Large-Margin Gaussian Mixture 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

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author .....
Department of Electrical Engineering and Computer Science
May 13 2008

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

Accepted by .....
Terry P. Orlando
Chair, Department Committee on Graduate Students

Large-Margin Gaussian Mixture Modeling for Automatic Speech Recognition

by

Hung-An Chang

Submitted to the Department of Electrical Engineering and Computer Science
on May 13 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Discriminative training for acoustic models has been widely studied to improve the performance of automatic speech recognition systems. To enhance the generalization ability of discriminatively trained models, a large-margin training framework has recently been proposed. This work investigates large-margin training in detail, integrates the training with more flexible classifier structures such as hierarchical classifiers and committee-based classifiers, and compares the performance of the proposed modeling scheme with existing discriminative methods such as minimum classification error (MCE) training. Experiments are performed on a standard phonetic classification task and a large vocabulary speech recognition (LVCSR) task. In the phonetic classification experiments, the proposed modeling scheme yields about 1.5% absolute error reduction over the current state of the art. In the LVCSR experiments on the MIT lecture corpus, the large-margin model has about 6.0% absolute word error rate reduction over the baseline model and about 0.6% absolute error rate reduction over the MCE model.

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

Acknowledgements

First of all, I would like to thank my advisor, James. R. Glass, for providing constant support, constructive encourage, useful guidance, broad picture of automatic speech recognition, and the freedom of proposing and testing out our own ideas.

I would like to give special thanks to Fei Sha, who I didn't have a chance to meet, for providing help for the large-margin training on TIMIT through mails. Special thanks to T. J. Hazen for providing me guidance to the MCE training on the MIT lecture corpus.

I would also like to thank Ken Schutte and Paul Hsu for providing constant helps on the SUMMIT recognizer and for insightful discussions. I would also like to thank other members of the Spoken Language Systems for providing an excellent and enjoyable working environment.

Thanks to all my friends, brothers, and sisters for providing me all kinds supports and constantly reminding me that I am not alone.

Thanks to my beloved family members, my parents, my big brother Hung-Yin, my young sister Chun-Yin, and my wife Pei, whose love and warmth I can still feel affectionately even though we are thousands of miles apart.

Finally, thanks to God for arranging all the things.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Over the years there has been much research devoted to improving the acoustic modeling performance for automatic speech recognition (ASR) systems. Among the acoustic modeling frameworks in existing ASR systems, Gaussian mixture models (GMMs) are typically used as classifiers to predict the acoustic labels in speech utterances. Traditionally, GMM parameters can be estimated efficiently via maximum-likelihood (ML) training using the Expectation-Maximization (EM) algorithm [5]. However, because the conditions for the optimality of the ML training, such as model correctness, generally do not hold [38], other parameter estimation approaches such as discriminative training of GMM parameters have been proposed to improve ASR performance.

While ML training determines model parameters that maximize the log-likelihood of the training data, discriminative training methods seek model parameters that can minimize the confusions in the training data made by the model. Generally, reduction of the confusions is achieved by optimizing the model parameters with respect to objective functions that are related to the degree of confusions. In the past ten years, several objective functions such as maximum mutual information (MMI)[38] training, minimum classification error (MCE) [17], and minimum word/phone error (MWE/MPE) [29] training have been proposed. Experi-

mental results on a variety of ASR tasks [25, 38, 29], including large vocabulary continuous speech recognition (LVCSR), have demonstrated the effectiveness of these methods in reducing the recognition error rate of ASR systems.

Although the training objectives are different, one common issue of all training methods is the generalization ability of the trained models; that is, the ability to translate the confusion reduction gained in training to unseen data. In the past, such a generalization ability has been maintained by applying smoothing techniques such as I-smoothing in MPE training [29], or by careful selection of the learning rate and smoothing coefficients [25]. More recently, as inspired by the success of support vector machines (SVM) [3] in other fields of pattern recognition, large-margin methods [33, 22, 39] that incorporate margin constraints in the training have been proposed to further improve the generalization ability of discriminatively trained models.

Large-margin training methods ensure generalization by requiring a log-likelihood margin between the well-classified samples and the decision boundary. Because of such margin, the trained model can tolerate some mismatch between the training data and the unseen data, and thus tends to have better generalization ability. Large-margin training is especially effective when the training error rate is low. This is because under low training error condition, the large-margin criterion will guide the training to select the set of parameters that has the maximal margin among all the sets of parameters that have low training error rate.

In addition to discriminative training, a better classifier structure has also been helpful in improving the acoustic model performance. For example, a hierarchical classifier was proposed [13] to reduce phonetic classification error. By dividing the classification problem into smaller sub-problems, a hierarchical classifier is potentially more robust and more generalizable to unseen data since there are more training exemplars in the pooled class. Also, hierarchies can also be used to partition a large feature vector into committees of smaller dimensionality classifiers. Considerable benefit has been observed by applying such committee-based classification framework in [14].

The goal of the thesis is to investigate the large-margin training criteria, to integrate the training with flexible modeling structures such as hierarchical classifiers or committee-based classification, and to compare the large-margin training with other discriminative training methods. The proposed modeling scheme will first be implemented and evaluated on the TIMIT phonetic benchmark task [12], and will be extended to a LVCSR task from the MIT lecture corpus [8].

1.2 Discriminative Training Methods for ASR

This section reviews discriminative training methods for GMMs models as proposed in the literature. The training methods reviewed include maximum mutual information (MMI) training [38], minimum phone error (MPE) training [29], and minimum classification error (MCE) [17, 24, 25] training. In the following sections, the objective functions and optimization algorithms of the discriminative methods are illustrated, followed by a brief comparison between the training methods.

1.2.1 MMI Training

This subsection briefly describes the MMI training of GMMs in a hidden Markov model (HMM) based speech recognizer. The basic idea of MMI training is to seek model parameters that can maximize the posterior probability of the correct transcription being generated by the model. Maximizing the posterior probability of the correct string increases the separation between the correct string and other competing hypotheses, and thus reduces confusion.

Objective Function

Given a set of training acoustic observation sequences $\{\mathbf{X}_1, \ldots, \mathbf{X}_N\}$, the corresponding transcription $\{\mathbf{Y}_1, \ldots, \mathbf{Y}_N\}$, and HMM parameter set $\boldsymbol{\lambda}$, the objective function of MMI

training can be expressed by

$$\begin{aligned} F_{MMI}(\boldsymbol{\lambda}) &= \log(p(\mathbf{Y}_1, \ldots, \mathbf{Y}_N | \mathbf{X}_1, \ldots, \mathbf{X}_N)) \\ &= \sum_{n=1}^{N} \log(p(\mathbf{Y}_n | \mathbf{X}_n)) \\ &= \sum_{n=1}^{N} \log(\frac{p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{Y}_n)^{\kappa} p_L(\mathbf{Y}_n)^{\kappa}}{\sum_{\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa})}), \end{aligned} \tag{1.1}$$

where $p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})$ is the probability of the observation sequence $\mathbf{X}_n$ being generated by $\boldsymbol{\lambda}$ given the hypothesis $\mathbf{S}$, $p_L(\mathbf{S})$ is the language model probability of hypothesis $\mathbf{S}$, and κ is a scaling factor that controls the relative weights between the acoustic model and the language model during the training.[1] Note that if the denominator term in Equation (1.1) is removed, the objective function becomes the same as what is used in ML training.

Auxiliary Function

Maximizing of the objective function in Equation (1.1) can be achieved either by applying gradient based methods such as Generalized Probabilistic Descent (GPD) [18] or by applying Extended Baum-Welch (EBW) update [10] with an appropriate auxiliary function, $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. The following paragraphs briefly describe the EBW update for MMI training used in [28]. More detailed mathematical descriptions can be found in Appendix A.

The basic EBW update procedures for MMI training are composed of the following steps:

1. Starting from HMM parameter set $\boldsymbol{\lambda}'$, construct an auxiliary function $G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ for $F_{MMI}(\boldsymbol{\lambda})$ around $\boldsymbol{\lambda}'$.

2. Update the parameter set to $\hat{\boldsymbol{\lambda}}$ such that $G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is maximized.

3. Repeat step 1 and 2 till the objective function $F_{MMI}(\boldsymbol{\lambda})$ converges.

The auxiliary function $G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ can be decomposed into the following form:

$$G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') + G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}'), \tag{1.2}$$

[1] For notational convenience, it is assumed that the language model probability $p_L(\mathbf{S})$ has been scaled by a factor $\frac{1}{\kappa}$, and thus further scaling by κ reverts the probability back to its original value.

where $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ corresponds to the numerator term in Equation (1.1), $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ corresponds to the denominator term in Equation (1.1), and $G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is a smoothing function that has maximum at $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$. $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is the same as the auxiliary function used in the E-M update of ML training; $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is similar to $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ but considers all hypotheses generated by the speech recognizer; and the smoothing function $G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is intended to make the auxiliary function converge better. Details about constructing the auxiliary function can be found in Appendix A.1.

The maximization of $G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ involves computation for the partial derivative of $G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ with respect to the parameter set $\boldsymbol{\lambda}$. Because the term $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ in Equation (1.2) is the same as the auxiliary function used in the E-M update of ML training and the $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is similar to $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, the procedures to compute the partial derivative are similar to what are used in ML training. The first step is to compute statistics such as the posterior probabilities of occupation of HMM states and the weighted-sum of training data with respect to the posterior probabilities based on the parameter set $\boldsymbol{\lambda}'$ from the previous iteration. Efficient computation of such statistics can be achieved by utilizing phone lattices generated by the recognizer. There are two types of phone lattices that are used in MMI training:

1. Numerator-lattices: Lattices used for computing partial derivatives of $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. Numerator-lattices are generated by the recognizer in forced-alignment mode that produces state sequences that match each observation sequence $\mathbf{X}_n$ with its corresponding transcription $\mathbf{Y}_n$.

2. Denominator-lattices: Lattices used for computing partial derivatives of $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. Denominator-lattices are generated by the recognizer in recognition mode that produces hypotheses for each observation sequence $\mathbf{X}_n$. Denominator-lattices can also be called recognition-lattices since they are generated by the recognition process.

Note that because of pruning operations during recognition, the correct transcription may not necessarily appear in the denominator-lattices. The next section describes how to compute statistics needed for MMI triaining.

Computing Statistics

The following procedures compute statistics needed for computing the partial derivative for $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$; similar procedures can be applied to compute statistics for $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. Given the start and end times of a phone arc q in a numerator-lattice, the HMM forward-backward procedure can be applied to compute the within-arc posterior probability $\gamma_{jmq}^{num}(t)$ of the m^{th} Gaussian mixture component of HMM state j at time t. Putting together all the within-arc posterior probabilities and running the forward-backward procedure across the entire lattice generates γ_q^{num}, the posterior probability of arc q being traversed. The occupation γ_{jm}^{num} of mixture component m of state j can be computed by summing over the within-arc posterior probabilities of all phone arcs in all the numerator-lattices:

$$\gamma_{jm}^{num} = \sum_{q=1}^{Q^{num}} \sum_{t=s_q}^{e_q} \gamma_{jmq}^{num}(t) \gamma_q^{num}, \tag{1.3}$$

where s_q and e_q denote the start and end times of phone arc q, and Q^{num} is total number of phone arcs in the numerator-lattices. The weighted sum of the training data can be computed by the following:

$$\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}) = \sum_{q=1}^{Q^{num}} \sum_{t=s_q}^{e_q} \gamma_{jmq}^{num}(t) \gamma_{jm}^{num} \mathbf{x}_t, \tag{1.4}$$

where $\mathbf{x}_t$ is the observation vector at time t. Also the weighted square sum of the training data can be computed by

$$\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}^2) = \sum_{q=1}^{Q^{num}} \sum_{t=s_q}^{e_q} \gamma_{jmq}^{num}(t) \gamma_{jm}^{num} \mathbf{x}_t \mathbf{x}_t^{\mathsf{T}}. \tag{1.5}$$

Applying similar procedures on the denominator-lattices, statisitcs for $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ such as γ_{jm}^{den}, $\boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X})$, and $\boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X}^2)$ can also be computed.

Parameter Update

The following paragraphs describe how to update the mean vectors and covariance matrices in $\boldsymbol{\lambda}$. The basic idea is to find the parameters such that the partial derivatives of the auxiliary function is zero. Because $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is the same as the auxiliary function used in ML training, the partial derivatives of $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ with respect to mean vectors and covariance matrices can be expressed by the following forms:

$$\frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \propto (\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}) - \gamma_{jm}^{num} \boldsymbol{\mu}_{jm}), \tag{1.6}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \propto (\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}^2) - \boldsymbol{\mu}_{jm} \boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X})^{\mathrm{T}} - \boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}) \boldsymbol{\mu}_{jm}^{\mathrm{T}} + \gamma_{jm}^{num} \boldsymbol{\mu}_{jm} \boldsymbol{\mu}_{jm}^{\mathrm{T}}), \tag{1.7}$$

where $\boldsymbol{\mu}_{jm}$ is the mean vector of the m^{th} Gaussian mixture component of HMM state j, and $\boldsymbol{\Sigma}_{jm}$ is the covariance matrix. Because $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is of similar form as $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, the partial derivatives of $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ can be also expressed by:

$$\frac{\partial}{\partial \boldsymbol{\mu}_{j,m}} G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \propto (\boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X}) - \gamma_{jm}^{den} \boldsymbol{\mu}_{jm}), \tag{1.8}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \propto (\boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X}^2) - \boldsymbol{\mu}_{jm} \boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X})^{\mathrm{T}} - \boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X}) \boldsymbol{\mu}_{jm}^{\mathrm{T}} + \gamma_{jm}^{den} \boldsymbol{\mu}_{jm} \boldsymbol{\mu}_{jm}^{\mathrm{T}}). \tag{1.9}$$

Note that the constant matrices for the $\propto$ in (1.6) and in (1.8) are the same given the same $\boldsymbol{\mu}_{jm}$ and $\boldsymbol{\Sigma}_{jm}$. The same thing also holds for the matrices in (1.7) and (1.9). Detailed derivation of the partial derivatives can be found in Appendix A.2. Also, because the $G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ in Equation (1.2) has maximum at $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$, the derivatives of $G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ with respect to $\boldsymbol{\mu}_{jm}$ and $\boldsymbol{\Sigma}_{jm}$ are of the forms:

$$\frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \propto (\boldsymbol{\mu}'_{jm} - \boldsymbol{\mu}_{jm}), \tag{1.10}$$

where $\boldsymbol{\mu}'_{jm}$ is the mean vector of mixture m of state j in $\boldsymbol{\lambda}'$; and

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \propto ((\boldsymbol{\Sigma}'_{jm} + \boldsymbol{\mu}'_{jm} \boldsymbol{\mu}'^{\mathrm{T}}_{jm}) - \boldsymbol{\mu}'_{jm} \boldsymbol{\mu}^{\mathrm{T}}_{jm} - \boldsymbol{\mu}_{jm} \boldsymbol{\mu}'^{\mathrm{T}}_{jm} + \boldsymbol{\mu}_{jm} \boldsymbol{\mu}'_{jm} - \boldsymbol{\Sigma}_{jm}). \tag{1.11}$$

Setting $\frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - \frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') + \frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \mathbf{0}$ and solving for the mean $\hat{\boldsymbol{\mu}}_{jm}$ yields

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}) - \boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X}) + \eta_{jm}\boldsymbol{\mu}'_{j,m}}{\gamma_{jm}^{num} - \gamma_{jm}^{den} + \eta_{jm}}, \tag{1.12}$$

where η_{jm} is a positive constant that controls the degree of smoothing. In similar way, combining the derivatives in (1.7), (1.9), and (1.11) and plugging in mean vector in Equation (1.12) to solve $\boldsymbol{\Sigma}_{jm}$ results in

$$\hat{\boldsymbol{\Sigma}}_{jm} = \frac{\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}^2) - \boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X}^2) + \eta_{jm}(\boldsymbol{\Sigma}'_{jm} + \boldsymbol{\mu}'_{jm}\boldsymbol{\mu}'^{\mathrm{T}}_{jm})}{\gamma_{jm}^{num} - \gamma_{jm}^{den} + \eta_{jm}} - \hat{\boldsymbol{\mu}}_{jm}\hat{\boldsymbol{\mu}}_{jm}^{\mathrm{T}}. \tag{1.13}$$

Note that the value of the smoothing constant η_{jm} is critical. If it is too small, some covariance matrices are not guaranteed to be positive semi-definite; if it is too large, the optimization will be slow. Also, a bad choice of the constant may potentially degrade the performance of MMI-trained models. Several heuristic selection criteria for η_{jm} can be found in [28].

For the update of mixture weights, another auxiliary function is suggested in [28] such that the sum-to-one constraint of mixture weights can be more easily incorporated into the optimization. For the mixture weights $\{w_{jm}\}_{m=1}^{M_j}$ of state j, the following auxiliary function is used:

$$\sum_{m=1}^{M_j} \gamma_{jm}^{num} \log(w_{jm}) - \frac{\gamma_{jm}^{den}}{w'_{jm}} w_{jm}, \tag{1.14}$$

where w'_{jm} is the mixture weight from the previous parameter set $\boldsymbol{\lambda}'$. The update weights $\{\hat{w}_{jm}\}_{m=1}^{M_j}$ are obtained by running the EBW procedure on Equation (1.14) for several iterations. Details of the update can also be found in Appendix A.2. Because HMM state transition weights also have the sum-to-one constraint, the transition weights can also be updated using a similar auxiliary function as in Equation (1.14).

To prevent overtraining, additional smoothing can be incorporated with MMI training. The I-smoothing that seeks to interpolate the ML-trained model with the MMI-trained model is typically used. Because the numerator terms in the MMI objective function is the same as the ML objective function, the effect of I-smoothing is the same as scaling γ_{jm}^{num} by a factor

$\frac{1+\tau}{\gamma_{jm}^{num}}$. However, the value of τ, in general, has to be tuned on the development set.

1.2.2 MPE Training

This section describes Minimum Phone Error (MPE) training proposed in [29]. The basic idea of MPE training is to maximize the average phone accuracy of hypotheses generated by the recognizer.

Objective Function

Given observation sequences $\{\mathbf{X}_1 \dots \mathbf{X}_N\}$, transcriptions $\{\mathbf{Y}_1 \dots \mathbf{Y}_N\}$, and HMM parameter set $\boldsymbol{\lambda}$, MPE training seeks to maximize

$$F_{MPE}(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \frac{\sum_{\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa} \mathrm{A}(\mathbf{S}, \mathbf{Y}_n)}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^{\kappa} p_L(\mathbf{U})^{\kappa}}, \tag{1.15}$$

where the $\mathrm{A}(\mathbf{S}, \mathbf{Y}_n)$ denotes raw phone accuracy that equals the number of phones in the reference transcription $\mathbf{Y}_n$ minus the number of phone errors. Because the term $\mathrm{A}(\mathbf{S}, \mathbf{Y}_n)$ are not necessarily positive, there is no log term in the MPE objective function as in the MMI objective function. As a result, the expected log-likelihood used in MMI training can not be directly used as auxiliary function for MPE training, and another auxiliary function has to be constructed.

