If each state has enough training examples, each state has its own GMM. Therefore, the corresponding output code matrix is an identity matrix, as shown in Figure 3-1(a). However, if certain states do not have enough examples, clustering is used to group the states. Because the states within a cluster share the same GMM, the corresponding rows in the output code matrix become identical, and cause the quantization effect, as shown in Figure 3-1(b).

Therefore, under the output code interpretation, to deal with data sparsity is to ensure each basic classifier has enough training examples, and to deal with the quantization effect is to ensure that there are no identical rows in the output code matrix. In the following section, we discuss several practical concerns when designing an output code framework

$$\begin{bmatrix} a_\lambda(\mathbf{x}, s_1) \\ a_\lambda(\mathbf{x}, s_2) \\ a_\lambda(\mathbf{x}, s_3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_\lambda(\mathbf{x}, s_1) \\ l_\lambda(\mathbf{x}, s_2) \\ l_\lambda(\mathbf{x}, s_3) \end{bmatrix}$$

(a) No clustering of states.

$$\begin{bmatrix} a_\lambda(\mathbf{x}, s_1) \\ a_\lambda(\mathbf{x}, s_2) \\ a_\lambda(\mathbf{x}, s_3) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} l_\lambda(\mathbf{x}, c_1) \\ l_\lambda(\mathbf{x}, c_2) \end{bmatrix}$$

(b) s_2 and s_3 grouped into the cluster c_2.

Figure 3-1: An output code illustration of the relations between acoustic scores and GMM log-likelihoods in a conventional acoustic modeling framework. Clustering is used to deal with data sparsity, but it also results in identical rows in the output code matrix, causing a quantization effect.

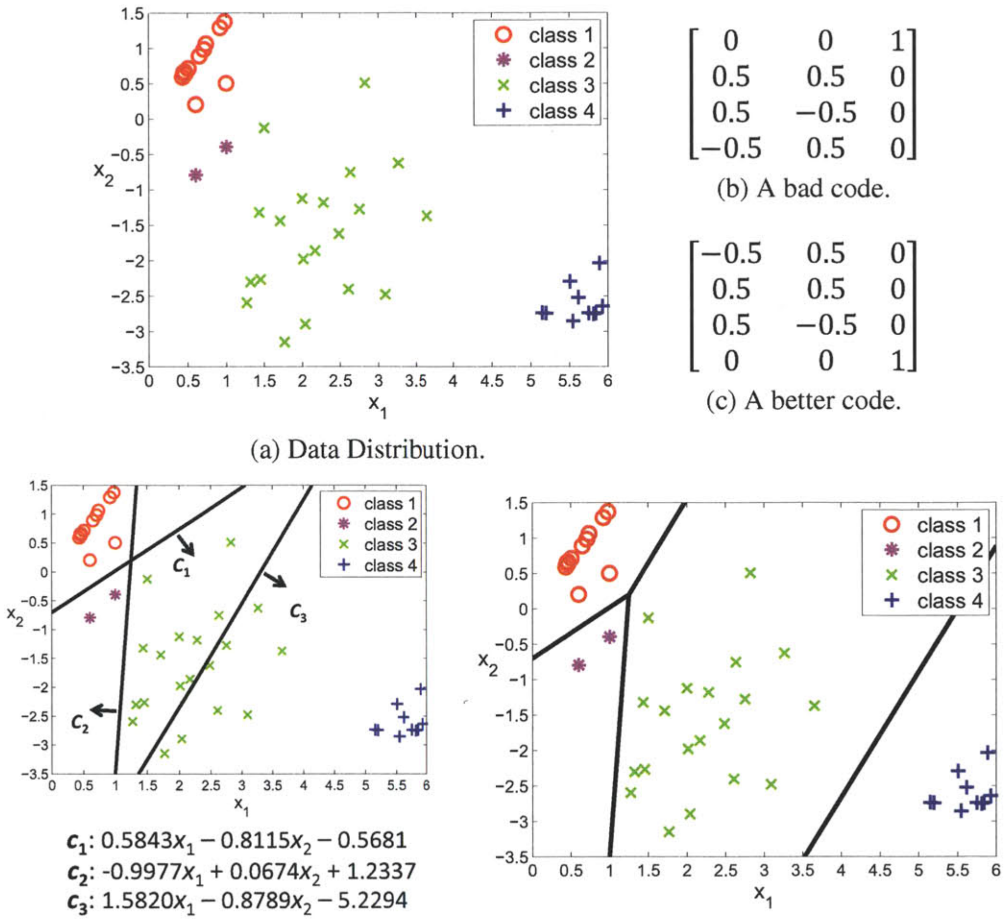
for context-dependent acoustic modeling.

3.1.2 Design Considerations

Under the output code framework, each context-dependent state is treated as an individual class, and let $\{s_1, \ldots, s_K\}$ be the set of states. To provide scores for the states, a set of L basic classifiers $\{c_1, \ldots, c_L\}$ is trained. Given an acoustic observation, the outputs of the basic classifiers are combined through a $K \times L$ output code matrix $\mathbf{M}$ to generate the scores for the states.

Designing a well-functioning output code matrix and its corresponding classifiers, however, is not trivial. Since the number of states, K, is large, there is a huge space of potential classifier choices. Moreover, as shown in Figure 3-2, one output code may be more suitable than another depending on the data. To function well, the design of an output code should capture some relationships between different classes of the data. In the following paragraphs, we discuss the relationships between the output code matrix, the classifiers, and the training data in more detail, and point out several practical design considerations.

If an entry M_{ij} is non-zero, the basic classifier c_j contributes to the score for the state s_i, and thus is associated with the state s_i. Because K (and L) can be large, when designing the output code, each state should only have a few associated classifiers to keep non-zero entries of $\mathbf{M}$ sparse. In theory, the value of M_{ij} can be negative as shown in Figure 3-2.



(a) Data Distribution.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0.5 & 0.5 & 0 \\ 0.5 & -0.5 & 0 \\ -0.5 & 0.5 & 0 \end{bmatrix}$$

(b) A bad code.

$$\begin{bmatrix} -0.5 & 0.5 & 0 \\ 0.5 & 0.5 & 0 \\ 0.5 & -0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(c) A better code.

$\boldsymbol{c_1}$: $0.5843x_1 - 0.8115x_2 - 0.5681$

$\boldsymbol{c_2}$: $-0.9977x_1 + 0.0674x_2 + 1.2337$

$\boldsymbol{c_3}$: $1.5820x_1 - 0.8789x_2 - 5.2294$

(d) Linear classifiers.

(e) Decision boundaries with the code in (c)

Figure 3-2: A 4-class example of designing an output code matrix with linear classifiers. Looking at the data in (a), we can see that there exist linear classifiers that can separate class 1 from class 2 and 3, class 1 and 2 from class 3, and class 3 from class 4. However, it is not possible for a linear classifier to correctly separate class 2 and 4 from class 3. In this sense, the output code in (b) is a bad code because the classifier corresponding to the second column seeks to perform the latter infeasible separation (i.e. the classifier has positive weight associated with class 2 and 4, treating their data as positive examples, and has next weight associated with class 3, treating its data as negative examples). On the other hand, the output code in (c) functions well for the data in (a). For example, the linear classifiers in (d) can be combined with the code in (c) to successfully classify the data, as shown in (e). Note that the two codes in (b) and (c) are the same except for the swapping of the first and the last rows. Therefore, the design of an output code matrix should not be independent of the data, and some guidance on the relationships between different classes of the data is needed. For speech, this is where knowledge such as broad classes can be helpful.

In such a situation, the examples of s_i are counted as negative training examples of the classifier c_j. However, when designing the output code for a context-dependent acoustic model, it is more convenient to constrain all entries to be non-negative for several reasons. First, the number of states is very large, and it is not obvious how to choose a small set of competing states for a particular state. Second, the ML training of GMMs only requires positive examples. Third, effective negative training examples can be considered in the parameter update via discriminative training procedures. To normalize the score of each state, we also require that each row of $\mathbf{M}$ sums to 1.

To avoid data sparsity, each basic classifier c_j should have enough training examples. If a state s_i does not have enough training examples, it should be grouped with other states to train each of its associated classifiers. However, the grouping of states should not be done arbitrarily; states grouped together should be similar with respect to certain perspective of speech, to avoid the situation shown in Figure 3-2(b). To ensure that there are no identical rows in the output code matrix, if both s_i and s_k are associated with a classifier c_j, there should be another classifier c_l that is associated with only one of s_i or s_k. In this way, each state will have a unique set of associated classifiers[1], and thus avoid the quantization effect. If s_i has enough training examples, a basic classifier can be trained directly for the occurrences of s_i. Note that even if s_i has enough data of its own, it may be beneficial to still include some other classifiers that are associated with s_i and other states, for the following reasons. First, the data of s_i may be helpful to learn a classifier for other states. Second, additional classifiers create a certain amount of redundancy, and redundancy can potentially help error correction based on communication theory [103], and might provide better generalization to novel test data.

We can summarize the design considerations of an output code framework for context-dependent acoustic modeling as follows.

- $M_{ij} \geq 0$ and $\sum_j M_{ij} = 1$.

[1] In theory, if two states have same set of associated classifiers but different combination weights, the two states can still be distinguishable. However, it only provides a weaker form of distinction, and the degree of distinction depends on how different the weights are. If the weights are to be learned automatically via some objective function, ensuring the weights of the states are different enough would make the optimization more complex. Ensuring that each state is associated with a unique set of classifiers is an easier and a more effective way of avoiding the quantization effect.

- $\mathbf{M}$ is sparse, but with a certain degree of redundancy to help error correction.
- No rows of $\mathbf{M}$ are identical to prevent the quantization effect.
- Each classifier c_j has enough training examples to avoid data sparsity.
- The training data for each classifier should be similar with respect to certain aspect of speech, to avoid the situation shown in Figure 3-2.

To achieve the above design considerations, one important question to answer is how to appropriately group each context-dependent unit with different sets of other units under different perspectives of speech such that the set of classifiers associated with the unit is unique among other units. One way of doing so is to group the context-dependent units with respect to different parts of local contexts. For example, the triphone "d-iy+n" (as in "d<u>ea</u>n") can be grouped with other triphones that have the center phone unit "iy" and the right context "n", such as the triphone "t-iy+n" (as in "t<u>ee</u>n"), to build a classifier that targets "iy" with right context "n". Similarly, another classifier that targets "iy" with left context "d" can also be constructed. Note that while each of the two classifiers is associated with multiple triphones, the triphone "d-iy+n" is the only triphone that is associated with both classifiers.

The notion of broad classes [57, 105] is also helpful in terms of constructing the classifiers. While each basic phonetic unit used in a language has its unique sets of properties, a broad class captures a perspective of speech, such as pronunciation manner or place of articulation, that is invariant among a subset of phonetic units. In general, the invariance captured by a broad class is relatively easier to model with higher accuracy [57], even in noisy conditions [105]. Broad classes can be used to group the contexts under different perspectives of speech. Take the triphone "d-iy+n" for example. Its center unit can belong to "Front_Vowel", "High_Vowel", or more broadly "Vowel"; its left context can belong to "Stop_Consonant", "Alveolar", or "Voiced_Consonant"; its right context can belong "Nasal_Consonant" or "Alveolar" or "Sonorant". Also, appropriately combining perspectives from different broad classes can help disambiguate confusions. For example, in English, if we can be sure that the left context belongs to both "Stop_Consonant" (based on pronunciation manner) and "Alveolar" (based on place of articulation), then the potential choice of the left context is reduced to either "d" or "t". Utilizing broad classes can provide a robust and guided way to group the context-dependent units when designing the output code.

In addition to ensuring each row of the output code distinct, maintaining the non-zero entries in the output code matrix sparse and compact is also desirable. This can be done by constructing a systematic, hierarchical structure of classifiers as described in the next section. Once the classifiers are built, the next step is to assign the combination weights of the classifiers. If the classifiers form a symmetric structure, an effective heuristic weight assignment can be applied, as shown in the next section. The combination weights can also be learned through discriminative training criteria, as shown in Section 3.2.3.

3.2 Multi-level Acoustic Model

Based on the above considerations, we propose a multi-level acoustic modeling framework. Under this multi-level framework, each context-dependent state is associated with a unique set of GMMs that target multiple levels of contextual resolution, and the log-likelihoods of the GMMs are linearly combined for scoring during decoding. The following sections introduce details of the proposed multi-level framework. We first use the commonly-used triphone-based model as an example to illustrate the multi-level idea, and then show how the framework can be expanded (or reduced) to model contexts of longer (or shorter) length.

3.2.1 Multi-Level Notations

The proposed multi-level framework involves grouping data and training of GMMs at multiple levels of contextual resolution. In this section, we define some common multi-level notations used throughout this thesis for convenience and reference. While the grouping of data and the training of classifiers happens at the HMM state-level, we use a phone-level notation for the sake of clarity. Extending the notation to the state-level is straightforward by using state-level alignment during the training.

- "p_l-p_c+p_r": The label denotes a triphone that has left context (previous phone label) "p_l", current phone label "p_c", and right context (next phone label) "p_r". For each triphone, we build multiple sets of classifiers targeting different levels of contextual resolution.

- $\langle p_l, p_c, p_r \rangle$: The label of a classifier that directly models the occurrences of the triphone "p_l-p_c+p_r". The label of a classifier is always enclosed by angle brackets.

- "*": This symbol is used when the classifier ignores certain contexts. For example, $\langle p_l, p_c, * \rangle$ denotes the label of a classifier that models the occurrences of "p_c" with left context "p_l" but with right context ignored.

- $B(\cdot)$: This function is used to reduce a phoneme into a broad class. In general we can associate each phonetic unit with one or more broad classes. For example, the phoneme "n" in the word "<u>n</u>oise" can belong to the broad class of "Nasal_Consonant". Under the broad class notation, $\langle B(p_l), p_c, * \rangle$ represents the label of a classifier that models the occurrence of "p_c" with left context belonging to the broad class "$B(p_l)$".

- $T_{ij}(\cdot)$: Each triphone is associated with multiple classifiers with different levels of contextual resolution. The function $T_{ij}(\cdot)$ is used to denote the label of the j^{th} classifier at the i^{th} contextual resolution level. A smaller value of i refers to a finer contextual resolution. For example, T_{11}("p_l-p_c+p_r") $= \langle p_l, p_c, p_r \rangle$, the label of the classifier targeting the finest level of contextual resolution.

3.2.2 Model Formulation

Basic Classifiers

Figure 3-3 shows the basic classifiers associated with the triphone "p_l-p_c+p_r" in the proposed framework. The classifier at the top level has the finest contextual resolution, but the fewest number of training examples. On the other hand, the classifiers at the bottom level have coarser contextual resolution, but the highest number of training examples. Except for the top level classifier, the other classifiers are also associated with other triphones.

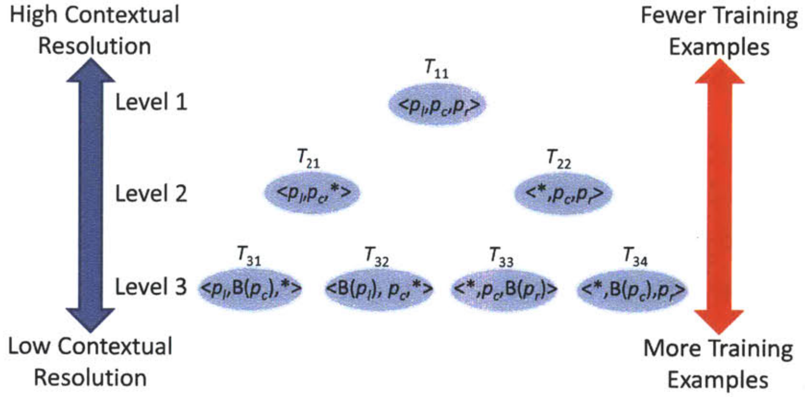


Figure 3-3: Basic classifiers for triphone "p_l-p_c+p_r".

Although each classifier has multiple associated triphones, in the proposed design each triphone has a unique set of basic classifiers (a more concrete example will be shown later). For each triphone, if an upper level classifier does not have enough training example, only the lower level ones will be used for scoring, and thus avoiding the data sparsity problem. In general, reducing the contextual resolution to Level 3 illustrated in Figure 3-3 is sufficient to obtain enough training examples for the triphone states appeared in the large-vocabulary recognition experiments reported in the thesis. If data sparsity still occurs at Level 3, the contextual resolution can be further reduced, and classifiers such as $\langle *, p_c, * \rangle$ or $\langle p_l, *, * \rangle$ can be used for scoring.

Scoring

For each classifier shown in Figure 3-3, a GMM can be trained for scoring. Given an acoustic feature vector $\mathbf{x}$, the log-likelihoods of $\mathbf{x}$ with respect to the basic classifiers of a triphone s are linearly combined to compute the acoustic score,

$$a_{\boldsymbol{\lambda}}(\mathbf{x}, s) = \sum_{i=1}^{3} \sum_{j=1}^{J_i} w_{ij}^{s} l_{\boldsymbol{\lambda}}(\mathbf{x}, T_{ij}(s)), \tag{3.1}$$

where J_i is the number of classifiers at level i, and the combination weight w_{ij}^s satisfies the normalization constraint $\sum_{i=1}^{s} \sum_{j=1}^{J_i} w_{ij}^{s} = 1$. For the actual values of the weights, we can use the default assignments based on the number of training examples as shown in Figure

3-4. Instead of using the default assignments, the combination weights can also be learned automatically, which will be discussed in later part of the chapter.

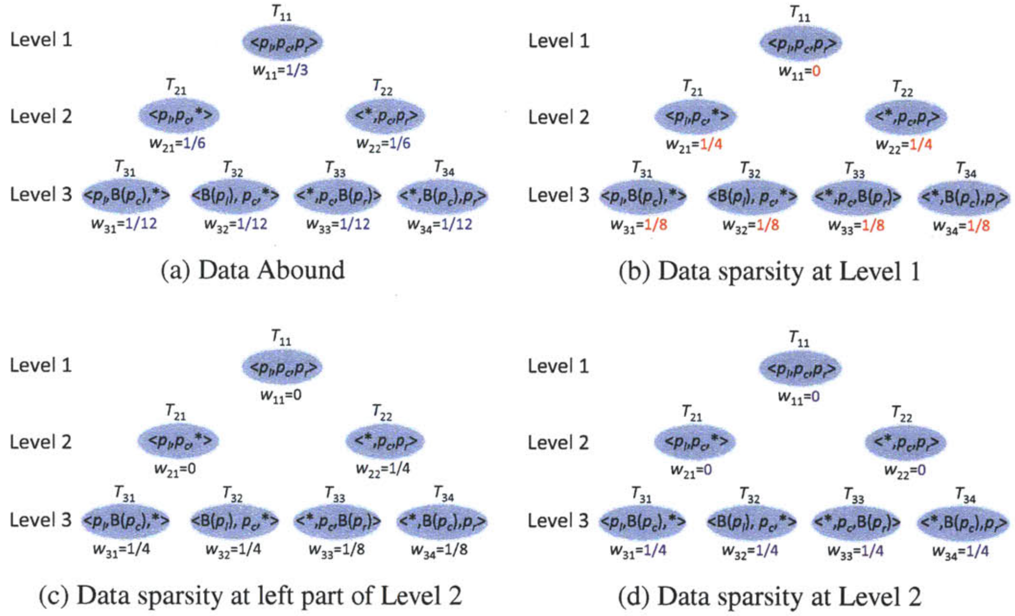
Properties

Under the proposed multi-level framework, each triphone is guaranteed to have a unique set of basic classifiers. This is because for any two different triphones s and s', they must differ at least in one of the following ways: left context, center phone, or right context. If they differ at the left context, the classifier $T_{31}(s)$ in Figure 3-3 is different from $T_{31}(s')$. Similarly, $T_{34}(s) \neq T_{34}(s')$ for a different right context, and $T_{32}(s) \neq T_{32}(s'), T_{33}(s) \neq T_{33}(s')$ for different center phone label. Since based on the default weight assignment in Figure 3-4, the weights for the classifiers at the bottom level are always non-zero, any two rows differ in at least one entry in the output code matrix. Figure 3-5 shows the sub-matrix of the output code matrix $\mathbf{M}$ where the entries of the two triphones "t-iy+n" (as in the word "teen") and "d-iy+n" (as in the word "dean") as a concrete example.

Also, if a triphone has enough training examples, all the classifiers in Figure 3-3 are used for scoring. Doing so creates some redundancy, and can potentially help error correction. On the other hand, if not enough data is available, only the lower level classifiers that have enough data are used for scoring. In this way, the data sparsity problem is avoided.

Remarks

While the design in Figure 3-3 is chosen for conducting experiments, the multi-level idea is not limited to the design. For example, classifiers such as $\langle p_l, p_c, B(p_r) \rangle$ and $\langle B(p_l), p_c, p_r \rangle$ can potentially be added to the scoring framework. There can be other ways of constructing classifiers as well. For example, the classifiers can be selected by a set of specifically designed decision trees such that the following property holds: For each lead node of a tree, and for each pair of the context-dependent units grouped in the node, there exists a leaf node in another tree that contains just one of the units. The basic principle is to make each context dependent unit be associated with a unique set of classifiers that have a reasonable number of training examples. Nevertheless, the design in Figure 3-3 represents a simple and effective realization of the multi-level idea.



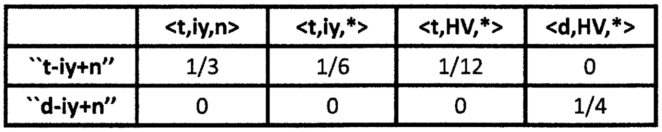
(a) Data Abound

(b) Data sparsity at Level 1

(c) Data sparsity at left part of Level 2

(d) Data sparsity at Level 2

Figure 3-4: Default assignment of combination weights. The intuition of the weight assignment is as follows. The set of classifiers at each of the three levels can uniquely specify the triphone "p_l-p_c+p_r"; therefore, the total weight assigned to each level is $1/3$ by default. By symmetry, the total weight assigned to each level is equally distributed to the corresponding classifiers, resulting in the weights shown in (a). If the top level classifier does not have enough data due to data sparsity, its weight is equally distributed to the other two levels, as shown in (b). If the left classifier at level 2 still does not have enough data, its weight is equally distributed to the left two classifiers at the bottom level, as shown in (c). Note that the two classifiers at the bottom level represent reductions of the classifier $\langle p_l, p_c, * \rangle$ with respect to one part of the contexts, and thus "inherit" the weight from $\langle p_l, p_c, * \rangle$. Finally, if all the upper level classifiers do not have enough data, each of the bottom level classifiers gets a $1/4$ weight, as shown in (d). The weights can also be automatically learned as shown in Section 3.2.3.



	<t,iy,n>	<t,iy,*>	<t,HV,*>	<d,HV,*>
“t-iy+n”	1/3	1/6	1/12	0
“d-iy+n”	0	0	0	1/4

Figure 3-5: Part of the output code matrix M where the rows for “t-iy+n” and “d-iy+n” differ. “HV” refers to the broad-class “High_Vowel”. Note that the two classifiers “⟨d,iy,n⟩” and “⟨d,iy,*⟩” do not have enough training examples, so do not show up in the output code matrix. The classifier “⟨d,HV,*⟩” inherits the weights from the two missing classifiers as illustrated in Figure 3-4(c).

3.2.3 Parameter Learning

Initializing GMMs

For each classifier, we can collect the acoustic observations that align with the classifier label in the references, and run the EM algorithm described in Section 2.1.5 on the collected observations to initialize the GMM parameters under the ML criterion. Unlike the decision tree clustering method where each acoustic feature is assigned to a single leaf node, in the proposed multi-level framework, each acoustic observation can contribute to multiple classifiers for training. For example, the acoustic feature vectors aligned with the phone “iy” in the word “teen” can be used to train ⟨t,iy,*⟩, ⟨t,HV,*⟩,⟨*,HV,n⟩,. . ., etc. In this way, the same data can be utilized with different perspectives, which can potentially benefit the overall model.

The Forward-Backward procedure described in the latter part of Section 2.1.5 can also be applied to further update the parameters. To derive the update equations, for each classifier c, let $w(c, s)$ be the combination weight of c used for scoring the context-dependent unit s. Take a classifier in Figure 3-5 for example, the weight $w($⟨t,iy,n⟩,“t-iy+n”$) = 1/6$. For each s, the posterior probability of s at time t, γ_{ts}, can be computed using Equation (2.38). For the m^{th} Gaussian mixture component of c, its posterior probability at time t, r^c_{tm}, can also be computed using Equation (2.25). With these two sets of posterior probabilities, we can compute the partial count of the m^{th} component of c over a training utterance by

$$\vartheta_{cm} = \sum_{t}^{T} r^c_{tm} \sum_{s:w(c,s)>0} w(c, s)\gamma_{ts}. \tag{3.2}$$

The intuition behind Equation (3.2) is as follows. Based on the Forward-Backward algorithm, and the EM theory described in Appendix B, the posterior probability γ_{ts} is the partial derivative of the ML training objective with respect to the acoustic score $a_{\boldsymbol{\lambda}}(\mathbf{x}_t, s)$. Since $a_{\boldsymbol{\lambda}}(\mathbf{x}_t, s)$ is a linear function of the GMM log-likelihood $l_{\boldsymbol{\lambda}}(\mathbf{x}_t, c)$, using the chain rule, s can contribute $w(c, s)\gamma_{ts}$ (if $w(c, s) > 0$) to the partial derivative of the ML training criterion with respect to $l_{\boldsymbol{\lambda}}(\mathbf{x}_t, c)$. By summing all the contributions from all s associated with c, we can have the last term in Equation (3.2) as the partial count of c at time t. After the partial count of c is computed, we can distribute the count to mixture components through the mixture posteriors as shown in Equation (3.2).