Auxiliary Function

The key to constructing a tractable auxiliary function for optimizing the MPE objective function is to partition the change in $F_{MPE}(\boldsymbol{\lambda})$ into a sum of change contributed by the change of log-probability of phone arc q in the lattices. More specifically, given the HMM parameter set $\boldsymbol{\lambda}'$ from the previous iteration, construct the auxiliary function $G_{MPE}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ by

$$G_{MPE}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \sum_{n=1}^{N} \sum_{q=1}^{Q_n} \frac{\partial F_{MPE}(\boldsymbol{\lambda})}{\partial \log(p_{\boldsymbol{\lambda}}(q))} \, |_{\boldsymbol{\lambda}=\boldsymbol{\lambda}'} \, G_{ML}(\boldsymbol{\lambda}, \boldsymbol{\lambda}', n, q), \tag{1.16}$$

where N is the number of training utterances, and Q_n is the number of phone arcs in the recognition lattice of the n^{th} utterance, $p_{\boldsymbol{\lambda}}(q)$ is the probability of phone arc q being generated by $\boldsymbol{\lambda}$, and $G_{ML}(\boldsymbol{\lambda}, \boldsymbol{\lambda}', n, q)$ is the auxiliary function for $\log(p_{\boldsymbol{\lambda}}(q))$ at $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$ that is used for ML training. Note that in this way the partial derivative of $F_{MPE}(\boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$ equals to that of $G_{MPE}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ at $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$, and therefore $G_{MPE}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is a valid auxiliary function for $F_{MPE}(\boldsymbol{\lambda})$.

Parameter Update

As in MMI training, optimizing $G_{MPE}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ requires statistics computed from the training data. The key statistics required in MPE training is

$$\gamma_q^{MPE} = \frac{1}{\kappa} \frac{\partial F_{MPE}(\boldsymbol{\lambda})}{\partial \log(p_{\boldsymbol{\lambda}}(q))}, \tag{1.17}$$

for each arc q. The statistics γ_q^{MPE} can be computed by:

$$\gamma_q^{MPE} = \gamma^q (c(q) - c_{avg}^n), \tag{1.18}$$

where γ_q is the posterior probability of phone arc q being traversed, $c(q)$ is the average raw phone accuracy of hypotheses passing through phone arc q, and c_{avg}^n is the average raw phone accuracy of all hypotheses in the recognition lattice of the n^{th} training utterance.

To show how Equation (1.18) holds, let us first break the numerator and denominator terms in $F_{MPE}(\boldsymbol{\lambda})$ into two parts according to whether the hypotheses contain phone arc q; that is,

$$F_{MPE}(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \frac{\sum_{\mathbf{S}:q\in\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^\kappa p_L(\mathbf{S})^\kappa \mathrm{A}(\mathbf{S}, \mathbf{Y}_n) + \sum_{\mathbf{S}:q\notin\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^\kappa p_L(\mathbf{S})^\kappa \mathrm{A}(\mathbf{S}, \mathbf{Y}_n)}{\sum_{\mathbf{U}:q\in\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^\kappa p_L(\mathbf{U})^\kappa + \sum_{\mathbf{U}:q\notin\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^\kappa p_L(\mathbf{U})^\kappa}. \tag{1.19}$$

Note that for hypotheses $\mathbf{S}$ that contains q, the differential of $p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^\kappa$ with respect to $\log(p_{\boldsymbol{\lambda}}(q))$ is $\kappa p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^\kappa$, and that for hypotheses $\mathbf{S}$ that does not contain q, the differential

of $p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa}$ with respect to $\log(p_{\boldsymbol{\lambda}}(q))$ is zero. As a result, γ_q^{MPE} can be represented by

$$\frac{1}{\kappa}\frac{\partial F_{MPE}(\boldsymbol{\lambda})}{\partial \log(p_{\boldsymbol{\lambda}}(q))} = \frac{\sum_{\mathbf{S}:q\in\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa} \mathrm{A}(\mathbf{S},\mathbf{Y}_n)}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^{\kappa} p_L(\mathbf{U})^{\kappa}} - \frac{\sum_{\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa} \mathrm{A}(\mathbf{S},\mathbf{Y}_n)}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^{\kappa} p_L(\mathbf{U})^{\kappa}} \frac{\sum_{\mathbf{S}:q\in\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa}}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^{\kappa} p_L(\mathbf{U})^{\kappa}}. \tag{1.20}$$

The term $\frac{\sum_{\mathbf{S}:q\in\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa}}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^{\kappa} p_L(\mathbf{U})^{\kappa}}$ in Equation (1.20) is the posterior probability of q being traversed and hence equals γ_q; the term $\frac{\sum_{\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa} \mathrm{A}(\mathbf{S},\mathbf{Y}_n)}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^{\kappa} p_L(\mathbf{U})^{\kappa}}$ is the average phone accuracy of all hypotheses in the lattice of the n^{th} utterance and hence equals c_{avg}^n. Note that $\frac{\sum_{\mathbf{S}:q\in\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa} \mathrm{A}(\mathbf{S},\mathbf{Y}_n)}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^{\kappa} p_L(\mathbf{U})^{\kappa}} = \frac{\sum_{\mathbf{S}:q\in\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa} \mathrm{A}(\mathbf{S},\mathbf{Y}_n)}{\sum_{\mathbf{S}:q\in\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa}} \frac{\sum_{\mathbf{S}:q\in\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa}}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})^{\kappa} p_L(\mathbf{U})^{\kappa}}$, and therefore the whole term equals to $c(q)\gamma_q$.

After γ_q^{MPE} for all phone arcs have been computed, other statistics required to update mean vectors and covariance matrices in $\boldsymbol{\lambda}$ can be computed using similar procedures as in the MMI training. More specifically, if we let

$$\gamma_q^{num} = \max(0, \gamma_q^{MPE}) \tag{1.21}$$

$$\gamma_q^{den} = \min(0, \gamma_q^{MPE}), \tag{1.22}$$

all the statistics γ_{jm}^{num}, γ_{jm}^{den}, $\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X})$, $\boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X})$, $\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}^2)$, and $\boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X}^2)$ can be computed using the same formula as in MMI training. As a consequence, the same update formula in Equations (1.12) and (1.13) can be directly used for MPE training. An intuition for MPE training is that if a phone arc q can help to produce hypotheses that have higher phone accuracy than the average, this phone arc should be consider a positive example in the training; on the other hand, if a phone arc tends to result in hypotheses that have lower phone accuracy than the average, this arc should be considered as a negative training example in the training.

Computing Phone Accuracy

To facilitate MPE training, it is necessary to have an efficient algorithm to compute the following quantities:

- $\mathrm{A}(\mathbf{S}, \mathbf{Y}_n)$: Raw phone accuracy of hypotheses $\mathbf{S}$ given correct transcription $\mathbf{Y}_n$.

- $c(q)$: Average accuracy of hypotheses that traverse through phone arc q.

- c_{avg}^n: Average accuracy of all hypotheses for the n^{th} utterance in the training data.

The following paragraphs describe the methods to compute these quantities as in [28].

The raw phone accuracy $\mathrm{A}(\mathbf{S}, \mathbf{Y}_n)$ can be computed by summing up the individual phone accuracy of all phone arcs in the hypothesis $\mathbf{S}$; that is

$$\mathrm{A}(\mathbf{S}, \mathbf{Y}_n) = \sum_{q:q \in \mathbf{S}} \mathrm{PhoneAcc}(q, \mathbf{Y}_n), \tag{1.23}$$

where $\mathrm{PhoneAcc}(q, \mathbf{Y}_n)$ is the accuracy of phone q. Ideally, the individual phone accuracy

$$\mathrm{PhoneAcc}(q, \mathbf{Y}_n) = \left\{ \begin{array}{l} 1 \text{ if correct phone} \\ 0 \text{ if substitution} \\ -1 \text{ if insertion} \end{array} \right\}, \tag{1.24}$$

but to compute the exact value above requires alignment between hypothesis $\mathbf{S}$ and the reference transcription $\mathbf{Y}_n$. To avoid the huge computation of aligning all hypotheses with the reference, a localized phone accuracy measure was proposed in [28]. Given a phone z in the reference transcription that overlaps in time with the hypothesized phone q, the following measure is computed:

$$\mathrm{Acc}(q, z) = \left\{ \begin{array}{l} -1 + 2e(q, z) \text{ if z and q are the same phone} \\ -1 + e(q, z) \text{ if different phones} \end{array} \right\}, \tag{1.25}$$

where $e(q, z)$ is the proportion of length of z that overlaps with q. Then, the individual phone accuracy can be computed by

$$\mathrm{PhoneAcc}(q, \mathbf{Y}_n) = \max_{z \in \mathbf{Y}_n} \mathrm{Acc}(q, z). \tag{1.26}$$

Efficient computation for $c(q)$ and c^n_{avg} can be achieved by utilizing the quantities computed in the HMM forward-backward algorithm. Let α_q denote the scaled likelihood of the HMM reaching phone arc q computed by the forward procedure, and let α'_q denote the average accuracy of phone sequences leading up to q. The value of α'_q can be computed by averaging the accuracy α'_r of each phone arc r preceding q and adding the average value with the accuracy of q; that is,

$$\alpha'_q = \frac{\sum_{r \text{ preceding } q} \alpha'_r \alpha_r t^\kappa_{rq}}{\sum_{r \text{ precdeing } q} \alpha_r t^\kappa_{rq}} + \text{PhoneAcc}(q, \mathbf{Y}_n), \tag{1.27}$$

where t^κ_{rq} is the scaled transition probability from r to q. Similarly, let β_q denote the scaled likelihood of the HMM following phone arc q computed by the backward procedure, and let β'_q denote the average accuracy of phone sequences following q. The value of β'_q can be computed by

$$\beta'_q = \frac{\sum_{r \text{ following } q} t^\kappa_{qr} p_{\boldsymbol{\lambda}}(r)^\kappa \beta_r (\beta'_r + \text{PhoneAcc}(r, \mathbf{Y}_n))}{\sum_{r \text{ following } q} t^\kappa_{qr} p_{\boldsymbol{\lambda}}(r)^\kappa \beta_r}. \tag{1.28}$$

As a result, the value of $c(q)$ can be computed by

$$c(q) = \alpha'_q + \beta'_q. \tag{1.29}$$

The average accuracy of all hypotheses in the lattice can be computed by averaging α'_q of all q at the end of the lattice:

$$c^n_{avg} = \frac{\sum_{q \text{ at the end of the lattice}} \alpha'_q \alpha_q}{\sum_{q \text{ at the end of the lattice}} \alpha_q}. \tag{1.30}$$

Smoothing

As in MMI training, I-smoothing can be applied to MPE training to prevent over-fitting. To perform I-smoothing, the ML statistics γ^{ML}_{jm}, $\boldsymbol{\vartheta}^{ML}_{jm}(\mathbf{X})$, and $\boldsymbol{\vartheta}^{ML}(\mathbf{X}^2)$ have to be computed

for each state j and mixture m. I-smoothing can be performed by:

$$\begin{aligned} \gamma_{jm}'^{num} &= \gamma_{jm}^{num} + \tau \\ \vartheta_{jm}'^{num}(\mathbf{X}) &= \vartheta_{jm}^{num}(\mathbf{X}) + \tfrac{\tau}{\gamma_{jm}^{ML}} \vartheta_{jm}^{ML}(\mathbf{X}) \\ \vartheta_{jm}'^{num}(\mathbf{X}^2) &= \vartheta_{jm}^{num}(\mathbf{X}^2) + \tfrac{\tau}{\gamma_{jm}^{ML}} \vartheta_{jm}^{ML}(\mathbf{X}^2) \end{aligned} \tag{1.31}$$

where τ is a positive constant that has to be tuned. I-smoothing is generally more important for MPE training than for MMI training.

1.2.3 MCE Training

This section briefly describes Minimum Classification Error (MCE) training [18, 17, 24, 25]. The goal of MCE training is to minimize the misclassifications of training data made by the model. For each utterance in the training data, if the best hypothesis generated by the recognizer does not match the transcription, the utterance is considered as being misclassified and is counted as a loss added to the objective function.

Objective Function

Given observation sequences $\{\mathbf{X}_1 \dots \mathbf{X}_N\}$ and transcriptions $\{\mathbf{Y}_1 \dots \mathbf{Y}_N\}$, the ideal MCE objective function (loss function) can be expressed by:

$$\mathcal{N}_{err} = \sum_{n=1}^{N} \text{sign}[-\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y_n})) + \max_{\mathbf{S} \neq \mathbf{Y}_n} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{S}))], \tag{1.32}$$

where $\text{sign}[z] = 1$ for $z > 0$ and $\text{sign}[z] = 0$ for $z \leq 0$, and $p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{S})$ is the probability of observation sequence $\mathbf{X}_n$ and hypothesis $\mathbf{S}$ being generated by model parameter set $\boldsymbol{\lambda}$. MCE training seeks to find $\boldsymbol{\lambda}$ such that the number of misclassified utterances is minimized.

However, because the sign function in Equation (1.32) is not differentiable, it is generally replaced by a differentiable, monotonically increasing function between 0 and 1 such that numerical optimization algorithms can be applied for training. A typical choice of such

function is the sigmoid function

$$\ell(d) = \frac{1}{1 + e^{-\zeta d}}, \tag{1.33}$$

where ζ is a positive constant. When d is very small (negative value), $\ell(d)$ is close to 0, meaning that the utterance is correctly classified; on the other hand, when d is large, $\ell(d)$ is close to 1, meaning the utterance is seriously misclassified. The value of ζ determines the steepness of the sigmoid function. A large value of ζ results in a steep transition close to the sign function in Equation (1.32). Because the absolute value of the likelihood gap $-\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y_n})) + \max_{\mathbf{S} \neq \mathbf{Y}_n} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{S}))$ in Equation (1.32) is generally larger in longer utterances, some MCE training frameworks in the literature normalize the log likelihood gap with respect to the length of the utterance [25] in order to confine the dynamic range of the log-likelihood gap.

The max function in Equation (1.32) can also be relaxed such that more than one competing hypotheses can be considered. One way of such relaxation is to average the scaled log-likelihood of the top C hypotheses:

$$\log\left(\left[\frac{1}{C}\sum_{\mathbf{S} \neq \mathbf{Y}_n}^{C} \exp(\nu \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{S})))\right]^{\frac{1}{\nu}}\right), \tag{1.34}$$

where ν is a positive scaling constant. In the limiting case, when ν approaches infinity, the expression in Equation (1.32) becomes the same as $\max_{\mathbf{S} \neq \mathbf{Y}_n} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{S}))$. As a result, the relaxed MCE objective function can be expressed by

$$F_{MCE}(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \ell\left(\frac{1}{\iota_n}(-\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n)) + \log(\left[\frac{1}{C}\sum_{\mathbf{S} \neq \mathbf{Y}_n}^{C} \exp(\nu \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{S})))\right]^{\frac{1}{\nu}}))\right), \tag{1.35}$$

where $\ell(d) = \frac{1}{1+\exp(-\zeta d)}$ is the sigmoid function, and ι_n equals to the number of frames in the string if the normalized version is used ($\iota = 1$ if no normalization). The values of ζ and ν, in general, have to be set heuristically, and several tips for choosing these values can be found in [25].

Parameter Update

Although similar EBW procedures can be applied for parameter update of MCE training [23], most MCE related work presented in the literature applies gradient-based methods for MCE training. The following paragraphs illustrate how to compute the gradient of the MCE objective function for gradient-based optimization methods. By investigating the computation for the gradient of MCE objective function, several intrinsic properties of MCE training can also be illustrated.

Let $d_n^{MCE}(\boldsymbol{\lambda}) = -\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \mathbf{Y}_n)) + \log(\left[\frac{1}{C}\sum_{\mathbf{S}\neq\mathbf{Y}_n}^{C}\exp(\nu\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{S}))\right]^{\frac{1}{\nu}})$, and the gradient of the MCE objective function $F_{MCE}(\boldsymbol{\lambda})$ can be computed by

$$\frac{\partial}{\partial\boldsymbol{\lambda}}F_{MCE}(\boldsymbol{\lambda}) = \sum_{n=1}^{N}\frac{1}{\iota_n}\frac{\partial\ell(d)}{\partial d}\mid_{d=d_n^{MCE}(\boldsymbol{\lambda})}\frac{\partial d_n^{MCE}(\boldsymbol{\lambda})}{\partial\boldsymbol{\lambda}}. \tag{1.36}$$

Note that term $\frac{\partial\ell(d)}{\partial d} = \zeta\ell(d)[1-\ell(d)]$ in Equation (1.36) has maximum of 0.25ζ at $d=0$ and that the ratio $\frac{\zeta\ell(d)[1-\ell(d)]}{0.25\zeta}$ decreases as ζ increases. As a result, MCE training gives more weight to training utterances close to the classification boundary; as the value of ζ increases, MCE training becomes more focusing on utterances near classification boundaries. Experiments in [24] show that choosing an appropriate large value of ζ for training results in better model performance than choosing a small value of ζ, where each training utterance is of similar weight.

For the gradient of $d_n^{MCE}(\boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$, the gradient can be decomposed into the following terms:

$$\begin{aligned}\frac{\partial d_n^{MCE}(\boldsymbol{\lambda})}{\partial\boldsymbol{\lambda}} &= -\frac{\partial}{\partial\boldsymbol{\lambda}}\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{Y}_n)) \\ &\quad + \sum_{\mathbf{S}\neq\mathbf{Y}_n}^{C}\frac{\exp(\nu\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{S})))}{\sum_{\mathbf{S}\neq\mathbf{Y}_n}^{C}\exp(\nu\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{S})))}\frac{\partial}{\partial\boldsymbol{\lambda}}\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{S})).\end{aligned} \tag{1.37}$$

Note that if the parameters in $\boldsymbol{\lambda}$ are moved along the direction of the gradient of $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{Y}_n))$ with respect to $\boldsymbol{\lambda}$, the value of $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n,\mathbf{Y}_n))$ increases after the update. Because the goal of MCE training is to minimize the objective function, gradient-based optimization methods move the parameters in $\boldsymbol{\lambda}$ in the reverse direction of the gradient $\frac{\partial}{\partial\boldsymbol{\lambda}}F_{MCE}(\boldsymbol{\lambda})$. As a

consequence, the intuition of the gradient in Equation (1.37) can be interpreted as follows. For each training utterance, MCE training seeks to increase the likelihood of the correct transcription being generated and to decrease the likelihood of competing hypotheses. Since there are many potential competing paths, MCE training uses the scaled posterior probability, $\frac{\exp(\nu \log(p_\lambda(\mathbf{X}_n,\mathbf{S})))}{\sum_{\mathbf{S}\neq\mathbf{Y}_n}^{\mathcal{C}} \exp(\nu \log(p_\lambda(\mathbf{X}_n,\mathbf{S})))}$, as a weight to penalize competing hypotheses; the higher the log-likelihood of the hypothesis is, the more penalty is imposed on the hypothesis.

As in other acoustic training framework, the gradient $\frac{\partial}{\partial \boldsymbol{\lambda}} F_{MCE}(\boldsymbol{\lambda})$ can be decomposed into a concatenation of partial derivatives with respect to GMM parameters and state transition weights. After all partial derivatives have been computed, the model can be updated by applying gradient-based optimization methods. Several kinds of optimization methods for MCE training have been compared in [25]. Based on the results reported in [25], the Quickprop algorithm resulted in the best MCE trained models. More details about the Quickprop algorithm can be seen in Appendix B.