Similarly, the first and second order statistics of the mixture component can be computed by

$$\mathcal{O}_{cm}(\mathbf{X}) = \sum_{t}^{T} \mathbf{x}_t r_{tm}^{c} \sum_{s:w(c,s)>0} w(c, s)\gamma_{ts}. \tag{3.3}$$

$$\mathcal{O}_{cm}(\mathbf{X}^2) = \sum_{t}^{T} \mathbf{x}_t r_{tm}^{c} \sum_{s:w(c,s)>0} w(c, s)\gamma_{ts}. \tag{3.4}$$

After the three sets of statistics in Equation (3.2-3.4) are computed, we can use the update formula in Equation (2.42-2.44) to update the GMM parameters. To smooth the model, we can also interpolate the statistics from the Forward-Backward procedure with that from the initial model parameter before applying the update formula.

Discriminative Training of GMMs

To integrate the proposed multi-level acoustic modeling framework with discriminative training methods, we have to be able to compute the discriminative objective function under the proposed framework and be able to update the multi-level model parameters to optimize the objective function. The first part involves implementing the multi-level framework in an existing ASR system, which will be discussed later in the chapter. In this section, we focus on the second part; that is, how to update the multi-level model parameters according to the discriminative objective function. As described in Section 2.3, there are two main update methods for discriminative training: the Gradient-based method and the EBW-based method. We will show how to update the multi-level model parameters under each kind of update method.

For the gradient based method, we have to be able to compute the gradient of the discriminative training objective function with respect to multi-level model parameters. As in the discriminative training procedure for a conventional clustering-based acoustic model, we can first take partial derivatives of the objective function with respect to each acoustic score as described in Section 2.3:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\lambda}} = \sum_{n=1}^{N} \sum_{\mathbf{x} \in \mathbf{X}_n} \sum_{s} \frac{\partial \mathcal{L}}{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}, s)} \frac{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}, s)}{\partial \boldsymbol{\lambda}}, \tag{3.5}$$

where N is the number of training utterance, $\mathbf{X}_n$ is the sequence acoustic feature vectors corresponding to the n^{th} training utterance. Because the acoustic score in the multi-level framework is a linear combination of the log-likelihoods of GMMs, we can compute the last term in Equation (3.5) by

$$\frac{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}, s)}{\partial \boldsymbol{\lambda}} = \sum_{i=1}^{3} \sum_{j=1}^{J_i} w_{ij}^{s} \frac{l_{\boldsymbol{\lambda}}(\mathbf{x}, T_{ij}(s))}{\partial \boldsymbol{\lambda}}, \tag{3.6}$$

where the last term is the partial derivative of the log-likelihood of a GMM with respect to its parameters, which has a closed-form solution. Therefore, the computation of the gradient of the multi-level model is similar to that of the conventional acoustic model except for the extra step of distributing the gradient contributions back to each GMM according to the combination weights, as described in Equation (3.6). After the gradient is computed, a Quasi-Newton method such as the Quickprop algorithm [85] can be used to update the parameters. Details of the Quickprop algorithm can be found in Appendix G.

For the EBW based method, as described in Section 2.3, the key is to compute the numerator statistics for the reference and the denominator statistics for the competing hypotheses. As for the discriminative training of conventional acoustic models, the posterior probability γ_{ts}^{num} in the numerator lattice and the posterior probability γ_{ts}^{den} in the denominator lattice can be computed using the Forward-Backward algorithm. After the posterior probabilities are computed, we can distribute the partial counts represented by the posterior probabilities to the GMMs based on the combination weights as shown in Equation (3.2-

3.4), and obtain the numerator statistics $\{\vartheta_{cm}^{num}, \mathcal{O}_{cm}^{num}(\mathbf{X}), \mathcal{O}_{cm}^{num}(\mathbf{X}^2)\}$ and the denominator statistics $\{\vartheta_{cm}^{den}, \mathcal{O}_{cm}^{den}(\mathbf{X}), \mathcal{O}_{cm}^{den}(\mathbf{X}^2)\}$ for each GMM c, and each mixture component m. Once the statistics are computed, the update formula in Equations (2.59), (2.60), and (2.63) can be used to update the means, variances, and mixture weights respectively.

Learning Weights

Instead of using the default weight assignment described in Figure 3-4, we can also seek to learn the combination weights automatically from data. If we fix the GMM parameters, we can represent the discriminative training objective function as a function of the combination weights. More specifically, let $\overline{\mathbf{W}} = \{\overline{w}_{ij}^s\}$ be the set of default combination weights, and let $\mathbf{W} = \{w_{ij}^s\}$ be the set of weights to be learned. We can use a constrained optimization to update the weights $\mathbf{W}$ as follows:

$$\begin{aligned}
&\text{minimize}_{\mathbf{W}} \mathcal{L}(\mathbf{W}) \\
&\text{subject to} \\
&\sum_{i,j} w_{ij}^s = 1 \;\; \forall s, \\
&w_{ij}^s \geq 0 \;\; \forall s, i, j, \\
&w_{ij}^s = 0 \;\; \forall \overline{w}_{ij}^s = 0,
\end{aligned} \tag{3.7}$$

where the term $\mathcal{L}(\mathbf{W})$ is the loss function of discriminative training represented as a function of the weights. In terms of the constraints, the first two sets of constraints ensure that the acoustic score is a convex combination of GMM log-likelhooods. The third set of constraints ensures that no classifier with an insufficient number of training examples will be used for scoring and thus prevents the data sparsity effect.

Because the gradient of the loss function with respect to the weights is relatively easy to compute, and because the constraints of Equation (3.7) form a polyhedra in the parameter space, we can use the gradient projection method described in [12] to update the weights. Figure 3-6 illustrates how the gradient projection method works. At each iteration of the gradient projection method, the parameters are updated to new values using the gradient-

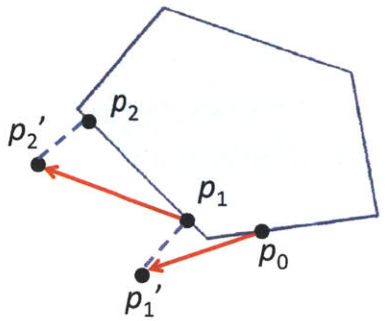


Figure 3-6: Illustration of the gradient projection method. Each red arrow represents an unconstrained gradient-based update, and each blue dashed line represents the projection back to the polyhedra.

based update as if there are no active constraints at the current point. Note that such an unconstrained update might potentially violate one or more constraints, and thus the resulting point might not be feasible. To find a feasible point, we can project the newly updated point back to the polyhedra, where the projection of a point p' with respect to the polyhedra is the point p in the polyhedra that is closest to the point p'. Because for any pair of states, s and s', there is no constraint that ties w^s_{ij} and $w^{s'}_{i'j'}$, the projection can be done separately for each s. Since there is only one sum to 1 constraint for each s, and since the second and the third sets of constraints are easy to check, the projection can be done fairly efficiently. (Basically, moving the free variables along the norm of the violated constraint, until hitting the polyhedra.)

To utilize the gradient projection method, we have to compute the partial derivative of the objective function in Equation (3.7) with respect to each weight w^s_{ij}. Since the GMM parameters are fixed during the optimization of the weights, the acoustic score $a_{\boldsymbol{\lambda}}(\mathbf{x}, s)$ is a linear function of the weight w^s_{ij}. Therefore, as for the discriminative training of GMMs, we can first take a partial derivative of the loss function with respect to the acoustic score, and then distribute the contribution to the weight as follows:

$$\frac{\partial \mathcal{L}}{\partial w^s_{ij}} = \sum_{n=1}^{N} \sum_{\mathbf{x} \in \mathbf{X}_n} \frac{\partial \mathcal{L}}{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}, s)} l_{\boldsymbol{\lambda}}(\mathbf{x}, T_{ij}(s)), \tag{3.8}$$

where the GMM log-likelihood $l_{\boldsymbol{\lambda}}(\mathbf{x}, T_{ij}(s))$ is fixed as a constant during the optimization of the weights.

One final remark is that in the optimization of Equation (3.7), each context-dependent unit s is associated with a separate set of weights w_{ij}^s that can be changed during the optimization. However, because many units may have only a small number of examples in the data, to increase the generalization ability of the weight learning, we can tie weights of different context-dependent units during the optimization. One way of doing the grouping is via the broad phonetic classes, and each triphone "p_l-p_c+p_r" can be reduced to a broad class triple "$B(p_l)$-$B(p_c)$+$B(p_r)$". For example, the triphone "d-iy+n" in the word "d$\underline{\text{ea}}$n" can be reduced to "SC-HV+N", where "SC" denotes the broad class of Stop Consonant, "HV" denotes High Vowel, and "N" denotes Nasal. The weights of s and s' can be tied if they both belong to the same broad class triple, and the default weights $\overline{w}_{ij}^s = \overline{w}_{ij}^{s'}$ $\forall i, j$. Note that the tying of weights does not create a quantization effect because each s still has a unique set of GMM classifiers. Let b denote the label of a weight group after the tying, the gradient with respect to the tied weight w_{ij}^b can be computed by

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^b} = \sum_{n=1}^{N} \sum_{\mathbf{x}\in\mathbf{X}_n} \sum_{s:\rho(s)=b} \frac{\partial \mathcal{L}}{\partial a_{\boldsymbol{\lambda}}(\mathbf{x}, s)} l_{\boldsymbol{\lambda}}(\mathbf{x}, T_{ij}(s)), \qquad (3.9)$$

where $\rho(s)$ is the mapping function from the context-dependent unit s to a weight group according to the tying. In terms of the projection, we can project the weights separately for each weight group b, similarly as in the original problem.

3.2.4 Potential Extensions

In the previous sections, we introduced the basic modeling framework of the multi-level model. In this section we discuss several potential extensions of the basic multi-level framework.

Multiple Broad Classes Extension

In Figure 3-3, a set of broad classes are used to reduce the contextual resolution and construct the classifiers in the bottom level. However, a phoneme can belong to different kinds

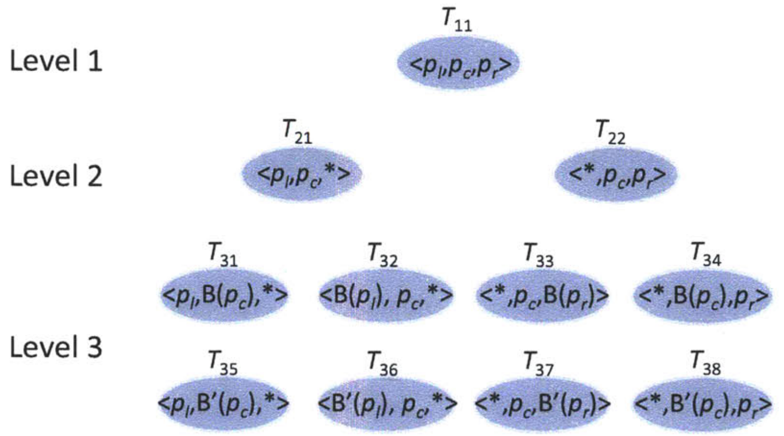
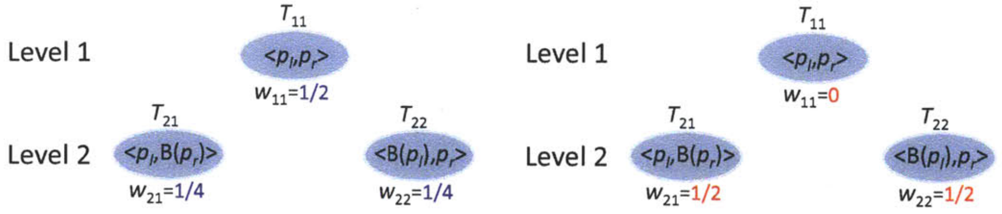


Figure 3-7: Extended classifiers for triphone "p_l-p_c+p_r". $B(\cdot)$ and $B'(\cdot)$ represent the broad class assignment based on two different aspects of acoustic-phonetic knowledge. For example, B("d") can be "Stop_Consonant" ("SC") and B'("d") can be "Alveolar" ("AL").

of broad classes under different perspectives. For example, the phoneme "d" can belong to the broad class "Stop_Consonant" based on production manner, but it can also belong to the broad class "Alveolar" based on place of articulation. Jointly considering different aspects of acoustic-phonetic knowledge can help disambiguate the phonemes, and can potentially help improve the model. The proposed multi-level framework can be extended by incorporating multiple sets of broad classes assignments in the bottom level as shown in Figure 3-7. In terms of the default weight assignment, since there are more classifiers at the bottom level in the multiple broad-classes extension, it makes more sense to assign more weight to the bottom level. For the classifier configuration in Figure 3-7, one reasonable way to assign the weights is to scale the weights in Figure 3-4 by $\frac{3}{4}$ and make $\{w_{3j} = w_{3j'},\ \ \forall 1 \leq j \leq 4,\ \ j' = j + 4\}$. As discussed in Section 3.2.2, there can be other ways of considering multiple sets of broad classes, but the design in Figure 3-7 shows an effective realization.

Modeling Different Length of Contexts

While the previous sections focus on the multi-level framework for triphone based system, the proposed multi-level framework can be modified to model contexts of different length.



(a) Data Abound

(b) Data sparsity at Level 1

Figure 3-8: Basic classifiers and combination weights for diphone based system.

For example, a two level framework can be used to model diphones as illustrated in Figure 3-8. The framework can also be extended to model longer contexts. For example, the acoustic score of a quintphone "p_{ll}-p_l-p_c+p_r+p_{rr}" can be decomposed by

$$\begin{aligned} a_{\boldsymbol{\lambda}}(\mathbf{x}, \text{“}p_{ll}\text{-}p_l\text{-}p_c\text{+}p_r\text{+}p_{rr}\text{”}) &= \upsilon_0 l_{\boldsymbol{\lambda}}(\mathbf{x}, \langle p_{ll}, p_l, p_c, p_r, p_{rr}\rangle) \\ &\quad + \upsilon_l a'_{\boldsymbol{\lambda}}(\mathbf{x}, \text{“}p_{ll}\text{-}p_l\text{-}p_c\text{”}) + \upsilon_r a'_{\boldsymbol{\lambda}}(\mathbf{x}, \text{“}p_c\text{+}p_r\text{+}p_{rr}\text{”}), \end{aligned} \tag{3.10}$$

where the scores of the two shifted triphone $a'_{\boldsymbol{\lambda}}(\mathbf{x}, \text{“}p_{ll}\text{-}p_l\text{-}p_c\text{”})$ and $a'_{\boldsymbol{\lambda}}(\mathbf{x}, \text{“}p_c\text{+}p_r\text{+}p_{rr}\text{”})$ can be computed using the triphone based multi-level framework, and the weights combination weights satisfy the normaliztion constraint $\upsilon_0 + \upsilon_l + \upsilon_r = 1$. If the quinphone has enough examples, υ_0 can be greater than 0; otherwise, it is set to 0 to avoid data sparsity.

3.3 Implementation Issues

3.3.1 Decoding

In this section, we discuss how to implement the multi-level acoustic model in the decoding framework of an existing large-vocabulary ASR system. While the actual implementation details may be complicated and might vary depending on the existing system, here we seek to present several implementation principles that can generalize across different existing systems. In order to do so, we first discuss some common properties of conventional large-vocabulary ASR decoding frameworks.

Conventional Decoding Frameworks

In the decoding frameworks of conventional large-vocabulary ASR systems, not all states are evaluated by the search module at any given point of time. The state space of a large-vocabulary ASR system can be large, and searching through all states at all times can be costly in terms of both running time and memory space. For computational efficiency, an active frontier that consists of a subset of states is kept at each point of time, and only the next states from the active frontier are evaluated during decoding. Typically, certain pruning criteria, such as the score difference from the current best score or the total number of active states, are commonly used to maintain the active frontier to be a reasonable size. Since only a subset of states are evaluated at each point of time, not all acoustic scores are needed during decoding, and the corresponding GMM log-likelihoods can be computed as needed.

Computing the log-likelihood of a GMM is a relatively costly operation compared with other operations needed during decoding. Computing the log-likelihood of a GMM requires a sequence of steps: computing the difference between the acoustic observation and the mean for each mixture component, multiplying the difference by an inverse variance, computing inner products, and taking the log-sum of the inner products. Compared with other operations needed in the Viterbi decoding algorithm (as described in Algorithm 2.1), which are addition, indexing, and tracking the maximum value, computing the log-likelihood of a GMM is typically the most computationally intensive operation in commonly available computational architectures.

The log-likelihood of a GMM might be accessed multiple times. Because of clustering, multiple context-dependent states can be associated with the same GMM for scoring. In addition, commonly-used post processing procedures, such as searching for N-Best hypotheses and computing posterior probabilities using the Forward-Backward algorithm, may also need to access the GMM log-likelihoods at different points of times. Since computing GMM log-likelihoods is costly, it is usually beneficial to store the computed log-likelihoods for future access.

Because of the above facts, although the detailed implementations may differ from system to system, it is still reasonable to assume that there is a lazy evaluation procedure for GMMs in the system. More specifically, when the log-likelihood of a GMM classifier c is needed at time point t during decoding, the decoding module calls the lazy evaluation procedure with the index c as one of its inputs. The lazy evaluation procedure then checks if the log-likelihood of c has been computed. If yes, the procedure accesses the memory location that stores the log-likelihood and returns the value; if not, the procedure accesses the acoustic observation $\mathbf{x}_t$, computes the log-likelihood $l_{\boldsymbol{\lambda}}(\mathbf{x}_t, c)$, stores the value at its designated memory location, and returns back the value. In the following sections, we assume the lazy evaluation procedure is built in to the existing system, and can be utilized when implementing the multi-level acoustic modeling framework. For convenience, we use LAZY_EVAL to denote the lazy evaluation procedure for computing GMM likelihood.

Multi-Level Implementation

There are several practical considerations when implementing the multi-level framework. For example, the implementation should not require too much computational overhead. Usage wise, it is preferred that the system is implemented in a way that is able to perform decoding under both the conventional framework and the multi-level framework. Coding wise, the system should reuse existing software when possible, and avoid extra memory management or drastic software changes.

Computationally, the cost of computing the acoustic score $a_{\boldsymbol{\lambda}}(\mathbf{x}_t, s)$ under the multi-level framework consists of two parts: the cost of computing the log-likelihood $l_{\boldsymbol{\lambda}}(\mathbf{x}_t, c)$ for each GMM c associated with the context-dependent state s, and the cost of computing a linear combination of the log-likelihoods. Since the LAZY_EVAL procedure is built-in, reusing the procedure for the log-likelihood computation can keep the implementation succinct. Also, the values of log-likelihoods can be stored, and can be used for future access, with all memory management issues being taken care of. One remaining question is whether we should also store the value of the acoustic score for future access. Storing the acoustic score can save the need for performing the linear combination in future access. However, because each context-dependent state s has a different score and the total number of states is large, managing the space to store the acoustic scores is not trivial and might not be as fast as expected[2]. Since the linear combination only involves additions and multiplications of a few numbers, recomputing the linear combination each time the acoustic score is accessed might only require a slightly more computational overhead. However, it can keep the overall implementation succinct and more memory efficient, compared with the one where the acoustic scores are stored.

To realize this idea, we can create a new procedure MULTI_LAZY_EVAL that accepts the same set of arguments as LAZY_EVAL and works as follows. MULTI_LAZY_EVAL first checks if the system is running under the conventional mode, or under the multi-level mode. If it is in the conventional mode, it passes the arguments to LAZY_EVAL and returns the log-likelihood it computes. If it is in the multi-level mode, MULTI_LAZY_EVAL treats the integer argument that originally represents the index of the GMM classifier for LAZY_EVAL as the index of the context-dependent state s. The procedure then calls LAZY_EVAL with the index of each GMM classifier that is associated with s, obtains the log-likelihoods, computes the linear combination, and then returns the value. Note that by setting the conventional mode as default, even if we replace every calling of LAZY_EVAL with MULTI_LAZY_EVAL, the modified decoder can still behave exactly the same as the original decoder, which is preferred in terms of usage.

With the MULTI_LAZY_EVAL procedure in mind, the main steps of the proposed multi-level implementation can be summarized as follows. First, modify the decoder module such that it can load in the tables that specify indices of GMMs, and combination weights for the linear combinations, and make MULTI_LAZY_EVAL accessible to the tables. Second, for each location in the original source code where LAZY_EVAL is called, replace it with the new procedure MULTI_LAZY_EVAL. In general, the locations where changes are needed are the forward procedure of Viterbi algorithm, the re-scoring procedure for higher-order language model, searching for N-Best paths, and the Forward-Backward procedure for computing lattices and posteriors. Third, depending on systems and coding styles, sometimes in the back-tracing step, or in some post-processing procedures (where the GMM

[2] Because of the large number of states, creating a static array for the acoustic scores at each time point can require a large amount of memory ($\sim 10^5$ states times $\sim 10^3$ observations per utterance). While there are several dynamic data structures, such as vector, or binary search tree, they often have trade-off between the time cost to insert an element and the time cost to retrieve an element. Initialization and reclamation of the data structure also takes time, and has to be done for every utterance.

log-likelihoods are supposedly computed), the GMM log-likelihoods are not accessed via `LAZY_EVAL` but by directly referencing the memory locations. These cases should be replaced by a call to `MULTI_LAZY_EVAL`, or some sub-routine that accesses the memory locations of log-likelihoods and computes the linear combination. Finally, under the conventional framework, there is a mapping table that maps the context-dependent state s to its associated GMM classifier c. Because the `MULTI_LAZY_EVAL` has taken care of the linear combination, such mapping table is no longer needed, and can be replaced with an identity mapping under the multi-level framework. Note that the state to GMM mapping table should be part of the input to the recognizer (not in the source code), and may be in a different location, or in a different form, depending on the system implementation. For example, in the FST-based recognizer we used to conduct the experiments, the table is embedded in the input labels of the C $\circ$ P $\circ$ L $\circ$ G FST. While here we only provide the basic idea of implementing the multi-level framework, the actual implementation details would be more complex and strongly depend on the existing system; they probably would be a good software engineering exercise!

Debugging Tips

Because the decoding framework of a large-vocabulary ASR system has to incorporate many practical engineering issues (e.g. pruning, re-scoring, memory management...) and has to be able to generate various kinds of recognition outputs (e.g. N-best hypotheses, lattices, phone alignments, state posterior probabilities...), the source code for the decoder is often large and complex. As a result, it would be non-trivial to check if all the places that need to be changed have been changed. To help debug the multi-level implementation, we can conduct a sanity check as follows.

After the first pass of modification is done, we can randomly permute the order of the GMMs stored in the acoustic model file, and at the same time, construct a matching table that can restore the random permutation. If the implementation was correct, when we switch the decoder to the multi-level mode and let the decoder accept the randomly permuted GMMs as an acoustic model, the matching table as the table for linear combination, and a combination weight vector $\mathbf{W} = \mathbf{1}$, the decoder should generate identical results as in the conventional mode, regardless of the input/output specifications. If the code passes the random permutation test, with high probability, the implementation is in good shape. Note that there are many kinds of outputs, and the modified decoder has to pass the random permutation for each kind of them. When doing the test, we suggest to begin testing with a short utterance (and probably with forced alignment) before testing on a larger set. In this way, the initial tests can run faster and can also identify the problems more efficiently. Bon débogage!

3.3.2 Discriminative Training

As described in Section 2.3.3, refining the acoustic model parameters with a discriminative training criterion requires several iterations of the following steps: First, to recognize the training utterances in both the forced-alignment and the recognition mode to generate recognition outputs, such as alignments, N-best hypotheses, lattices, and state posteriors. Then, process the recognition outputs to accumulate gradient (for gradient-based update) or sufficient statistics (for EBW-based update). Finally, use the gradient or sufficient statistics to update model parameters. If the multi-level decoding framework described in Section 3.3.1 is implemented correctly, the recognition outputs generated under the multi-level mode should be in correct format and ready to be used. In this section, we discuss some implementation and debugging tips for the accumulator that processes the recognition outputs and accumulates gradient, or sufficient statistics, to be used for the parameter update of the multi-level acoustic model.

Although the actual format will differ depending on the implementation, in general, the function of the accumulator can be summarized as follows. First, for each state s appearing in the recognition outputs at time t, compute a scaling factor α_{ts} that reflects the importance of s at time t based on the discriminative training objective function[3]. Second, compute the posterior probability r^s_{tm} for each mixture component for the GMM associated

[3] The value of α_{ts} depends both on the posterior probabilities and the type of discriminative objective function, and can be either positive or negative. For example, under the Maximum Mututal Information (MMI) criterion, $\alpha_{ts} = \gamma^{num}_{ts} - \gamma^{den}_{ts}$, the difference between the posterior probability of state s at time t in the numerator (forced-alignment) lattice and that in the denominator (recognition) lattice. For Minimum Classification Error (MCE) training, α_{ts} also depends on the smoothed sentence error, as described by the terms before the partial derivative of the acoustic score in Equation (2.69).

with s. Third, for a gradient-based update, add the partial derivatives with respect to the mean and variance of m, as described in Appendix A, to the gradient, and add $\alpha_{ts}r^{s}_{tm}$ to the dimension of the gradient with respect to the mixture weight of m; for an EBW-based update, add $\alpha_{ts}r^{s}_{tm}$ to the count statistics of the m^{th} mixture of s, $\alpha_{ts}r^{s}_{tm}\mathbf{x}_t$ to the first order statistics, and $\alpha_{ts}r^{s}_{tm}\mathbf{x}_t\mathbf{x}_t^T$ to the second order statistics. Since under the multi-level framework, each s is associated with multiple GMMs, to modify the accumulator for the multi-level framework, we have to change the second and third part of the procedure: Modify the second part such that it can compute r^{c}_{tm} for each GMM c associated with s. Modify the third part such that it is scaled by $\alpha_{ts}r^{c}_{tm}w(c,s)$, where $w(c,s)$ is the linear combination weight of c when computing the score for s. Note that depending on the implementation of the original accumulator, the two sets of modifications might be tied together in a subroutine.