1.2.4 Comparisons of Discriminative Training Methods

In previous sections, discriminative training methods including MMI, MPE, and MCE training have been reviewed. This section first briefly compares how the discriminative training methods select training examples for GMMs in the parameter set and how the methods determine the weight of each training utterance. Optimization methods used for the discriminative training methods are also compared.

Selecting Training Examples for GMMs

Discriminative training for GMMs, in some sense, can be viewed as a process of using objective function to guide the selection of training examples (observation vectors) for GMM parameters. For each GMM, its training examples can be divided into two types: positive training examples whose likelihood of being generated by the GMM should be increased; and negative training examples whose likelihood should be decreased. In contrast to ML training where only positive training examples are considered, discriminative training considers

both positive and negative training examples and seeks to increase the likelihood difference between the two types of examples. Depending on the type of objective function, different signs and weights can be assigned to observation vectors in the training data.

MMI and MCE training both utilize the forced-alignment of reference transcriptions to allocate positive training examples for the GMMs. For selecting negative training examples, MMI training treats the observation vectors corresponding to phone arcs in recognition-lattices as negative training examples for GMMs related to those phone arcs. MCE training collects negative training examples from recognition outputs (N-best list or lattices, depending on the implementation) in a similar manner as MMI but does not consider the portions contributed by correct hypotheses in the recognition outputs. The weight of each training example in these two training methods is determined according to the posterior probability of occupation of the example.

On the other hand, MPE training uses the average phone accuracy c^n_{avg} of each training utterance as a threshold to decide whether the observation vectors within a phone arc are positive training examples for the GMMs related to the arc. If $c(q)$, the average phone accuracy of hypotheses passing through phone arc q, is higher than c^n_{avg}, the observation vectors within arc q are considered as a positive training examples for the GMMs related to q; otherwise, the vectors are considered as negative training examples. Further, instead of using posterior probabilities of occupation as weights for training examples, MPE scales the posterior probability of each arc q by a factor of $|c(q) - c^n_{avg}|$, and uses the scaled probabilities as weights for training.

By scaling the posterior probability with $|c(q) - c^n_{avg}|$, MPE tends to focus on differentiating hypotheses with high accuracy from those with low accuracy, and may potentially enhance the reduction of confusion. However, because an incorrect phone can be considered as a positive example if the hypotheses passing through it have high average accuracy, the positive training examples selected by MPE training can potentially be noisier than those selected by MMI or MCE training. This fact may explain why I-smoothing is more important to MPE training than to MMI training. The ML statistics added in the I-smoothing can

provide additional positive training examples for MPE and thus enhance the performance of MPE.

Weight of Trainning Utterance

Because of the sigmoid function in its objective function, MCE training gives more weight to utterance that is near the classification boundary. Although this weighting helps to reduce confusions by focusing on errors that are easier to correct, it also has a potential side-effect of penalizing longer utterances. This is because longer utterances tend to have larger dynamic range of likelihood gap between the reference and competing hypotheses. Larger dynamic range of likelihood gap tends to make the sigmoid function assign smaller weights to the utterances. Although normalizing the likelihood gap with respect to the length of utterance helps to confine the dynamic range, the normalization also introduces a scaling inversely proportional to the length of the utterance to the gradient of the utterance. Whether the normalization is performed or not, longer utterances are penalized in some sense. As a result, appropriate chopping of the training data may be necessary for MCE training to avoid the penalizing effects on longer utterance.

On the other hand, MMI and MPE training do not use other function to adjust the weight of each utterance. However, because longer utterances can generally provide more training examples, longer utterances tend to contribute more effects to the training than shorter utterances. In MPE training, the effects of longer utterances can potentially be enhanced further. This is because the dynamic range of $c(q) - c^n_{avg}$ in longer utterances tends to be larger, and potentially can give more weights to the arcs in longer utterances.

Comparisons of Optimization Methods

EBW algorithm and gradient-based methods are major techniques used for the parameter optimization of discriminative training methods. EBW algorithm optimizes the objective function by re-estimating the parameters such that an auxiliary function related to the objective function can be maximized. Gradient-based methods, on the other hand, use

the gradient as a reference to compute the update-step of the parameters. The following paragraphs discuss practical issues of applying the two types of optimization methods to discriminative training.

To use EBW algorithm for discriminative training, the first thing to do is to construct an appropriate auxiliary function. Generally, a tractable strong-sense auxiliary function which can guarantee to increase the objective function after each update is desired. However, because such kind of strong-sense auxiliary functions are not known to exist for the objective functions of discriminative training methods, weak-sense auxiliary functions which only guarantee to have the same partial derivatives as the objective functions are used instead. To make a weak-sense auxiliary function vary more consistently with the objective function, smoothing terms are added to the auxiliary function. The smoothing terms can make the maximum of the auxiliary function closer to the initial point of the parameter set, and can make better guarantee to increase the objective function. However, how to set the smoothing terms is critical. If the smoothing terms are too small, the training can become unstable; if they are too large, the training can be slowed down and are more easily to get trapped at a bad local extrema. Having appropriate smoothing for the auxiliary function is the key to successfully apply EBW algorithm for discriminative training methods.

For the gradient-based methods, the key issue is to select an appropriate learning rate of updating the parameters. If the learning rate is too large, the optimization may become unstable; if it is too small, the optimization may easily fall to a poor local extrema. Although several gradient-based methods [1, 21] have more sophisticated ways of choosing proper scaling of the update step, a task-dependent initial learning rate generally has to be heuristically specified as well as other learning parameters. Generally, applying gradient-based methods has fewer parameter tuning compared with applying EBW algorithm, but the number of iterations required before the training converge can be larger.

1.3 Multi-level classification

This section introduces multi-level classification proposed in [13] and [14]. Two types of multi-level classifiers are discussed. The hierarchical classifiers divide the classification problem into a set of sub-problems, while the committee-based classifiers combine the outputs of different classifiers to make a joint decision. Both types of the classifiers have been shown to have potential to improve the acoustic model performance. Details of the classifiers are illustrated in the following subsections.

1.3.1 Hierarchical classifiers

The basic idea of hierarchical classifiers is to use a hierarchical structure to divide and conquer the whole classification problem. Figure 1-1 is an example of a two-level hierarchical classifier. The structure of the hierarchy can be constructed either by automatic clustering algorithms or by acoustic-phonetic knowledge. For a hierarchical GMM classifier, each non-root node in the hierarchical tree has its own set of GMM parameters. Given a feature vector $\mathbf{x}$, the node c can return a distance metric $d(\mathbf{x}, c)$ which is the negative log-likelihood of $\mathbf{x}$ being generated by the GMM parameter of c with a constant shift. Because the parent node in the hierarchy has the training examples of all its children nodes, the parameter estimation for the parent node is generally more robust.

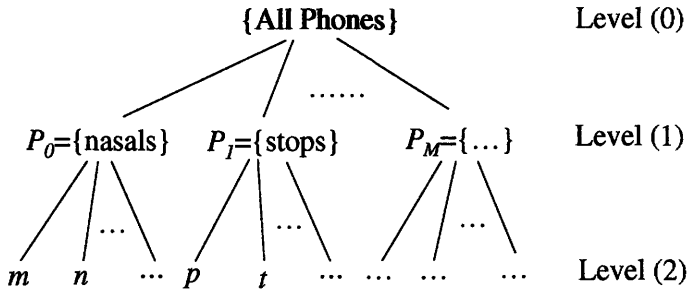

Figure 1-1: Hierarchical classifier. The leaf nodes are the possible output labels of the classifier.

Given a feature vector $\mathbf{x}$, the hierarchical GMM classifier can choose the output label

by comparing the weighted sum of the distance metrics from the root to the leaves. For example, given a two-level hierarchical GMM classifier, the output label $\hat{y}$ can be predicted by

$$\hat{y} = \arg\min_{c} w_C d(\mathbf{x}, c) + w_P d(\mathbf{x}, \mathcal{P}(c)), \tag{1.38}$$

where $\mathcal{P}(c)$ is the index of the parent node of class c, w_C and w_P are relative weights that reflect the importance of the two levels in the hierarchy. The values of w_C and w_P can be specified by cross-validation.

1.3.2 Committee-based classifiers

Committee-based classifiers are classifiers that can combine the classification results from models trained by different types of features. Several decision making criteria of committee-based classifiers have been studied in [14], including decisions based on voting, decisions based on linear combination of log-likelihood ratio (LCLR), and decisions based on an independence assumption.

The voting criterion works by simply counting the classification result from each committee member, and choosing the output that gets the highest number of votes. Ties are solved by assigning priorities to the committee members. The LCLR criterion first sums up the log-likelihood ratio (posterior probability) of each output class across all the committee members and picks the output class with highest total log-likelihood ratio. The independence assumption criterion assumes that the features used by the committee members are statistically independent, and a decision is made by comparing the summed log-likelihood across the committee members. Experiments in [14] showed that the LCLR criterion and the independence assumption criterion have similar performance.

1.4 Organization of the Thesis

In this thesis, a recently proposed large-margin discriminative training framework for Gaussian mixture models is investigated. In chapter 2, the large-margin training framework is

integrated with multi-level classifiers to target a benchmark problem of TIMIT phonetic classification [20]. In chapter 3, the effort of expanding the large-margin training framework to a large vocabulary speech recognition task is presented, and the large-margin models are compared with MCE trained models on the MIT lecture corpus [8]. Chapter 4 concludes the thesis and proposes several possible future research directions.

Chapter 2

Large-Margin GMMs for Phonetic Classification

This chapter introduces how to integrate the large-margin discriminative training framework with multi-level classifiers to tackle the problem of phonetic classification. The large-margin training framework in [33] is first reviewed, and then a training approach that combines the large-margin training criterion with hierarchical GMM classifiers is proposed. The set up of TIMIT corpus is illustrated, and experimental results of the proposed modeling scheme on the benchmark task of TIMIT phonetic classification are reported. Several issues about the large-margin training on phonetic classification are also discussed at the end of the chapter.

2.1 Large-Margin GMMs

While several variants of large-margin GMM training have been proposed in the literature [33, 34, 22, 39], this section focus on the framework proposed by Sha and Saul [33] in that their framework provides a more direct perspective of how the large-margin constraints can be incorporated in discriminative training. In the following subsections, the loss function of large-margin training is first illustrated, and then some practical issues about the large-margin training are discussed.

Loss function

The basic principle of large-margin training is to make the distance metric of the correct class be smaller than that of the competing class by at least some margin $\xi \geq 0$ if possible. To be more specific, consider the multi-way classification problem with features $\{\mathbf{x}_n\}_{n=1}^N$ and corresponding labels $\{y_n\}_{n=1}^N$, where $y_n \in \{1, 2, \ldots, C\}$.

For each token in the training data, the large-margin criterion requires that

$$\forall c \neq y_n, d(\mathbf{x}_n, c) \geq \xi + d(\mathbf{x}_n, y_n), \tag{2.1}$$

where $d(\mathbf{x}_n, c)$ denotes the distance metric of feature vector $\mathbf{x}_n$ with respect to class c computed by the model. Typically, in the GMM framework, the distance metric $d(\mathbf{x}_n, c)$ can be expressed by

$$d(\mathbf{x}_n, c) = -\log(p(\mathbf{x}_n, c)) + \theta, \tag{2.2}$$

where $p(\mathbf{x}_n, c)$ is the GMM probability and θ is a constant that is common for all the classes. A sufficiently large value of θ is typically selected such that all the distances metrics will be greater than or equal to 0. For each violation of the constraint in (2.1), the training criterion will add the difference to the training objective function, resulting in a token-level loss function

$$l_n = \sum_{c \neq y_n} [\xi + d(\mathbf{x}_n, y_n) - d(\mathbf{x}_n, c)]_+, \tag{2.3}$$

where $[f]_+ = \max(0, f)$. The overall training loss of the data is derived by summing up the token-level losses

$$\mathfrak{L} = \sum_{n=1}^{N} l_n, \tag{2.4}$$

and the model parameters can be derived by minimizing the loss in (2.4).

Outlier handling

One practical issue of large-margin training is to handle the problem of outliers. Outliers are training examples that lie at the opposite side of the decision boundary and are far away

from the boundary. Examples of outliers are shown in Figure 2-1.

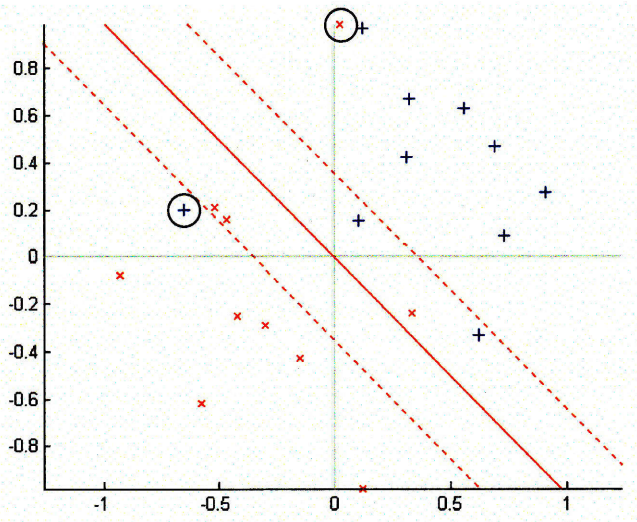

Figure 2-1: The circled data-points are examples of outliers. The red solid line is the decision boundary, and the region within the two red dash lines is the space with margin less than 0.25.

As shown in Figure 2-1, outliers can contribute a large amount of margin violation, and thus can dominate and potentially mislead the training. To reduce the effect of outliers, several heuristic approaches have been proposed.

In [33], a token-wise re-weighting method was proposed to handle the outlier problem. The basic idea of re-weighting is to multiply each training token with a weight that is inversely proportional to its initial loss. More specifically, let l_n^{ML} be the loss of the n^{th} training token computed by the initial maximum-likelihood model, a weight w_n can be chosen by $w_n = \min(\frac{1}{\xi}, \frac{1}{l_n^{ML}})$, and results in a weighted loss function

$$\mathfrak{L} = \sum_{n=1}^{N} w_n l_n. \tag{2.5}$$

By doing such re-weighting, outliers contribute an equal amount of loss as all other examples, and thus avoid impacting training. Another approach proposed in [39] suggested that picking a smaller margin value at the beginning of the training and gradually increasing the margin. Although these methods are shown to be effective, better algorithms to handle the outlier

are still desired for large-margin training.

Parameter optimization

While GMM parameters can be determined by directly applying conventional gradient-based numerical optimization with respect to the large-margin loss function, better convex optimization algorithms can be applied by doing the following modifications as described in [33].

1. Transform the GMM parameters into positive semi-definitive matrices.
2. Modify the $d(\mathbf{x}_n, y_n)$ term in (2.1) such that the distance metric is computed by a single, pre-specified, mixture at each token.

To transform the GMM parameters into positive semi-definitive matrices, consider the m^{th} mixture component of class c with mean $\boldsymbol{\mu}_{cm}$, inverse covariance matrix $\boldsymbol{\Sigma}_{cm}^{-1}$, and mixture weight w_{cm}. Given the feature vector $\mathbf{x}$, the log-likelihood contributed by this component can be computed by

$$\rho(\mathbf{x}, c, m) = -\frac{1}{2}((\mathbf{x} - \boldsymbol{\mu}_{cm})^{\mathrm{T}}\boldsymbol{\Sigma}_{cm}^{-1}(\mathbf{x} - \boldsymbol{\mu}_{cm}) + \theta_{cm}), \tag{2.6}$$

where $\theta_{cm} = \log(\det(\Sigma)) - 2\log(w_{cm})$. By setting the matrix

$$\boldsymbol{\Phi}_{cm} = \begin{bmatrix} \boldsymbol{\Sigma}_{cm} & -\boldsymbol{\Sigma}_{cm}\boldsymbol{\mu}_{cm} \\ -\boldsymbol{\mu}_{cm}^{\mathrm{T}}\boldsymbol{\Sigma}_{cm} & \boldsymbol{\mu}_{cm}^{\mathrm{T}}\boldsymbol{\Sigma}_{cm}\boldsymbol{\mu}_{cm} + \theta_{cm} \end{bmatrix}, \tag{2.7}$$

and letting $\mathbf{z} = [\mathbf{x}^{\mathrm{T}}\ 1]^{\mathrm{T}}$, the mixture log-likelihood can be expressed by

$$\rho(\mathbf{x}, c, m) = -\frac{1}{2}\mathbf{z}^{\mathrm{T}}\boldsymbol{\Phi}_{cm}\mathbf{z}, \tag{2.8}$$

and the log-likelihood of the mixture model can be computed by

$$\log(p(\mathbf{x}, c)) = \log(\sum_{m} \exp(\rho(\mathbf{x}, c, m))). \tag{2.9}$$

If a sufficiently large constant is added to the last element of $\mathbf{\Phi}_{cm}$, the matrix $\mathbf{\Phi}_{cm}$ can become positive semi-definite. If the proper value is selected such that $\{\mathbf{\Phi}_{cm}\}$ is positive semi-definite for all c and m, the distance metric $d(\mathbf{x}, c)$ will be of the form as in (2.2).

Since the function $-\log(\sum_m \exp(-d_m))$ is a concave function with respect to each d_m, the distance metric $d(\mathbf{x}, c)$ will be a concave function of the matrices $\{\mathbf{\Phi}_{cm}\}$. As a result, the $-d(\mathbf{x}, c)$ term is a convex to the matrices $\{\mathbf{\Phi}_{cm}\}$. Furthermore, since $\rho(\mathbf{x}, y_n, m)$ is a linear function with respect to $\mathbf{\Phi}_{y_n m}$, it is also a convex function to $\mathbf{\Phi}_{cm}$. Therefore, given a pre-specified mixture index m_n, the token-level loss l_n can be modified by

$$l_n = \sum_{c \neq y_n} [\xi - \rho(\mathbf{x}, y_n, m_n) - d(\mathbf{x}, c)]_+, \tag{2.10}$$

such that l_n is convex function with respect to $\{\mathbf{\Phi}_{cm}\}$. Note that the index m_n can be specified by picking the mixture component with the largest log-likelihood in the initial model. In this way, since the overall loss is convex with respect to the $\mathbf{\Phi}$ matrices, the problem of spurious local minimum is avoided, and efficient convex optimization algorithms such as convex conjugate (CG) algorithm [31] can be applied.

2.2 Hierarchical Large-Margin GMM Training

This section illustrates how to combine hierarchical GMM classifiers with large-margin training. Although the proposed training framework focuses on the training of 2-level hierarchical classifier as shown in Figure 1-1, it is generalizable to classifiers with higher level hierarchies. In the following, the joint margin criterion of the training is first introduced, and the training procedures are presented.

2.2.1 Joint Margin Criterion

Given a 2-level hierarchical GMM classifier as in Figure 1-1, all its GMM parameters can be transformed into positive semi-definite matrices by applying the transform in Equation 2.7

and adding an appropriate shift. For convenience, let us call the leaf nodes in the 2-level hierarchy by class-level nodes, and call the non-leaf nodes (except the root) by cluster-level nodes. For each class-level node c, its GMM parameters can be represented by a set of matrices $\mathbf{\Phi_c} = \{\mathbf{\Phi}_{cm}\}_{m=1}^{M_c}$, where M_c is the number of mixture components in c; for each cluster-level node p, the parameters can be represented by $\mathbf{\Theta}_p = \{\mathbf{\Theta}_{pk}\}_{k=1}^{K_p}$, where $\mathbf{\Theta}_{pk}$ is the parameter matrix for the k^{th} mixture component of p and K_p is the total number of mixture components of p. For each class-level node c, its corresponding cluster-level node can be tracked by the parent pointer $\mathcal{P}(c)$.

Given the feature vector $\mathbf{x}_n$, the distance metric of the hierarchical classifier for the competing class c can be thus computed by

$$w_C d(\mathbf{x}_n, \mathbf{\Phi}_c) + w_P d(\mathbf{x}_n, \mathbf{\Theta}_{\mathcal{P}(\mathbf{c})}). \tag{2.11}$$

The joint margin constraint can be constructed by requiring that for each competing class c the weighted margin computed by the class-level classifier and the cluster-level classifier be greater than a positive value:

$$\forall c \neq y_n, \quad w_C(d(\mathbf{x}_n, \mathbf{\Phi}_c) - d(\mathbf{x}_n, \mathbf{\Phi}_{y_n})) + w_P(d(\mathbf{x}_n, \mathbf{\Theta}_{\mathcal{P}(\mathbf{c})}) - d(\mathbf{x}_n, \mathbf{\Theta}_{\mathcal{P}(y_n)})) \geq \xi, \tag{2.12}$$

where $d(\mathbf{x}_n, \mathbf{\Phi}_c) - d(\mathbf{x}_n, \mathbf{\Phi}_{y_n})$ is the margin of class-level classifier and $d(\mathbf{x}_n, \mathbf{\Theta}_{\mathcal{P}(\mathbf{c})}) - d(\mathbf{x}_n, \mathbf{\Theta}_{\mathcal{P}(y_n)})$ is the margin of cluster-level. Similar to the original large-margin training, every violation of the constraint in (2.12) contributes to the token-level loss l_n^h; furthermore, the distance metrics related to the correct class y_n can be relaxed such that l_n^h is a convex function with respect to the parameter matrices. As a result, the token-level loss of each training token can be expressed by

$$l_n^h = \sum_{c \neq y_n} [\xi + w_C(-\rho(\mathbf{x}_n, \mathbf{\Phi}_{y_n m_n}) - d(\mathbf{x}_n, \mathbf{\Phi}_c)) + w_P(-\rho(\mathbf{x}_n, \mathbf{\Theta}_{\mathcal{P}(y_n)k_n}) - d(\mathbf{x}_n, \mathbf{\Theta}_{\mathcal{P}(c)}))]_+, \tag{2.13}$$

where m_n and k_n are pre-specified mixture component for node y_n and $\mathcal{P}(y_n)$ respectively,

and the values $\rho(\mathbf{x}_n, \mathbf{\Phi}_{y_n m_n})$ and $\rho(\mathbf{x}_n, \mathbf{\Theta}_{\mathcal{P}(y_n)k_n})$ can be computed by a formula similar to Equation (2.8). After the token-level loss has been computed for each training token, the GMM parameters can be optimized by minimizing the weighted sum

$$\mathfrak{L} = \sum_{n=1}^{N} w_n l_n^h, \tag{2.14}$$

where w_n is the weight to handle the issue of outliers as mentioned in the previous section.

2.2.2 Parameter Optimization

Margin Scaling Factor

Different setting of the margin ξ affect the decision boundary of the models and therefore can potentially affect the performance of a large-margin trained model. As a result, choosing an appropriate margin value is important for large-margin training. Since the possible dynamic range of ξ can be large, instead of searching for the margin value ξ directly, it is more convenient to search for its reciprocal $\alpha = \frac{1}{\xi}$. This can be achieved by scaling the loss function in Equation (2.14) with α. The scaled token-level loss $l_n^{h'}$ becomes of the following form:

$$l_n^{h'} = \sum_{c \neq y_n} [1 + \alpha \Delta_{cy_n}]_+, \tag{2.15}$$

where Δ_{cy_n} contains all the remaining terms in Equation (2.13) except ξ.

Note that by the above scaling, the loss function is transformed into a function of the margin scaling factor α. In this way, the problem of searching the margin value ξ is reduced to the problem of searching the margin scaling factor α for the training. The value of α has a two-sided effect on the large-margin model. Effectively, a smaller α results in a larger margin, and will potentially make more training samples have a positive loss and thus make more training examples be considered during the training. In general, more samples being considered in the optimization can result in a more robust decision boundary so that the resulting model can be more generalizable to unseen data. However, if α is set too small,

many examples that are included by large-margin training may not be very informative for selecting a good decision boundary and will therefore limit the gain of the large-margin training. In the phonetic classification experiments presented in this chapter, several values of α were used for training to evaluate its effect on model performance. A heuristic algorithm of selecting α was also develop to see whether effect value of α can be selected efficiently.

Turbo Training

The optimization of $\mathfrak{L}$ in Equation (2.14) can be achieved by alternatively updating the two levels of classifiers in the hierarchy using convex optimization algorithm such as conjugate gradient (CG) algorithm. More specifically, the optimization procedure first fixes one set of the matrices and optimize the matrices in the other set; after several iteration, the roles of the two sets are changed and the alternative procedure is used until both of the two sets of models converge. In this way, the information learned from one set of classifiers can be used in the training for the other set. This procedure is similar to the turbo decoding used in the communication society [2]. The pseudo code of the training procedure is shown in Algorithm 1. In the TIMIT phonetic classification experiments, the value t_1 and t_2 are set to 50 and 60 respectively, and the maximum number of rounds, r, is set to 3. Because the CG algorithm determines the update step size at each iteration according to the length of gradient, separating the optimizations for $\mathbf{\Phi}$ and $\mathbf{\Theta}$ can prevent their update step size being affected by one another and may improve the efficiency of the update.

Algorithm 1 Turbo Training

1: Fix all cluster-level matrices $\mathbf{\Theta}$, run CG on class-level matrices $\mathbf{\Phi}$ for t_1 iterations to minimize $\mathfrak{L}$.
2: Fix all class-level matrices $\mathbf{\Phi}$, run CG on cluster-level matrices $\mathbf{\Theta}$ for t_2 iterations to minimize $\mathfrak{L}$.
3: Repeat 1 and 2 until CG stops or r rounds have reached.
4: Use held-out training data to choose the final models.

2.3 TIMIT Corpus

TIMIT [19] 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, 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 [12]. There are 61 ARPAbet symbols used for transcription and their example occurrences are listed in Table 2.1.

2.3.1 TIMIT Data Sets

There are three types of sentences in the TIMIT corpus: dialect (SA), phonetically-compact (SX), and phonetically-diverse (SI). The dialect sentences were designed to reveal the dialectical variation of the speakers, and were spoken by all 630 speakers. The phonetically-compact sentences were designed such that the sentences are both phonetically-comprehensive and compact. The phonetically-diverse sentences were drawn from existing text sources to reveal contextual variance. 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 2.2.

Because the SA sentences were spoken by all the speakers, they were excluded from the training and evaluation of acoustic models. The standard training set selected by NIST consists of 462 speakers and 3,696 utterances. The utterances of the other 168 speakers form a "complete" test set. Note that there is no overlap between the texts read by the speakers in the training and "complete" test set. 400 utterances of 50 speakers in the "complete" test set are extracted to form a development set for model development. Utterances of the remaining 118 speakers are called the "full" test set. Among the utterances of the "full" test set, 192 utterances by 24 speakers, 2 males and 1 females from each of 8 dialect regions, are selected as a "core" test set. Typically, acoustic model performance reported in the literature

ARPAbet	Example	ARPAbet	Example
aa	b*o*b	ix	deb*i*t
ae	b*a*t	iy	b*ee*t
ah	b*u*t	jh	*j*oke
ao	b*ough*t	k	*k*ey
aw	b*ou*t	kcl	k closure
ax	*a*bout	l	*l*ay
ax-h	p*o*tato	m	*m*om
axr	butt*er*	n	*n*oon
ay	b*i*te	ng	si*ng*
b	*b*ee	nx	wi*nn*er
bcl	b closure	ow	b*oa*t
ch	*ch*oke	oy	b*oy*
d	*d*ay	p	*p*ea
dcl	d closure	pau	pause
dh	*th*en	pcl	p closure
dx	mu*dd*y	q	glottal stop
eh	b*e*t	r	*r*ay
el	bott*le*	s	*s*ea
em	botto*m*	sh	*sh*e
en	butto*n*	t	*t*ea
eng	Wash*ing*ton	tcl	t closure
epi	epenthetic silence	th	*th*in
er	b*ir*d	uh	b*oo*k
ey	b*ai*t	uw	b*oo*t
f	*f*in	ux	t*oo*t
g	*g*ay	v	*v*an
gcl	g closure	w	*w*ay
hh	*h*ay	y	*y*acht
hv	a*h*ead	z	*z*one
ih	b*i*t	zh	a*z*ure
h#	utterance initial and final silence		

Table 2.1: ARPAbet symbols for phones in TIMIT with examples. Letters in the examples corresponding to the phones are put in italic.

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 2.2: Sentence type information of TIMIT [12].

Set	#Speakers	#Utterances	#Hours	#Tokens	#Tokens w/o q
Train	462	3696	3.14	142,910	140,225
Development	50	400	0.34	15,334	15,056
Core Test	24	192	0.16	7,333	7,215
"Full" Test	118	944	0.81	36,347	35,697

Table 2.3: Data set information of TIMIT.

was evaluated on the "core" test set.

Data set information that includes number of speakers, utterances, hours, and phonetic tokens is summarized in Table 2.3. Because the glottal stop q is generally not considered in phonetic-classification experiment, the number of tokens after the removal of q for each set is also listed in the table.

2.3.2 TIMIT Phonetic-Classification

TIMIT [19] phonetic-classification is a benchmark task used to evaluate the performance of acoustic phonetic models. The basic content of the phonetic-classification task is that, given the locations and boundaries of the phones in the utterances, try to predict the unknown phone labels. In this sense, phonetic-classification is a task that evaluates the ability of the acoustic models to distinguish different acoustic-phonetic units.

Generaly, the NIST training set is used for acoustic model training, and the core test set is used for model evaluation. The development set is used to help model development. Although the TIMIT corpus has 61 phonetic labels, a collapsed set of 39 labels is used for evaluation [20]. The mapping from 61 classes to 39 classes is listed in Table 2.4. Also, as the

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 2.4: Mapping from 61 classes to 39 classes used for scoring, from [12].

common practice of phonetic classification, the glottal stops q are ignored for both training and testing.

Table 2.5 lists some recent results reported on the TIMIT phonetic-classification task. All the methods listed in the table used the NIST training set and the core test set. While the hidden conditional random fields (CRF) method [11], the large-margin GMM method [33], and the regularized least squares with second-order features (RLS2) method [30] use a single set of acoustic features, the Hierarchical GMM [13] and the committee [14] methods utilize the multi-level classification techniques to combine the classification information from multiple types of acoustic measurements. Note that for the models using single sets of features the recently reported error rate improvement is much less than 1%, showing the difficulty of this task.

Method	Feature	Error Rate
Hierarchical GMM[13]	Seg	21.0%
Hidden CRF[11]	Frame	21.7%
Large Margin GMM[33]	Frame	21.1%
RLS2[30]	Seg	20.9%
Committee[14]	Seg	18.3%

Table 2.5: Recent reported classification results on TIMIT core test set. Feature type refers to segmental (1 vector/phone) or frame-based.

2.4 Experiments

This section reports the experimental results of hierarchical and committee-based large-margin GMM on TIMIT phonetic classification. The classification results of models trained based on different types of features and margin specifications are presented. The way of heuristically choosing an appropriate margin scaling is also discussed.

2.4.1 Features

In the experiments, eight different segmental feature measurements proposed in [14] were used to train GMMs. The feature measurements are computed by the following. For each phone segment, the time region occupied by the phone segment plus the 30ms regions beyond the start and end time of the segment are used to extract spectral representations such as Mel-frequency Cepstral Cepstral Coefficients (MFCCs) and perceptual linear prediction (PLP) coefficients. The entire time region for feature extraction is then divided into several sub-regions to capture temporal information. For each sub-region, the spectral representations computed in the sub-region are consolidated into a fixed-dimensional vector using a temporal function which can either be an average or cosine transform. The vectors computed from all sub-regions and the log-duration of the phone are concatenated into a single vector, resulting in a fixed-dimensional feature for each phone segment.

The eight different features, S1-S8 are summarized in Table 2.6. The primarily differences between the feature measurements are as follows:

	# Dims	Window [ms]	Spectral Representation	Temporal Basis
S1	61	10	12MFCC	5 avg
S2	61	30	12MFCC	5 avg
S3	61	10	12MFCC	5 cos
S4	61	30	12MFCC	5 cos
S5	64	10	9MFCC	7 cos
S6	61	30	15MFCC	4 cos
S7	61	20	12PLPCC	5 avg
S8	61	20	12PLPCC	5 cos

Table 2.6: Summary of features used for experiments.

1. The duration of Hamming window used to compute the short-time Fourier transform.

2. The number of MFCCs or PLP coefficients used.

3. The temporal basis function used to consolidate feature coefficients.

The number of dimensions of each type of feature is determined by the number of spectral coefficients and the number of temporal basis regions; for example, S1 has $5 * 12 + 1 = 61$ dimensions.

2.4.2 Baselines

For the classification experiments, the 61 TIMIT phone labels were reduced to 48 classes to train the acoustic models as in [20, 33]. The mapping from 61 classes to 48 classes is listed in Table 2.7. When doing an evaluation, the 48 classes are mapped down to the 39 classes to calculate the classification error rate as in other works in the literature.

To enable hierarchical classification, a 2-level hierarchy is built by clustering the phone classes into nine clusters according to their broad manner of articulation. The nine clusters are stops, nasals/flaps, strong fricatives, weak fricatives, high vowels, low vowels, short vowels, semi-vowels and closures (including silences). Table 2.8 lists the phone labels in each manner clusters.

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 2.7: Mapping from 61 classes to 48 classes in [33].

For each of the eight types of features, 5 kinds of ML baseline models are trained: "Gauss", "2-mix", "4-mix", "H(1,2)", and "H(2,4)". "Gauss" refers to a single full covariance Gaussian model, while "2-mix" and "4-mix" represent GMMs with at most two and four Gaussian components per classes respectively. "H(1,2)" is a hierarchical model using "Gauss" for leaf-nodes and "2-mix" for cluster-level nodes. "H(2,4)" is defined similarly. Each type of GMM was trained by Cross-Validation EM (CV-EM) algorithm [35] which has better prevention to overtraining than traditional EM, and the one with lower error rate on the development set among two independent trails was selected. For "H(1,2)" and "H(2,4)", the development set was also used to find a proper set of relative weights w_C and w_P between the two levels of the hierarchy.

Figure 2-2 shows the error rates of models trained from different feature sets, S1-S8, on the development, and Figure 2-3 shows the error rates on the core test set. Table 2.9 summarizes Figures 2-2 and 2-3 by averaging the error rates of the models trained from the eight feature sets. From Table 2.9, we can see that, on average, the performance of "H(1,2)" is close to "Gauss" and "H(2,4)" is close to "2-mix" showing that ML training does not

Cluster Type	Phone Labels
Stops	b d g p t k
Nasals/Flaps	m em n en nx ng eng dx
Strong Fricatives	s sh z zh ch jh
Weak Fricatives	v f dh th hh hv
High Vowels	ae ay eh ey ih iy oy
Low Vowels	aa au aw ow uh uw ux
Short Vowels	ah ax ax-h axr er ix
SemiVowels	el l w y r
Closures	bcl dcl gcl pcl tcl kcl epi q pau h#

Table 2.8: Phone labels in manner clusters.

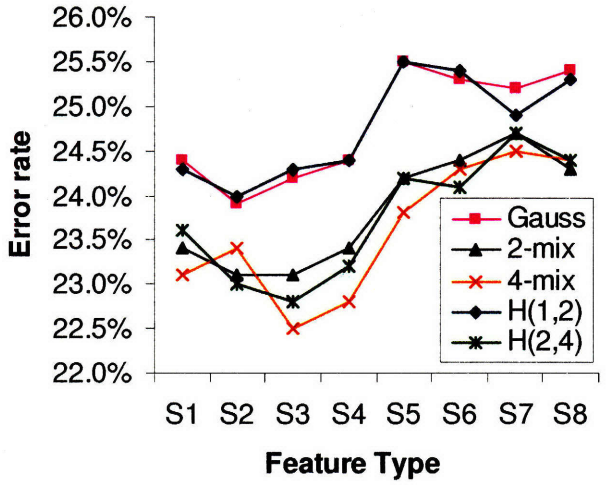

Figure 2-2: Error rates of ML GMM classifiers on Dev set.

derive much benefit from the hierarchical framework. Also, some "4-mix" models performed worse than corresponding "2-mix" models, showing that the models may start over-fitting the training data.

2.4.3 Large-Margin Classifiers

In this section the classification results of the models for each type of feature after large-margin training are presented. The large-margin models "LM Gauss", "LM 2-mix", "LM 4-mix" are trained as in [33], while "LM H(1,2)" and "LM H(2,4)" are trained by the scheme

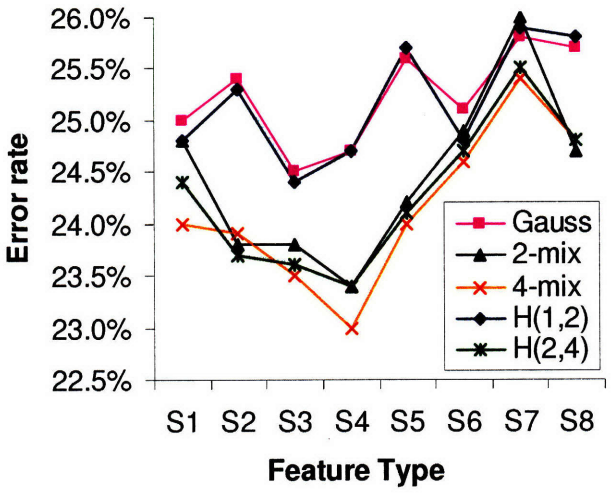

Figure 2-3: Error rates of ML GMM classifiers on Core Test set.

Set	Gauss	2-mix	4-mix	H(1,2)	H(2,4)
Dev	24.8%	23.8%	23.5%	24.7%	23.8%
Test	25.2%	24.4%	24.1%	25.2%	24.3%

Table 2.9: Average error rates of the ML GMM classifiers.

presented in the previous section.

As mentioned previously, the margin scaling factor α can significantly affect the performance of the models. To illustrate how the model performances vary according to α, several values of α were sampled and the average error rate of the eight features on the development set were plotted in Figure 2-4. As shown in Figure 2-4, the error rate decreases as α is reduced from 0.25 to 0.05, and increases as α gets smaller than 0.05. The trend of the curves is the same as discussed in Section 2.2.2. Another interesting observation is that the more complex the model is, the greater variation in classification performance; indicating that finding a good value of α becomes important as the model becomes more complex.