One debugging technique to check if the modified accumulator correctly applies the weights when accumulating the gradient or sufficient statistics is as follows. First, generate recognition outputs of an utterance with a conventional acoustic model, and compute the gradient or sufficient statistics using the original accumulator. Second, randomly permute the GMMs in the acoustic model, and record the mapping $P(\cdot)$ that can restore the original order. Third, create a two-level framework such that the acoustic score $a_{\boldsymbol{\lambda}}(\mathbf{x}, s)$ is computed by $w_1 l_{\boldsymbol{\lambda}}(\mathbf{x}, P(s)) + w_2 l_{\boldsymbol{\lambda}}(\mathbf{x}, P(s))$, where w_1 and w_2 are two non-zero random weights that sum to one. Finally, run the modified accumulator under the two-level framework with the permuted acoustic model, and check the output with that of the original accumulator. If the modified accumulator functions correctly, the two outputs should be the same except for some rounding errors. If the modified accumulator passes the single utterance test, test it again on multiple utterances (maybe $\sim$ 100 utterances), before running it on the entire training set.

For a large-vocabulary speech recognition task, there can be a large number of training utterances, and using a single accumulator to accumulate the gradient, or sufficient statistics over all training utterances would take a very long time. One practical approach is to divide the training utterances into R sets, and run R accumulators in parallel on different machines. Then, the outputs of the R accumulators can be added up by a merging procedure. If R is also large, we can run $\sqrt{R}$ merging procedures in parallel where each one of them takes care of the outputs of $\sqrt{R}$ accumulators. The outputs of the $\sqrt{R}$ merging procedures can be added up by a final merging procedure. In this way, the time requirements for accumulating the gradient or sufficient statistics can be largely reduced.

3.4 Chapter Summary

In this chapter, we first used output codes as an illustration to show how the quantization effect occurs as a trade-off to deal with data sparsity in the conventional clustering-based context-dependent acoustic model, and discussed criteria that could avoid both the data sparsity and the quantization effect. Based on the criteria, we proposed a multi-level acoustic modeling framework, where each context-dependent unit is associated with a unique set of GMM classifiers that target multiple levels of contextual resolution. Then, we showed how to learn the model parameters of the multi-level framework under both maximum-likelihood and discriminative criteria, and discussed some possible extensions of the multi-level framework. Finally, we introduced some implementation and debugging guidelines. In the following chapters, we will conduct several experiments to evaluate the performance of the multi-level acoustic model compared with the conventional clustering-based acoustic model.

Chapter 4

Phonetic Recognition Experiment

4.1 Motivation

In Chapter 3, we proposed a multi-level acoustic modeling framework that addresses both the data sparsity problem and the quantization effect. Since the multi-level model ensures that different context-dependent phonetic units always have different acoustic scores, it should be able to provide better phonetic discrimination, and hence should help reduce phonetic confusions during recognition. In this chapter, we conduct a phonetic recognition experiment on the TIMIT corpus [71, 132] to see whether the multi-level model has better phonetic recognition accuracy than a conventional clustering-based acoustic model, without lexicographic and word-level constraints. Note that the focus of the experiment is to see if the multi-level framework can provide improvement, not to optimize the performance of the model on the TIMIT corpus.

4.2 TIMIT Corpus

TIMIT [71, 132] is an acoustic-phonetic continuous speech corpus that was recorded in Texas Instrument (TI), transcribed at the Massachusetts Institute of Technology (MIT), and verified and prepared for CD-ROM production by National Institute of Standard Technology (NIST). The corpus contains $6,300$ phonetically-rich utterances spoken by 630 speakers, including 438 males and 192 females, from 8 major dialect regions of American

English. For each utterance, the corpus includes waveform files with corresponding time-aligned orthographic and phonetic transcriptions [71, 44]. There are 61 APRAbet symbols used for transcription and their example occurrences are listed in Table 4.1. Because the size of the TIMIT corpus is not too large (about 5.4 hours of audio overall), and because it provides phonetically-rich data and expert transcribed time alignments of phonetic units which are generally not available in other corpora, TIMIT is an excellent test-bed for initial evaluations of new acoustic modeling techniques.

4.2.1 Data Sets

There are three types of sentences in the TIMIT corpus: dialect (SA), phonetically-compact (SX), and phonetically-diverse (SI). The two dialect sentences were designed to reveal the dialectical variations of speakers and were spoken by all 630 speakers. The 450 phonetically-compact sentences were designed such that the sentences are both phonetically-comprehensive and compact. The phonetically-diverse sentences were drawn from the Brown corpus [68] to reveal contextual variations. The sentences were organized such that each speaker spoke exactly 2 SA sentences, 5 SX sentences, and 3 SI sentences. The sentence type information is summarized in Table 4.2.

Because the SA sentences were spoken by all the speakers, they are typically excluded from the training and evaluation of acoustic models reported in the literature [77, 44]. The standard training set selected by NIST consists of 462 speakers and 3,696 utterances. The utterances spoken by the other 168 speakers form a "complete" test set. Note that there is no overlap between the texts spoken by the speakers in the training and the "complete" test set. 400 utterances of 50 speakers in the "complete" test set are extracted to form a development set for model development and parameter tuning. Utterances of the remaining 118 speakers are called the "full" test set. Among the utterances of the "full" test set, 192 utterances spoken by 24 speakers, 2 males and 1 female from each of the 8 dialect regions, are selected as the "core" test set. Typically, the evaluation results reported in the literature are based on the "core" test set rather than on the "full" test set. The data set information that includes number of speakers, utterances, audio length, and number of phonetic tokens

ARPAbet	Example	ARPAbet	Example
aa	bob	ix	debit
ae	bat	iy	beet
ah	but	jh	joke
ao	bought	k	key
aw	bout	kcl	k closure
ax	about	l	lay
ax-h	potato	m	mom
axr	butter	n	noon
ay	bite	ng	sing
b	bee	nx	winner
bcl	b closure	ow	boat
ch	choke	oy	boy
d	day	p	pea
dcl	d closure	pau	pause
dh	then	pcl	p closure
dx	muddy	q	glottal stop
eh	bet	r	ray
el	bottle	s	sea
em	bottom	sh	she
en	button	t	tea
eng	Washington	tcl	t closure
epi	epenthetic silence	th	thin
er	bird	uh	book
ey	bait	uw	boot
f	fin	ux	toot
g	gay	v	van
gcl	g closure	w	way
hh	hay	y	yacht
hv	ahead	z	zone
ih	bit	zh	azure
h#	utterance initial and final silence		

Table 4.1: ARPAbet symbols for phones in TIMIT with examples from [44]. Letters in the examples corresponding to the phones are underlined.

Sentence Type	#Sentences	#Speakers/ Sentence	Total	#Sentences/ Speaker
Dialect (SA)	2	630	1260	2
Compact (SX)	450	7	3150	5
Diverse (SI)	1890	1	1890	3
Total	2342	-	6300	10

Table 4.2: Sentence type information of TIMIT [44].

Set	#Speakers	#Utterances	#Hours	#Tokens
Train	462	3696	3.14	142,910
Development	50	400	0.34	15,334
Core Test	24	192	0.16	7,333
“Full” Test	118	944	0.81	36,347

Table 4.3: Data set information of TIMIT.

is summarized in Table 4.3.

4.2.2 Conventional Training and Evaluation Setup

Training Setup

Although the TIMIT corpus has been designed in a phonetically-rich way, some phonetic labels used by TIMIT still have limited amount of occurrences in the data. For example, the label “eng”, syllabic “ng” as in the word “Washington”, has less than 20 examples in the training set. Also, the distinct between certain pairs of labels, such as “ax” (“about”) and “ax-h” (“potato”), are relatively subtle. Due to these reasons, in the experiments reported in the literature [75, 77, 63, 90, 112, 30, 106], it was common that the 61 phonetic labels were reduced to 48 classes when training the acoustic model. Table 4.4 illustrates the 61 to 48 mapping. In this work, we also adopted the 48-class mapping when training the acoustic models in the experiments.

Evaluation Setup

Phone Error Rate (PER) is commonly used as the evaluation metric for quantifying model performance on the TIMIT corpus. To compute PER, for each test utterance, the hypothe-

1	iy	17	l	33	g
2	ih	18	el	34	p
3	eh	19	r	35	t
4	ae	20	y	36	k
5	ix	21	w	37	z
6	ax ax-h	22	er axr	38	zh
7	ah	23	m em	39	v
8	uw ux	24	n nx	40	f
9	uh	25	en	41	th
10	ao	26	ng eng	42	s
11	aa	27	ch	43	sh
12	ey	28	jh	44	hh hw
13	ay	29	dh	45	pcl tcl kcl q
14	oy	30	b	46	bcl dcl gcl
15	aw	31	d	47	epi
16	ow	32	dx	48	h# pau

Table 4.4: Mapping from 61 classes to 48 classes in [77].

sized phonetic string generated by the recognizer is aligned with a reference phonetic string using dynamic programming [21], and the total numbers of Substitution (Sub) errors, Insertion (Ins) errors, and Deletion (Del) errors are computed based on the alignment. One commonly-used software to align the hypotheses and references is the `sclite` toolkit [1] developed by NIST. The per utterance errors are then summed up over all test utterances, and the PER can be computed by dividing the total errors with the total number of phonetic tokens in the test utterances:

$$\text{PER} = \frac{\#\text{Sub Errors} + \#\text{Ins Errors} + \#\text{Del Errors}}{\text{Total phonetic tokens in test utterances}}. \tag{4.1}$$

In the experiment results reported in the literature, the phonetic labels are further reduced to 39 classes when computing the PER. Table 4.5 illustrates the label mapping under the 39-class setup. Since the 39-class mapping has been used in almost every work reported in the literature since it was first utilized in [75, 76], we also adopted the mapping when doing the evaluation for consistency with the literature.

1	iy	20	n en nx
2	ih ix	21	ng eng
3	eh	22	v
4	ae	23	f
5	ax ah ax-h	24	dh
6	uw ux	25	th
7	uh	26	z
8	ao aa	27	s
9	ey	28	zh sh
10	ay	29	jh
11	oy	30	ch
12	aw	31	b
13	ow	32	p
14	er axr	33	d
15	l el	34	dx
16	r	35	t
17	w	36	g
18	y	37	k
19	m em	38	hh hv
39	bcl pcl dcl tcl gcl kcl q epi pau h#		

Table 4.5: Mapping from 61 classes to 39 classes used for scoring, from [75].

4.3 Experiment

4.3.1 Acoustic Observations

To compute the acoustic observations (feature vectors) used in the experiment, a series of signal processing procedures are applied as follows. Given a speech utterance, the mean (DC component) of the waveform samples (with 16K Hz sampling rate) is first removed, and the samples are passed to a pre-emphasis (high-pass) filter

$$\tilde{s}[n] = s[n] - 0.97s[n-1], \tag{4.2}$$

where n denotes sample index, $s[n]$ denotes the sample sequence (with DC removed), and $\tilde{s}[n]$ denotes the output of pre-emphasis filter. Then, a Short Time Fourier Transform (STFT) is performed every 5ms, using a 25.6ms Hamming window [95], and the first 14 Mel-Frequency Cepstrum Coefficients (MFCC) [23] are computed for each 5ms analysis

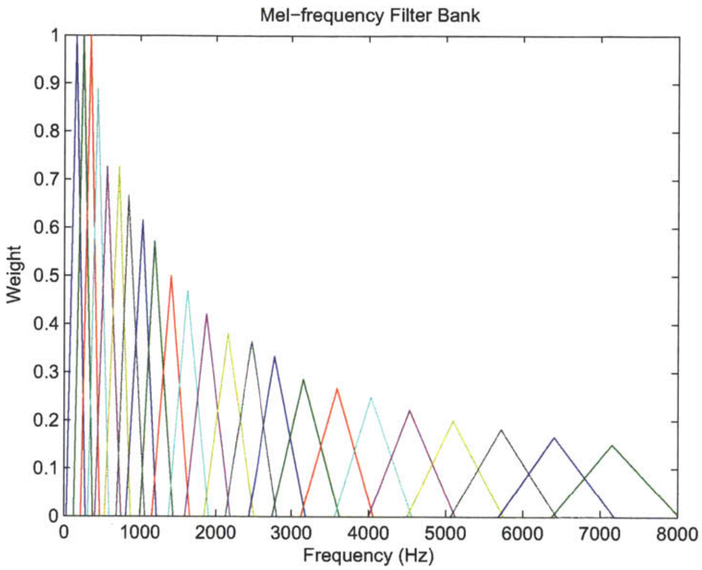


Figure 4-1: The Mel-Frequency filter banks used to compute the MFCCs.

frame. Figure 4-1 shows the Mel-Frequency filter banks used to compute the MFCCs. Figure 4-2 illustrates the signal processing flow for computing the MFCCs. After the MFCCs are computed, Cepstrum Mean Normalization [81] (with a 0.5s step) is applied as a basic normalization procedure against noise and inter-speaker variations.

The acoustic observations are then extracted at a 10ms frame rate as follows. To incorporate temporal information embedded in the speech signal, the average values of the (CMN normalized) MFCCs from 8 telescoping regions, ranging from 75ms before the time point of the frame to 75ms after the time point of the frame, are concatenated together to form a 112-dimensional (8 × 14) feature vector [85]. Detailed specifications of the telescope regions are listed in Table 4.6. The 112-dimensional feature is then rotated and reduced to 50 dimensions by a composition of Neighbourhood Components Analysis [39] and Principal Component Analysis [60], as in [114].

4.3.2 HMM Topology

Triphone based context-dependent acoustic models were used for the phonetic recognition experiments. Each triphone is represented by a 3-state, left-to-right, HMM. Because some

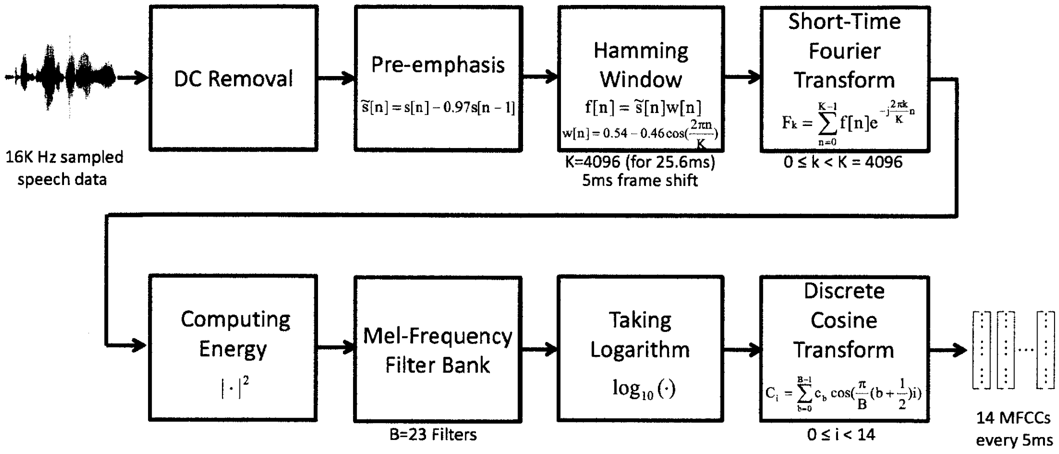

16K Hz sampled speech data
DC Removal
Pre-emphasis
s̃[n] = s[n] - 0.97s[n - 1]
Hamming Window
f[n] = s̃[n]w[n]
w[n] = 0.54 - 0.46cos(2πn/K)
K=4096 (for 25.6ms)
5ms frame shift
Short-Time Fourier Transform
F_k = Σ_{n=0}^{K-1} f[n]e^{-j2πk n/K}
0 ≤ k < K = 4096
Computing Energy
|·|²
Mel-Frequency Filter Bank
B=23 Filters
Taking Logarithm
log₁₀(·)
Discrete Cosine Transform
C_i = Σ_{b=0}^{B-1} c_b cos(π/B (b + 1/2)i)
0 ≤ i < 14
14 MFCCs every 5ms

Figure 4-2: The signal processing flow for computing MFCCs. The Hamming window and Short Time Fourier Transform are applied every 5ms for spectral analysis. (j denotes $\sqrt{-1}$.) The energy at each frequency band from the spectral analysis is passed to the Mel-Frequency filter banks. Logarithm is used to reduced the dynamic range of the outputs of the filter bank. Discrete Cosine Transform is applied to the log of the outputs of the filter bank, $\{c_b\}$, to extract the first 14 MFCCs.

Region	b-4	b-3	b-2	b-1	b+1	b+2	b+3	b+4
Start Time(ms)	-75	-35	-15	-5	0	+5	+15	+35
End Time(ms)	-35	-15	-5	-0	+5	+15	+35	+75

Table 4.6: Specifications of telescope regions for feature extraction. The minus sign "-" refers to before the time point of the frame, while "+" sign refers to after the time point of the frame.

phone labels, such as the release of stop consonant "b" and "d", can have very short acoustic realizations, skipping of states is also allowed in the topology. The HMM topology can be represented as a Finite State Transducer (FST), and as shown in Figure 4-3.

4.3.3 Phone-Level Language Model

While phonetic recognition does not use lexicographic or word-level information, in previous experiments reported in the literature, bigram phone-level language models are typically used during decoding [77, 63, 44, 30, 106]. In the phonetic recognition experiment reported in this chapter, a bigram phone-level language model was trained as follows. First, the original 61 phone labels in the phonetic references from the training set were reduced

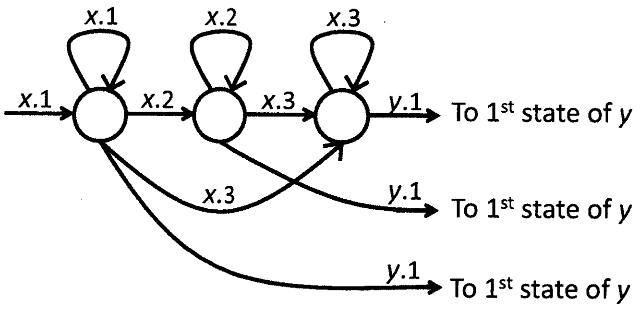


Figure 4-3: The HMM state topology used to model the triphones in the experiment (represented in an FST format after determinizing epsilon transitions). x and y represent two consecutive triphones. $x.1, x.2, x.3$, and $y.1$ denote the model indices that are used to compute the acoustic scores.

to 48 classes based on Table 4.4. Then, the reduced phonetic references were fed to the `ngram-count` script of SRILM [116] (with `-interpolate1 -addsmooth2` 0.25) to train the bigram phone-level model. The resulting bigram phone-level language model had a perplexity of 14.6 on the dev set[1]. After the phone-level language model was trained, it was then converted into an FST to constrain the search space during decoding.

4.3.4 Clustering-Based Acoustic Model

Decision Tree Clustering

Decision tree clustering, described in Section 2.2.5, was used to cluster the triphone states. For each triphone state, its associated acoustic observations were collected based on the expert transcribed phonetic time-alignments[2], and the mean and the variance of the observations were computed for clustering usage. The triphone states were divided into $48 \times 3 = 144$ groups based on the center phone unit and the state index in the topology. For each group, a decision tree was used to split the contexts. To grow the tree, a question

[1] This is the perplexity when the labels are reduced to 48 classes. If using the same language model training procedure without the label reduction, the resulting perplexity on the dev set became 15.8, the same as what was reported in [44].

[2] Supposed there were k frames, indexed by $\{0, 1, \ldots, k-1\}$, aligned to a triphone based on the expert transcriptions. If $k < 3$, all three states of the triphone collected the k frames as examples. If $k \geq 3$, the first state got $[0, \lfloor \frac{k}{3} \rfloor]$, the second state got $[\lfloor \frac{k}{3} \rfloor, \lfloor \frac{2k}{3} \rfloor]$, and the third state got $[\lfloor \frac{2k}{3} \rfloor, k-1]$. Note that the acoustic feature vector corresponding to the boundary frame were assigned to both states. In this way, each state could get slightly more data that might potentially help parameter initialization.

from Table F.1 was selected to split a node into two. For a question to be selected, it must satisfy the following two conditions: First, both of the two new nodes have at least MIN_DP examples. Second, it must provide the largest log-likelihood gain based on Equation (D.7). The tree keeps growing until either the number of nodes has reached MAX_NODES or the normalized log-likelihood improvement[3] of the best question was less than 0.001.

After the clustering was finished, a GMM was trained for each leaf node under the Maximum-Likelihood (ML) criterion by running the Expectation-Maximization (EM) algorithm on the data aligned with the node. The GMM was allowed to increase one mixture component per 50 examples until MAX_MIX components. Table 4.7 lists the PERs on the Development (Dev) set under different settings. Based on the Dev set performance, the model with MIN_DP= 500, MAX_NODES= 25, and MAX_MIX= 30 was used as the initial model for discriminative training. For convenience, we call the model ML-CL, meaning Maximum-Likelihood (ML) trained Clustering (CL) based model. The PER of the ML-CL model on the core test set was 29.0%.

Discriminative Training

Minimum Classification Error (MCE) [62, 85, 86] training was used as a discriminative criterion to refine the GMM parameters. Because the PER of the ML-CL model on the training data was low ($< 10\%$), in order to enhance the parameter learning, the recognition score of each competing hypothesis was boosted by a margin proportional to the approximated phone-level string distance[4]. A simplified lattice based variant of the MCE criterion[5] was used for training. No phone-level language model was used during the train-

[3] log-likelihood improvement divided by the number of examples. Another commonly-used criterion is the relative improvement of log-likelihood.

[4] For the triphone state s_t at time t in the competing hypothesis, check the corresponding state s_t^{ref} in the forced-alignment to see if the two states correspond to the same (center) phone. If not, add the acoustic score for s_t by a margin ρ/P_t^{ref}, where P_t^{ref} is the length (in terms of the number of frames) of the phone unit covering time point t in the forced alignment. The information used to compute the margin can be precomputed, and can be loaded in when doing decoding. To enable margin based decoding, modify the source code of the decoding module similarly as in Section 3.3.1. In the experiment, ρ was set to 10.

[5] Since the number of competing hypotheses can be many in phonetic recognition, two simplifications are made. First, set $\eta = 1$ and do not remove the reference string if it showed up in the recognition lattice. By doing so, $\gamma_{ts}^{comp_n}$ in Equation (2.68) would be the same as the posterior probability γ_{ts} computed by the conventional Forward-Backward algorithm. Second, instead of counting the number of paths $\mathcal{K}$ in the lattice to compute the average path score, just use the best path score when computing d_n in Equation (2.65).

MIN_DP	MAX_NODES	CL_STATES	MAX_MIX	TOTAL_MIX	PER
1,000	25	846	15	12.7K	28.0%
1,000	25	846	30	22.9K	27.6%
1,000	25	846	60	28.0K	27.3%
1,000	25	846	120	28.6K	27.2%
1,000	50	854	15	12.8K	28.1%
1,000	50	854	30	23.1K	27.3%
1,000	50	854	60	28.0K	27.3%
1,000	50	854	120	28.6K	27.6%
500	50	1,626	15	22.1K	27.6%
500	50	1,626	30	27.2K	26.9%
500	50	1,626	60	28.3K	27.0%
500	50	1,626	120	28.7K	27.3%
500	25	1,511	15	20.6K	27.3%
500	25	1,511	30	26.0K	**26.8**%
500	25	1,511	60	27.9K	27.2%
500	25	1,511	120	28.6K	27.0%
200	50	3,421	15	25.7K	27.6%
200	50	3,421	30	27.5K	27.8%
200	50	3,421	60	28.3K	27.5%
200	50	3,421	120	28.7K	27.6%

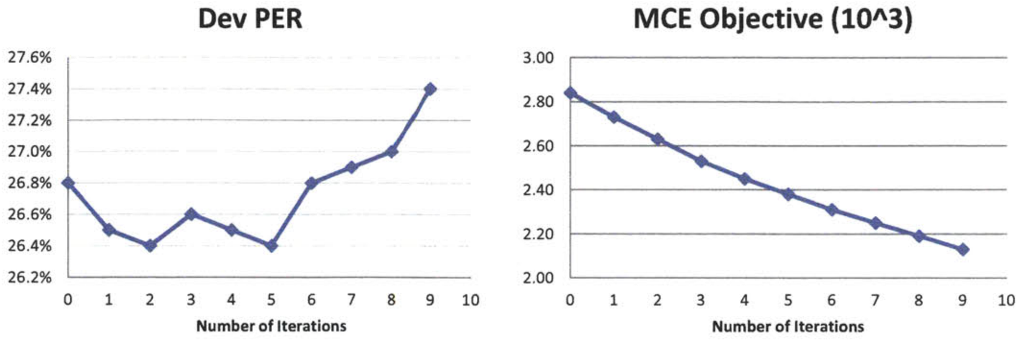
Table 4.7: PERs on the dev set based on different clustering criteria and maximum number of mixture components. CL_STATES denotes the total number of clustered states (total number of leaf nodes). TOTAL_MIX denotes the total number of mixture components summing over all clustered states.

ing. A Quasi-Newton-based Quickprop algorithm was used to update model parameters. Details of the Quickprop algorithm are described in Appendix G, and settings for the MCE training are listed in Appendix H. Figure 4-4 shows the PERs on the Dev set and the values of the MCE objective functions on the training set across different iterations. Based on the PER on the Dev set, the model at iteration 5 was picked. For convenience, we call the model MCE-CL. The PER of the MCE-CL model on the core test set was 28.2%.

4.3.5 Multi-Level Acoustic Model

Model Initialization

The model structure in Figure 3-3 was used to construct the multi-level model. As in the training for the clustering-based model, the expert transcribed phone alignments were used



(a)PER on the Dev set

(b)MCE objective function

Figure 4-4: MCE training results of the clustering-based acoustic model on TIMIT. While the training objective function monotonically decreased, the PER on the Dev set started increasing at iteration 6.

to collect acoustic observations for each triphone state. For each triphone state with more than 300 training examples, a 5-component GMM was trained using the EM algorithm, resulting in about 500 classifiers at the top level. For the second level, a GMM was trained if the corresponding node had more than 100 examples, resulting in about 4.7K classifiers. The GMMs at the second level were allowed to add one mixture component per 50 training examples until there were at most 10 components. For the bottom level, the broad classes listed in Table 4.8 were used to construct classifiers. No cut-offs on the number of examples were used for the bottom level, and the GMMs were allowed to add one mixture component per 50 training examples until there were 20 components. There were in total 10.2K classifiers used over all three levels, and 86.3K mixture components. The combination weights were assigned according to the principles described in Figure 3-4. For convenience, we call the model ML-Multi. The ML-Multi model had a PER of 26.2% on the Dev set, and 27.3% on the core test set.

Discriminative Training

The same setting of Minimum Classification Error (MCE) training used to train the clustering-based model was also used to train the multi-level model. Figure 4-5 illustrates the PERs of the multi-level acoustic model on the Dev Set cross different MCE iterations. According

Broad Class	Phone Labels
Front_Vowel	iy ih ix eh ey ae aw
Back_Vowel	uw uh ow ao aa ay oy
Mid_Vowel	ah ax er
Semi_Vowel	w l y r el
Weak_Fric	v f th dh hh
Strong_Firc	z s sh zh jh ch
Voiced_Stop	b d g
Unvoiced_Stop	p t k
Nasal	m n en ng dx
Silence	<> cl vcl epi sil

Table 4.8: Broad classes used in the TIMIT experiment.

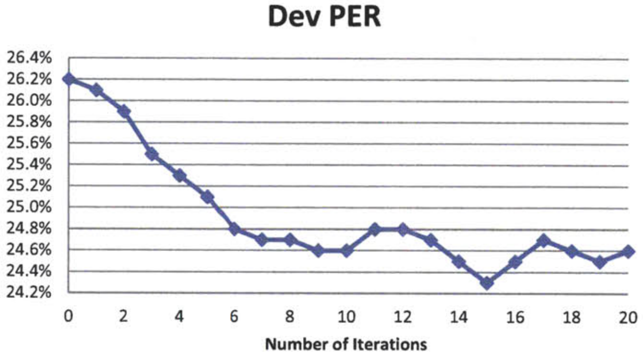


Figure 4-5: MCE Training results for the multi-level acoustic model on TIMIT.

to the PER on the Dev set, the model at iteration 15 was selected. For convenience, we call the model MCE-Multi. The PER of the MCE-Multi model on the core test set was 25.8%.

4.3.6 Comparisons

Table 4.9 summarizes the performances of the clustering-based acoustic model (CL) and the multi-level acoustic model (Multi) on the Dev set and the core-test set of TIMIT. Comparing with the MCE-CL model, the MCE-Multi model has about 8% relative improvement in PER on the core-test set. Based on the McNemar significance test [33], the improvement was statistically significant (with $p < 0.001$). The result suggests that the multi-level structure can help improve phonetic prediction accuracy. Also, the relative improvement of

Model	Dev PER	Core-Test PER
ML-CL	26.8%	29.0%
MCE-CL	26.4 %	28.2%
ML-Multi	26.2%	27.3%
MCE-Multi	24.3%	25.8%

Table 4.9: Comparison of the PERs of the clustering-based (CL) acoustic model and the multi-level (Multi) acoustic model on TIMIT.

MCE-Multi over MCE-CL is larger than that of ML-Multi over ML-CL, suggesting that the multi-level model can potentially benefit more from discriminative training than the clustering-based model. This result is consistent with the hypothesis that by providing different scores for different context-dependent states, the multi-level acoustic model can have better discriminative power that can be learned via the discriminative training algorithm. While the size of the multi-level model was larger, further increasing the size of the clustering-based model did not further improve PER, as shown in Table 4.7.

Table 4.10 lists the performance of several discriminatively trained context-dependent acoustic models on the core-test set reported in the literature[6]. Compared with the Maximum Mutual Information (MMI) model in [63] and the Phone-discriminating MCE (P-MCE) model in [30], the MCE-Multi model yielded better performance. However, it did not perform as well as the Boosted MMI (BMMI) model with discriminative feature-space transform in [106]. Since the multi-level framework can be generalized to other types of feature vectors, it would be interesting to see how the multi-level structure can further improve the system with specially trained feature-space transform as in [106]. Nevertheless, the focus of the phonetic recognition experiment was to see whether the multi-level model can improve phonetic prediction accuracy, rather than optimizing the performance on TIMIT. Since the improvement shown in Table 4.9 has demonstrated the point, we can move on to the next step; that is, to test whether the multi-level framework can be utilized

[6] The methods compared here were all speaker-independent models using a single type of acoustic feature vector. Utilizing multiple types of acoustic observations during recognition has shown significant improvement on PER as described in [44]. Speaker adaptation can also provide additional improvement as shown in [106]. Since the focus here is to see if the multi-level framework can provide better performance than clustering-based models given that other conditions are the same, we did not investigate using multiple types of acoustic features. However, the multi-level framework can generalize to the system with multiple types of acoustic observations. For speaker adaptation, see Chapter 6 for more details.

Method	Authors	Year	PER
MMI[63]	Kapadia et al	1993	28.2%
P-MCE [30]	Fu et al	2007	27.0%
fBMMI [106]	Sainath et al	2009	22.7%

Table 4.10: Performances of several discriminatively trained context-dependent acoustic model on the TIMIT core-test set.

by a large-vocabulary ASR system and provide better recognition performance.

4.4 Chapter Summary

In this chapter we conducted a phonetic recognition task to see that without lexicon and word-level language model if the multi-level acoustic model could provide better phonetic prediction accuracy than a conventional clustering-based acoustic model. The result was positive since the multi-level model provided about 8% relative improvement in terms of Phone Error Rate (PER). In the next chapter, we will deploy the multi-level framework to a large-vocabulary ASR task to see if the multi-level framework can be well integrated into existing decoding framework and provide better recognition accuracy.

Chapter 5

Large-Vocabulary Speech Recognition

5.1 Motivation

In Chapter 3, we showed that the multi-level acoustic model addresses the data sparsity problem for context-dependent acoustic modeling while preventing the quantization effect. Such a property ensures that each context-dependent state can have a unique and robust acoustic score for a given acoustic observation, and thus can potentially provide more discriminating power to reduce confusions during decoding. While the phonetic recognition experiments in Chapter 4 showed that the multi-level framework can provide better phonetic prediction accuracy, another important question is whether the multilevel framework can be well integrated into a large-vocabulary ASR system, and provide performance improvement on a real world task. To test the performance of the multi-level model on a large-vocabulary real world ASR task, in this chapter we conduct large-vocabulary lecture transcription experiments on the MIT Lecture Corpus.

5.2 MIT Lecture Corpus

The MIT Lecture Corpus contains audio recordings and manual transcriptions for approximately 300 hours of MIT lectures (number is still growing) from eight different courses, and over 100 MITWorld seminars given on a variety of topics [96]. The audio data were recorded with omni-directional microphones, and were generally recorded in a classroom environment. The recordings were manually transcribed including disfluencies such as filled pausess, and false starts. In addition to the spoken words, the following annotations are also included in the transcriptions: occasional time markers at obvious pauses or sentence boundaries, locations of speaker changes (labeled with speaker identities if known), and punctuation, based on the transcriber's subjective assessment of spoken utterances [37].

The lecture corpus is a difficult data set for ASR systems for the following reasons. First, the recorded data consist of spontaneous speech. The lecture speech contains many disfluencies such as filled pauses, false starts, and partial words. In addition, the spontaneous nature also results in less formal sentences, and poorly organized, or ungrammatical sentences can be frequently observed. Second, the lectures contain many lecture-specific words that are uncommon in daily life. The lecture-specific words can result in serious Out-Of-Vocabulary (OOV) problems. As shown in [97], even using the 10K most frequent conversational words[1] as the vocabulary, the OOV rate is still around 10%. Third, the speech data potentially suffer from reverberation, coughing, laughter, or background noise such as students' talking. The above reasons make it challenging to build a robust ASR system for lectures.

5.2.1 Data Sets

In the experiments reported in this chapter, about 119 hours of lectures were selected as the training data for the acoustic model. The training lectures include two lectures of linear algebra (18.06), two lectures of introduction to computer programs (6.001), one lecture of physics (8.224), two lectures of Anthropology, and 99 lectures from 4 years of MITWorld lecture series on a variety of topics. Table 5.1 lists the sizes of the training lectures. In addition to the word references, the phonetic time-alignments of the training lectures, which were computed based on the forced-alignment algorithm described in [47], were also used for acoustic model training.

Two held-out MITWorld lectures, Thomas Friedman's "The World is Flat" lecture (MIT-World-2005-TF) and Thomas Leighton's lecture about a startup company called Akamai (MIT-World-2004-TL), were used as a development (Dev) set. Table 5.2 lists the sizes

[1] Selected from SWITCHBOARD [38], a telephone conversation-based speech corpus.

of the Dev lectures. Eight lectures from four subjects were selected as the test set for model evaluation. The test set includes two lectures of differential equation (18.03), two lectures of introduction to algorithms (6.046), two lectures of automatic speech recognition (6.345), and two lectures of introduction to biology (7.012). Table 5.3 lists the sizes of the test lectures. To evaluate the performance of the ASR system under a generic, speaker-independent scenario, there were no speaker overlaps among all the three sets of data.

Class	#Lectures	#Hours
18.06-1999	2	1.45
6.001-1986	2	2.19
8.224-2003	1	0.94
MIT-World-2002	30	36.24
MIT-World-2003	51	54.60
MIT-World-2004	7	8.79
MIT-World-2005	11	12.39
St_Marys-Anthropology	2	2.16
Total	106	118.76

Table 5.1: Sizes of lectures in the training set.

Class	#Lectures	#Hours
MIT-World-2005-TF	1	1.25
MIT-World-2004-TL	1	0.92
Total	2	2.17

Table 5.2: Sizes of lectures in the development set.

Class	#Lectures	#Hours
18.03-2004	2	1.66
6.046-2005	2	2.48
6.345-2001	2	2.54
7.012-2004	2	1.41
Total	8	8.11

Table 5.3: Sizes of lectures in the test set.

5.3 Experiment

Two large-vocabulary ASR systems were used in the experiments: a diphone, landmark-based system and a triphone, frame-based system. The diphone, landmark-based system aims to decode quickly and would be suitable for applications that require a faster ASR response. However, its recognition results might be less accurate. The triphone, frame-based system decodes more accurately, but it runs much more slowly. Since the two systems have different properties, evaluating both systems can provide a better indication as to how well the multi-level acoustic modeling framework can be applied to different types of ASR related applications.

5.3.1 Evaluation Metric

Word Error Rate (WER) was used as the evaluation metric for the system performance in the automatic lecture transcription experiment. To compute the WER, for each utterance, the reference word string and the best scoring hypothesis string from ASR are aligned using dynamic programming to compute the number of Substituted (Sub), Inserted (Ins), and Deleted (Del) words, and then the total number of errors are accumulated over all test utterances and are normalized by the number of words in the references:

$$\text{WER} = \frac{\#\text{Sub Words} + \#\text{Ins Words} + \#\text{Del Words}}{\text{Total number of words the in references}}. \tag{5.1}$$

As in the phonetic recognition experiments of Chapter 4, the `sclite` toolkit [1] developed by NIST was used to compute WERs.

5.3.2 Acoustic Observations

The acoustic observations (feature vectors) used by the two systems were extracted in the same way except for the time locations where the features were extracted. For the diphone, landmark-based system, the observations were extracted only at the acoustic landmarks [34] that distribute non-uniformly over the utterance. For the triphone, frame-based system, the observations were extracted uniformly at a 10ms frame rate, as in the phonetic recognition experiment of Chapter 4. At each time point to extract the acoustic observation, the same procedure described in Section 4.3.1 was used. Namely, the average values of first 14 MFCCs from 8 telescoping regions were concatenated to a 112-dimensional vector, and a composition of Neighbourhood Components Analysis [39] and Principal Component Analysis [60] was used to reduce the dimension into 50. The projection matrix used for dimension reduction was the same as in [114].

5.3.3 Phonetic Labels

The phonetic labels used by the triphone, frame-based system are listed in Table 5.4. Compared with the 61 TIMIT labels listed in Table 4.1, the changes are as follows. "ax-h", "eng", "hv", "ix", "hx", "q", and "ux" are removed. "h#" is replaced with "–", and "pau" with "_". "ah_fp", "_b", "_c", "_l", and "_n" are added to model filled pause, background sound, cough, laughter, and noise, respectively. There are $59(61 - 7 + 5 = 59)$ phonetic labels in total used by the triphone, frame-based system. (60 kinds of contexts including sentence boundary "<>".)

For the diphone, landmark-based system, the phonetic labels are similar except that more labels are used to model different types of background sounds, coughs, laughter, and noises. For backgrounds, coughs, and laughter, each is modeled by 4 labels. For noise, 6 labels are used. As a result, there are in total 73 $(54 + 1 + 4 \times 3 + 6 = 73)$ phonetic labels used by the diphone-landmark system. (74 kinds of contexts including sentence boundary "<>".)

5.3.4 HMM Topologies

For the triphone, frame-based system the same 3-state topology illustrated in Figure 4-3 was used. For the diphone, landmark-based system, a 2-state topology illustrated in Figure 5-1 was used to model each diphone. The idea behind the 2-state topology is as follows. Each landmark location computed by the algorithm in [34] represents a time point where the spectral change of the acoustic signal is larger than a threshold. The spectral change might correspond to a transition from one phone unit to another, or might correspond to an

Label	Example	Label	Example
aa	bob	ih	bit
ae	bat	iy	beet
ah	but	jh	joke
ao	bought	k	key
aw	bout	kcl	k closure
ax	about	l	lay
axr	butter	m	mom
ay	bite	n	noon
b	bee	ng	sing
bcl	b closure	ow	boat
ch	choke	oy	boy
d	day	p	pea
dcl	d closure	pcl	p closure
dh	then	r	ray
dx	muddy	s	sea
eh	bet	sh	she
el	bottle	t	tea
em	bottom	tcl	t closure
en	button	th	thin
epi	epenthetic silence	uh	book
er	bird	uw	boot
ey	bait	v	van
f	fin	w	way
g	gay	y	yacht
gcl	g closure	z	zone
hh	hay	zh	azure
–	utterance initial and final silence		
ah_fp	filled pause	_	inter-world pause
_b	background sound	_c	cough
_l	laughter	_n	noise

Table 5.4: Phonetic labels used in the triphone, frame-based recognizer for lectures. Letters in the examples corresponding to the phones are underlined. The labels above the double horizontal lines are the same as those in TIMIT. The labels below the double horizontal lines are used to represent silence, pause, and other sound effects.

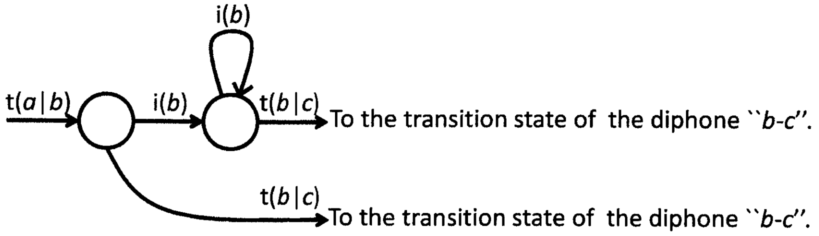

i(*b*)
t(*a*|*b*)
i(*b*)
t(*b*|*c*)
To the transition state of the diphone "*b-c*".
t(*b*|*c*)
To the transition state of the diphone "*b-c*".

Figure 5-1: The 2-state topology used to model the diphones in the experiment. A diphone "*a-b*" (phone "*b*" with left context "*a*") is modeled by a transition state followed by an internal state. t(*a*|*b*) denotes the model label that provides the score of an acoustic observation being a transition from phone "*a*" to phone "*b*", while i(*b*) denotes the model label that provides the score of an observation being an internal part of the phone "*b*".

internal transition within a phone unit (such as diphthongs). The between phone transitions are scored by a set of transition models, while the internal transitions are scored by internal models. Since there is only one between phone transition for two consecutive phone units, there is no looping at the transition state in the topology of Figure 5-1. While in principle the internal models can also be context-dependent, making the internal models context-independent can reduce the size of the search space, and significantly increase the decoding speed.

5.3.5 Language Model and FSTs

The same topic-independent language model in [35] was used in the experiment. Three text sources, the lecture transcriptions[2], the SWITCHBOARD corpus [38], and the MICASE[3] corpus, were used as the training data, and a vocabulary of size 37.4K was selected from the texts. The `ngram-count` script of SRILM [116] (with `-unk -prune` 0.0000015) was used to train a trigram language model. The trigram language model has a perplexity of 202.7 on the two Dev lectures.

Finite State Transducers (FSTs) were used to modularize and combine the search constraints. The lexicon and pronunciation rules were converted into a Lexicon (L) FST[4] and a

[2] Mainly the training lectures in Table 5.1, but some other lectures such as 8.01-1999 were also included.

[3] MIchigan Corpus of Academic Spoken English. `http://quod.lib.umich.edu/m/micase/`

[4] Some reduction rules were also applied during the construction of the L FST to convert casual usages of words to formal usages, such as changing "gonna" to "going to".

Pron-rule (P) FST [52], respectively. The bigram language model reduced from the trigram language model was converted into a Language Model/Grammar (G) FST. Then, the P, L, and G FSTs were composed and determinized to form a P $\circ$ L $\circ$ G FST. Depending on the type of context-dependent acoustic model, different Context (C) FST was composed with the P $\circ$ L $\circ$ G FST to form the search space during decoding, as described in Section 2.4.2. The differences between the trigram and the bigram language models were also converted into an FST to be used for a second pass re-scoring.

5.3.6 Clustering-Based Diphone Acoustic Models

As in [50], the $5,549$ diphone units[5] used in the diphone-landmark system were reduced to $1,871$ classes via clustering. For each of the $1,871$ classes, a GMM was trained under the ML criterion by running the EM algorithm on the acoustic observations corresponding to the class, based on the phonetic alignments generated in [47]. Each GMM was allowed to add one mixture component per 50 examples until `MAX_MIX` was reached. To create models of different sizes, different values of `MAX_MIX`, 30, 60, and 100, were used. For convenience, we call the models ML-CL-Di$_{30}$ (Maximum-Likelihood trained CLustering-based Diphone model with `MAX_MIX`= 30), ML-CL-Di$_{60}$, and ML-CL-Di$_{100}$.

For each of the three ML models, Minimum Classification Error (MCE) training was used to refine the GMM parameters. As in other researches on discriminative training for large-vocabulary ASR, such as [86], a unigram (rather than bigram or trigram) language model was used for the MCE training to force the acoustic model to correct more errors during training. To speed up the training process, only the best $\mathcal{K} = 20$ hypotheses were considered during the gradient computation. Since the WERs on training lectures were already high, no margins were used to boost the scores of competing hypotheses. As in the phonetic recognition experiments, the Quickprop algorithm (as described in Appendix G) was used to update parameters. The operational constants used for MCE training are listed in Table H.2. The WER on the Dev set was used as the selection criterion over the MCE iterations. For convenience, we call the MCE trained models MCE-CL-Di$_{60}$, MCE-

[5] 74×74 for transition models plus 73 for internal models. There is no internal model for sentence boundary "<>".

Model	#Mixtures	Dev WER
ML-CL-Di$_{30}$	31.9K	45.7%
ML-CL-Di$_{60}$	49.5K	44.2%
ML-CL-Di$_{100}$	65.3K	43.8%
MCE-CL-Di$_{30}$	31.9K	41.2%
MCE-CL-Di$_{60}$	49.5K	41.5%
MCE-CL-Di$_{100}$	65.3K	41.5%

Table 5.5: WER of clustering-based diphone models on the Dev lectures.

Model	6.046	6.345	7.012	18.03	Test WER
ML-CL-Di$_{30}$	42.7%	34.5%	33.6%	42.3%	38.2%
MCE-CL-Di$_{30}$	37.6%	28.5%	28.9%	36.8%	32.8%

Table 5.6: WER of clustering-based diphone models on the Test lectures.

CL-Di$_{60}$, and MCE-CL-Di$_{100}$, respectively.

Table 5.5 lists the WERs of the clustering-based diphone models on the Dev lectures. While the WERs of these ML models improved as the number of mixture components increased, the MCE models did not show such trend. Investigating the loss functions of the MCE models on both the training and Dev lectures, we found that the training loss decreased as the number of mixture components increased, while the loss on the Dev set increased, suggesting that some over-training might have occurred. Based on the WER on the Dev set, the MCE-CL-Di$_{30}$ model was selected. For convenience, we use ML-CL-Di and MCE-CL-Di to refer the ML-CL-Di$_{30}$ model and the MCE-CL-Di$_{30}$ model, respectively, in the following sections. Table 5.6 lists the WERs of the ML-CL-Di$_{30}$ and MCE-CL-Di$_{30}$ models on the subsets of the Test lectures. As shown in the table, the MCE training significantly improved the WER on each subset of the test lectures, showing the effectiveness of the discriminative training.

5.3.7 Multi-level Diphone Acoustic Models

The model structure in Figure 3-8 was used to construct a basic multi-level diphone acoustic model. Broad classes in Table 5.7 were used to construct bottom level classifiers. The same phonetic alignments generated in [47] were used to collect training examples for the GMM

Broad Class	Phone Labels
Low_Vowel	aa ah ah_fp ao aw ax ow uh uw w
High_Vowel	ae ay eh ey ih iy oy y
Retroflex	er r axr
Lateral	el l
Fric	f v th dh s z sh zh ch jh hh
Closures	bcl dcl gcl pcl tcl kcl epi
Stop	b d g p t k
Nasal	m em n en ng dx
Silence	– <>
Noise	_ _b1 _b2 _b3 _b4 _c1 _c2 _c3 _c4 _l1 _l2 _l3 _l4 _n1 _n2 _n3 _n4 _n5 _n6

Table 5.7: Manner-based broad classes used to construct the basic multi-level diphone acoustic model.

classifiers, and the EM algorithm was used to train the GMM parameters under the ML criterion. Each GMM was allowed to add one mixture component per 50 training examples until there were a maximum of 30 components. For convenience, we call the model ML-B-Multi-Di (Maximum-Likelihood trained Basic Multi-level Diphone model).

As described in Section 3.2.4, multiple sets of broad classes can be used to construct bottom level classifiers. Table 5.8 lists another set of broad classes based on place of articulation that was used to construct an extended version of multi-level model. Each additional GMM classifier from the additional broad classes was also allowed to add one mixture components per 50 training examples until there were a maximum of 30 components. In the extended model, the default combination weight for a top level classifier was set to $1/3$, and the default weight for each bottom level classifier was set to $1/6$. If the top level classifier did not have enough training examples, its weight was equally distributed to the bottom level classifiers. For convenience, we call the model ML-E-Multi-Di (Maximum-Likelihood trained Extended Multi-level Diphone acoustic model).

MCE training was used to refine the GMM parameters of both the basic and extended multi-level models. The same set of operational constants as in the MCE training of the clustering-based diphone acoustic models were used. The WER on the Dev lectures was used as the selection criterion over different MCE iterations. Table 5.9 lists the WERs on

Broad Class	Phone Labels
Labial	m em b p bcl pcl f v
Alveolar	n en d t dcl tcl s z th dh
Palatal	sh zh jh ch
Velar	ng k g kcl gcl
Front	ae aw eh ey ih iy
Back	aa ao ay ow oy uw uh
Mid	ah ah_fp ax axr er hh
Semi_Vowel	l el r w y
Silence	– <>
Noise	_ _b1 _b2 _b3 _b4 _c1 _c2 _c3 _c4 _l1 _l2 _l3 _l4 _n1 _n2 _n3 _n4 _n5 _n6

Table 5.8: Place of articulation-based broad classes used for the extended multi-level diphone acoustic model.

Model	#Mixtures	Dev WER	Test WER
ML-B-Multi-Di	58.7K	45.5%	37.8%
ML-E-Multi-Di	85.0K	45.7%	38.0%
MCE-B-Multi-Di	58.7K	40.0%	30.3%
MCE-E-Multi-Di	85.0K	39.2%	29.9%

Table 5.9: WERs of multi-level diphone acoustic models on Dev and Test lectures.

the Dev and Test lectures before and after the MCE training. Unlike the clustering-based model, although the extended model has more parameters than the basic model, it still provided better WERs after MCE training. Also, increasing the MAX_MIX for the bottom level classifiers in the basic multi-level model to 60 further reduced the WERs on both the Dev and Test lectures to 39.3% and 30.1%, respectively, for the MCE trained models. These facts suggest that the multi-level model might be less sensitive to over-training as the number of parameters increases.

5.3.8 Clustering-Based Triphone Acoustic Models

As in Section 4.3.4, the acoustic observations corresponding to each triphone state were collected based on the phonetic alignments from [47]. The question set in Table F.2 was used to grow the decision trees. The same decision tree clustering procedure as in Section

MIN_DP	MAX_NODES	CL_STATES	Dev WER
50	100	14.3K	39.4%
50	200	24.3K	40.2%
100	50	7.7K	39.8%
100	100	13.9K	39.6%
100	200	22.4K	40.6%
200	25	4.2K	40.2%
200	50	7.5K	**39.4%**
200	100	13.2K	39.5%
200	200	19.8K	40.2%

Table 5.10: WERs on the Dev lectures under different clustering settings.