Figure 2-5 shows the error rates of the models for the eight feature sets on the development set under $\alpha = 0.05$, and Figure 2-6 shows the corresponding error rates on the core test set. Table 2.10 summarizes Figures 2-5 and 2-6 by showing the average error rates of the models trained from the eight features sets. As shown in Table 2.10, although "LM 4-mix" has

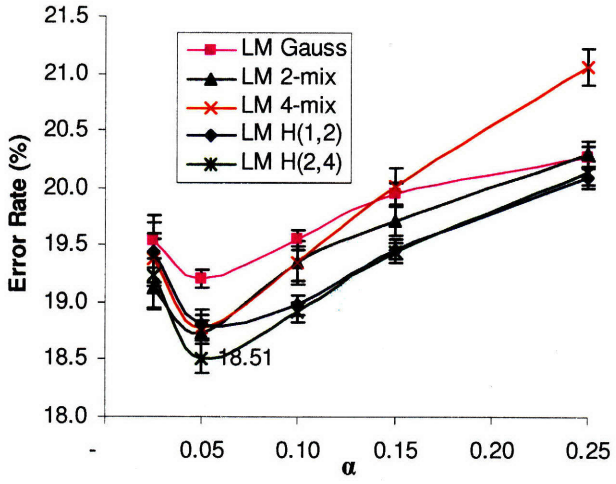

Figure 2-4: Average error rate on Dev set under different margin scaling factor. Error bars show 0.25 standard deviation across feature sets.

Set	Gauss	2-mix	4-mix	H(1,2)	H(2,4)
Dev	19.2%	18.7%	18.8%	18.8%	**18.5%**
Test	20.6%	20.0%	20.0%	19.9%	**19.6%**

Table 2.10: Average error rates of the large-margin GMM classifiers.

almost twice the number of parameters as "LM 2-mix", the performances of the two kinds of models are quite similar on average. This shows that simply increasing the number of mixtures may not necessarily improve the overall performance, since the model may overfit the training data. On the other hand, "LM H(1,2)" and "LM H(2,4)" achieve better performance than the other three types of models on average, showing that the hierarchical models tend to generalize better to unseen data.

To see whether the hierarchical large-margin GMM has significant improvement over the current state of art, the outputs of "LM H(2,4)" were compared with that of RLS2 model [30] and a McNemar significance test [6] was conducted. Six out of the eight models were significantly different at the 0.001 level. This also includes the model trained with feature set S2, which was also used for the RLS2 experiment.

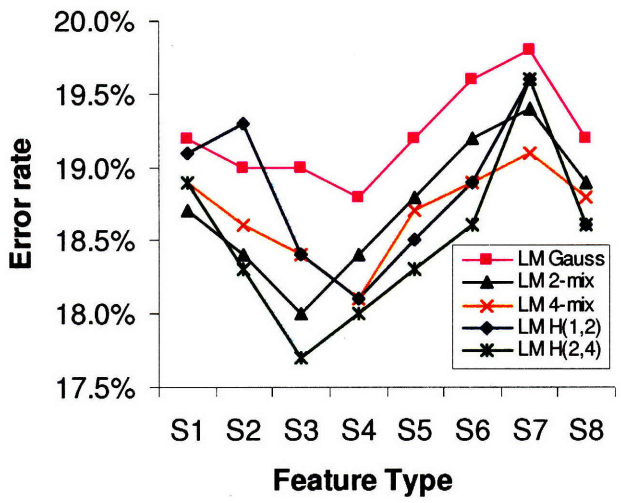

Figure 2-5: Error rates of large-margin GMM classifiers on Dev set.

Set	Gauss	2-mix	4-mix	H(1,2)	H(2,4)
Dev	17.0%	16.2%	16.1%	16.5%	**15.9%**
Test	17.8%	17.1%	17.1%	17.2%	**16.8%**

Table 2.11: Error rates of committee classifiers.

2.4.4 Committee-based Classifiers

This section presents the experimental results of committee-based classifiers. The independent assumption in [14] was utilized, and the committee-based classifier combined the outputs of the individual classifiers trained from S1-S8 by summing their log probabilities. The performances of the committee-based classifier was also affected by the margin scaling factor α. Figure 2-7 shows the performances of the committee-based classifiers on the development set under different value of α. The detailed performances of the committee-based classifiers on the development and on the core test set (using $\alpha = 0.1$) are list in Table 2.11. As in the earlier large-margin experiments, the "H(2,4)" model yields the best result.

An interesting observation is that for all five types of classifiers, the optimal explored value of α for an individual classifier did not result in the best committee classifier. As shown in Figure 2-7 the optimal value of α for the committee-based classifiers tended to be

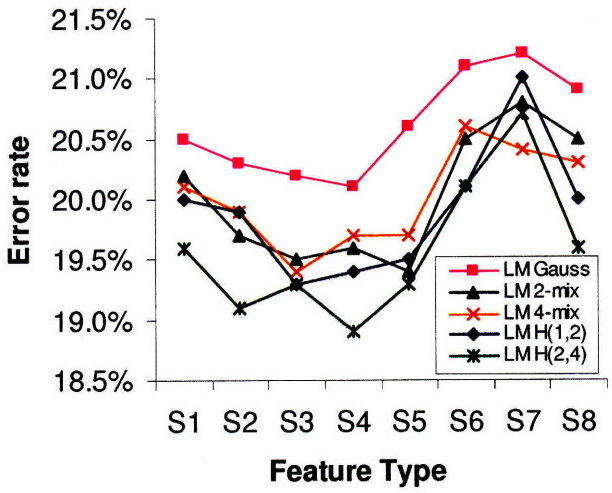

Figure 2-6: Error rates of large-margin GMM classifiers on Core Test set.

consistently slightly larger than that of individual classifiers. One possible explanation for this observation could be that the diversity of the individual classifiers may tend to decrease as α becomes smaller. Because a small α results in stricter margin constraints, the number of margin violations tends to increase. The increase of margin violations would increase the overlap of the training examples used by the large margin training on different types of features. As a result, although each individual classifier becomes more accurate, they become less complementary of each other and thus the overall committee was not as effective as the one with a set of more diverse but reasonably accurate committee members.

2.4.5 Heuristic Selection of Margin Scaling Factor

In the previous experiments, the optimal value of margin scaling factor α was found by using a brute force search on the development set. Although this way was effective, it was also time consuming in that the large-margin model has to be trained before it can be evaluated. It would be much preferable if a suitable scaling factor can be found before training the model, since the large-margin training takes a considerable amount of time.

One possible heuristic approach to select α was inspired from the observation that, for a training token with positive loss, the value of the loss has a convex shape variation according

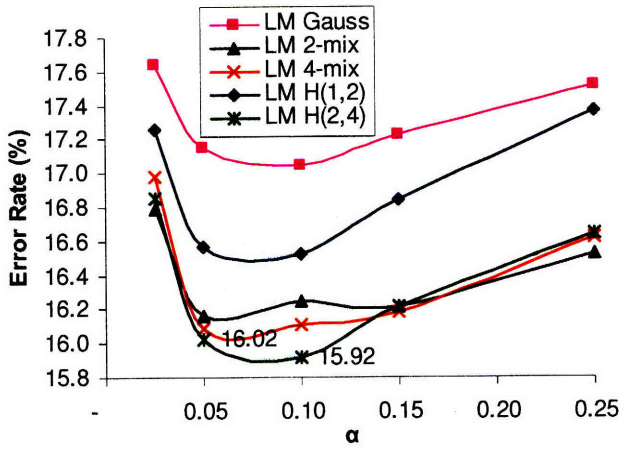

Figure 2-7: Error rates of committee-based classifier on Dev set under different margin scaling factor.

to α. To explain this, consider the case where the n^{th} training example has a positive loss. Such training examples are called "effective" in that only these examples would be considered in the large-margin training. For convenience, the expression of token loss l_n is shortened by $l_n = \sum_{c \neq y_n} [1 + \alpha \Delta_c]_+$.

A small α can have a two-sided effect on l_n. On the one hand, for a strong competing class c_1 with $\Delta_{c_1} > 0$, a small α can make $[1 + \alpha \Delta_{c_1}]_+$ small, and in this sense, may decrease the loss value. On the other hand, a small α may also make a weak competing class c_2 with $\Delta_{c_2} \ll 0$ contribute to the loss by making $1 + \alpha \Delta_{c_2} > 0$, and in this sense, may increase l_n. Because the loss is piece-wise linear to α, there exists an $\hat{\alpha}_n$ such that l_n can be minimized.

The value $\hat{\alpha}_n$ can potentially be a good choice for this training example, because it balances two effects: trying to increase the margin as much as possible (or, equally, choosing α as small as possible) while not letting too many weak competing classes disrupt the training. Following this idea, if a value $\hat{\alpha}$ is picked such that the average loss of all the "effective" examples is minimized, the value may also be a suitable choice of α for the whole training set.

The heuristic approach above was applied to select the α value for "LM H(1,2)" and "LM H(2,4)". The performances of the resulting models for all eight feature sets are listed

	LM H(1,2)			LM H(2,4)		
	dev	test	α	dev	test	α
S1	18.9%	20.0%	0.070	19.1%	20.4%	0.070
S2	18.9%	19.8%	0.070	18.3%	19.4%	0.072
S3	18.3%	19.6%	0.063	17.9%	19.7%	0.069
S4	18.3%	19.4%	0.067	18.1%	**18.7%**	0.069
S5	18.7%	19.7%	0.064	18.6%	19.6%	0.062
S6	19.1%	20.2%	0.068	18.7%	20.4%	0.069
S7	19.6%	21.0%	0.076	19.2%	20.6%	0.071
S8	18.7%	19.9%	0.067	18.7%	19.8%	0.071
avg	18.8%	20.0%		18.6%	19.8%	
com	16.6%	17.2%		16.0%	**16.7%**	

Table 2.12: Error rates of classifiers with pre-determined α.

in Table 2.12. The "com" in the table refers to committee classifier. Both the two sets of models have performances close to the models with $\alpha = 0.05$, demonstrating the effectiveness of the heuristic method. As shown in the table, the best overall result for a classifier trained from a single feature set reaches 18.7% error rate, while the best overall committee-based classifier obtains an error rate of 16.7%.

2.5 Discussion

In this chapter, the large-margin training framework for phonetic classification in [33] was introduced, and a training approach that combined the large-margin training criterion with hierarchical GMM classifiers was proposed. The proposed hierarchical large-margin GMM classifiers were evaluated on the benchmark task of TIMIT phonetic-classification. Based on the experiment results, several observations are discussed in the following paragraphs.

The first observation is the relation of training errors and the test errors of the large-margin models. Although not shown explicitly, the training errors of the large-margin models trained from different feature sets had similar variation patterns with respect to the margin scaling factor α: the models trained under $\alpha = 0.1$, 0.15, and 0.25 have low and similar training error rates; the models trained under $\alpha = 0.05$ have a little higher training error

rates; and the models trained under $\alpha = 0.025$ have high training error rates. By comparing the patterns of training errors with that of test errors, several properties of large-margin training can be revealed:

1. Under similar training error rates, using larger margin (smaller α) for training can help increase the generalization ability of the trained model.

2. Sacrificing some loss in training errors according to margin constraints may help improve the generalization ability of the model since some types of errors in the training may not be generalizable to other data sets.

3. If the margin value is set too large, the resulting model will become very poor since caring too much for the margin may distort the decision boundaries of the model.

These properties also demonstrate the importance of selecting an approptiate margin value for large-margin training.

The second observation is about the effect of integrating the large-margin training with hierarchical classifiers. As shown in Figure 2-6, for seven of the eight different feature sets, the "LM H(2,4)" model has better performance than "LM 4-mix" and uses fewer number of parameters. This fact reveals an advantage of using a hierarchy. Because the cluster-level nodes get the training examples of its children nodes, the parameter estimation of cluster-level nodes can be more robust and more generalizable to unseen data. Such an effect may become more salient when context-dependent acoustic modeling is applied since some context-dependency of phones may be very rare and have very limited amount of training data. Using a hierarchy can cluster the context-dependent labels with less training data together and provide a more stable parameter estimation.

In sum, the phonetic classification experiments presented in this chapter have shown that choosing an appropriate margin value is important to large margin training and that combining multi-level classification with large-margin training has a great potential to improve acoustic model performance. By integrating multi-level classifiers with large margin

training, the proposed modeling scheme obtained the best known result on the benchmark task of TIMIT phonetic classification.

Chapter 3

Large-Margin GMMs for LVCSR

This chapter presents the efforts of expanding the large-margin training framework to a large vocabulary continuous speech recognition (LVCSR) task. Practical issues of expanding large-margin training are first discussed, and then the environment of the LVCSR experiments is introduced. Preliminary experimental results of large-margin training on the MIT lecture corpus [8] are reported, and the model performances are compared with that of the model trained under MCE criterion. Discussion about the experiment results are presented at the end of the chapter.

3.1 Issues of Expending to LVCSR

In this section, practical issues of expanding the large-margin training framework to the task of LVCSR are discussed. The issues include possible modification of the loss function, diagonalization of GMMs, convexity of the loss function, and parallelization of computation. Details of each issue is described in the following subsections.

3.1.1 Loss Function

While the token-level loss function used in the previous chapter serves well in the TIMIT classification task, it may need some modification before being used as a training objective

on LVCSR tasks. This is because the token-level loss function has two intrinsic properties:

1. The loss of each token is assumed to be independent.

2. Each class can be a potential competitor against any other label at any given time.

These two properties, however, generally do not hold for most ASR systems designed for LVCSR tasks. This is because ASR systems for LVCSR tasks, in general, use context-dependent acoustic models, making errors and possible competitors in nearby time points dependent. Also, the lexicons and language models used in ASR systems also help to constrain the sets of possible confusions that can appear in the search, strengthening the dependencies between nearby confusions. As a result, modification of the loss function is needed to better fit the nature of ASR systems for LVCRS tasks. There are several possible ways to modify the loss function, one of which is to expand the token-level loss to string-level as in [34].

Let $\mathbf{X}_n$ be the observation sequence of the n^{th} utterance in the training data, and $\mathbf{Y}_n$ be the corresponding label sequence. The distance metric of $\mathbf{X}_n$ with respect to $\mathbf{Y}_n$ can be computed by summing up the token-level distance metrics in the string as follows:

$$D(\mathbf{X}_n, \mathbf{Y}_n) = \sum_{t=1}^{T_n} d(\mathbf{x}_{n_t}, y_{n_t}), \tag{3.1}$$

where $\mathbf{x}_{n_t}$ is the t^{th} observation vector in $\mathbf{X}_n$, y_{n_t} is the t^{th} token label in $\mathbf{Y}_n$, T_n is the total number of tokens in the string, and $d(\mathbf{x}_{n_t}, y_{n_t})$ is the token-level distance metrics as in Equation (2.2).

The string-level loss of the utterance can then be then expressed by

$$L_n = [D(\mathbf{X}_n, \mathbf{Y}_n) + \log(\sum_{\mathbf{S} \neq \mathbf{Y}_n,\ \mathbf{S} \in \boldsymbol{\mathcal{S}}_n} \exp(\xi H(\mathbf{Y_n}, \mathbf{S}) - D(\mathbf{X}_n, \mathbf{S}))]_+, \tag{3.2}$$

where $D(\mathbf{X}_n, \mathbf{S})$ is the distance metric of $\mathbf{X}_n$ with respect to hypothesis $\mathbf{S}$, $H(\mathbf{Y}_n, \mathbf{S})$ is the edit distance between the correct label sequence $\mathbf{Y}_n$ and hypothesis $\mathbf{S}$, and $\boldsymbol{\mathcal{S}}_n$ is the set of N-best hypothesis generated by ASR systems. In this way, the loss function of the

large-margin training can be expressed by

$$\mathcal{L} = \sum_{n=1}^{N} w_n L_n, \tag{3.3}$$

where w_n is the weight to prevent outliers, as mentioned in the previous chapter.

By generalizing the margin constraints to the string-level, the model can jointly update more parameters, and the optimization procedure can become more flexible. Also, the log-sum term in Equation (3.2) can help decide the importance of the hypothesis in the N-best list. When two hypotheses have similar distance metrics, the log-sum term will make the optimization focus more on the hypothesis with the larger edit distance. Significant error reduction on TIMIT phonetic-recognition was reported in [34] by switching from the token-level loss function to the string-level loss function.

3.1.2 Diagonalization of GMMs

In the experiments in [33], [34], and chapter 2, full-covariance GMMs were used. Full-covariance GMMs, can model the relation between different dimensions of the features, and in theory can better describe the data than diagonal GMMs. On the other hand, full-covariance GMMs need more training data than diagonal GMMs to have robust parameter estimation. In general, full-covariance GMMs can perform very well if the amount of training data is sufficient and is roughly balanced for each label.

In the case of LVCSR, however, the amounts of training data for each acoustic label are usually very imbalanced: some labels such as silence may have millions of training examples while some labels may only have very few (< 50) training examples. In such imbalanced data scenarios, it is necessary to conduct extra smoothing for the model parameters of the full-covariance GMMs that have few training examples. To avoid the additional work of smoothing, most existing ASR systems for LVCSR use diagonal GMMs. As a result, most existing discriminative training algorithms for LVCSR tasks are implemented on diagonal GMMs. To make a fair comparison with other existing implementation of discriminative

training methods, the large-margin GMMs used in the experiments reported in this chapter are also diagonal. In this way, the factor of using different types of models is removed, and the comparison can be more focus on the difference between the training objective functions of different discriminative training methods.

3.1.3 Convexity of Loss Function

The relaxation of the loss in Equation (2.3) into the form in Equation (2.10) makes the loss function of large-margin training convex with respect to the parameter matrices in Equation (2.7). Such relaxation is based on the assumption that the log-likelihood, $\log(p(\mathbf{x}_n, y_n))$, of the observation vector $\mathbf{x}_n$ is mainly contributed by the log-likelihood of one single Gaussian mixture component, $\rho(\mathbf{x}_n, y_n, m_n)$. This assumption generally holds if the number of Gaussian mixture components for each label is not too many. However, in the case of LVCSR, such an assumption may not necessarily hold. This is because the amount of training data for certain acoustic labels in LVCSR can be large, and to model the variation of such an amount of training data, a great number of mixture components are needed. Instead of using the likelihood of a single mixture component to approximate the likelihood of an entire GMM, another convex approximation may be needed in the case where the model has many mixture components.

One possible approximation of the log-likelihood is as follows: Let $d_n(m)$ be the distance metric for the m^{th} mixture component of model y_n; the distance metric of the model can be computed by $d(\mathbf{x}_n, y_n) = -\log \sum_m \exp(-d_n(m))$. Although the distance metric $d(\mathbf{x}_n, y_n)$ is a concave function for all the $d_n(m)$, it can be upper-bounded by a convex function; that is,

$$d(\mathbf{x}_n, y_n) \leq \sum_{m=1} \gamma_n^{ML}(m) d_n(m) + \sum_{m=1} \gamma_n^{ML}(m) \log(\gamma_n^{ML}(m)), \tag{3.4}$$

where $\gamma_n^{ML}(m)$ is the posterior of the mixture component m based on the initial parameters trained by the Maximum-Likelihood criterion. Because all the values of $\gamma_n^{ML}(m)$ are computed by the initial ML model and would not change with large-margin training, the

right hand side of the inequity is a linear combination of $d_n(m)$ (plus some shift) and thus is convex with respect to all $d_n(m)$. Note that the inequality above becomes equality if $d_n(m)$ is computed by the initial ML parameters.