4.3.4 was used. Several sets of clustering settings were tested. For each clustering setting, a GMM was trained for each leaf node, and the GMM was allowed to add one mixture component per 50 training examples until there were 60 components. The WER on the Dev lectures was used as the selection criterion among different settings. Table 5.10 lists the numbers of resulting clustered states and WERs on the Dev lectures under different clustering settings. While the model with 7.5K clustered states had slightly better WER on the Dev set than the model with 14.3K clustered states, its WER on the Test set was better by 0.4% absolute. Based on the results in Table 5.10, the setting with 7.5K clustered states was used for the following steps.

Different values of MAX_MIX were used to obtain models of different size. As before, each GMM was allowed to add one mixture component per 50 examples until MAX_MIX components were reached, and the GMMs were trained under ML criterion using the EM algorithm based on the phonetic alignments. For each ML model under a particular MAX_MIX, MCE training with operational constants in Table H.3 was applied to refine the GMM parameters. Table 5.11 lists the WERs of the models on the Dev and Test lectures before and after MCE training. As shown in the table, for smaller models, MCE training provided larger improvement over the original ML models, but the Test WER did not necessarily become better. (Since their ML starting point were worse.) Based on the results in Table 5.11, MCE-CL-Tri$_{60}$ was selected. For convenience, we call ML-CL-Tri$_{60}$ and MCE-CL-Tri$_{60}$ by ML-CL-Tri and MCE-CL-Tri respectively in the following sections.

Model	#Mixtures	Dev WER	Test WER
ML-CL-Tri$_{15}$	103K	41.6%	34.5%
ML-CL-Tri$_{30}$	265K	40.1%	33.0%
ML-CL-Tri$_{60}$	310K	39.4%	32.4%
ML-CL-Tri$_{120}$	471K	39.0%	32.2%
ML-CL-Tri$_{240}$	641K	39.0%	32.0%
MCE-CL-Tri$_{15}$	103K	38.0%	31.2%
MCE-CL-Tri$_{30}$	265K	38.0%	30.7%
MCE-CL-Tri$_{60}$	310K	37.3%	30.6%
MCE-CL-Tri$_{120}$	471K	37.5%	31.1%
MCE-CL-Tri$_{240}$	641K	37.5%	31.4%

Table 5.11: WERs of clustering-based triphone acoustic models of different sizes. The numbers in the subscripts refers to the value of MAX_MIX.

5.3.9 Multi-level Triphone Acoustic Models

Similar to the diphone case, both a basic and an extended multi-level triphone acoustic model were trained. The model structure in Figure 3-4 was used to build the ML-B-Multi-Tri (Maximum-Likelihood trained Basic Multi-level Triphone) model. Count statistics were used as the selection criteria for classifiers at the top level. Table 5.12 lists the number of triphone states that have training examples larger than several thresholds. Jointly considering the number of states and the number of training examples, 800 was selected as the cut-off, resulting in about 10.3K classifiers, and a 15-component GMM was trained for each classifier using the EM algorithm. For the second level, 200 was chosen as the cut-off threshold to cover over 90% of diphone contexts that appeared in the lexicon, resulting in about 9.5K classifiers. For each GMM classifier at the second level, the GMM was allowed to add one mixture component per 50 training examples until there were 30 components. For the bottom level, the manner-based broad classes in Table 5.7 were used except that the Noise class was merged into the Silence class (and the labels for noises and artifacts were also simplified). Each classifier at the bottom level was allowed to add one mixture component per 50 training examples until 60 components were reached. Adding up the classifiers over the three levels, the resulting model had 25.2K classifiers and 646K Gaussian mixture components.

For the ML-E-Multi-Tri model, classifiers from the place-of-articulation-based broad

Example Counts	Number of Triphone States
> 0	102K
> 100	37.1K
> 200	26.3K
> 400	17.2K
> 800	10.3K
> 1600	5.6K
> 3200	2.7K

Table 5.12: Counts statistics of triphone phone states in the training lectures.

Model	#Mixtures	Dev WER	Test WER
ML-B-Multi-Tri	646K	39.6%	31.3%
ML-E-Multi-Tri	907K	39.7%	31.3%
MCE-B-Multi-Tri	646K	34.9%	27.4%
MCE-E-Multi-Tri	907K	34.9%	27.1%

Table 5.13: WERs of the multi-level triphone acoustic models on the Dev and Test lectures.

classes in Table 5.8 (with the Noise class merged into the Silence class) were added to the bottom level. The resulting extended model had 30.8K classifiers and 907K components in total. In terms of the assigning the combination weights, the principles described in Section 3.2.4 were used.

For both the ML-B-Multi-Tri and ML-E-Multi-Tri models, MCE training was used to refine the GMM parameters. To speed up the training process, a larger initial learning rate ϵ for the Quickprop algorithm was used[6], where other operational constants were the same as the training for the clustering-based triphone model. The WER on the Dev lectures was used as the selection criterion among different MCE iterations. Table 5.13 lists the WERs of the multi-level models on the Dev and Test lectures. As in the diphone scenario, the multiple broad-class extension provided extra WER improvement on the Test lectures after MCE training.

[6] Both of them started with $\epsilon = 2.5$, but the ϵ for the basic model was set to 1.0 after iteration 7 because the length of the gradient suddenly became much larger at that iteration. For the extended model, however, the learning rate was kept the same all the way.

Learning Combination Weights

Instead of using the default assignments, the combination weights could also be automatically learned as described in the later part of Section 3.2.3. One question is whether the learning of weights can further improve the system performance with MCE trained GMM parameters. The following summarizes several attempts to further improve the MCE-B-Multi-Tri and MCE-E-Multi-Tri models via adjusting the weights.

The first attempt was to directly run the constrained optimization in Equation (3.7) on the training lectures. Although doing so could keep improving the MCE objective function, the WER on the Dev lectures got worse. Looking at the resulting weights, we found that a lot of weights of the top level classifiers were very close to 1. One possible reason for this might be that the MCE training might have reduced the confusions of a significant amount of triphone states that have top level classifiers, and made the model become more confident on those top level classifiers. Since each top level classifier was associated to one unique triphone state, setting a larger combination weight on a confident top level classifier might potentially further reduce confusions on the training lectures. This self amplification might potentially hurt the generalization ability of the model.

Based on the lesson from the first attempt, the second attempt was to run the weight optimization on the Dev lectures. While doing so successfully reduced the WER on the Dev set, it resulted in almost no changes on the WER of the Test lectures. One possible reason for this might be that there were too many free parameters in the optimization. Because each triphone state could have its own independent set of combination weights based on Equation (3.7), there could be $O(10^5)$ free parameters in the weight optimization. It might be that the set of triphone states appearing in the Dev lectures might just weakly overlap with those appearing in the Test lectures. Therefore, if the weights of all the triphone states were allowed to move independently, what was learned from the Dev lectures might not be transferable to the Test lectures.

As a result, the third attempt was to tie the weights according to broad class triples during the optimization, as described in the last paragraph of Section 3.2.3. As shown in Table 5.14, doing so reduced the WERs on both the Dev and Test lectures. Also, the weight

Model	Dev WER	Test WER
MCE-B-Multi-Tri	34.9%	27.4%
wMCE-B-Multi-Tri	34.8%	27.3%
MCE-E-Multi-Tri	34.9%	27.1%
wMCE-E-Multi-Tri	34.7%	27.0%

Table 5.14: WERs of the multi-level triphone acoustic model before and after weight optimization. The models with w as the prefix denote the ones with weight optimization.

optimization provided better WER improvement for the extended model on the Dev lectures than for the basic model. This result was consistent with the fact that the extended model has more free parameters and should perform better on the data set where the optimization were conducted. One final remark is that while model with the weight optimization does not seem to have a large improvement over the model with default weights, the weights can be influential to WERs. For example, randomly assigning positive weights (but with the weights of each state summed to one) can make the WERs of both ML and MCE trained models get worse by $1 \sim 2\%$ absolute, compared with models using the default weight assignment. This fact suggests that the default weight assignment might be close to a local optimum.

5.3.10 Comparisons and Discussions

Figure 5-2 summarizes the WERs of the acoustic models on the Test lectures. As shown in the figure, MCE training significantly improved the test WER cross all types of models. Among the MCE trained models, the multi-level (Multi) models outperformed the clustering-based (CL) models on both the diphone and triphone-based systems. This result shows that the multi-level framework can be integrated into different types of systems and combine well with discriminative training methods. Among the MCE trained multi-level models, the extended (E-Multi) models provided better WERs than the basic (B-Multi) models. Based on the McNemar significant test [33], the improvements of MCE-Multi over MCE-CL and the improvements of MCE-E-Multi over MCE-B-Multi were all statistically significant (with $p < 0.001$). As a comparison, the MCE-CL-Tri model has similar performance as the triphone model in [87] on the lecture data.

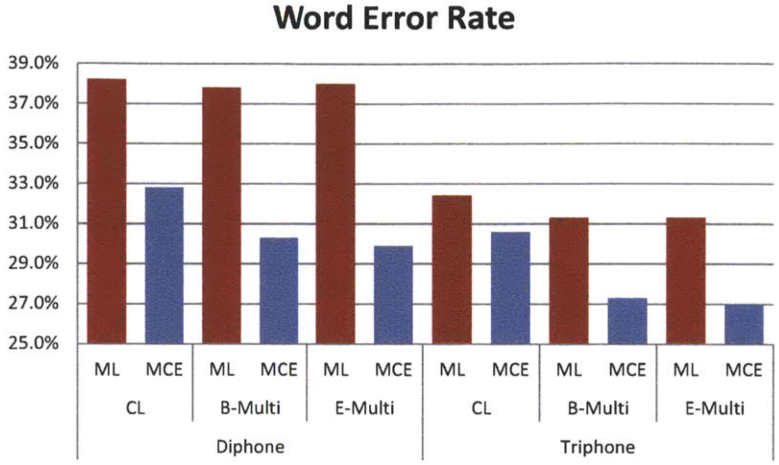


Figure 5-2: WERs of the acoustic models on the Test lectures.

In terms of computation, under the same pruning threshold during decoding, the MCE-B-Multi-Di model was about $10 \sim 20\%$ slower than the MCE-CL-Di model, while the MCE-B-Multi-Tri was about two times slower than the MCE-CL-Tri model. One possible cause of the difference between the diphone and triphone-based systems might be that the acoustic models and the FSTs are much larger for the triphone-based system, resulting in higher memory overhead. Different pruning thresholds were also used to evaluate the models. If the pruning threshold was adjusted for the MCE-CL-Tri model such that it took similar amount of decoding time as the MCE-Multi-B-Tri model under the default pruning threshold, its WER on the test lectures improved about 0.2% absolute. On the other hand, if the pruning threshold was adjusted for the MCE-Multi-B-Tri model such that it took similar amount of decoding time as the MCE-CL-Tri model under the default pruning threshold, its WER on the test set degraded about 1.0% absolute. While there were trade-offs between accuracy and decoding speed, the improvements of the multi-level models over the clustering-based models were still significant.

While the multi-level models provided improvements over clustering-based models under both ML and MCE training criteria, the gains under MCE training were larger. One possible explantation is as follows. During the MCE training of clustering-based models, if there were two states s_1 and s_2 belonging to the same cluster but one showing up in the reference and the other showing up in one of the competing hypotheses, then the MCE training could not correct this type of confusion. On the other hand, under the multi-level framework, there would be at least one classifier associated with s_1 and one classifier associated with s_2 that could be adjusted to correct this confusion. In this way, MCE training can potentially correct more confusions for the multi-level models, resulting in larger performance gains.

Also, as shown in Table 5.5 and 5.11, at some point, the MCE training became less effective for the clustering-based models as the number of parameters became larger. Investigating the MCE objective functions showed that the training objective functions kept improving as the number of parameters increased. These facts suggest that some over-training effects might have occurred. On the other hand, the multi-level model kept improving while the number of parameters increased. Also, although there were many more parameters in the extended multi-level models than the basic multi-level models, the MCE training still provided some improvements. These facts suggests that the multi-level framework is potentially less likely to suffer over-training as the number of parameters increases.

While the models used in the experiments of this chapter were all trained with the 119-hour training lectures listed in Table 5.1, it would also be interesting to compare how the clustering-based and the multi-level models perform with different amounts of training data. When the amount of training data is reduced, the data sparsity problem becomes more severe, and it would be an interesting scenario to evaluate how effectively the model addresses the data sparsity problem. Although not reported in this chapter, part of the experimental results reported in Chapter 6 suggest that the multi-level model provides better performance improvement over the clustering-based model when the amount of training data is more limited. This can potentially be an advantage when deploying an ASR system for a language with more limited resources.

5.4 Chapter Summary

In this chapter, a large-vocabulary lecture transcription task on the MIT Lecture Corpus was used to evaluate whether the multi-level framework could be integrated into existing

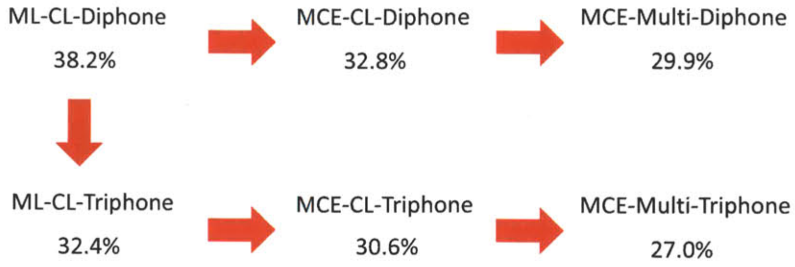


Figure 5-3: Summary of the word error rate improvements of different combinations of modeling techniques on the Test lectures.

ASR systems and provided better recognition accuracy over conventional clustering-based acoustic model. Different modeling techniques were compared in the experiment: in terms of training criteria, Maximum-Likelihood (ML) v.s. Minimum Classification Error (MCE); in terms of length of context-dependency, Diphone v.s. Triphone; and in terms of ways of addressing data sparsity, Clustering (CL) v.s. Multi-level (Multi). Figure 5-3 summarizes the word error rate improvements of different combinations of modeling techniques on the Test lectures. The experiment results confirmed that the multi-level framework can be integrated into existing large-vocabulary ASR systems, and combines well with discriminative training methods.

Chapter 6

Adaptive Acoustic Modeling

6.1 Motivation

Although most ASR systems are pre-trained, it is usually desirable for them to be able to adapt to incoming speech data since significant performance improvement are often achievable [133, 134]. Therefore, when introducing a new modeling component to the system, it is important to evaluate how the new component might affect the adaptability of the system. In this chapter, a large-vocabulary speaker adaptation experiment is conducted to test whether the multi-level acoustic model can adapt better than the conventional clustering-based acoustic model. Several speaker adaptation algorithms and adaptation scenarios are first reviewed, and then, the experimental results are reported.

6.2 Overview of Speaker Adaptation

6.2.1 Speaker Adaptation Algorithms

Speaker adaptation methods have been studied extensively for GMM-based ASR. One of the simplest and most effective method is Maximum A Posteriori (MAP) adaptation that updates GMM parameters of a Speaker Independent (SI) model to maximize the posterior probability of the adaptation data with respect to the updated parameters [32]. Details of MAP-based adaptation can be found in Appendix C.

Instead of updating each Gaussian component individually as in the MAP adaptation algorithm, the Maximum Likelihood Linear Regression (MLLR) algorithm groups Gaussian components and estimates a linear transform of the SI GMM means to the corresponding Speaker Dependent (SD) distribution in order to maximize the likelihood of the adaptation data [79]. Eigenvoice [69] and reference speaker weighting [49] use multiple reference speakers to represent a speaker vector that is a concatenation of Gaussian means. The adapted speaker vector is determined using a Maximum-Likelihood (ML) criterion to derive a linear combination of the reference speaker vectors. Although these methods are all effective with limited adaptation data, MAP adaptation typically provides the largest improvement in WER when there is a significant quantity of adaptation data [69].

While the aforementioned methods use ML-based criterion, discriminative methods have also been investigated, and several types of discriminative objective functions have been successfully used for speaker adaptation. For example, the statistics of Maximum Mutual Information (MMI) training have been used to formulate a Conditional MLLR (CMLLR) adaptation framework [43]. Minimum Phone Error (MPE) training has also been shown to be effective in estimating the regression transformation matrix [122]. If enough adaptation data are available, the entire set of GMM parameters can be adapted via the discriminative MAP method [102]. In addition, training criteria such as Minimum Classification Error (MCE) has also been shown to be effective for speaker adaptation [50].

6.2.2 Adaptation Scenarios

Depending on whether the reference transcriptions of the adaptation data are available, the adaptation scenarios can be categorized into a supervised adaption or an unsupervised adaptation. Under a supervised adaptation scenario, both the speech data from the speaker and the corresponding reference transcriptions are available. On the other hand, only the speech data are available under an unsupervised adaption scenario. While supervised adaptation provides much better performance improvement than unsupervised adaptation, investigating model performances under unsupervised scenario is also important. This is because collecting the speech data is generally much easier than collecting both the speech data and the corresponding references. In the adaptation experiments reported in this chapter, the performance of the clustering-based and multi-level models were evaluated under both supervised and unsupervised adaptation scenarios.

Depending on whether the system considers previous adaptation data, and the initial model, the adaptation scenarios can also be categorized into a batch adaptation scenario or an online adaptation scenario. Under a batch adaptation scenario, the model is not updated until a significant amount of adaptation data is available. When doing the update, the initial model is used as the starting point of the adaptation, and all the previous adaptation data are used during the update. On the other hand, under an online adaptation scenario, the model is adjusted based on the currently available adaptation data, and the model keeps changing as new adaptation data comes in. Typically, batch adaptation takes more time to update the model, but it usually provides better and more stable performance improvement. In the experiments reported in this chapter, only the model performance under batch adaptation were investigated since batch adaptation provides a more stable measure on how well the model adapt.

Finally, in many cases, there can be many potential speakers that will use the ASR system. In such a multi-speaker scenario, the speakers and corresponding adaptation data are typically clustered into several groups, and a speaker adaptive model is constructed for each group. During recognition, either a speaker identification procedure can be used to select which speaker adaptive model to use; or the systems can run in parallel, and the best scoring one can be selected as the output. Since the focus of this research is to compare the adaptation abilities between the clustering-based model and the multi-level model, investigating model performance under a single speaker scenario is sufficient.

6.3 Supervised Adaptation Experiment

6.3.1 Setup

A series of lectures on introductory mechanical physics (8.01-1999) taught by a Dutch-accented lecturer in the MIT Lecture Corpus were used for the speaker adaptation exper-

Model	SI-ML-CL	SI-MCE-CL	SI-MCE-Multi
WER	32.3%	31.7%	28.6%

Table 6.1: WER of Speaker-Independent (SI) models on the three evaluation lectures of 8.01-1999.

iment. The audio and transcripts of the first 30 lectures were used as potential adaptation data, while the last 3 recorded lectures (L31, L32, and L34) were used as the evaluation data. As in the experiments in Chapter 5, WER was used as the evaluation metric.

The same ASR configuration as described in Chapter 5 was used for the evaluation except that the transcriptions of 8.01-1999 lectures were removed from the training text of the language model. Other parts of the configuration, including acoustic observation extraction, HMM topology, pronunciation rules, lexicon, and vocabulary, stayed the same. More details about the impacts of the language model can be found in Section 6.3.3.

6.3.2 Acoustic Models

Speaker-Independent Acoustic Models

Several triphone-based acoustic models trained in the experiments of Chapter 5 were used as Speaker-Independent (SI) models to compare the effect of adaptation. For the clustering-based models, the ML-CL-Tri and MCE-CL-Tri (with MAX_MIX=60) models in Chapter 5 were used. Since only the triphone models were compared in the adaptation experiments, we call the models SI-ML-CL and SI-MCE-CL, respectively. For the multi-level model, the MCE-B-Multi-Tri (without weight optimization) was used. Similarly, we call the model SI-MCE-Multi for the rest of the chapter. The WERs of the SI models on the three evaluation lectures are listed in Table 6.1.

Speaker-Adaptive Acoustic Models

To evaluate how the amount of adaptation data might affect model performance, the adaptation lectures were arranged into four incremental groups: "L01 to L05", "L01 to L10", "L01 to L20", and "L01 to L30". To construct the MCE-based Speaker-Adaptive (SA) model, the SI-MCE-CL and SI-MCE-Multi models were used as initial models, and several iterations of MCE training were conducted on the data of each incremental group, resulting in SA-MCE-CL and SA-MCE-Multi models shown in Figure 6-1. As in the experiments in Chapter 5, unigram language models[1] were used during the MCE training. Similar settings as in Table H.3 were used for the MCE training except that the `STEP_LIMIT` was set to 1.0 and that the learning rate ϵ was set to 25.0, 15.0, 10.0, and 7.5 respectively for each incremental group, respectively. Depending on the size of the data group, the MCE training converged in around $5-7$ iterations.

As a baseline to compare with the MCE-based adaptation, the commonly used Maximum A Posteriori(MAP) based adaptation was also implemented. The SI-ML-CL model was used as the initial model for the MAP adaptation, and the modified EM algorithm described in Appendix C was used to update the model. The performance of the MAP adaptation algorithm on different amounts of adaptation data correspond to the SA-MAP-CL line plotted in Figure 6-1.

Speaker-Dependent Acoustic Model

Speaker-dependent acoustic models were also trained on the adaptation data as a comparison to the effectiveness of adaptation. For the clustering-based models, the same stopping criteria described in Section 5.3.8 were used for growing the decision trees. Different amounts of adaptation data resulted in different amounts of clustered states. For the GMM of each clustered state, it was allowed to add one mixture component until 60 components were reached. Both ML and MCE[2] training were applied to learn the GMM parameters. Table 6.2 lists the WERs of the speaker-dependent acoustic models with respect to different amounts of adaptation (training) data. As shown in the table, when the number of training lectures increased, the WERs of the speaker-dependent models on both the training and test lectures also improved. While the MCE training significantly reduced the training WERs, the reductions of the test WERs were not as much. This result might be due to the fact that

[1] The transcriptions of the adaptation lectures were included when constructing the unigram language models.

[2] The operational constants for the MCE training were the same as what used to train the Speaker-Adaptive (SA) MCE models except that the learning rate ϵ was set to 7.5 for all the four data groups.

#Lectures	#States	#Mixtures	Training WER		Test WER	
			ML	MCE	ML	MCE
5	2.3K	23K	8.8%	4.8%	27.8%	27.9%
10	3.7K	43K	8.5%	4.9%	24.1%	24.0%
20	5.0K	78K	6.8%	4.4%	20.5%	20.0%
30	5.5K	107K	6.4%	4.1%	18.7%	18.1%

Table 6.2: WERs of speaker-dependent, clustering-based acoustic models.

#Lectures	#Classifiers	#Mixtures	Training WER		Test WER	
			ML	MCE	ML	MCE
5	7.2K	94K	6.3%	3.8%	24.7%	24.2%
10	8.7K	157K	7.2%	4.3%	21.5%	20.7%
20	10.8K	242K	7.2%	4.2%	19.1%	17.9%
30	12.2K	295K	7.7%	4.4%	17.8%	16.2%

Table 6.3: WERs of speaker-dependent, multi-level acoustic models.

the training errors were already low, and the models might have suffered certain degree of over-training. For comparison, the WERs of the MCE trained speaker-dependent models were plotted as the SD-MCE-CL line in Figure 6-1.

For the multi-level models, the basic structure in Figure 3-4 was used to construct the classifiers. As in Section 5.3.9, each top level classifier had to have more than 800 examples, and 15 Gaussian mixture components were trained for each top level classifier. Each second level classifier had to have more than 200 training examples and was allowed to add one mixture component every 50 training examples until there were 30 components. The manner-based broad classes used in Section 5.3.9 were also used to construct the bottom level classifiers, and each classifier was allowed to add one mixture component every 50 training examples until there were 60 components. The model parameters were initialized under the ML criterion, and were refined using MCE training. The operational constants used for the MCE training were the same as what were used for training the speaker-dependent, clustering-based models. Table 6.3 lists the sizes and WERs of the speaker-dependent, multi-level acoustic models with different amounts of training lectures. While the speaker-dependent, multi-level models had much larger sizes than the speaker-dependent, clustering-based models, the multi-level models had larger improve-

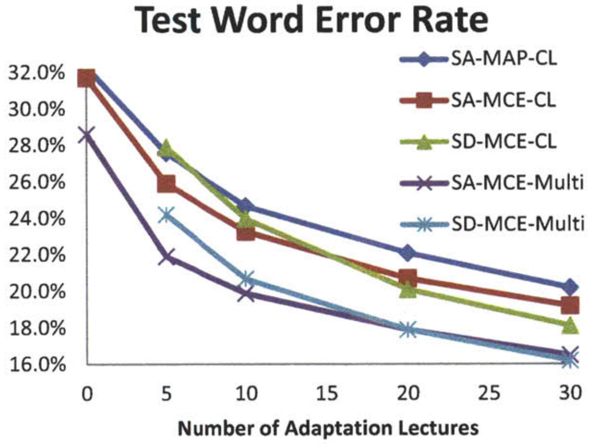


Figure 6-1: Supervised adaptation results.

ments after the MCE training. This fact suggests that the multi-level models are less likely to suffer over-training as the number of parameters increases. For comparison, the WERs of the MCE trained, speaker-dependent, multi-level acoustic models were plotted as the SD-MCE-Multi line in Figure 6-1.

6.3.3 Comparisons and Discussions

Figure 6-1 summarizes the performance of different types of acoustic models. As shown in Figure 6-1, both the MAP and MCE-based adaptation effectively reduced the WERs, and the gains over SI models (the points with 0 adaptation lectures) kept increasing as the amount of adaptation data increased. Also, the gains of the MCE adaptation were consistently larger than that of the MAP adaptation, suggesting that discriminative-based criteria can potentially provide better adaptation capability.

As shown by the "SA-MCE-Multi vs. SA-MCE-CL" line in Figure 6-2, the relative improvement of the SI-MCE-Multi model over the SI-MCE-CL model was about 10% (the point with 0 adaptation lectures). However, the relative improvements became $14 \sim 15\%$ when the models started adapting on the data. This result suggests that the multi-level model can potentially adapt better than the clustering-based model. One possible reason that contributes to such improvement might be as follows. Consider two triphone units s_1

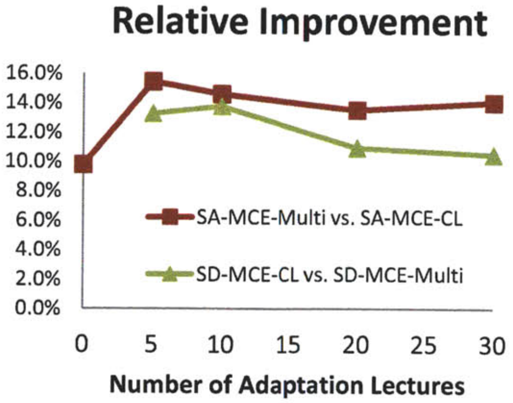


Figure 6-2: Relative improvements of multi-levels models over clustering-based models.

and s_2 that were clustered together by the speaker-independent, clustering-based model. The two units were clustered together because "on average" they were similar across the speakers that appeared in the training data. However, for the particular speaker to whom the model sought to adapt, the two units could be very different. Since the two units were clustered together, the clustering-based model was not able to capture the difference. On the other hand, under the multi-level framework, there would be at least one classifier that could be modified to capture the difference.

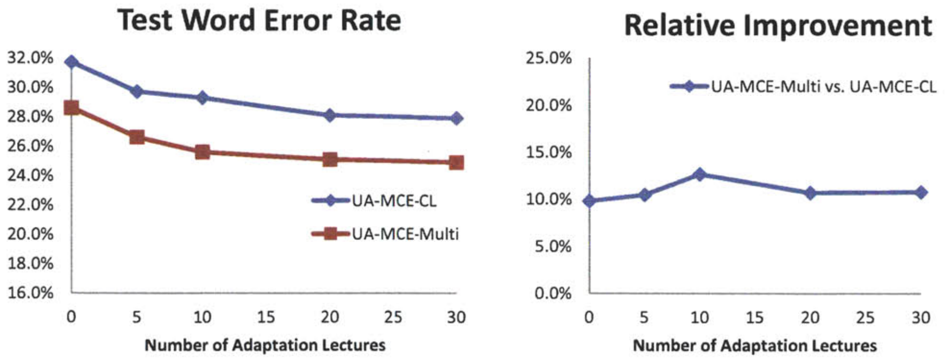
Comparing the SA-MCE-CL and SD-MCE-CL models, the speaker-adaptive models performed better than the speaker-dependent models when less adaptation data were available, but the relative performance of the two models switched when there were about 15 adaptation lectures. On the other hand, the SA-MCE-Multi model had at par performance as the SD-MCE-Multi model when there were 20 adaptation lectures, and it was only slightly worse than the SD-MCE-Multi model when all the 30 lectures were used. This result also supports that the multi-level model has better adaptation capability than the clustering-based model in the sense that under the multi-level framework, it requires more data for the speaker-dependent model to provide competitive performance as the speaker-adaptive model.

Comparing the speaker-dependent models, as shown by the "SD-MCE-Multi vs. SD-MCE-CL" line in Figure 6-2, the speaker-dependent, multi-level model had larger relative improvement over the speaker-dependent, clustering-based model when there were fewer training lectures. When fewer training data were available, the data sparsity problem could become more severe. Having larger improvement when fewer data were available suggests that the multi-level model can potentially handle the data sparsity problem more effectively, and might be less likely to suffer over-training. This can be advantageous when deploying an ASR system for a language that has relatively limited resources.

Although not shown, the effects of using different language models were also evaluated. When including the transcriptions of all the 8.01-1999 lectures (including the three evaluation lectures) to the training texts of the language model, the performances of the models still followed the same trend as shown in Figure 6-1, but the WERs of the models improved about 2% absolute. Another experiment included only the transcriptions from the adaptation lectures to the training texts. Doing so could also consistently improved the WERs, and the gain increased slightly as the amount of adaptation data increased. However, even at 30 adaptation lectures, the improvements were only around 0.5% absolute. Therefore, the main impact was from including the transcriptions of the test lectures (the last three in the lecture series). Nevertheless, the usages of different language models did not affect the analysis described in the previous paragraphs.

6.4 Unsupervised Adaptation Experiment

To start the unsupervised adaptation, the audio data of the adaptation lectures were first recognized by the MCE trained Speaker-Independent (SI) model with the trigram language model. Then, the best scoring recognition hypothesis for each utterance was used as the "reference", and was used for the MCE based adaptation to generate a Unsupervised-Adaptive (UA) model. The results of the unsupervised adaptation are plotted in Figure 6-3. As shown in the figure, although improvement of the UA-MCE-Multi model over the UA-MCE-CL model was not as large in the supervised scenario, it still provided about 10% relative improvement consistently over different amounts of adaptation data. This result is positive in the sense that the improvement from the SI training stayed relatively constant in the adaptation, and that the multi-level model was consistently better through the adaptation



(a) WERs on the evaluation lectures.

(b) Relative improvements of the UA-MCE-Multi model.

Figure 6-3: Unsupervised adaptation results.

process.

Several other ways of doing unsupervised adaptation were also investigated. For example, after the adapted model was generated, it could be used to re-recognize the speech again and generate a potentially more accurate "transcription" that could be used to adapt the model parameters. However, such an iterative procedure did not provide better performance on the test lectures, probably because some errors were accumulated during the iterative process. Another attempt was to first recognize a subset of the adaptation data, adapt on the subset, use the adapted model to recognize another subset, and so on so forth. However, the incremental procedure produced some cumulative errors over different subsets [3]. Such effect of cumulative errors might be mitigated if a "reference-lattice" rather than the best hypothesis was used for adaptation. While several details might need further investigation, the current experimental result still shows that the multi-level model can potentially adapt better against new speech data even in a unsupervised scenario.

[3] For example, on one of the subset of data, some silence part might be miss-recognized as a filler word, and when the model adapted on the "reference", it would try to generate a filler word on those miss-recognized parts, resulting in generating more insertions of the filler words on the next subset of data.

6.5 Chapter Summary

In this chapter, a large-vocabulary speaker adaptation experiment was conducted to compare the adaptation capabilities of the conventional clustering-based acoustic model and the multi-level acoustic model against new speech data. The experiment results showed that the multi-level model had about $14 \sim 15\%$ relative improvement over the clustering- based model under a supervised adaptation scenario, and had about 10% relative improvement under an unsupervised adaptation scenario. These results support the hypothesis that the multi-level model can adapt better against new data, which can potentially be beneficial when deploying ASR systems.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

One key message delivered in this thesis is that "not everything has enough training data, but different things can still be modeled differently by appropriately combining multiple levels of perspectives." The conventional context-dependent acoustic modeling framework deals with the data sparsity problem by grouping context-dependent phonetic units under a single perspective, typically guided by a heuristic decision tree. As a result, the conventional framework is "treating things that are supposed to be different the same" in the sense that the units grouped into the same cluster always have the same acoustic scores. This quantization side effect can potentially limit the performance of the acoustic model, in that certain types of confusions can become uncorrectable, and that the acoustic scores might become less informative to the recognizer.

In this work, a multi-level acoustic modeling framework was proposed to address both the data sparsity problem and the quantization effect. By appropriately combining classifiers that target at multiple levels of context specifications, each context-dependent unit can have a unique score, and therefore effectively "models different things differently" as it should be. Experiment results have shown that the multi-level framework can provide better phonetic prediction accuracy, significantly reduce recognition errors for large-vocabulary ASR systems, and potentially adapt better to incremental data. In this sense, we have shown that "modeling different units differently under data sparsity" is achievable for acoustic modeling of ASR systems, and doing so can provide a significant amount of performance improvement. While there can be many potential areas of future research, we point out several possible research directions as follows.

7.2 Future Work

Combining with Deep Belief Networks

In most existing ASR systems, including the work presented in this thesis, the acoustic scores are computed by GMMs. While GMMs have been used for several decades, recently, several alternatives [91, 22] have been proposed to replace the likelihoods of GMMs with posterior probabilities generated by Deep Belief Networks (DBNs) during decoding. The DBN-based framework has shown significant improvement over the GMM-based framework both in a context-independent phonetic recognition task [91] and a context-dependent large-vocabulary speech recognition task [22].

While providing significant improvements, the work in [22] still used clustering to group context-dependent states, and the DBNs were trained to only distinguish different clustered states. In other words, it still suffers the quantization effect, and the multi-level idea proposed in this work can potentially be applied to provide further improvement. One way to incorporate the multi-level idea is as follows. Note that each type of classifier in Figure 3-3 implicitly provides a perspective that specifies how the context-dependent units differ. For example, under the perspective of $\{<p_l,p_c,*>\}$, the triphone "r-iy+l" (as in "real") is the same as "r-iy+d" (as in "read") but is different from "d-iy+l" (as in "deal"). Under each kind of perspective, a number of different groups can be formed, and a set of DBNs can be trained to distinguish the groups that have enough training data. As a result, multiple sets of DBNs can be trained under multiple types of angles, and the (log) posterior probabilities of the DBNs can be appropriately combined to form the acoustic score for each context-dependent unit. In this way, the quantization effect can be avoid, and the performance of the DBN-based framework can potentially be further improved.

ASR for Languages with Limited Resources

In the conventional acoustic modeling framework, expert knowledge is required to construct separating questions through which the decision tree can grow. However, such kind of language-specific expert knowledge might not be easily available when deploying an ASR system for a language where knowledge sources are relatively scarce. On the other hand, the proposed multi-level framework utilizes only the notion of broad classes to construct classifiers which requires less expert knowledge, and can potentially generalize better across different languages. Also, the broad classes can potentially be learned from data [105], which can further reduce the required amount of expert knowledge. In addition, because multi-level structures incorporate classifiers targeting both high and low levels of context resolution, even when the total amounts of training data are relatively small, the low level classifiers can still have a reasonable number of training examples. As a result, the multi-level framework should still function reasonably well when fewer amounts of training data are available. These facts suggest that the multi-level framework can be potentially advantageous when deploying an ASR system for a language with relatively limited resources, in terms of both expert knowledge and quantity of training data. It would be interesting to see how it actually works in the limited resources scenario.

Incremental Learning with Crowdsourcing

Recently, several fields of research have been starting to use crowdsourcing platforms such as Amazon Mechanical Turk to utilize the power of a large number of non-experts for data analysis and labeling, and there have been several successful experiments that utilize crowdsourcing to transcribe speech data [88, 84, 74]. Although the non-experts might make more errors than experts when transcribing the data, the crowds can generate much larger amounts of data with lower cost. Also, it has been shown that by dividing the entire transcription task into a sequence of easier sub-problems and appropriately utilizing ASR for automatic quality control, the transcription errors can be significantly reduced [74]. Since the multi-level framework can potentially generate better ASR results, and since the multi-level framework might have better adaptation capability (as shown in Chapter 6), having the multi-level framework interacting with the crowdsourcing mechanism can be beneficial, in the sense that it might provide better quality control and learn better. How to combine the multi-level framework with crowdsourcing to construct an incremental learning framework (potentially with feedback loops) would be also an interesting research direction.

Optimizing System Design

While the proposed multi-level framework provided significant improvements, there can be many possible ways of design the classifiers, as discussed in Chapter 3. Investigating classifier designs that can further improve the system performance would also be a possible future research direction. Also, combining the multi-level framework with quinphone based acoustic model or iterative lexicon learning methods can also potentially further improve the system, and would also be a possible direction to explore.

Appendix A

Maximum-Likelihood Estimation for Multivariate Gaussian Distribution

A D-dimensional random vector $\mathbf{x}$ belongs to a multivariate Gaussian distribution, if its probability density function is of the form

$$\mathcal{N}(\mathbf{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}}\exp(\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})), \tag{A.1}$$

where $\boldsymbol{\mu}$ is a D-dimensional mean vector, $(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}$ is the transpose of the column vector $(\mathbf{x}-\boldsymbol{\mu})$, $\boldsymbol{\Sigma}$ is a D by D symmetric positive-definite covariance matrix, and $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix. Given a set of K vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}$ independently drawn from a multivariate Gaussian distribution, the parameters of the distribution, mean $\boldsymbol{\mu}$ and covariance Σ, can be estimated under the ML criterion as follows. Since the K vectors are drawn independently, the log-likelihood of seeing the vectors can be expressed by

$$\sum_{k=1}^{K}\log(\mathcal{N}(\mathbf{x}_k;\boldsymbol{\mu},\boldsymbol{\Sigma})) = -\frac{KD}{2}\log(2\pi) - \frac{K}{2}\log(|\boldsymbol{\Sigma}|) - \sum_{k=1}^{K}\frac{1}{2}(\mathbf{x}_k-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_k-\boldsymbol{\mu}). \tag{A.2}$$

To find the $\boldsymbol{\mu}$ that can maximize the log-likelihood, we can take the partial derivative of Equation (A.1) with respect to $\boldsymbol{\mu}$ and make it equal to zero. As a result, the equation to solve $\boldsymbol{\mu}$ becomes

$$\sum_{k=1}^{K} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu}) = 0, \tag{A.3}$$

where the differentiation can be done using the formula in [2]. Solving Equation (A.3), we can find the Maximum-Likelihood estimate of the mean

$$\hat{\boldsymbol{\mu}} = \frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_k. \tag{A.4}$$

To find the ML estimate of the covariance matrix requires additional steps. First, using the chain rule and the differential formula of the determinant from [2], we can get

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \log(|\Sigma|) = \frac{1}{|\boldsymbol{\Sigma}|}|\Sigma|(\boldsymbol{\Sigma}^{-1})^{\mathrm{T}} = \boldsymbol{\Sigma}^{-1}. \tag{A.5}$$

Second, using the identity of the trace operation, and the chain-rule for matrix calculus, we can compute the following partial derivative by

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\Sigma}}(\mathbf{x}_k - \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu}) &= \frac{\partial}{\partial \boldsymbol{\Sigma}}\mathrm{tr}((\mathbf{x}_k - \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu})), && \text{(A.6)}\\
&= \frac{\partial}{\partial \boldsymbol{\Sigma}}\mathrm{tr}(\boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^{\mathrm{T}}), && \text{(A.7)}\\
&= \frac{\partial \boldsymbol{\Sigma}^{-1}}{\partial \boldsymbol{\Sigma}}\frac{\partial}{\partial \boldsymbol{\Sigma}^{-1}}\mathrm{tr}(\boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^{\mathrm{T}}), && \text{(A.8)}\\
&= \frac{\partial \boldsymbol{\Sigma}^{-1}}{\partial \boldsymbol{\Sigma}}(\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^{\mathrm{T}}, && \text{(A.9)}\\
&= -\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^{\mathrm{T}}, && \text{(A.10)}
\end{aligned}$$

where the second equality utilizes the trace identity, the third equality utilizes the chain-rule of the matrix calculus, the fourth equality utilizes the differential formula of the trace, and the final equality utilizes the identity

$$\frac{\partial \boldsymbol{\Sigma}\boldsymbol{\Sigma}^{-1}}{\partial \boldsymbol{\Sigma}} = \frac{\partial \boldsymbol{\Sigma}}{\partial \boldsymbol{\Sigma}}\boldsymbol{\Sigma}^{-1} + \boldsymbol{\Sigma}\frac{\partial \boldsymbol{\Sigma}^{-1}}{\partial \boldsymbol{\Sigma}} = 0. \tag{A.11}$$

Plugging in Equations (A.5) and (A.10) into the partial derivative of the log-likelihood with respect to $\mathbf{\Sigma}$, the equation to solve the ML estimate of $\mathbf{\Sigma}$ becomes

$$-\frac{K}{2}\mathbf{\Sigma}^{-1} + \sum_{k=1}^{K}\frac{1}{2}\mathbf{\Sigma}^{-1}\mathbf{\Sigma}^{-1}(\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^{\mathrm{T}} = 0. \tag{A.12}$$

Solving the above equation and plugging in the ML estimate for the mean, we can have the ML estimate for the covariance matrix

$$\hat{\mathbf{\Sigma}} = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{x}_k - \hat{\boldsymbol{\mu}})(\mathbf{x}_k - \hat{\boldsymbol{\mu}})^T = \frac{1}{K}\sum_{k=1}^{K}\mathbf{x}_k\mathbf{x}_k^{\mathrm{T}} - \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^{\mathrm{T}}. \tag{A.13}$$

Note that to ensure the ML estimate of the covariance matrix is positive-definite, the number of the training vectors K should be greater than the dimension D. For practical concerns, the covariance matrix might be assumed to be diagonal. In this case, each dimension can be treated independently, and parameters for each dimension can be estimated separately by changing the vector notations in Equations (A.4) and (A.13) into scalars.

Appendix B

Auxiliary Function for Optimization

The auxiliary function is a commonly-used tool when it is difficult to directly optimize an objective function. Let $F(\boldsymbol{\lambda})$ be the function to be optimized and let $\bar{\boldsymbol{\lambda}}$ denote the current model parameters. An auxiliary function $Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ is a function that is closely related to $F(\boldsymbol{\lambda})$, but is much easier to optimize, typically with a closed-form optimum. If updating the parameters to the optimum of the auxiliary function also improves the original objective function, then we can construct a new auxiliary function at the new operating point and do the update again. If, at each iteration, we can guarantee improving the objective function, then the iterative procedure can make the model converge to a stationary point, typically a local optimum. Depending on how well an auxiliary function can guarantee improvement, it can be either a strong-sense auxiliary function, or a weak-sense auxiliary function as described in [99].

B.1 Strong-sense and Weak-sense Auxiliary Functions

Without loss of generality, we can assume that we are maximizing $F(\boldsymbol{\lambda})$. A strong-sense auxiliary function is one that satisfies the following property:

$$F(\boldsymbol{\lambda}) - F(\bar{\boldsymbol{\lambda}}) \geq Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}}) - Q(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\lambda}}) \qquad \forall \boldsymbol{\lambda}. \tag{B.1}$$

On the other hand, a weak-sense auxiliary function satisfies

$$\left.\frac{\partial F(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}=\bar{\boldsymbol{\lambda}}} = \left.\frac{\partial Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}=\bar{\boldsymbol{\lambda}}}. \tag{B.2}$$

Note that if $Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ is a continuously differentiable strong-sense auxiliary function, it is also a weak-sense auxiliary function. To show this, let f_i be the value of $\left.\frac{\partial F(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}=\bar{\boldsymbol{\lambda}}}$ at the i^{th} dimension, q_i be that of $\left.\frac{\partial Q(\boldsymbol{\lambda},\bar{\boldsymbol{\lambda}})}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}=\bar{\boldsymbol{\lambda}}}$, and $\boldsymbol{e}_i$ be the unit vector of the i^{th} dimension. If $f_i > q_i$, then there exists a small enough $d > 0$ such that $\frac{F(\bar{\boldsymbol{\lambda}}-d\boldsymbol{e}_i)-F(\bar{\boldsymbol{\lambda}})}{-d} > \frac{Q(\bar{\boldsymbol{\lambda}}-d\boldsymbol{e}_i,\bar{\boldsymbol{\lambda}})-Q(\bar{\boldsymbol{\lambda}},\bar{\boldsymbol{\lambda}})}{-d}$, which contradicts Inequality (B.1). Similarly, $f_i < q_i$ can also cause a contradiction. Therefore, f_i has to be equal to q_i for every i, resulting in Equation (B.2).

For a strong-sense auxiliary function, an improvement in the auxiliary function guarantees a larger or equivalent improvement in the objective function, and thus the objective function is guaranteed to improve at each iteration. Also, from Equation (B.2), if the auxiliary function converges to a stationary point that $\left.\frac{\partial Q(\boldsymbol{\lambda},\bar{\boldsymbol{\lambda}})}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}=\bar{\boldsymbol{\lambda}}} = 0$, then the gradient of $F(\boldsymbol{\lambda})$ at $\bar{\boldsymbol{\lambda}}$ is also 0. So the iterative procedure will keep improving the objective function until it converges to a stationary point, and to a local optimum for many problems.

On the other hand, a weak-sense auxiliary function does not have such a guarantee. However, if the update of an iteration does not move the parameters too far, it can also improve the objective function. Also, from Equation (B.2), if the auxiliary function converges to a local optimum, then the objective function also converges. As described in [99], a smoothing term can be added to a weak-sense auxiliary function at each iteration such that the optimum of the auxiliary function can be closer to the starting point in order to ensure the improvement.

B.2 Auxiliary Function for the EM Algorithm

In this section, we show that the auxiliary function used in the EM algorithm is a strong-sense auxiliary function, and thus the procedure can guarantee to converge to a local optimum. At each iteration, let $q_{\mathbf{S}}$ be the posterior probability $q_{\mathbf{S}} = \frac{p_{\bar{\boldsymbol{\lambda}}}(\mathbf{X},\mathbf{S})}{\sum_{\mathbf{S}'} p_{\bar{\boldsymbol{\lambda}}}(\mathbf{X},\mathbf{S}')}$ based on the current parameter vector $\bar{\boldsymbol{\lambda}}$, and the auxiliary function can be expressed by

$$Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}}) = \mathbb{E}_{\mathbf{S}|\mathbf{X};\bar{\boldsymbol{\lambda}}}[\log(p_{\boldsymbol{\lambda}}(\mathbf{X}, \mathbf{S}))] = \sum_{\mathbf{S}} q_{\mathbf{S}} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}, \mathbf{S})). \tag{B.3}$$

As a result, the difference $F(\bar{\boldsymbol{\lambda}}) - Q(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\lambda}})$ becomes

$$F(\bar{\boldsymbol{\lambda}}) - Q(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\lambda}}) = \log(\sum_{\mathbf{S}} p_{\bar{\boldsymbol{\lambda}}}(\mathbf{X}, \mathbf{S})) - \sum_{\mathbf{S}} q_{\mathbf{S}} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}, \mathbf{S})) \tag{B.4}$$

$$= -\sum_{\mathbf{S}} q_{\mathbf{S}} \log(q_{\mathbf{S}}). \tag{B.5}$$

Similarly, let $r_{\mathbf{S}} = \frac{p_{\boldsymbol{\lambda}}(\mathbf{X},\mathbf{S})}{\sum_{\mathbf{S}'} p_{\boldsymbol{\lambda}}(\mathbf{X},\mathbf{S}')}$, and the difference $F(\boldsymbol{\lambda}) - Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}})$ can be expressed by

$$F(\boldsymbol{\lambda}) - Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}}) = \log(\sum_{\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}, \mathbf{S})) - \sum_{\mathbf{S}} q_{\mathbf{S}} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}, \mathbf{S})) \tag{B.6}$$

$$= -\sum_{\mathbf{S}} q_{\mathbf{S}} \log(r_{\mathbf{S}}). \tag{B.7}$$

Therefore, if we subtract Equation (B.5) from (B.7),

$$F(\boldsymbol{\lambda}) - Q(\boldsymbol{\lambda}, \bar{\boldsymbol{\lambda}}) - (F(\bar{\boldsymbol{\lambda}}) - Q(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\lambda}})) = -\sum_{\mathbf{S}} q_{\mathbf{S}} \log(\frac{r_{\mathbf{S}}}{q_{\mathbf{S}}}) \tag{B.8}$$

$$= D_{KL}(q_{\mathbf{S}}||r_{\mathbf{S}}), \tag{B.9}$$

where $D_{KL}(q_{\mathbf{S}}||r_{\mathbf{S}})$ is the K-L divergence [70] of the two distributions, and is greater than or equal to zero. In this way, we have shown that Inequality (B.1) holds, and that the auxiliary function used in the EM algorithm is a strong-sense auxiliary function.

Appendix C

Maximum a Posteriori Adaptation for Gaussian Mixture Model

In this appendix, we summarize the formulation of the Maximum A Posteriori (MAP) based adaption for Gaussian Mixture Model (GMM), which was derived in [32]. The MAP framework in [32] assumes that the prior distribution of the mixture weights $\{\omega_1, \ldots, \omega_M\}$ follows the Dirichlet distribution [59]:

$$g(\omega_1, \ldots, \omega_M; \nu_1, \ldots, \nu_M) \propto \prod_{m=1}^{M} \omega_m^{\nu_m - 1}, \tag{C.1}$$

where $g(\cdot)$ denotes the probability density function, and $\{\nu_1, \ldots, \nu_M\}$ are hyper parameters that can be thought as prior counts of the mixture components. Note that the normalization constant is omitted in the above formula. Under the Dirichlet distribution above, the expectation of ω_m can be computed by

$$\mathbb{E}[\omega_m] = \frac{\nu_m}{\sum_{m'=1}^{M} \nu_{m'}}, \tag{C.2}$$

which is of exactly the same form as if doing Maximum-Likelihood (ML) estimation on the mixture weights when seeing the m^{th} mixture component ν_m times, respectively. This is why we can call $\{\nu_1, \ldots, \nu_M\}$ prior counts.