By using the upper-bound in Equation (3.4) to replace $d(\mathbf{x}_n, y_n)$ in Equation (2.3), the loss function can still be convex, and convex optimization algorithms can still be applied to refine the searching of parameters. However, due to the time constraint, although the convex relaxation in Equation (3.4) was implemented in the LVCSR experiments, the model parameters were only updated by Quickprop, without using other convex optimization algorithms to refine the parameters.

3.1.4 Parallelization of Computation

For LVCSR tasks, it is typical that the training corpus contains more than 100 hours of speech data. To conduct discriminative training on such an amount of data requires a large amount of computation, and appropriate parallelization of the computation is crucial to the success of discriminative training on LVCSR tasks.

In general, the computations for the forced alignments and recognition results (lattices or N-best lists) of training utterances can be distributed to multiple machines and the outputs can be stored in separated files. The gradients or other statistics required for the training can then be computed by accumulating the contributions of each training utterance based on the output files. Such an accumulation process can also be parallelized by first distributing the accumulation to multiple machines and using a single machine to merge the results. Note that if a sophisticated optimization algorithm is used for the training, it is critical to implement the parallelization in a way that the state of the optimization module can be recorded before distributing the computation to multiple machines.

3.2 Experimental Environment

In the experiments reported in this chapter, speech recognition was performed using the SUMMIT landmark-based speech recognizer, and acoustic models were trained and evaluated on the MIT Lecture Corpus. The following subsections briefly describe the speech recognizer and the corpus used to conduct the experiments.

3.2.1 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 nearly 100 MITWorld seminars given on a variety of topics [26]. The audio data were recorded with omni-directional microphones and were generally recorded in a classroom environment. The recordings were manually transcribed in a way that disfluencies such as filled pauses and false starts are kept. In addition to the spoken words, the following annotations are also included in the transcriptions: (1) occasional time makers at obvious pauses or sentence boundaries, (2) locations of speaker changes (labeled with speaker identities if known), (3) punctuation based on the transcribers' subjective assessment of spoken utterances [8].

The lecture corpus is a difficult data set for ASR systems for the following reasons. (1) The data recorded are spontaneous speech. The lecture speech contains many disfluencies such as filled pause, false start, and partial words. In addition, the spontaneous nature also results in less formal sentences, and poorly organized or ungrammatical sentences can be frequently observed.(2) The lectures contain many lecture-specific words that are uncommon to daily life. The lecture-specific words can result in serious Out-Of-Vocabulary (OOV) problems. Even using the 10K most frequent words in switchboard as the vocabulary, the OOV rate is still around 10% [27]. (3) The speech data potentially suffer from reverberation, coughing, laughter, or background noise such as students' talking. The above reasons make a robust recognizer for lecture corpus difficult to construct.

In the experiments reported in this chapter, the acoustic models were created from a training set of about 119 hours that includes 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 lectures series with a variety of topics. Two held-out MITWorld lectures, Thomas Friedman's "The World is Flat" lecture (MIT-World-2005-TF) and Thomas Leighton's lecture about Akamai startup (MIT-World-2004-TL), are used as a development set to select the discriminatively trained models. A test set that 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) are used to evaluate the trained models. Note that there are no speaker overlaps between the training lectures and the test lectures. The sizes of training, development, and test sets are listed in Table 3.1, 3.2, and 3.3 respectively.

3.2.2 SUMMIT landmark-based speech recognizer

In conventional hidden Markov model (HMM) based ASR systems, the speech signal is chopped into fixed-length frames, and for each frame, acoustic measurements for the speech signal within the frame are computed to form a feature vector for the frame. Instead of extracting feature vectors at a constant frame-rate, the SUMMIT landmark-based speech recognizer first compute a set of perceptually important time points as landmarks based on an acoustic difference measure, and extracts a feature vector for each landmark. In this way, SUMMIT tends to be more focused on time-points where the acoustic nature of speech changes.

SUMMIT uses Finite-State Transducers (FSTs) to represent all constraints and conduct search in recognition. By utilizing FSTs, the language model and the lexicon related information, including context dependent state mapping, pronunciation rules, and reduction rules can all be modularized. The search module for the recognizer can then be easily constructed by composing the modules using efficient FST composition algorithm. In addition, by modularizing using FSTs, the recognizer can also be easily adapted to new constraints such as additional pronunciation rules. More details about the FST implementation can be seen in [15].

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	104	118.76

Table 3.1: 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 3.2: 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	2	1.41
Total	8	8.11

Table 3.3: Sizes of lectures in the test set.

The following paragraphs describe the setup of SUMMIT for the Lecture Corpus, including the feature extraction, baseline acoustic model, and language model.

Landmark Features

The landmark features used for SUMMIT in the experiments are computed as follows: For each landmark selected by SUMMIT, 8 telescoping regions, ranging from 75ms before the landmark to 75ms after the landmark, are used to extract acoustic measurements. Detailed specifications of the telescope regions are listed in Table 3.4. Within each telescoped region,

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 3.4: Specifications of telescope regions for landmark features. The minus sign "-" refers to before the landmark, while "+" sign refers to after the landmark.

a 25.6ms Hamming window is used to collect speech samples and the window shifts at a 5ms frame-rate. For each group of samples selected by the window, short time Fourier analysis is applied and the first 14 MFCCs corresponding to the samples are computed. The average values of MFCCs of the 8 telescoping regions are then concatenated into a 112 dimensional vector, and the 112 dimensional vector is reduced to 50 dimension by Principle Component Analysis (PCA), resulting a 50 dimensional feature vector per landmark.

Baseline Acoustic Model

There are 74 phonetic labels used in the baseline acoustic model for the lecture corpus. The 74 phonetic labels can be derived from the 61 labels used in TIMIT by making the following changes :

1. Remove 'ax-h', 'eng', 'hv', 'ix', 'nx', 'q', and 'ux' from the 61-label set.
2. Add 4 labels '_b1', '_b2', '_b3', and '_b4' for background.
3. Add 4 labels '_c1', '_c2', '_c3', and '_c4' for coughing.
4. Add 4 labels '_l1', '_l2', '_l3', and '_l4' for laughter.
5. Add 6 labels '_n1', '_n2', '_n3', '_n4', '_n5', and '_n6' for noise.
6. Add 'ah-fp' for filled pause.
7. Add '<>' for sentence boundary.
8. Replace 'h#' by '_' and 'pau' by '-'.

For each landmark, the left and right contexts of the landmark are modeled by a set of diphones. Because a landmark can either be a transitional point from one phone to another, e.g. t(bcl|b), or just an internal point of a phone, e.g. i(aa), there are (74*74)+(74−1) = 5549 possible di-phones (-1 is for i(<>)). The 5549 di-phones are then clustered by a decision tree, resulting in 1871 classes.

For each of the 1871 diphone classes, a diagonal GMM is used to model the training feature vectors for the diphone. The maximum number of mixture components for each GMM is set to 30, and the model is allowed to add one mixture component for every 50 training examples available. The GMM parameters are trained based on standard E-M training except that the mixture components are added one at a time by splitting the mixture component with largest variance when the log-likelihood converges, until the maximum allowed number of mixture components is reached.

In addition to the landmark models, log-duration is used as a feature to model the phone segments during recognition. Details about the segment-based modeling can be seen in [7]. Although both the landmark models and segment models are used for recognition, only the parameters in landmark models were updated in the experiments.

Language Model

The training data for the language model used in the experiments come from the transcripts of the following three sources: (1) The training lectures in Table 3.1, (2) Switchboard telephone conversations [9], (3) Michigan Corpus of Academic Spoken English. The SRILM toolkit [36] was used to compute the N-gram counts of these three text sources with the default setting of smoothing. The resulting N-gram counts were expressed by a language model FST, and the language model FST was composed with the other lexicon level FSTs to form the search module of the recognizer.

3.3 Experiments

In this section, preliminary experimental results for large-margin GMMs on the MIT Lecture Corpus are presented and the performance of the large-margin models are compared with that of MCE trained models [25]. The training procedures of MCE and large-margin models are first described, followed by a performance comparison and discussion.

3.3.1 MCE Models

The MCE training in the experiments can be summarized by the following steps:

1. For each training utterance, generate the forced alignment of words and phone sequences based on the reference.

2. For each training utterance, generate the N-best hypotheses of the utterance.

3. Compute the gradient of the normalized MCE objective function using the outputs of step 1 and 2.

4. Update the model parameters using Quickprop (Appendix B) based on the gradient computed in step 3.

In the reported experiments, the number of hypotheses in the N-best list was set to 20, and the training terminated after 15 iterations of the above procedure. For the training parameters in the MCE objective function, the ζ in the sigmoid function was set to 10, and the value of ν was set to 1. For the parameters of Quickprop, the scale limit was set to 1.75, step limit was set to 10, and the learning rate ϵ was set to 0.2.

The Word Error Rate (WER) of the MCE model after each training iteration was computed on the two lectures in the development set, and the best performing model on the development set was used to compare with large-margin models on the test lectures.

3.3.2 Large-Margin Models

Selecting Margin Value

For the large-margin models, two different values of the margin ξ were used for training. The first value was simply 1, and the other value was selected by the following heuristic procedure:

1. For each training utterance, compute the normalized log-likelihood difference $d'_n = \frac{-\log(p(\mathbf{X}_n,\mathbf{Y}_n))+\max_{\mathbf{S}}\log(p(\mathbf{X}_n,\mathbf{S}))}{T_n}$, where T_n is the total number of landmarks in the utterance.

2. Compute the average $d^{\mathcal{P}} = \mathrm{E}[d'_n | d'_n \geq 0]$ and $\sigma^{\mathcal{P}} = \mathrm{E}[\sqrt{(d'_n - d^{\mathcal{P}})^2} | d'_n \geq 0]$ of all utterances with $d'_n \geq 0$.

3. Compute $d^{\mathcal{N}} = \mathrm{E}[d'_n | d'_n < 0]$ and $\sigma^{\mathcal{N}} = \mathrm{E}[\sqrt{(d'_n - d^{\mathcal{N}})^2} | d'_n < 0]$ in similar way.

4. For each utterance, compute the ratio $r_n = \frac{\Delta_n}{T_n}$, where Δ_n is the number of different diphones that appeared in the best competing hypothesis and the correct transcription.

5. Compute the average ratio $\hat{r} = \mathrm{E}[r_n]$.

6. Set margin $\hat{\xi} = \mathrm{abs}(d^{\mathcal{N}}) + \frac{\sigma^{\mathcal{P}}\sigma^{\mathcal{N}}}{\sigma^{\mathcal{P}}+\sigma^{\mathcal{N}}}$

The intuition behind the heuristic estimation of margin is as follows. Given a margin value ξ, if $d'_n < -\xi\frac{\Delta_n}{T_n} = -\xi r_n$, the large-margin criterion would consider the utterance to be recognized with strong confidence and would remove the utterance from further training. In this sense, the problem of choosing a margin value can be reduced to a problem of setting a threshold that determines whether an utterance has been recognized with sufficient confidence. The threshold selected in this heuristic approach is approximately $\mathrm{abs}(d^{\mathcal{N}})+\sigma^{\mathcal{N}}$; meaning that if an utterance is correctly recognized with a log-likelihood gap larger than the average log-likelihood gap of all utterances being correctly recognized by at least one standard deviation, the utterance should be considered confident enough. The reason why $\frac{\sigma^{\mathcal{P}}\sigma^{\mathcal{N}}}{\sigma^{\mathcal{P}}+\sigma^{\mathcal{N}}}$ is used instead of $\sigma^{\mathcal{N}}$ is to consider the thought that if $\sigma^{\mathcal{P}}$ is small, the performance

of the recognizer should be more reliable and the required log-likelihood gap can be set to a smaller value. By running the heuristic algorithm above, a heuristic margin value $\hat{\xi} = 1.13$ was selected.

Training

For the large-margin training in the experiments, the following loss function was used:

$$\mathcal{L} = \sum_{n=1}^{N} \frac{w_n}{\iota_n} \left[D(\mathbf{X}_n, \mathbf{Y_n}) + \log(\frac{1}{C} \sum_{\mathbf{S} \neq \mathbf{Y}_n}^{C} \exp(\xi H(\mathbf{Y}_n, \mathbf{S} - D(\mathbf{X}_n, \mathbf{S})))) \right]_+ \tag{3.5}$$

where w_n is the weight to handle outliers as mentioned before, ι is the number of diphones in the n^{th} utterance, and C is the size of the N-best list. In the experiments, the weight w_n is set to the same value as what the sigmoid function would give to the utterance when computing the gradient for MCE training; more specifically,

$$w_n = \zeta \ell(d_n)[1 - \ell(d_n)], \tag{3.6}$$

where $\ell(\cdot)$ is the sigmoid function, and d_n is the difference between the average log-likelihood of the competing hypotheses and the correct transcription of the n^{th} utterance. In addition, the size of N-best list is set to the same value as in the MCE training. By using the above specifications of the loss function, the initial scale of the gradients of the large-margin training and the MCE training can be kept the same, and thus the same setting of optimization parameters can be used for both of the two training schemes. As a result, the factor of using different optimization scheme is removed, and the comparison between the two types of models can focus more on whether introducing the large-margin constraints can help improve model performance.

As mentioned in section (3.1.3), the loss function in Equation (3.5) can be made convex by applying the relaxation as in Equation (3.4). Making the loss function convex can prevent the training being trapped by a bad local minimum. Also, utilizing the posteriors probabilities computed by the ML model as in Equation (3.4) can make the training become less

Models	ML	MCE	LM 1	LM h	LM ch
WER	45.7%	41.2%	41.2%	41.2%	41.0%

Table 3.5: Word error rates on the development set.

aggressive; potentially prevent the model from over-fitting the training data. The convex criterion was used to train one of the large-margin models in the experiments to see whether the convex relaxation can enhance the generalization ability of the model as expected.

In the experiments, three variants of the large-margin models, "LM 1", "LM h", and "LM ch", were trained. "LM 1" denotes the large-margin model trained with $\xi = 1$, while "LM h" denotes the model trained with heuristic setting of ξ (1.13). "LM ch" denotes the model trained with convex loss function and heuristic setting of ξ. For all the tree types of models, the gradients used in training were computed using a similar distributed computation scheme as described in the previous section, and the parameters were updated using the Quickprop algorithm with the same setting of optimization parameters as in the MCE training. The 2-lecture development set was used to selected the models across iterations, and the best performing models on the development set were used to in the performance comparison in the next section.

3.3.3 Comparisons and Discussion

This section compares the model performance between the baseline ML model, MCE model, and large-margin models. Table 3.5 lists the word error rates (WERs) of the models on the development. As shown in Table 3.5, the four discriminatively trained models yielded significant improvement over the baseline ML model. Also, "LM ch" yielded slightly better performance than other models, showing that the convex relaxation can potentially enhance the generalization ability of the model.

Table 3.6 lists the WERs of the models on the test lectures. As shown in the table, all the four types of discriminatively trained models yielded 5-6% absolute WER reduction over the ML baseline on all the four kinds of lectures in the test set, showing the effectiveness

Models	18.03-2004	6.345-2001	6.046-2005	7.012-2004	Total	Relative Reduction
ML	42.3%	34.5%	42.7%	33.6%	38.2%	-
MCE	36.8%	28.5%	37.6%	28.9%	32.8%	14.2%
LM 1	37.0%	28.4%	37.5%	28.3%	32.6%	14.5%
LM h	37.0%	28.0%	37.2%	28.7%	32.5%	14.9%
LM ch	35.9%	27.9%	36.9%	28.8%	32.2%	15.7%

Table 3.6: Word error rates on test lectures.

of the discriminative training methods. While "MCE", "LM 1", and "LM h" have similar WER on the development set, the two large-margin models performed better than the MCE model on most of the test lectures. This fact confirms the prediction that the large-margin models tend to have better generalization ability to unseen data. Among all the large-margin models, "LM ch" has the best performance on the development set and the test set, showing that the convex relaxation in Equation (3.4) can help enhance the generalization ability of the large-margin model.

To see whether there were significant differences between the models, McNemar significance tests [6] were conducted based on the WER of the models. The p-vlues of the significance tests were summarized in Table 3.7. The p-values for the tests between the baseline ML model and all the other models were very small, showing that the discriminatively trained models significantly are different from the baseline model. The p-value for the test between "MCE" and "LM h" was less than the 0.01 threshold, showing that the difference between the two models was at least marginally significant. The p-value between "MCE" and "LM ch" was smaller than 0.001 threshold, showing that the two models were significant different. The results of significance test also confirm that the convex relaxation helps to enhance the generalization ability of the model.

In the experiments reported in [34], the large-margin training had a large improvement over the MCE training on the task of TIMIT phonetic recognition, while in the experiments on the lecture corpus, the gain of large-margin training was not as large. There are several possible reasons for this.

	Baseline	MCE	LM 1	LM h
MCE	<0.001	-	-	-
LM 1	<0.001	0.066	-	-
LM h	<0.001	0.008	0.124	-
LM ch	<0.001	<0.001	<0.001	0.008

Table 3.7: The p-values of McNemar significance tests of the models on WER.

In the training in [34], the objective function for large-margin training was kept convex, and an additional convex optimization algorithm was used to refine the search for parameters [32]. Although the loss function for "LM ch" was convex, the parameters were updated based on Quickprop and no further refinement of the parameters was performed. It would be worthwhile to investigate whether applying an additional convex optimization algorithm to refine the search can provide additional performance gain.

Another possible reason is that the way that the edit distance between the reference and the hypothesis string is computed is not the best. In the current implementation, the edit distance is counted by the number of different diphone labels between the correct string and the hypothesis. Although such an edit distance is easy to compute, it may not be directly related to WER, since many diphone label differences may result in only one word error. A more appropriate kind of edit distance may be the number of phone errors or number of word errors of the hypothesis. Such a kind of distance can be computed without aligning the hypothesis with the reference by using similar approximations as in MPE training.

In sum, the preliminary experiments on the lecture corpus have shown that large-margin training can result in models potentially more generalizable to unseen data than MCE training. While the current implementation of large-margin training had marginal performance gain over MCE training, several possible modifications that can potentially increase the gain have been discussed. To investigate how much performance gain can be achieved by such modifications are left as future work.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

In this work, a large-margin training criterion for ASR systems has been investigated. In chapter 2, possible ways of integrating large-margin training integrated with a multi-level classification framework were studied, and experimental results on the TIMIT phonetic classification have shown that the integration has great potential to improve the acoustic model performance. In chapter 3, the large-margin training is expended to LVCSR tasks, and the preliminary experimental results showed that the large-margin training can potentially make the model more generalizable to unseen data than MCE training. Possible modifications to improve the large-margin training were also discussed.

4.2 Future Work

In the future, research to further improve the large-margin training for LVCSR tasks will be studied. Several possible research directions are: (1) Applying convex optimization algorithm to refine the model parameters, (2) changing the way of computing string distance, (3) constructing a hierarchy for diphones, and (4) utilizing lattice information. The following paragraphs briefly summarize these possible modifications of the large-margin training.

4.2.1 Applying Convex Optimization to Refine Parameters

In the experiments in [34], additional convex optimization algorithm was used to refine the parameters of the large-margin models. In the experiments reported in Chapter 3, however, no further refinement of the parameters were performed after the Quickprop update. It would be worthwhile to investigate whether applying additional convex optimization algorithm to refine the search can provide additional performance gain.