For the mean $\boldsymbol{\mu}_m$ and covariance Σ_m of the m^{th} mixture component, it is assumed that their prior distribution follows the normal-Wishart distribution [24]. Under this assumption, the prior distribution of the inverse covariance matrix $\mathbf{\Sigma}_m^{-1}$ follows the Wishart distribution with a hyper parameter κ_m and a positive-definite symmetric matrix $\mathbf{C}_m$, and the prior distribution of the mean $\boldsymbol{\mu}_m$ follows the normal (Gaussian) distribution with prior mean $\mathbf{u}_m$ and covariance matrix $\mathbf{\Sigma}_m$. As a result, the density of the prior distribution of $\boldsymbol{\mu}_m$ and $\mathbf{\Sigma}_m$ is of the following form:

$$\begin{aligned} h(\boldsymbol{\mu}_m, \mathbf{\Sigma}_m; \tau_m, \mathbf{u}_m, \kappa_m, \mathbf{C}_m) \propto & \; |\mathbf{\Sigma}_m^{-1}|^{\frac{\kappa_m - D - 1}{2}} \exp(-\frac{1}{2}\mathrm{tr}(\mathbf{C}_m \mathbf{\Sigma}_m^{-1})) \\ & |\mathbf{\Sigma}_m|^{-\frac{1}{2}} \exp(-\frac{\tau_m}{2}(\boldsymbol{\mu}_m - \mathbf{u}_m)^{\mathrm{T}} \mathbf{\Sigma}_m^{-1} (\boldsymbol{\mu}_m - \mathbf{u}_m)), \end{aligned} \tag{C.3}$$

where the first line corresponds to the Wishart distribution of the inverse covariance matrix $\mathbf{\Sigma}_m^{-1}$, the second line corresponds to the normal distribution of the mean $\boldsymbol{\mu}_m$, D denotes the dimension, and τ_m is a hyper parameter that determines how close is $\boldsymbol{\mu}_m$ to its prior $\mathbf{u}_m$.

Given a set of adaptation data $\{\mathbf{x}_1, \ldots \mathbf{x}_T\}$, the MAP adaptation method seeks to find the GMM parameters that can maximize the sum of the two terms: the log-likelihood of the data and the log of the prior density. A similar EM algorithm as described in Section 2.1.5 can be used by adjusting the update formula as follows. As in the Maximum-Likelihood (ML) estimation, given the current GMM parameters, the posterior probability of $\mathbf{x}_t$ being drawn from the m^{th} mixture component, r_{tm}, can be computed by Equation (2.25). For the mixture weights, if we add the log of Equation (C.1) to the objective function, it is equivalent to adding the count of the m^{th} mixture component by $\nu_m - 1$. As a result, the update formula for the mixture weight becomes

$$\hat{\omega}_m = \frac{(\nu_m - 1) + \sum_{t=1}^{T} r_{tm}}{\sum_{m'=1}^{M} (\nu_m - 1) + T}. \tag{C.4}$$

For the update of mean $\boldsymbol{\mu}_m$, the difference between MAP adaptation and ML estimation is the second line of Equation (C.3). Adding the log of the term into the objective function is equivalent to adding τ_m training examples with value equals to $\mathbf{u}_m$. As a result, the update formula for the mean becomes

$$\hat{\boldsymbol{\mu}}_m = \frac{\tau_m \mathbf{u}_m + \sum_{t=1}^{T} r_{tm} \mathbf{x}_t}{\tau_m + \sum_{t=1}^{T} r_{tm}}. \tag{C.5}$$

The update for the covariance matrix $\boldsymbol{\Sigma}_m$ is more complex. Taking the log of Equation (C.3), the terms related to the determinant $|\boldsymbol{\Sigma}_m|$ sum to $-\frac{\kappa_m - D}{2}\log(|\boldsymbol{\Sigma}_m|)$. Similar to Appendix A, the coefficient $\kappa_m - D$ will be added to the denominator of the update formula of $\boldsymbol{\Sigma}_m$. So we can ignore the $-1/2$ in the exponents when deriving the update formula. The term $\frac{\tau_m}{2}(\boldsymbol{\mu}_m - \mathbf{u}_m)^T(\boldsymbol{\mu}_m - \mathbf{u}_m)$ contributes to the numerator of the covariance update by $\tau_m(\boldsymbol{\mu}_m - \mathbf{u}_m)(\boldsymbol{\mu}_m - \mathbf{u}_m)^T$. For the term $\frac{-1}{2}\mathrm{tr}(\mathbf{C}_m\boldsymbol{\Sigma}_m^{-1})$, the trace identity in [2] can be use to compute its partial derivative with $\boldsymbol{\Sigma}_m$, and it contributes to the numerator of the covariance update by $\mathbf{C}_m^T$. Since $\mathbf{C}_m$ is symmetric, the update formula of the covariance matrix becomes

$$\hat{\boldsymbol{\Sigma}}_m = \frac{\mathbf{C}_m + \tau_m(\hat{\boldsymbol{\mu}}_m - \mathbf{u}_m)(\hat{\boldsymbol{\mu}}_m - \mathbf{u}_m)^T + \sum_{t=1}^{T} r_{tm}(\mathbf{x}_t - \hat{\boldsymbol{\mu}}_m)(\mathbf{x}_t - \hat{\boldsymbol{\mu}}_m)^T}{(\kappa_m - D) + \sum_{t=1}^{T} r_{tm}}, \tag{C.6}$$

where $\hat{\boldsymbol{\mu}}_m$ is the updated mean from Equation (C.5).

In terms of setting the hyper parameters of the prior distribution, commonly used settings are as follows. $(\nu_m - 1)$, τ_m, and $(\kappa_m - D)$ are set to the count of the m^{th} mixture component in the original (non-adapted) model. $\mathbf{u}_m$ is set to the mean of the original model, and $\mathbf{C}_m$ is set to the covariance matrix of the original model times the count of the mixture component in the original model. While not covered in this appendix, a similar update formula as Equation (C.4)-(C.6) can be derived for the Forward-Backward algorithm by plugging in the posterior probability of the state at each time point.

Appendix D

Splitting Nodes in Decision Tree Clustering

In this appendix, we describe the commonly-used criterion of selecting the splitting questions for the decision tree clustering algorithm. The objective function used in decision tree clustering to select the splitting question is the increase of the log-likelihood after the splitting. The clustering algorithm assumes that the data within a node follows a Gaussian distribution and computes the mean and covariance for each node using ML estimation.

Suppose a question splits a node into two: the first node contains K_1 examples with mean $\boldsymbol{\mu}_1$ and covariance Σ_1, and the second node contains K_2 examples with mean $\boldsymbol{\mu}_2$ and covariance Σ_2. Let the original node have mean $\boldsymbol{\mu}_0$ and covariance Σ_0. Note that because the distribution is Gaussian, the means and covariances of these three nodes have the following relations.

$$\boldsymbol{\mu}_0 = \frac{K_1\boldsymbol{\mu}_1 + K_2\boldsymbol{\mu}_2}{K_1 + K_2}. \tag{D.1}$$

$$\Sigma_0 = \frac{K_1\Sigma_1 + K_1\boldsymbol{\mu}_1\boldsymbol{\mu}_1^\intercal + K_2\Sigma_2 + K_2\boldsymbol{\mu}_2\boldsymbol{\mu}_2^\intercal}{K_1 + K_2} - \boldsymbol{\mu}_0\boldsymbol{\mu}_0^\intercal, \tag{D.2}$$

where $K_1\Sigma_1 + K_1\boldsymbol{\mu}_1\boldsymbol{\mu}_1^\intercal$ represent the second order statistics of the data in the first new node, and $K_2\Sigma_2 + K_2\boldsymbol{\mu}_2\boldsymbol{\mu}_2^\intercal$ is that for the data in the second new node.

We can compute the log-likelihood of the first new node using the following identity

$$\begin{aligned}
\sum_{k=1}^{K_1}(\mathbf{x}_k - \boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{\Sigma}_1^{-1}(\mathbf{x}_k - \boldsymbol{\mu}_1) &= \mathrm{tr}(\sum_{k=1}^{K_1}\boldsymbol{\Sigma}_1^{-1}(\mathbf{x}_k - \boldsymbol{\mu}_1)^{\mathrm{T}}(\mathbf{x}_k - \boldsymbol{\mu}_1)) && \text{(D.3)}\\
&= \mathrm{tr}(\boldsymbol{\Sigma}_1^{-1}\sum_{k=1}^{K_1}(\mathbf{x}_k - \boldsymbol{\mu}_1)^{\mathrm{T}}(\mathbf{x}_k - \boldsymbol{\mu}_1)) && \text{(D.4)}\\
&= K_1 D, && \text{(D.5)}
\end{aligned}$$

where D is the dimension of the vector. The first equality above holds because of the trace identity [2], the second equality holds because of the linearity of matrix multiplication, and the third equality holds because of the fact that $\boldsymbol{\Sigma}_1 = \frac{\sum_{k=1}^{K_1}(\mathbf{x}_k - \boldsymbol{\mu}_1)^{\mathrm{T}}(\mathbf{x}_k - \boldsymbol{\mu}_1)}{K_1}$ under ML estimation. Using the above identity and Equation (A.2), the log-likelihood of the first new node can be computed by

$$-\frac{K_1 D}{2}\log(2\pi) - \frac{K_1}{2}\log(|\boldsymbol{\Sigma}_1|) - \frac{K_1 D}{2}, \tag{D.6}$$

where $|\boldsymbol{\Sigma}_1|$ is the determinant of $\boldsymbol{\Sigma}_1$. As a result, we can compute the increase of log-likelihood after the split by

$$\frac{K_1 + K_2}{2}\log(|\boldsymbol{\Sigma}_0|) - \frac{K_1}{2}\log(|\boldsymbol{\Sigma}_1|) - \frac{K_2}{2}\log(|\boldsymbol{\Sigma}_2|). \tag{D.7}$$

In terms of implementation, for each node in the tree, the algorithm can use Equation (D.7) to compute the log-likelihood increase with respect to each binary question and find the best splitting question for the node. The best splitting question and the increase of log-likelihood can be stored in a priority queue. At each step, the algorithm can pop up the best node, split the node, and push the two new nodes back into priority queue. Typically the algorithm keeps splitting nodes until the log-likelihood increase of the best node is smaller than a threshold, or the number of nodes has reached pre-defined maximum. After the splitting is finished, the cluster for each context-dependent unit (including the unseen ones) can be found by walking down the tree based on each splitting question.

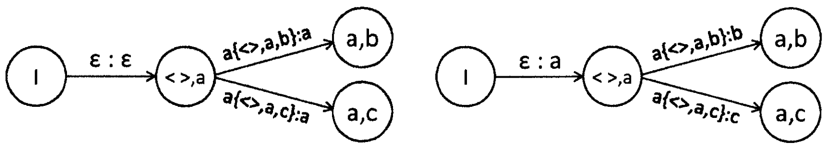
Appendix E

Composition Issues and Dynamic Expansion of FST

In this appendix, we discuss several practical issues when composing the FSTs for large-vocabulary speech recognition tasks. More specifically, we discuss practical issues about how to compose (integrate) different levels of search constraints, including HMM topology constraints, represented by a M FST, contextual constraints from a context-dependent acoustic model, represented by a C FST, and linguistic constraints, represented by a P $\circ$ L $\circ$ G FST. For convenience, we use $\tilde{\mathrm{L}}$ to denote the P $\circ$ L $\circ$ G through out this appendix.

We first discuss an issue that might occur when composing the C FST with the $\tilde{\mathrm{L}}$ FST. Note that a state s in the composed C $\circ$ $\tilde{\mathrm{L}}$ FST can be thought of as a composite of a state c from the C FST and a state l from the $\tilde{\mathrm{L}}$ FST. During composition, the procedure checks that if c has an arc whose output symbol matches the input symbol of one of the arc of l. If there is a match, saying that the arc a_c of state c matches with the arc a_l from state l, the procedure creates a new state s' that is a composite of c' and l', where c' is the destination state of the arc a_c and l' is the destination state of the arc a_l. The composition procedure also creates an arc from s to s' with input symbol the same as that of a_c, the output symbol the same as that of a_l, and the weight equals to the product ($\otimes$) of the weights of a_c and a_l based on the semiring. On the other hand, if there is no match (and if s is not a finial state), there is no permissible path that pass through s, and therefore we call s a "dead-end" state.

The issue here is that if the composition of C and $\tilde{\mathrm{L}}$ is not handle carefully, a large num-



(a) "Theoretical" C FST for triphone. (b) Shifted C FST for triphone

Figure E-1: A simplified illustration of a Context (C) FST for triphpone. Each circle represent a state. The circle with the symbol "I" denotes the initial state. The labels in other circles represent the memory of the left context and the current center phone unit of the state, respectively. The "<>" symbol denotes the sentence boundary. Each arrow denotes an FST arc, where the first symbol before ":" denotes the input symbol, and the one after that denotes output symbol. If either the input or output symbol is not required, the "ε" symbol is used. The label "a{<>,a,b}" denotes the FST label for the triphone with the left context "<>", the center phone unit "a", and the right context "b". During recognition, the time points aligned with the triphone "a{<>,a,b}" should belong to the phoneme "a", not "b", and in theory, the output of symbol of the arc should be "a", as shown in (a). The time point of when to output the phoneme label can also be shifted forward without affecting the weight of each permissible path, as shown in (b).

ber of "dead-end" states can be generated during the composition, and might potentially blow up the memory. For example, if we directly use the triphone-based C FST in Figure E-1(a) to compose with $\tilde{\text{L}}$, the following situation might occur. Assuming that there is a word w in $\tilde{\text{L}}$ whose pronunciation starts with "a" and followed by "b". Both of the two arcs "a{<>,a,b}:a" and "a{<>,a,c}:a" for the state "<>,a" in Figure E-1(a) matches the arc from the initial state of $\tilde{\text{L}}$ to the first state of the word w. However, the arc "a{<>,a,c}:a" results in a "dead-end" state. This is because the state "a,c" in Figure E-1(a) only generates "c" as its output, but the first state of w expect "b" to proceed to the second state. Suppose there are K basic phonetic unit, $K - 1$ "dead-end" states might be generated at each step, and can blow up the memory. To resolve such issue, a shifted C FST as shown in Figure Figure E-1(b) can be used. When using the shifted FST, the uncertainty about the right context that results in the "dead-end" states is resolved, and the $\text{C} \circ \tilde{\text{L}}$ can be successfully composed. Note that while we use a triphone-based C FST for explanation, the issue can also occur for a higher order context-dependent models.

Although the $\text{C} \circ \tilde{\text{L}}$ can be successfully composed, its size is typically not small for a triphone (or higher order) large-vocabulary system. Take triphone systems used the exper-

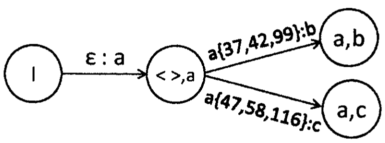


Figure E-2: A simplified illustration of a C FST for a three-state triphone framework. The numbers embedded in the input symbol represent the corresponding acoustic model index for each state.

iments in Chapter 5 for example. The sizes of the FSTs are around 100 $\sim$ 200 Megabytes. Given the size of the C $\circ$ $\tilde{\mathrm{L}}$, if we directly compose the M FST with the C $\circ$ $\tilde{\mathrm{L}}$ FST, it can potentially further increase the size of the resulting FST by another order of magnitude, into the order of Gigabytes. This is because the composition is effectively replacing each arc in the C $\circ$ $\tilde{\mathrm{L}}$ FST with the HMM topology in the M FST, and describing a HMM topology is much more complex (in terms of storage) than describing an FST arc. Having a Gigabyte-sized FST for decoding can create a very heavy I/O and Memory loads on currently available computation architectures.

To avoid using a huge-sized FST for decoding, the dynamic FST expansion implemented by Dr. L. Hetherington was used to conduct the triphone-based recognition experiments reported in this thesis. Under the dynamic expansion framework, the C FST is modified as illustrated in Figure E-2 to record the acoustic model indices described in the M FST. During decoding, each FST state is attached with an (one byte) integer to indicate the current state index in the HMM topology, and therefore appropriate acoustic model can be selected for scoring from the input symbol of the statically composed FST. Using the dynamic expansion can keep the statically composed FST in moderate size, while the memory used during the running time can be controlled by appropriated pruning. As a result, the dynamic expansion can be a practical solution for building ASR system with triphone or higher order context-dependent acoustic model.

Appendix F

Questions for Decision Tree Clustering

F.1 Phonetic Questions for TIMIT

LEFT: aa ah ao aw ax ow uh uw ae ay eh ey ih ix iy oy w y	RIGHT: aa ah ao aw ax ow uh uw ae ay eh ey ih ix iy oy w y
LEFT: aa ah ao aw ax ow uh uw w	RIGHT: aa ah ao ax ay ow oy uh uw w
LEFT: aa ah ao ax uh	RIGHT: aa ah ao ax ay ow oy uh
LEFT: aa ao	RIGHT: aa ay
LEFT: ah ax uh	RIGHT: ah ax
LEFT: aw ow uw w	RIGHT: ao oy
LEFT: aw ow uw	RIGHT: uw w
LEFT: aw ow	RIGHT: ow uh
LEFT: ae ay eh ey ih ix iy oy y	RIGHT: ae aw eh ey ih ix iy y
LEFT: ae eh ih ix	RIGHT: ae aw eh ey
	RIGHT: ae aw eh
LEFT: ae eh	RIGHT: ae eh
LEFT: ay ey oy iy y	
LEFT: ey iy y	RIGHT: iy y
LEFT: ey iy	
LEFT: ay oy	

LEFT: b d g p t k	RIGHT: b d g p t k
LEFT: b p	RIGHT: b p
LEFT: d t	RIGHT: d t
LEFT: g k	RIGHT: g k
LEFT: cl vcl epi	RIGHT: cl vcl epi
LEFT: vcl	RIGHT: vcl
LEFT: cl epi	RIGHT: cl epi
LEFT: jh ch zh sh z s	RIGHT: jh ch zh sh z s
LEFT: jh ch zh sh	
LEFT: ch sh	RIGHT: ch jh
LEFT: jh zh	RIGHT: zh sh
LEFT: z s	RIGHT: z s
LEFT: dh th v f dx hh	RIGHT: dh th v f dx hh
LEFT: dh th v f	RIGHT: dh th v f
LEFT: dh th	RIGHT: dh th
LEFT: v f	RIGHT: v f
LEFT: dx hh	RIGHT: dx hh
LEFT: er r	RIGHT: er r
LEFT: er	RIGHT: er
LEFT: m n en ng	RIGHT: m n en ng
LEFT: m	RIGHT: m
LEFT: n en ng	RIGHT: n en ng
LEFT: n en	RIGHT: n en
LEFT: el l	RIGHT: el l
LEFT: sil <>	RIGHT: sil <>

Table F.1: Phonetic questions used for decision tree clustering in the phonetic recognition experiment on TIMIT. The 61 to 48 mapping in Table 4.4 is used. The merging of labels are as follows. {"h#", "pau"}→"sil". {"pcl", "tcl", "kcl", "q"}→"cl". {"bcl","dcl","gcl"}→"vcl". {"er", "axr"}→"er". {"ng", "eng"}→"ng". {"ax", "ax-h"}→"ax". {"uw", "ux"}→"uw". The label "<>" denotes the sentence boundary.

F.2 Phonetic Questions for Lectures

LEFT: aa ah ah_fp ao aw ax ow uh uw ae ay eh ey ih iy oy w y	RIGHT: aa ah ah_fp ao aw ax ow uh uw ae ay eh ey ih iy oy w y
LEFT: aa ah ah_fp ao aw ax ow uh uw w	RIGHT: aa ah ah_fp ao ax ay ow oy uh uw w
LEFT: aa ah ah_fp ao ax uh	RIGHT: aa ah ah_fp ao ax ay ow oy uh
LEFT: aa ao	RIGHT: aa ay
LEFT: ah ah_fp ax uh	RIGHT: ah ah_fp ax
	RIGHT: ah ax
LEFT: aw ow uw w	RIGHT: ao oy
LEFT: aw ow uw	RIGHT: uw w
LEFT: aw ow	RIGHT: ow uh
LEFT: ae ay eh ey ih iy oy y	RIGHT: ae aw eh ey ih iy y
LEFT: ae eh ih	RIGHT: ae aw eh ey
	RIGHT: ae aw eh
LEFT: ae eh	RIGHT: ae eh
LEFT: ay ey oy iy y	
LEFT: ey iy y	RIGHT: iy y
LEFT: ey iy	

LEFT: ay oy	
LEFT: b d g p t k	RIGHT: b d g p t k
LEFT: b p	RIGHT: b p
LEFT: d t	RIGHT: d t
LEFT: g k	RIGHT: g k
LEFT: pcl tcl kcl bcl dcl gcl epi	RIGHT: pcl tcl kcl bcl dcl gcl epi
LEFT: bcl dcl gcl	RIGHT: bcl dcl gcl
LEFT: pcl tcl kcl epi	RIGHT: pcl tcl kcl epi
LEFT: tcl epi	RIGHT: tcl epi
LEFT: jh ch zh sh z s	RIGHT: jh ch zh sh z s
LEFT: jh ch zh sh	
LEFT: ch sh	RIGHT: ch jh
LEFT: jh zh	RIGHT: zh sh
LEFT: z s	RIGHT: z s
LEFT: dh th v f dx hh	RIGHT: dh th v f dx hh
LEFT: dh th v f	RIGHT: dh th v f
LEFT: dh th	RIGHT: dh th
LEFT: v f	RIGHT: v f
LEFT: dx hh	RIGHT: dx hh
LEFT: er axr r	RIGHT: er axr r
LEFT: er axr	RIGHT: er axr
LEFT: m em n en ng	RIGHT: m em n en ng
LEFT: m em	RIGHT: m em
LEFT: n en ng	RIGHT: n en ng
LEFT: n en	RIGHT: n en
LEFT: el l	RIGHT: el l
LEFT: – <>	RIGHT: – <>

LEFT: _	RIGHT: _
LEFT: _c _l _n _b	RIGHT: _c _l _n _b
LEFT: _c _l	RIGHT: _c _l
LEFT: _c	RIGHT: _c
LEFT: _l	RIGHT: _l
LEFT: _n _b	RIGHT: _n _b
LEFT: _n	RIGHT: _n
LEFT: _b	RIGHT: _b

Table F.2: Phonetic questions used for decision tree clustering in the large-vocabulary ASR experiment.

Appendix G

Quickprop Algorithm

This appendix illustrates the Quickprop algorithm used in [86] in more detail. Quickprop is an approximated second-order optimization method based on the classic Newton's method. Newton's method is an iterative optimization method. At each iteration, Newton's method builds a quadratic approximation $M(\boldsymbol{\lambda})$ to the function of interested $F(\boldsymbol{\lambda})$ using the first three terms of the Taylor series expansion of the function $F(\boldsymbol{\lambda})$ around the current point $\boldsymbol{\lambda}^{(p)}$. The update criterion of Newton's method is to choose the parameter set $\boldsymbol{\lambda}^{(p+1)}$ such that the gradient of the approximation $\nabla M(\boldsymbol{\lambda}^{(p+1)})$ equals $\mathbf{0}$. As a result, the solution of $\boldsymbol{\lambda}^{(p+1)}$ can be expressed by

$$\boldsymbol{\lambda}^{(p+1)} = \boldsymbol{\lambda}^{(p)} + \mathbf{s}^{(p)}, \tag{G.1}$$

where the step $\mathbf{s}^{(p)}$ can be computed by

$$\mathbf{s}^{(p)} = -(\nabla^2 F(\boldsymbol{\lambda}^{(p)}))^{-1} \nabla F(\boldsymbol{\lambda}). \tag{G.2}$$

In general, if the Hessian matrix $\nabla^2 F(\boldsymbol{\lambda})$ is positive definite, and the initial value $\boldsymbol{\lambda}^{(0)}$ is sufficiently close to the optimum, Newton's method converges rapidly to a local minimum of function $F(\boldsymbol{\lambda})$ [86]. However, in general, there is no guarantee that the Hessian matrix is positive definite. Also, representing the true Hessian matrix becomes impractical as the dimension of $\boldsymbol{\lambda}$ becomes large.

To address the two issues above, Quickprop makes the following two major changes with regards to the original Newton's method:

1. Use a diagonal approximation for the Hessian.
2. Use certain criterion to check the condition of the Hessian, adding a term proportional to the gradient to the update step if the criterion does not hold.

The i^{th} diagonal element of the Hessian matrix can be approximated by:

$$\frac{\partial^2 F(\boldsymbol{\lambda}^{(p)})}{\partial \lambda_i^2} \approx \frac{\nabla F(\boldsymbol{\lambda}^{(p)})_i - \nabla F(\boldsymbol{\lambda}^{(p-1)})_i}{\Delta \lambda_i^{(p-1)}}, \tag{G.3}$$

where $\nabla F(\boldsymbol{\lambda}^{(p)})_i$ is the i^{th} element of the gradient at $\boldsymbol{\lambda}^{(p)}$, and $\Delta\lambda_i^{(p-1)}$ is the i^{th} component of the update step $\mathbf{s}^{(p-1)}$. The approximation is accurate when the update step is small, but in general, the approximation can provide helpful information to guide the optimization. For each element in $\boldsymbol{\lambda}$, the product $[\nabla F(\boldsymbol{\lambda}^{(p)})_i \nabla F(\boldsymbol{\lambda}^{(p-1)})_i]$ is used as a measure to check the condition of the Hessian: if the product is negative (different sign), the minimum is considered likely to exist between $\lambda^{(p)_i}$ and $\lambda^{(p-1)_i}$, and the update step of Newton's method is used; otherwise, a term proportional to the gradient is added to the update step, resulting in

$$s_i = -[(\nabla^2 F(\boldsymbol{\lambda})_i)^{-1} + \epsilon]\nabla F(\boldsymbol{\lambda})_i, \tag{G.4}$$

where $\nabla^2 F(\boldsymbol{\lambda})_i$ is approximated by (G.3) and ϵ is a positive learning rate. Generally, setting a proper value of ϵ is important. However, once it is set within a reasonable range, the optimization results will be similar.

There are also several additional controls on the update step used by Quickprop to enhance the numerical stableness of the algorithm. For example, the absolute value of step size can not grow ϖ times larger than the previous step size; if the gradient and the modified Newton step are of the same sign, a simple gradient step is used instead. Details of Quickprop update can be seen in the pseudo code of Algorithm (G.1).

Algorithm G.1 Quickprop Update [86]

```
##Quickprop Update for iteration $p$.
for $i = 1 \ldots L$ do
    # Loop over each element in $\boldsymbol{\lambda}$
    $\Delta F_1 \Leftarrow \nabla F(\boldsymbol{\lambda}^{(p)})_i$            # get first derivative from current iteration.
    $\Delta F_0 \Leftarrow \nabla F(\boldsymbol{\lambda}^{(p-1)})_i$          # get first derivative from previous iteration.
    $\Delta\lambda \Leftarrow \boldsymbol{\lambda}_i^{(p)} - \boldsymbol{\lambda}_i^{(p-1)}$          # get last step size.

    # Calculate approximate diagonal second derivative
    $\Delta^2 F \Leftarrow (\Delta F_1 - \Delta F_0)/\Delta\lambda$

    # Calculate modified Newton step
    $g_1 \Leftarrow -\epsilon \Delta F_1$
    if ($\Delta^2 F > 0$) then
        $g_2 \Leftarrow -\Delta F_1/\Delta^2 F$
        if ($\Delta F_1 \Delta F_0 > 0$) then
            # gradients point the same way
            $d \Leftarrow g_1 + g_2$
        else
            # gradients change sign
            $d \Leftarrow g_2$
        end if
    else
        $d \Leftarrow g_1$
    end if

    # Limit absolute step size
    if (abs($d$) $>$ abs($\varpi * \Delta\lambda$)) or abs($d$) >STEP_LIMIT then
        $d \Leftarrow$ sign($d$) $*$ min(STEP_LIMIT, abs($\varpi * \Delta\lambda$))
    end if

    # If going uphill or update step is near zero, use simple gradient
    if (($d * \Delta F_1$) $>$ 0.0) or (abs($\Delta$) $<$ TINY) then
        $d \Leftarrow$ sign($g_1$) $*$ min(abs($g_1$), STEP_LIMIT)
    end if

    # Update parameter
    $\boldsymbol{\lambda}_i^{(p+1)} \Leftarrow \boldsymbol{\lambda}_i^{(p)} + d$
end for
```

Appendix H

Settings of Experimental Constants

H.1 TIMIT Experiment

H.1.1 MCE Settings

Constant	Value	Description
η	1.0	Reflecting the relative importance of the best scoring hypothesis.
ζ	$1.5/\mathcal{F}_n$	Determining sharpness of the sigmoid function.
ρ	10.0	Margin value per unit distance in the phone alignment.
ϵ	25	Learning rate of the Quickprop algorithm.
ϖ	1.75	Growth factor of step size used in the Quickprop algorithm
`STEP_LIMIT`	1.0	Step limit of the Quickprop algorithm.
`vprune`	10	Viterbi pruning threshold for path scores.
`vprunenodes`	3,000	Maximum number of nodes in the active frontier.

Table H.1: Constant settings for MCE training on TIMIT. $\mathcal{F}_n$ is the number of frames in the n^{th} utterance.

H.2 Lecture Experiment

H.2.1 MCE Settings for Diphone Models

Constant	Value	Description
η	1.0	Reflecting the relative importance of the best scoring hypothesis.
ζ	$10.0/\mathcal{F}_n$	Determining sharpness of the sigmoid function.
ϵ	0.2	Learning rate of the Quickprop algorithm.
ϖ	1.75	Growth factor of step size used in the Quickprop algorithm
STEP_LIMIT	10.0	Step limit of the Quickprop algorithm.
vprune	10	Viterbi pruning threshold for path scores.
vprunenodes	250	Maximum number of nodes in the active frontier.
$\mathcal{K}$	20	Maximum number of competing hypotheses.

Table H.2: Constant settings for MCE training of diphone models on the lecture corpus. $\mathcal{F}_n$ is the number of landmarks in the n^{th} utterance.

H.2.2 MCE Settings for Triphone Models

Constant	Value	Description
η	1.0	Reflecting the relative importance of the best scoring hypothesis.
ζ	$10.0/\mathcal{F}_n$	Determining sharpness of the sigmoid function.
ϵ^{CL}	2.0	Learning rate of the Quickprop algorithm for the training of clustering based models.
ϵ^{Multi}	2.5	Learning rate of the Quickprop algorithm for the training of multi-level models.
ϖ	1.75	Growth factor of step size used in the Quickprop algorithm
STEP_LIMIT	2.0	Step limit of the Quickprop algorithm.
vprune	10	Viterbi pruning threshold for path scores.
vprune^f	1000	Viterbi pruning threshold for path scores in the forced-alignment mode.
vprunenodes	3000	Maximum number of nodes in the active frontier.
vprunenodes^f	3000	Maximum number of nodes in the active frontier in the forced-alignment mode.
$\mathcal{K}$	20	Maximum number of competing hypotheses.

Table H.3: Constant settings for MCE training of triphones models on the lecture corpus. $\mathcal{F}_n$ is the number of frames in the n^{th} utterance.

Appendix I

Frequently Used Acronyms

Acronym	Meaning
ASR	Automatic Speech Recognition
EBW	Extended Baum-Welch
EM	Expectation-Maximization
FST	Finite-State Transducer
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
MCE	Minimum Classification Error
ML	Maximum-Likelihood
PER	Phone Error Rate
WER	Word Error Rate

Table I.1: List of frequently used acronyms

Bibliography

[1] `ftp://jaguar.ncsl.nist.gov/current_docs/sctk/doc/sclite.htm`.

[2] `http://en.wikipedia.org/wiki/Matrix_calculus`.

[3] J. Aldrich. R.A. Fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 12(3):162–176, 1997.

[4] I. Badr. Pronunciation learning for automatic speech recognition. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2011.

[5] I. Badr, I. McGraw, and J. Glass. Pronunciation learning from continuous speech. *Proc. Interspeech*, pages 549–552, 2011.

[6] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding for linear codes for minimizing symbol error rate. *IEEE Trans. Information Theory*, 20(2):284–287, 1974.

[7] L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for porbabilistic function of Markov processes anf to a model for Ecology. *Bulletin of American Mathematical Society*, 73:360–363, 1967.

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

[9] I. Bazzi. *Modelling out-of-vocabulary words for robust speech recognition*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2002.

[10] I. Bazzi and J. R. Glass. Modeling out-of-vocabulary words for robust speech recognition. *Proc. ICSLP*, pages 401–404, 2000.

[11] R. E. Bellman. *An introduction to the theory of dynamic programming*. RAND Corporation, 1953.

[12] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[13] M. Bisani and H. Ney. Open vocabulary speech recognition with flat hybrid models. *Proc. Eurpspeech*, pages 725–728, 2005.

[14] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[15] H.-A. Chang. Large-margin Gaussian mixture modeling for automatic speech recognition. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2008.

[16] H.-A. Chang and J. R. Glass. Discriminative training of hierarchical acoustic models for large vocabulary continuous speech recognition. *Proc. ICASSP*, pages 4481–4484, 2009.

[17] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Technial Report TR-10-98, Computer Science Group, Harvard University*, 1998.

[18] W. Chou, B.-H. Juang, and C.-H. Lee. Segmental GPD training of HMM based speech recognizer. *Proc. ICASSP*, pages 473–476, 1992.

[19] G. F. Choueiter. *Linguistically-motivated sub-word modeling with applications to speech recognition.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2009.

[20] Y. Chow, M. Dunham, O. Kimball, M. Krasner, G. Kubala, J. Makhoul, P. Price, S. Roucos, and R. Schwartz. BYBLOS: the BBN continuous speech recognition system. *Proc. ICASSP*, pages 89–92, 1987.

[21] T. H. Cormen, C. E. Leiserson, and R. L. Rovest amd C. Stein. *Introduction to algorithms.* MIT Press & McGraw-Hill, 2001.

[22] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMs. *Proc. ICASSP*, pages 4688–4691, 2011.

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

[24] M. H. DeGroot. *Optimal statistical decisions.* John Wiley & Sons, Inc., 1970.

[25] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algrithm. *Journal of the Royal Statistical Society. Series B(Methodological)*, 39(1):1–38, 1977.

[26] T. G. Dietterich and G. Bakiri. Solving multiclass learning problem via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[27] J. G. Fiscus. A post-processing system to yield reduced word error rates: Recognizer Output Voting Error Reduction (ROVER). *Proc. ASRU*, pages 347–354, 1997.

[28] A. Franz and B. Milch. Searching the web by voice. *Proc. Computational Linguistics*, pages 1213–1217, 2002.

[29] S. H. Friedberg, A. J. Insel, and L. E. Spence. *Linear algebra*. Prentice Hall, 2003.

[30] Q. Fu, X. He, and L. Deng. Phone-discriminative Minimum Classification Error (P-MCE) training for phonetic recognition. *Proc. Interspeech*, pages 2073–2076, 2007.

[31] S. Furui. Automatic speech recognition and its application to information extraction. *Proc. ACL*, pages 11–20, 1999.

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

[33] L. Gillick and S. J. Cox. Some statistical issues in the comparison of speech recognition algorithms. *Proc. ICASSP*, pages 532–535, 1989.

[34] J. Glass. A probabilistic framework for segment-based speech recognition. *Computer Speech and Language*, 17:137–152, 2003.

[35] J. Glass, T. J. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay. Recent progress in the MIT spoken lecture processing project. *Proc. Interspeech*, pages 2553–2556, 2007.

[36] J. Glass, T. J. Hazen, S. Cyphers, K. Schutte, and A. Park. The MIT spoken lecture processing project. *Proc. EMNLP*, pages 28–29, 2005.

[37] J. R. Glass, T. J. Hazen, L. Hetherington, and C. Wang. Analysis and processing of lecture audio data: preliminary investigations. *HLT-NAACL Workshop on Speech Indexing and Retrieval*, 2004.

[38] J. J. Godfrey, E. C. Holliman, and J. McDaniel. SWITCHBOARD: telephone speech corpus for research and development. *Proc. ICASSP*, pages 517–520, 1991.

[39] J. Goldberger, S. T. Roweiss, R. R. Salakhutdinov, and G. E Hinton. Neighbourhood components analysis. *Proc. NIPS*, pages 513–520, 2004.

[40] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264, 1953.

[41] P. S. Gopalakrishnan, D. Kanevsky, A. Nadas, and D. Nahamoo. An inequality for rational functions with applications to some statistical estimation problems. *IEEE Trans. Information Theory*, 37(1):107–113, 1991.

[42] A. Gruenstein, J. Orszulak, S. Liu, S. Roberts, J. Zabel, B. Reimer, B. Mehler, S. Seneff, J. Glass, and J. Coughlin. City Browser: developing a conversational automotive HMI. *Proc. CHI*, pages 4291–4296, 2009.

[43] A. Gunawardana and W. Byrne. Discriminative speaker adaptation with conditional maximum likelihood linear regression. *Proc. Eurospeech*, pages 1203–1206, 2001.

[44] A. K. Halberstadt. *Heterogeneous acoustic measurements and multiple classifiers for speech recognition.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 1998.

[45] K. Harrenstien. Automatic captions in YouTube. Blog, document available at http://googleblog.blogspot.com/2009/11/automatic-captions-in-youtube.html.

[46] T. Hastie, T. Robert, and F. Jerome. *The elements of statistical learning*, chapter 14.3.12, pages 520–528. New York: Springer, 2009.

[47] T. J. Hazen. Automatic alignment and error correction of human generated transcripts for long speech recordings. *Proc. Interspeech*, pages 1606–1609, 2006.

[48] T. J. Hazen and I. Bazzi. A comparison and combinations of methods for OOV word detection and word confidence scoring. *Proc. ICASSP*, pages 397–400, 2001.

[49] T. J. Hazen and J. R. Glass. A comparison of novel techniques for instantaneous speaker adaptation. *Proc. Eurospeech*, pages 2047–2050, 1997.

[50] T. J. Hazen and E. McDermott. Discriminative MCE-based speaker adaptation of acoustic models for a spoken lecture processing task. *Proc. Interspeech*, pages 1577–1580, 2007.

[51] A. Heidel, H.-A. Chang, and L.-S. Lee. Language model adaptation using latent Dirichlet allocation and an efficient topic inference algorithm. *Proc. Interspeech*, pages 2361–2364, 2007.

[52] I. L. Hetherington. An efficient implementation of phonological rules using finite-state transducers. *Proc. Eurospeech*, pages 1599–1602, 2001.

[53] I. L. Hetherington. MIT finite-state transducer toolkit for speech and language processing. *Proc. ICSLP*, pages 2609–2612, 2004.

[54] B.-J. Hsu and J. Glass. Style & topic language model adaptation using HMM-LDA. *Proc. EMNLP*, pages 373–381, 2006.

[55] B.-J. (P.) Hsu. *Language modeling for limited-data domains*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2009.

[56] X. Huang, A. Acero, and H.-W. Hon. *Spoken language processing: a guide to theory, algorithm, and system development*. Prentice Hall PTR, 2001.

[57] D. P. Huttenlocher and V. W. Zue. A model of lexical access from partial phonetic information. *Proc. ICASSP*, pages 391–394, 1984.

[58] M. Hwang, X. Huang, and F. Alleva. Predicting unseen triphones with senones. *IEEE Trans. Speech and Audio Processing*, 4(6):412–419, 1996.

[59] N. L. Johnson and S. Kotz. *Distribution in statistics.* New York: Wiley, 1972.

[60] I.T. Jolliffe. *Principal component analysis.* Springer Series in Statistics. Springer, NY, 2 edition, 2002.

[61] D. Jones, E. Gibson, W. Shen, N. Granoien, M. Herzog, D. Reynolds, and C. Weinstein. Measuring human readability of machine generated text: three case studies in speech recognition and machine translation. *Proc. ICASSP*, pages 1009–1012, 2005.

[62] B.-H Juang, W. Chou, and C.-H. Lee. Minimum classificatoin error rate methods for speech recognition. *IEEE Trans. Audio, Speech, and Language Processing*, 5(3):257–265, 1997.

[63] S. Kapadia, V. Valtchev, and S. J. Young. MMI training for continuous phoneme recognition on the TIMIT database. *Proc. ICASSP*, pages 491 – 494, 1993.

[64] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 35(3):400–401, 1987.

[65] M. Killer, S. Stuker, and T. Schultz. Grapheme based speech recognition. *Proc. Interspeech*, pages 3141–3144, 2003.

[66] R. Kneser and H. Ney. Improved backing-off for M-gram language modeling. *Proc. ICASSP*, pages 181–184, 1995.

[67] R. Kneser and V. Steinbiss. On the dynamic adaptation of stochastic language models. *Proc. ICASSP*, pages 586–589, 1993.

[68] H. Kucera and W. N. Francis. *Computational analysis of present-day American English.* Brown University Press, 1967.

[69] R. Kuhn, J. Junqua, P. Nguyen, and N. Niedzielski. Rapid speaker adaptation in eigenvoice space. *IEEE Trans. on Speech and Audio Processing*, 8:695–707, 2000.

[70] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[71] L. Lamel, R. Kassel, and S. Seneff. Speech database development: design and analysis of acoustic-phonetic corpus. *Proc. DARPA Speech Recognition Workshop*, 1986. Report No. SAIC-86/1546.

[72] A. Lavie, A. Waibel, L. Levin, M. Finke, D. Gates, M. Gavalda, T. Zeppenfeld, and P. Zhan. Jansus-III: speech-to-speech translation in multiple languages. *Proc. ICASSP*, pages 99–102, 1997.

[73] A. Lee, T. Kawahara, and K. Shikano. Julius – an open source real-time large vocabulary recognition engine. *Proc. Eurospeech*, pages 1691–1694, 2001.

[74] C.-Y. Lee and J. Glass. A transcription task for crowdsourcing with automatic quality control. *Proc. Interspeech*, pages 3041–3044, 2011.

[75] K.-F. Lee. *Large-vocabular speaker-independent continuous speech recognition: the SPHINX system*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1988.

[76] K.-F. Lee. *Automatic speech recognition: the development of the SPHINX system*. Kluwer Academic Publishers, 1989.

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

[78] L.-S. Lee. Voice dictation of Mandarin Chinese. *IEEE Signal Processing Magazine*, 14(4):63–101, 1997.

[79] C. J. Leggetter and P. C. Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech and Language*, 9:171–185, 1995.

[80] J. Li, M. Yuan, and C.-H. Lee. Soft margin estimation of hidden Markov model parameters. *Proc. Eurospeech*, pages 2422–2425, 2006.

[81] F.-H. Liu, R. M. Stern, X. Huang, and A. Acero. Efficient cepstral normalization for robust speech recognition. *Proc. ARPA Speech and Natural Language Workshop*, pages 69–74, 1993.

[82] A. Ljolje. High accuracy phone recognition using context clustering and quasitriphone models. *Computer Speech and Language*, 8:129–151, 1994.

[83] S. P. Lloyd. Least square quantixation in PCM. *IEEE Trans. Information Theory*, 28(2):129–137, 1982.

[84] M. Marge, S. Banerjee, and A. Rudnicky. Using the Amzon Mechanical Turk for transcription of spoken language. *Proc. ICASSP*, pages 5270–5273, 2010.

[85] E. McDermott and T. J. Hazen. Minimum classification error training of landmark models for real-time continuous speech recognition. *Proc. ICASSP*, pages 937–940, 2004.

[86] E. McDermott, T. J. Hazen, J. L. Roux, A. Nakamura, and S. Katagiri. Discriminative training for large-vocabulary speech recognition using minimum classification error. *IEEE Trans. Audio, Speech, and Language Processing*, 15(1):203–223, 2007.

[87] E. McDermott, S. Watanabe, and A. Nakamura. Margin-space integration of MPE loss via differencing of MMI functionals for generalized error-weighted discriminative training. *Proc. Interspeech*, pages 224–227, 2009.

[88] I. McGraw, C.-Y. Lee, L. Hetherington, S. Seneff, and J. Glass. Collecting voices from the cloud. *Proc. LERC*, pages 19–26, 2010.

[89] F. Metze, C. Fugen, Y. Pan, and A. Waibel. Automatically transcribing meetings using distant microphones. *Proc. ICASSP*, pages 989–992, 2005.

[90] J. Ming, P. O'Boyle, M. Owens, and F. J. Smith. A bayesian apporach for building triphone models for continuous speech recognition. *IEEE Trans. Speech and Audio Processing*, 7(6):678–683, 1999.

[91] A. Mohamed, G. E. Dahl, and G. Hinton. Acoustic modeling using Deep Belief Networks. *IEEE Trans. on Audio, Speech, and Language Processing*, 20(1):14–22, 2012.

[92] M. Mohri. Finite-state transducers in language and speech processing. *Computation Linguistics*, 23(2):269–311, 1997.

[93] H. Murveit, J. Butzberger, V. Digalakis, and M. Weintraub. Large-vocabulary dictation using SRI's DECIPHER speech recognition system: progressive search techniques. *Proc. ICASSP*, pages 319–322, 1993.

[94] A. Nadas. A decision theoretic formulation of a training problem in speech recognition and a comparison of training by unconditional versus conditional maximum likelihood. *IEEE Trans. Acoustic, Speech, and Signal Processing*, 31(4):814–817, 1983.

[95] A. V. Oppenheim and R. W. Schafer with J. R. Buck. *Discrete-time signal processing*. Prentice-Hall Signal Processing Series. Prentice-Hall, 2 edition, 1999.

[96] A. Park, T. J. Hazen, and J. R. Glass. Automatic processing of audio lectures for information retireval: vocabulary selection and language modeling. *Proc. ICASSP*, pages 497–500, 2005.

[97] A. S. Park. *Unsupervised pattern discovery in speech: applications to word acqusition and speaker segmentation.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2006.

[98] M. A. Peabody. *Methods for pronunciation assessment in computer aided language learning*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2011.

[99] D. Povey. *Discriminative training for large vocabulary speech recognition*. PhD thesis, University of Cambridge, Cambridge, U.K, 2003.

[100] D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah. Boosted MMI for model and feature-space discriminative training. *Proc. ICASSP*, pages 4057–4060, 2008.

[101] D. Povey and P. C. Woodland. Minimum phone error and I-smoothing for improved discriminative training. *Proc. ICASSP*, pages 203–223, 2002.

[102] D. Povey, P. C. Woodland, and M. J. F. Gales. Discriminative MAP for acoustic model adaptation. *Proc. ICASSP*, pages 312–315, 2003.

[103] J. G. Proakis. *Digital communications*. McGraw-Hill, 2001.

[104] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceesings of the IEEE*, 77(2):257–286, 1989.

[105] T. N. Sainath. *Applications of broad class knowledge for noise robust speech recognition*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2009.

[106] T. N. Sainath, B. Ramabhadran, and M. Picheny. An exploration of large vocabulary tools for small vocabulary phonetic recognition. *Proc. ASRU*, pages 359–364, 2009.

[107] G. A. Sanders, A. N. Le, and J. S. Garofolo. Effects of word error wrate in the DARPA communicator data during 2000 and 2001. *Proc. ICSLP*, pages 277–280, 2002.

[108] R. E. Schapire. Using output codes to boost multiclass learning problems. *Proc. ICML*, pages 313–321, 1997.

[109] R. Schluter, W. Macherey, B. Muller, and H. Ney. Comparison of discriminative training criteria and optimization methods for speech recognition. *Speech Comunication*, 34(3):287–310, 2001.

[110] R. Schwartz, Y. Chow, O. Kimball, S. Roucos, M. Krasner, and J. Makhoul. Context-dependent modeling for acoustic-phonetic recongition of continuous speech. *Proc. ICASSP*, pages 1205–1208, 1985.

[111] K. Seymore and R. Rosenfeld. Using story topic language model adaptation. *Proc. Eurospeech*, pages 1987–1990, 1997.

[112] F. Sha and L. K. Saul. Comparison of large margin training to other discriminative training methods for phonetic recognition by hidden Markov models. *Proc. ICASSP*, pages 313–316, 2007.

[113] H. Shu. *Multi-tape finite-state transducer for asynchronous multi-stream pattern recognition with application to speech*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2006.

[114] N. Singh-Miller. *Neighborhood analysis methods in acoustic modeling for automatic speech recognition*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2010.

[115] O. Siohan, B. Ramabhadran, and B. Kingsbury. Constructing ensembles of ASR systems using randomized decision trees. *Proc. ICASSP*, pages 197–200, 2005.

[116] A. Stokle. SRILM: an extensible language modeling toolkit. *Proc. ICSLP*, pages 901–904, 2002.

[117] H. Strik and C. Cucchiarini. Modeling pronunciation variation for ASR: a survey of literature. *Speech Communication*, 29:225–246, 1999.

[118] Y.-H. Sung, T. Hughes, F. Beaufays, and B. Strope. Revisiting graphemes with increasing amounts of data. *Proc. ICASSP*, pages 4449–4452, 2009.

[119] V. Valtchev, J. J. Odell, P. C. Woodland, and S. J. Young. MMIE training of large vocabulary recognitoin systems. *Speech Communication*, 22:303–314, 1997.

[120] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. on Information Theory*, 13(2):260–269, 1967.

[121] W. Wahlster. *Verbmobil: foundations of speech-to-speech translation*. Springer-Verlag Berlin Heidelberg, 2000.

[122] L. Wang and P. C. Woodland. MPE-based discriminative linear transform for speaker adaptaion. *Proc. ICASSP*, pages 321–324, 2004.

[123] S. Wang. Using graphone models in automatic speech recognition. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2009.

[124] S. Witt and S. Young. Language learning based on non-native speech recognition. *Proc. Eurospeech*, pages 633–636, 1997.

[125] P. C. Woodland and D. Povey. Large scale MMIE training for conversational telephone speech recognition. *Proc. NIST Speech Transctription Workshop*, 2000.

[126] P. C. Woodland and D. Povey. Large scale discriminative training of hidden Markov models for speech recognition. *Computer Speech and Language*, 16:25–47, 2002.

[127] Y. Xu, A. Goldie, and S. Seneff. Automatic question generation and answer judging: a Q&A game for language learning. *Proc. SIGSLaTE*, 2009.

[128] S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. *The HTK book*. Cambridge, U.K.: Entropic, 1999.

[129] S. J. Young, J. J. Odell, and P.C. Woodland. Tree-based state tying for high accuracy acoustic modeling. *Proceeding of the workshop on Human Language Technology*, pages 309–312, 1994.

[130] D. Yu, L. Deng, and A. Acero. Large-margin minimum classification error training for large-scale speech recognition task. *Proc. ICASSP*, pages 1137–1140, 2007.

[131] V. Zue, J. R. Glass, D. Goodline, M. Phillips, and S. Seneff. The SUMMIT speech recognition system: phonological modeling and lexical access. *Proc. ICASSP*, pages 49–52, 1990.

[132] V. Zue, S. Seneff, and J. Glass. Speech database development at MIT: TIMIT and beyond. *Speech Communication*, 9:351–356, 1990.

[133] V. Zue, S. Seneff, J. Glass, J. Polifroni, C. Pao, T. Hazen, and L. Hetherington. JUPITER: a telephone-based conversational interface for weather information. *IEEE Trans. on Speech and Audio Processing*, 8(1):100–112, 2000.

[134] V. W. Zue and J. R. Glass. Conversational Interfaces: Advances and Challenges. *Proceedings of the IEEE*, 88(8):1166–1180, 2000.