4.2.2 Changing the Way of Computing String Distance

One key objective of large-margin training is to have a margin between the correct string and the incorrect string that is proportional to the string distance. As a result, how the string distance is measured affects the importance of the hypothesis in the training. In the current implementation of large-margin training for LVCSR, the string distance between the correct and incorrect hypotheses is measured by the number of different diphone labels between the two strings. However, the string distance can also be measured in terms of phone errors or word errors of the incorrect hypothesis. Part of future research effort could focus on investigating which kind of string distances are most effective for large-margin training.

4.2.3 Constructing a Hierarchy for Diphones

In the phonetic classification experiments in chapter 2, the hierarchical classifiers have been shown to have better performance than single level classifiers. Introducing a similar hierarchical framework to acoustic models for LVCSR tasks may also have potential performance gains. The current diphone models used in the lecture tasks were formed by the leaf-nodes of the decision tree that were used to cluster the set of possible diphone labels. Instead of just modeling the leaf-nodes, jointly modeling the higher parents nodes and the leaf-nodes in the decision tree can naturally form a hierarchical classifier. Investigating the ASR performance of such hierarchical diphone classifiers can also be an interesting topic of future research.

4.2.4 Utilizing Lattices

In the previous experiments, the large-margin training updated the parameters based on the correct transcription and the N-best list generated by the recognizer. Instead of using N-best lists, utilizing lattices for large-margin training can also potentially improve the model performance. One possible way of utilizing lattices for large-margin training is to use lattices to compute the expected log-likelihood and string distance of hypotheses and incorporate these two quantities into the large-margin training criterion. More specifically, given an observation sequence $\mathbf{X}_n$, transcription $\mathbf{Y}_n$, and acoustic parameter set $\boldsymbol{\lambda}$, the loss l_n can be computed by

$$l_n = [-\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{Y}_n)p_L(\mathbf{Y}_n)) + \mathrm{E}_{\mathbf{S}}[\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))] + \mathrm{E}_{\mathbf{S}}[\mathcal{D}(\mathbf{S},\mathbf{Y}_n)]]_+ , \qquad (4.1)$$

where $p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})$ and $p_L(\mathbf{S})$ denote the acoustic model probability of hypothesis $\mathbf{S}$ and the language model probability, respectively, as in previous chapters, $\mathrm{E}_{\mathbf{S}}[\cdot]$ denotes taking an expectation over all hypotheses, and $\mathcal{D}(\mathbf{S},\mathbf{Y}_n)$ denotes the string distance between $\mathbf{S}$ and $\mathbf{Y}_n$.

With the help of lattices, the loss l_n in Equation (4.1) and its gradient with respect to $\boldsymbol{\lambda}$ can be compute efficiently by running forward-backward algorithms on the phone arcs in the lattices. Details of the mathematical derivations of the lattice-based computation can be seen in Appendix D. By incorporating lattices into the training, more competitors can be considered in the training which can potentially enhance the performances of the large-margin training.

Appendix A

Optimization for MMI Training

A.1 Auxiliary Function for MMI Training

This section describes how to construct an appropriate auxiliary function for MMI training. The basic ideas of derivation came from [28]. Auxiliary functions are frequently used tools for optimization, especially when direct optimization of the original objective function is difficult. The idea of how auxiliary function works is illustrated in Figure A-1. The optimization starts with $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$, and an auxiliary function $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is constructed. By maximizing with respect to $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, $\boldsymbol{\lambda}$ is updated to $\hat{\boldsymbol{\lambda}}$, and the value of objective function $F(\boldsymbol{\lambda})$ increases. The update procedure continues until $F(\boldsymbol{\lambda})$ converges to a local maximum.

A smooth (continuous in first order differential) function of $\boldsymbol{\lambda}$, $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, is said to be a strong-sense auxiliary function [28] for $F(\boldsymbol{\lambda})$ around $\boldsymbol{\lambda}'$ iff

$$G(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - G(\boldsymbol{\lambda}', \boldsymbol{\lambda}') \leq F(\boldsymbol{\lambda}) - F(\boldsymbol{\lambda}'), \tag{A.1}$$

whereas $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is a weak-sense auxiliary function for $F(\boldsymbol{\lambda})$ around $\boldsymbol{\lambda}'$ if

$$\frac{\partial}{\partial \boldsymbol{\lambda}} G(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \mid_{\boldsymbol{\lambda}=\boldsymbol{\lambda}'} = \frac{\partial}{\partial \boldsymbol{\lambda}} F(\boldsymbol{\lambda}) \mid_{\boldsymbol{\lambda}=\boldsymbol{\lambda}'} . \tag{A.2}$$

Note that a function $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ can be both a strong-sense and a weak-sense auxiliary function

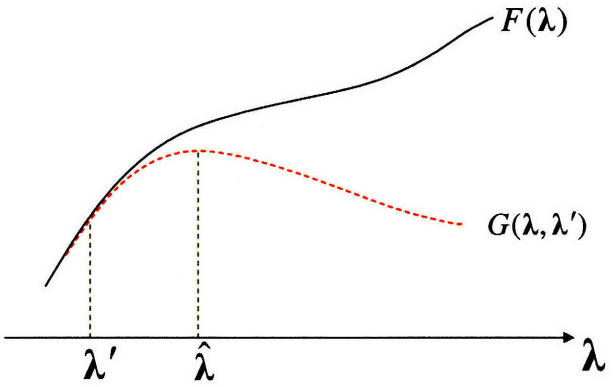

Figure A-1: Illustration of an auxiliary function. The solid curve is the function $F(\boldsymbol{\lambda})$ to be maximized, and the red dashed curve is the auxiliary function $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. By maximizing with respect to $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. $\boldsymbol{\lambda}$ is updated to $\hat{\boldsymbol{\lambda}}$, and the value of the objective function $F(\boldsymbol{\lambda})$ increases. The update procedure continues until $F(\boldsymbol{\lambda})$ converges to a local maxima.

of $F(\boldsymbol{\lambda})$ at the same time. The expected log-likelihood used in the Expectation-Maximization update of ML training is an example of such a case. For a strong sense auxiliary function $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, because the difference $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - G(\boldsymbol{\lambda}', \boldsymbol{\lambda}')$ is always less than or equal to $F(\boldsymbol{\lambda}) - F(\boldsymbol{\lambda}')$, the value of the objective function is guaranteed to increase if the new parameter set $\boldsymbol{\lambda}$ is chosen such that $G(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ can be maximized. For a weak-sense auxiliary function, however, this property is not guaranteed to hold. Although using a weak-sense auxiliary function to update the parameters can not guarantee increasing the objective function as using a strong-sense auxiliary function would, if the update converges, the function $F(\boldsymbol{\lambda})$ is guaranteed to be at a local maximum.

While a tractable strong-sense auxiliary function for MMI objective function in Equation (1.1) is difficult to construct (even not known to exist), a weak-sense auxiliary function can be constructed by the following steps. First, separate the numerator and denominator in the log term of Equation (1.1) into two terms:

$$\begin{aligned} F_{MMI}(\boldsymbol{\lambda}) &= \sum_{n=1}^{N} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{Y}_n)^{\kappa} p_L(\mathbf{Y}_n)^{\kappa}) - \sum_{n=1}^{N} \log(\sum_{\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa}) \\ &= F_{MMI}^{num}(\boldsymbol{\lambda}) - F_{MMI}^{den}(\boldsymbol{\lambda}). \end{aligned} \tag{A.3}$$

Second, given the previous HMM parameter set $\boldsymbol{\lambda}'$, construct weak-sense auxiliary functions

$G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ and $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ for $F^{num}_{MMI}(\boldsymbol{\lambda})$ and $F^{den}_{MMI}(\boldsymbol{\lambda})$ around $\boldsymbol{\lambda}'$ respectively. Note that in this way $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ is also a weak-sense auxiliary function for $F_{MMI}(\boldsymbol{\lambda})$ around $\boldsymbol{\lambda}'$. Third, add an appropriate smoothing function of $\boldsymbol{\lambda}$ that has a maximum at $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$ to the difference $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ such that the overall auxiliary function can be a concave down function as shown in Figure A-1. In this way, the auxiliary function for $F_{MMI}(\boldsymbol{\lambda})$ around $\boldsymbol{\lambda}'$ can be expressed in the following form:

$$G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') + G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}'). \tag{A.4}$$

Because the $F^{num}_{MMI}(\boldsymbol{\lambda})$ in Equation (A.3) is the same as the objective function of ML training, the expected log-likelihood used in ML training can be directly used as $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. More specifically, let

$$G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \sum_{n=1}^{N} \sum_{\boldsymbol{\varkappa} \in \mathfrak{M}(\mathbf{Y_n})} \gamma^{num}_{\boldsymbol{\varkappa}} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \boldsymbol{\varkappa}|\mathbf{Y}_n)^{\kappa} p_L(\mathbf{Y}_n)^{\kappa}), \tag{A.5}$$

where $\boldsymbol{\varkappa}$ denotes the state-mixture sequence of a HMM that lists the state and mixture assignment at each time frame, $\mathfrak{M}(\mathbf{Y}_n)$ is the set of possible HMM state-mixture sequences that can generate $\mathbf{Y}_n$, $\gamma^{num}_{\boldsymbol{\varkappa}}$ is the posterior probability of the state-mixture sequence $\boldsymbol{\varkappa}$ being generated by the previous HMM parameter set $\boldsymbol{\lambda}'$, and $p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \boldsymbol{\varkappa}|\mathbf{Y}_n)^{\kappa}$ is the scaled probability of the observation sequence $\mathbf{X}_n$ and the state-mixture sequence $\boldsymbol{\varkappa}$ being generated by the HMM parameter set $\boldsymbol{\lambda}$ given transcription $\mathbf{Y}_n$. Similarly, the auxiliary function $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ can be set by

$$G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \sum_{n=1}^{N} \sum_{\mathbf{S}} \sum_{\boldsymbol{\varkappa} \in \mathfrak{M}(\mathbf{S})} \gamma^{den}_{\boldsymbol{\varkappa}} \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n, \boldsymbol{\varkappa}|\mathbf{S})^{\kappa} p_L(\mathbf{S})^{\kappa}). \tag{A.6}$$

Given the auxiliary function $G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, the parameters are updated according to the following criterion:

$$\hat{\boldsymbol{\lambda}} = \arg_{\boldsymbol{\lambda}} \frac{\partial}{\partial \boldsymbol{\lambda}} G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \mathbf{0}. \tag{A.7}$$

A.2 Parameter Update

This section describes how to maximize the auxiliary function in Equation (A.4) and how to update the parameters for MMI training in more detail. Maximization of auxiliary function $G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ in Equation (A.4) requires computing the partial derivative $\frac{\partial}{\partial \boldsymbol{\lambda}} G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. The parameters are then updated by solving $\frac{\partial}{\partial \boldsymbol{\lambda}} G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \mathbf{0}$.

Because there are exponentially many $\gamma_{\mathscr{s}}$ in Equation (A.5) and (A.6), it is necessary to translate the two functions into forms that are much easier to analyze. Knowing that the observations become independent given the state assignments of HMM, the summation over all possible state-mixture sequences in Equation (A.5) and (A.6) can be broken down into a summation over individual states and mixtures. More specifically, the function $G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ can be expressed using the numerator-lattices and corresponding statistics in section 1.2.1:

$$G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \sum_{j} \sum_{m=1}^{M_j} \sum_{q=1}^{Q^{num}} \sum_{t=s_q}^{e_q} \kappa \gamma_{jmq}^{num}(t) \gamma_q^{num} \log(p(\mathbf{x}_t | \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})) + K_L^{num}, \tag{A.8}$$

where M_j is the total number of mixtures in state j, $p(\mathbf{x}_t | \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$ is the Gaussian probability of observation vector $\mathbf{x}_t$ given mean vector $\boldsymbol{\mu}_{jm}$ and $\boldsymbol{\Sigma}_{jm}$, and K_L^{num} is the summation of language model scores that are not related to HMM parameters. Similarly, $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ can also be expressed by

$$G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \sum_{j} \sum_{m=1}^{M_j} \sum_{q=1}^{Q^{den}} \sum_{t=s_q}^{e_q} \kappa \gamma_{jmq}^{den}(t) \gamma_q^{den} \log(p(\mathbf{x}_t | \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})) + K_L^{den}. \tag{A.9}$$

To compute the partial derivatives, let us first consider the log probability

$$\log(p(\mathbf{x}_t | \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})) = \frac{-1}{2}[(\mathbf{x}_t - \boldsymbol{\mu}_{jm})^{\mathrm{T}} \boldsymbol{\Sigma}_{jm}^{-1} (\mathbf{x}_t - \boldsymbol{\mu}_{jm}) + \log(\det(\boldsymbol{\Sigma}_{jm}))] + C, \tag{A.10}$$

where $\det(\boldsymbol{\Sigma}_{jm})$ is the determinant of $\boldsymbol{\Sigma}_{jm}$ and C is a normalization constant. Taking partial

derivative of Equation (A.10) with respect to $\boldsymbol{\mu}_{jm}$ results in

$$\frac{\partial}{\partial \boldsymbol{\mu}_{jm}} \log(p(\mathbf{x}_t|\boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})) = \frac{1}{2}[(\boldsymbol{\Sigma}_{jm}^{-1} + \boldsymbol{\Sigma}_{jm}^{-\mathsf{T}})(\mathbf{x}_t - \boldsymbol{\mu}_{jm})] = \boldsymbol{\Sigma}_{jm}^{-1}(\mathbf{x}_t - \boldsymbol{\mu}_{jm}). \tag{A.11}$$

Note that the only term related to $\boldsymbol{\mu}_{jm}$ in Equation (A.8) is $\log(p(\mathbf{x}_t|\boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}))$ and that $\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}) = \sum_{q=1}^{Q^{num}} \sum_{t=s_q}^{e_q} \gamma_{jmq}^{num}(t)\gamma_q^{num}\mathbf{x}_t$, the partial derivative

$$\frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \boldsymbol{\Sigma}_{jm}^{-1}(\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}) - \gamma_{jm}^{num}\boldsymbol{\mu}_{jm}). \tag{A.12}$$

Similarly, the partial derivative of $G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ can be computed by

$$\frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \boldsymbol{\Sigma}_{jm}^{-1}(\boldsymbol{\vartheta}_{jm}^{den}(\mathbf{X}) - \gamma_{jm}^{den}\boldsymbol{\mu}_{jm}). \tag{A.13}$$

Since the smoothing function $G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ has maximum at $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$, the partial derivative of $G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ with respect to $\boldsymbol{\mu}_{jm}$ can also be expressed by

$$\frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \boldsymbol{\Sigma}_{jm}^{-1}(\eta_{jm}\boldsymbol{\mu}'_{jm} - \eta_{jm}\boldsymbol{\mu}_{jm}), \tag{A.14}$$

where η_{jm} is a positive constant, and $\boldsymbol{\mu}'_{jm}$ is the mean vector from the previous iteration. Because $\boldsymbol{\Sigma}_{jm}^{-1}$ is positive definite, the solution for equation

$$\frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - \frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') + \frac{\partial}{\partial \boldsymbol{\mu}_{jm}} G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \mathbf{0} \tag{A.15}$$

is the $\hat{\boldsymbol{\mu}}_{jm}$ in Equation (1.12).

For the update of covariance matrices, consider the following partial derivative:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} \log(p(\mathbf{x}_t|\boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})) &= \frac{-1}{2}[(\mathbf{x}_t - \boldsymbol{\mu}_{jm})(\mathbf{x}_t - \boldsymbol{\mu}_{jm})^{\mathsf{T}}(-\boldsymbol{\Sigma}_{jm}^{-2}) + \boldsymbol{\Sigma}_{jm}^{-1}] \\ &= \frac{1}{2}[(\mathbf{x}_t - \boldsymbol{\mu}_{jm})(\mathbf{x}_t - \boldsymbol{\mu}_{jm})^{\mathsf{T}} - \boldsymbol{\Sigma}_{jm}]\boldsymbol{\Sigma}_{jm}^{-2}. \end{aligned} \tag{A.16}$$

By incorporating $\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X}^2)$ and $\boldsymbol{\vartheta}_{jm}^{num}(\mathbf{X})$, the partial derivative $\frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ would be of the form in (1.7), and similarly $\frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ would be of the form in (1.9). Again,

since $\boldsymbol{\Sigma}_{jm}^{-2}$ is positive definite, the solution for

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} G^{num}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - \frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} G^{den}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') + \frac{\partial}{\partial \boldsymbol{\Sigma}_{jm}} G^{sm}(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = \mathbf{0} \tag{A.17}$$

is unique. Plugging in $\hat{\boldsymbol{\mu}}_{jm}$ in Equation (1.12) to Equation (A.17) yields the update formula in Equation (1.13).

Instead of using auxiliary function in Equation (A.4) directly to update mixture weights, the following auxiliary function was used in [28]:

$$\sum_{m=1}^{M_j} \gamma_{jm}^{num} \log(w_{jm}) - \frac{\gamma_{jm}^{den}}{w'_{jm}} w_{jm}, \tag{A.18}$$

where w'_{jm} is mixture weight from the previous HMM parameter set $\boldsymbol{\lambda}'$. The reason for switching to the auxiliary function in Equation (A.18) is that the sum-to-one constraint of mixture weights can be easily incorporated into the maximization. Note that the partial derivative of the auxiliary function in Equation (A.18) with respect to w_{jm} equals to

$$\frac{\gamma_{jm}^{num}}{w_{jm}} - \frac{\gamma_{jm}^{den}}{w'_{jm}}, \tag{A.19}$$

and is the same as $\frac{\partial}{\partial w_{jm}} G_{MMI}(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$ at $w_{jm} = w'_{jm}$. As a result, the function in Equation (A.18) is also an weak-sense auxiliary function. The mixture weights are then updated by running another EBW procedure on Equation (A.18).

To illustrate the update procedure for the mixture weights, let $\{w_{jm}^{(p)}\}_{m=1}^{M_j}$ denotes the mixture weights after p iteration. The objective function that the update procedure seek to maximize is

$$F^w(\boldsymbol{\lambda}^{(p+1)}) = \sum_{m=1}^{M_j} \gamma_{jm}^{num} \log(w_{jm}^{(p+1)}) - \frac{\gamma_{jm}^{den}}{w'_{jm}} w_{jm}^{(p+1)}. \tag{A.20}$$

To make the function above more analytically tractable, a smoothing function is added to

$F^w(\boldsymbol{\lambda}^{(p+1)})$, resulting in an auxiliary function

$$G^w(\boldsymbol{\lambda}^{(p+1)}, \boldsymbol{\lambda}^{(p)}) = \sum_{m=1}^{M_j} \gamma_{jm}^{num} \log(w_{jm}^{(p+1)}) - \frac{\gamma_{jm}^{den}}{w'_{jm}} w_{jm}^{(p+1)} + k_{jm}(w_{jm}^{(p)} \log(w_{jm}^{p+1}) - w_{jm}^{(p+1)}), \quad \text{(A.21)}$$

where each k_{jm} is a constant greater or equal to 0, and $w'_{jm} = w_{jm}^{(0)}$. Note that because of the sum-to-one constraint of mixture weights, choosing $k_{jm} = \frac{-\gamma_{jm}^{den}}{w'_{jm}} + \max_m \frac{\gamma_{jm}^{den}}{w'_{jm}}$ reduces the linear term in Equation (A.21) to a constant independent of the weights:

$$G^w(\boldsymbol{\lambda}^{(p+1)}, \boldsymbol{\lambda}^{(p)}) = \sum_{m=1}^{m_j} [\gamma_{jm}^{num} \log(w_{jm}^{(p+1)}) + k_{jm} w_{jm}^{(p)} \log(w_{jm}^{(p+1)})] + \max_m \frac{\gamma_{jm}^{den}}{w'_{jm}}. \quad \text{(A.22)}$$

Applying Lagrangian multipliers to Equation (A.22) with the sum-to-one constraint of $\{w_{jm}^{(p+1)}\}_{m=1}^{M_j}$, the update formula of mixture weights become

$$w_{jm}^{(p+1)} = \frac{\gamma_{jm}^{num} + k_{jm} w_{jm}^{(p)}}{\sum_{m=1}^{M_j} \gamma_{jm}^{num} + k_{jm} w_{jm}^{(p)}}, \quad \text{(A.23)}$$

where

$$k_{jm} = (\max_m \frac{\gamma_{jm}^{den}}{w'_{jm}}) - \frac{\gamma_{jm}^{den}}{w'_{jm}}. \quad \text{(A.24)}$$

Because the transition probabilities of HMM states also have sum-to-one constraint, similar technique can be applied for the update of transition probabilities.

Appendix B

Quickprop Algorithm

This appendix illustrates the Quickprop algorithm used in [25] in more detail. Quickprop is a second-order optimization method loosely 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 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

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

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

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

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 $F(\boldsymbol{\lambda})$ [25]. However, in general, there is no guarantee that the Hessian matrix is positive definite, and representing true Hessian matrix becomes impractical as the length of $\boldsymbol{\lambda}$ becomes large.

To address the two issues above, Quickprop makes the following two major changes with regard to 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 gradient to the update step if the criterion does not hold.

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

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

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 above 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

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

where $\nabla^2 F(\boldsymbol{\lambda})_i$ is approximated by (B.3) and ϵ is a positive learning rate. Generally, setting a proper value of ϵ is important.

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 k times larger than previous step size; if the gradient and the modified Newton step are of the same sign, simple gradient step is used instead. Details of Quickprop update can be seen in the pseudo code of Algorithm (2).

Algorithm 2 Quickprop Update [25]

##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($k * \Delta\lambda$)) or abs(d) >TASK_LIMIT **then**
    $d \Leftarrow \text{sign}(d) * \min(\text{TASK\_LIMIT}, \text{abs}(k * \Delta\lambda))$
  end if

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

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

Appendix C

Conjugate Gradient Algorithm

This appendix presents pseudo-code of the Conjugate Gradient (CG) algorithm provided in [31]. Given a function f taking n parameters, a starting value $\mathbf{x}$, and an error tolerance $\epsilon < 1$, the CG algorithm can minimize $f(\mathbf{x})$ by doing the procedures in Algorithm 3. There are two loops in the algorithm. The outer loop decides the direction $\mathbf{d}$ to update the parameters, while the inner loop seeks to determine the best step size along the direction of $\mathbf{d}$.

The algorithm terminates when the maximum number of iteration i_{max} or when the length of gradient after i iteration $f'(\mathbf{x}_{(i)})$ is smaller than a ratio of the length of original gradient $f'(\mathbf{x}_{(0)})$. The pre-conditioner $\mathbf{M}$ is to make the procedure less sensitive to numerical errors. $\mathbf{M}$ must be positive definite, but not necessarily be in matrix form. β is kept to be greater 0 to ensure better convergence. The value σ_0 determines the first step of linear search. Unfortunately, the value of σ_0 may need to be adjust for different types of functions.

Algorithm 3 Conjugate Gradient

$i \Leftarrow 0$
$j \Leftarrow 0$
$\mathbf{r} \Leftarrow f'(\mathbf{x})$
Calculate a preconditioner $\mathbf{M} \approx f''(\mathbf{x})$
$\mathbf{s} \Leftarrow \mathbf{M}^{-1}\mathbf{r}$
$\mathbf{d} \Leftarrow \mathbf{s}$
$\delta_{new} \Leftarrow \mathbf{r}^{\mathrm{T}}\mathbf{d}$
$\delta_0 \Leftarrow \delta_{new}$
while $i < i_{max}$ and $\delta_{new} > \epsilon^2\delta_0$ **do**
  $j \Leftarrow 0$
  $\delta_d \Leftarrow d^{\mathrm{T}}$
  $\alpha \Leftarrow -\sigma_0$
  $\eta_{prev} \Leftarrow [f'(\mathbf{x} + \sigma_0\mathbf{d})]^{\mathrm{T}}\mathbf{d}$
  repeat
    $\eta \Leftarrow [f'(\mathbf{x})]^{\mathrm{T}}\mathbf{d}$
    $\alpha \Leftarrow \alpha\frac{\eta}{\eta_{prev} - \eta}$
    $\mathbf{x} \Leftarrow \mathbf{x} + \alpha\mathbf{d}$
    $\eta_{prev} \Leftarrow \eta$
    $j \Leftarrow j + 1$
  until $j \geq j_{max}$ or $\alpha^2\sigma_d > \epsilon^2$
  $\mathbf{r} \Leftarrow -f'(\mathbf{x})$
  $\delta_{old} \Leftarrow \delta_{new}$
  $\delta_{mid} \Leftarrow \mathbf{r}^{\mathrm{T}}\mathbf{s}$
  Calculate a preconditioner $\mathbf{M} \approx f''(\mathbf{x})$
  $s \Leftarrow \mathbf{M}^{-1}\mathbf{r}$
  $\delta_{new} \Leftarrow \mathbf{r}^{\mathrm{T}}\mathbf{s}$
  $\beta \Leftarrow \frac{\delta_{new} - \delta_{mid}}{\delta_{old}}$
  $k \Leftarrow k + 1$
  if $k == n$ or $\beta \leq 0$ **then**
    $\mathbf{d} \Leftarrow \mathbf{s}$
    $k \Leftarrow 0$
  else
    $\mathbf{d} \Leftarrow \mathbf{s} + \beta\mathbf{d}$
  end if
  $i \Leftarrow i + 1$
end while

Appendix D

Large-Margin Training on Lattices

This appendix describes how to compute the loss in Equation 4.1 and its gradient using the lattice. The key idea is to break down the expectations into a summation over the contributions of all phone arcs in the lattices. Given a phone lattice of an utterance, a standard HMM forward-backward algorithm can be performed, and for each phone arc q, the following quantities can be computed:

- $p(q)$: Probability of phone arc q being generated by the model.
- α_q: Probability of hypotheses leading up to q. Computed by the forward procedure.
- β_q: Probability of hypotheses leaving q. Computed by the backward procedure.
- γ_q: The posterior probability of phone arc q being traversed. Computed by $\frac{\alpha_q \beta_q}{\sum_{r \text{ at the end}} \alpha_r}$.

Given these quantities of the phone arcs in the lattice, the expectation terms in Equation 4.1 can be computed by the following.

D.1 $\mathrm{E}_{\mathbf{S}}[\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))]$

For convenience, let ϱ_q^n denote the expected log-likelihood $\mathrm{E}_{\mathbf{S}}[\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))]$; that is,

$$\varrho_{avg}^n = \mathrm{E}_{\mathbf{S}}[\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))] = \frac{\sum_{\mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}) \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))}{\sum_{\mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})p_L(\mathbf{U})}. \tag{D.1}$$

Because the probability of hypothesis S being generated can be decomposed into the probability of a phone sequence $\{q_1^{\mathbf{S}} \dots q_S^{\mathbf{S}}\}$ being generated, the log probability of $\mathbf{S}$ can be computed by

$$\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S})) = \sum_{s=1}^{S} \log(p(q_s^{\mathbf{S}})) + \sum_{s=1}^{S-1} \log(t_{q_s q_{s+1}}), \tag{D.2}$$

where $t_{q_s q_{s+1}}$ is the transition probability from phone arc q_s to q_{s+1} (including the lexicon and language model probabilities).

Using the decomposition in Equation (D.2), ϱ_{avg}^n can be computed by the following forward procedures: Let α_q^l be the expected log-likelihood of the phone sequences leading up to q; α_q^l can be computed by

$$\alpha_q^l = \frac{\sum_{r \text{ preceding } q} \alpha_r t_{rq}(\alpha_r^l + \log(t_{rq}))}{\sum_{r \text{ preceding } q} \alpha_r t_{rq}} + \log(p(q)), \tag{D.3}$$

where α_q is forward probability leading up to q as defined above, and t_{rq} is the transition probability from r to q. The expected log-likelihood of all hypotheses can be computed by doing a weighted sum over all α_q^l; that is,

$$\varrho_{avg}^n = \frac{\sum_{q \text{ at the end of lattice}} \alpha_q \alpha_q^l}{\sum_{q \text{ at the end of lattice}} \alpha_q}. \tag{D.4}$$

For the gradient of ϱ_{avg}^n with respect to $\boldsymbol{\lambda}$, it can be computed by using the chain rule

$$\frac{\partial \varrho_{avg}^n}{\partial \boldsymbol{\lambda}} = \sum_q \frac{\partial \varrho_{avg}^n}{\partial \log(p(q))} \frac{\partial \log(p(q))}{\partial \boldsymbol{\lambda}}. \tag{D.5}$$

Similar to MPE training, to compute the partial derivative $\frac{\partial \varrho_{avg}^n}{\partial \log(p(q))}$ needs to compute the average log-likelihood of hypotheses passing phone arc q

$$\varrho(q) = \frac{\sum_{\mathbf{S}:\ q \in \mathbf{S}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}) \log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))}{\sum_{\mathbf{U}:\ q \in \mathbf{U}} p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{U})p_L(\mathbf{U})}. \tag{D.6}$$

Note that ϱ_{avg}^n is of similar form as the objective function of MPE except that $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))$

is also a function of $\log(p(q))$. The partial derivative of $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))$ with respect to $\log(p(q))$ is 1 if $q \in \mathbf{S}$ and is 0 otherwise. As a result, $\frac{\partial \varrho_{avg}^n}{\partial \log(p(q))}$ can be computed by

$$\frac{\partial \varrho_{avg}^n}{\partial \log(p(q))} = \gamma_q(\varrho(q) - \varrho_{avg}^n) + \gamma_q, \tag{D.7}$$

where the $\gamma_q(\varrho(q) - \varrho_{avg}^n)$ term is similar to the term γ_q^{MPE} in MPE training, and the term γ_q reflects the partial derivative of $\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))$ with respect to $\log(p(q))$.

To compute the value of $\varrho(q)$, a additional backward procedure is required. Let β_q^l be the average likelihood of hypotheses leaving phone arc q; the value β_q^l can be computed by a weighted average with respect to all phone arc r following q:

$$\beta_q^l = \frac{\sum_{r \text{ following } q} t_{qr}p(r)\beta_r(\beta_r^l + \log(t_{qr}) + \log(p(r)))}{\sum_{r \text{ following } q} t_{qr}p(r)\beta_r}. \tag{D.8}$$

After β_q^l is computed, $\varrho(q)$ can be computed by

$$\varrho(q) = \alpha_q^l + \beta_q^l. \tag{D.9}$$

D.2 $\mathbf{E}_{\mathbf{S}}[\mathcal{D}(\mathbf{S}, \mathbf{Y}_n)]$

Similar to the case for computing $\mathrm{E}_{\mathbf{S}}[\log(p_{\boldsymbol{\lambda}}(\mathbf{X}_n|\mathbf{S})p_L(\mathbf{S}))]$, let δ_{avg}^n denotes the average string distance of all hypotheses $\mathrm{E}_{\mathbf{S}}[\mathcal{D}(\mathbf{S}, \mathbf{Y}_n)]$, and let $\delta(q)$ denote the average string distance of hypotheses passing through q. If the string distance $\mathcal{D}(\mathbf{S}, \mathbf{Y}_n)$ can also be broken down into a summation of phone distances $\sum_{q \in \mathbf{S}} \Delta(q, \mathbf{Y}_n)$, δ_{avg}^n and its gradient can be computed as follows.

Let α_q^d denote the average string distance of hypotheses leading up to q, and let β_q^d denote the average string distance of hypotheses leaving q. The value of α_q^d and β_q^d can be computed by the following forward and backward procedures:

$$\alpha_q^d = \frac{\sum_{r \text{ preceding } q} \alpha_r t_{rq} \alpha_r^d}{\sum_{r \text{ preceding } q} \alpha_r t_{rq}} + \Delta(q, \mathbf{Y_n}). \tag{D.10}$$

$$\beta_q^d = \frac{\sum_{r \text{ following } q} \beta_r t_{qr} p(r)(\beta_r^d + \Delta(r, \mathbf{Y}_n))}{\sum_{r \text{ following } q} \beta_r t_{qr} p(r)}. \tag{D.11}$$

After computing all α_q^d, δ_{avg}^n can be computed by

$$\delta_{avg}^n = \frac{\sum_{q \text{ at the end of lattice}} \alpha_q \alpha_q^d}{\sum_{q \text{ at the end of lattice}} \alpha_q}. \tag{D.12}$$

The partial derivative $\frac{\partial \delta_{avg}^n}{\partial \log(p(q))}$ can be computed by

$$\frac{\partial \delta_{avg}^n}{\partial \log(p(q))} = \gamma_q(\delta(q) - \delta_{avg}^n), \tag{D.13}$$

where $\delta(q) = \alpha_q^d + \beta_q^d$. The gradient of δ_{avg}^n with respect to $\boldsymbol{\lambda}$ can then be computed by

$$\frac{\partial \delta_{avg}^n}{\partial \boldsymbol{\lambda}} = \sum_q \gamma_q(\delta(q) - \delta_{avg}^n) \frac{\partial \log(p(q))}{\partial \boldsymbol{\lambda}}. \tag{D.14}$$

References

[1] R. Battiti. First- and second- order methods for learning: Between steepest descent and Newton's method. *Neural Comput.*, 4:141–166, 1992.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. *IEEE International Conference on Communications*, pages 1064–1070, 1993.

[3] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2), 1998.

[4] H.-A. Chang and J. R. Glass. Hierarchical large-margin Gaussian mixture models for phonetic classification. *Automatic Speech Recognition and Understanding Workshop*, pages 272–277, 2007.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubi. Maximum likelihood from incomplete data via EM algorithm. *Journal of Royal Statistical Society*, 39:1–88, 1977.

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

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

[8] 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.

[9] J. J. Godfrey, E. C. Holliman, and J. McDaniel. SWITCHBOARD: telephone speech corpus for research and development. *Proceedings of ICASSP*, pages 517–520, 1992.

[10] P. S. Gopalakrishnan, D. Kanevsky, A. Nadas, and Nahamoo. Generalization of the Baum algorithm to rational objective functions. *Proceedings of ICASSP*, pages 631–634, 1989.

[11] A. Gunawardana, M. Mahajan, A. Acero, and J. C. Platt. Hidden conditional random fields for phone classification. *Proceedings of Eurospeech*, pages 1117–1120, 2005.

[12] A. K. Halberstadt. *Heterogenous acoustic measurements and multiple classifiers for speech recognition.* PhD thesis, Massachusetts Institute of Technology, 1998.

[13] A. K. Halberstadt and J. R. Glass. Heterogeneous acoustic measurements for phonetic classification. *Proceedings of Eurospeech*, pages 401–404, 1997.

[14] A. K. Halberstadt and J. R. Glass. Heterogeneous acoustic measurements and multiple classifiers for speech recognition. *Proceedings of ICSLP*, pages 995–998, 1998.

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

[16] B.-H. Juang. Maximum-likelihood estimation for mixture multivariate stochastic observations of Markov chains. *AT&T Technical Journal*, 64(6), 1985.

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

[18] B.-H. Juang and S. Katagiri. Discriminative learning for minimum error classification. *IEEE Trans. on Signal Processing*, 40(12):3043–3053, 1992.

[19] L. Lamel, R. Kassel, and S. Seneff. Speech database development: design and analysis of acoustic-phonetic corpus. *Proceedings of DARPA Speech Recognition Workshop*, 1986.

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

[21] J. Leroux and E. McDermott. Optimization methods for discriminative training. *Proceedings of Eurospeech*, pages 3341–3344, 2005.

[22] J. Li, M. Yuan, and C.-H. Lee. Soft margin estimation of hidden Markov model parameters. *Proceedings of Interspeech*, pages 2422–2425, 2006.

[23] W. Macherey, L. Haferkamp, R. Schlüter, and H. Ney. Investigations on error minimizing training criteria for discriminative training in automatic speech recognition. *Proceedings of Eurospeech*, pages 2133–2136, 2005.

[24] E. McDermott and T. J. Hazen. Minimum classification error training of landmark models for real-time continuous speech recognition. *Proceedings of ICASSP*, pages 937–940, 2004.

[25] 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. on Audio, Speech, and Language Processing*, 15(1), 2007.

[26] A. Park, T. J. Hazen, and J. R. Glass. Automatic processing of audio lectures for information retrieval: vocabulary selection and language modeling. *Proceedings of ICASSP*, pages 497–500, 2005.

[27] A. S. Park. *Unsupervised pattern discovery in speech: applications to word acquisition and speaker segmentation.* PhD thesis, Massachusetts Institute of Technology, 2006.

[28] D. Povey. *Discriminative training for large vocabulary speech recognition.* PhD thesis, University of Cambridge, 2003.

[29] D. Povey and P. C. Woodland. Minimum phone error and I-smoothing for improved discriminative training. *Proceedings of ICASSP*, pages 105–108, 2002.

[30] R. Rifkin, K. Schutte, M. Saad, J. Bouvire, and J. R. Glass. Noise robust phonetic classification with linear regularized least squares and second-order features. *Proceedings of ICASSP*, pages 881–884, 2007.

[31] J. R. Schewchuk. An introduction to conjugate gradient without the agonizing pain. *C.M.U.*, 1994.

[32] F. Sha. *Large margin training for acoustic models for speech recognition.* PhD thesis, University of Pennsylvania, 2006.

[33] F. Sha and L. K. Saul. Large margin Gaussian mixture modeling for phonetic classification and recognition. *Proceedings of ICASSP*, pages 265–268, 2006.

[34] F. Sha and L. K. Saul. Comparison of large margin training to other discriminative methods for phonetic recognition by hidden Markov models. *Proceedings of ICASSP*, pages 313–316, 2007.

[35] T. Shinozaki and M. Ostendorf. Cross-validation em training for robust parameter estimation. *Proceedings of ICASSP*, pages 473–440, 2007.

[36] A. Stokle. SRILM: An extensible language modeling toolkit. *Proceedings of ICSLP*, pages 901–904, 2002.

[37] L. Vandenberghe and S. P. Boyd. Semidefinite programing. *SIAM Review*, 38(1):49–95, 1996.

[38] P.C. Woodland and D. Povey. Large scale MMIE training for conversational telephone speech recognition. *Proceedings of NIST Speech Transcription Workshop*, 2000.

[39] D. Yu, L. Deng, and A. Acero. Large-margin minimum classification error training for large-scale speech recognition tasks. *Proceedings of ICASSP*, pages 1137–1140, 2007.