Chapter 1

Introduction

Communication through speech is an integral part of our lives. Automatic speech recognition for machines (computers) therefore provides a natural and effective way of communicating with machines. We are, however, far from creating a machine which can understand spoken discourse on any subject, by all speakers and for all conditions. Much research and development of speech recognition systems will be required before this goal is achieved.

Hidden Markov models (HMMs) have been the dominant approach to speech recognition since the 1980s [94, 92]. HMMs are statistical models used to characterize the spectral properties of the frames of a pattern.

A *Markov model* is a system that can be described as being in one of N distinct states at any given time. At regularly spaced, discrete times, the system undergoes a change of state, depending on a set of probabilities associated with each state. This is known as a discrete-time Markov process. *Hidden* Markov models are doubly embedded stochastic processes with an underlying stochastic process that is not directly observable (hidden) but can be observed indirectly through another set of stochastic processes that produce the sequence of observations. The observation is therefore a probabilistic function of the state. Section 2.1 will describe HMMs and their use in speech recognition in more detail.

Hidden Markov model speech recognition systems typically consist of two main parts, namely

1. *Feature extraction.* Here features, which will be used to recognize the utterance, are extracted from the speech signal. One of the most common features used are the Mel-frequency cepstral coefficients (MFCCs), which can be obtained from the power spectrum of the speech signal. Section 2.1.1 gives a detailed description of the feature extraction process as used in this thesis.

2. *Hidden Markov models.* HMMs are used to represent the temporal nature of the speech signal. A word or phonetic unit is typically represented by a single HMM, with each state in the HMM representing an acoustic unit within the word or phonetic unit. In a continuous digit recognition task, for example, HMMs would be used to represent the digits 0 through 9 and other events such as silence and inter word pauses. In continuous density HMMs, a weighted sum of Gaussian distributions is used to represent the probability of an observation (features extracted) being generated by that state. Section 2.1 presents a detailed description of the HMM system used in this thesis.

The assumption made when using a hidden Markov model (HMM) or any statistical model, is that the process can be characterized as a parametric random process. It is furthermore assumed that these parameters can be determined or estimated in a precise, well-defined manner. HMMs are usually trained using the maximum likelihood (ML) criterion. When creating continuous speech recognition systems, sparse training data is often a problem. This limits the effectiveness of the conventional approaches, such as maximum likelihood parameter estimation.

The collection of large speech databases is not a trivial task (if done properly). It is not always possible to collect, segment and annotate large databases for every task, language or dialect. The collection of large speech databases is an expensive and time consuming task. It is doubtful whether there will ever be a substitute for sufficient, well recorded and annotated data when creating speech recognition systems. However, given that it is not always possible to collect sufficient data, this thesis focuses on using Bayesian and discriminative training algorithms to improve continuous speech recognition systems in scenarios where there is a limited amount of training data available.

1.1 Adaptation

Adaptation is a process for adjusting seed models or training data (non-task-specific) to create more specialized models using a small amount of task-specific adaptation data. There are many applications of adaptation algorithms, including:

- **Speaker adaptation** Speaker adaptation is a well researched and documented [67, 74, 32, 6, 20, 31, 21] example of adaptation, where little speaker-dependent data is available for creating a speaker-specific model. Using the speaker-independent model or dataset of many speakers, adaptation techniques can be used to create an improved speaker-dependent model. It is impractical to use the algorithms described in this thesis for speaker adaptation due to their computational expensive nature.

- **Gender adaptation** It is well documented [117, 51, 118] that usage of gender dependent models for male and female speakers improves performance. Adaptation can be used to improve the gender dependent performance in situations where there is limited training data available. If, for example, we have a reasonable amount of training data for female speakers but little for male speakers, we could adapt the female model or data using the male training data, thereby improving the recognition performance for male speakers.

- **Language and dialect adaptation** As mentioned, the collection of a comprehensive database for a new language or dialect is a difficult and time consuming procedure. Much of the work in multi-language research has focused on creating speech recognition systems which can recognize speech from multiple languages [47, 10, 29, 25], or bootstrapping of new monolingual models for a new language using existing models [116, 102].

  Alternatively, we can apply adaptation techniques to reduce the amount of language or dialect-specific training data required. Recently, some studies [112, 15, 30, 63, 85] have therefore used adaptation techniques to improve the recognition performance for a new language or dialect, using existing models of other languages or dialects.

Two main families of adaptation schemes have been proposed in the past, namely Bayesian adaptation and transformation-based adaptation procedures.

1.1.1 Bayesian adaptation

The maximum *a-posteriori* (MAP) estimation procedure [67, 43, 45] attempts to find the parameters (θ) which maximize the posterior probability of the parameters given the training data, i.e.

$$\theta_{MAP} = \underset{\theta}{argmax} P(\theta|\mathbf{X}) = \underset{\theta}{argmax} P(\mathbf{X}|\theta)P(\theta) \tag{1.1}$$

where X is the training data. $P(\theta)$ is the prior distribution of the model parameters and expresses any knowledge about the parameters or model prior to any data being observed. $P(\mathbf{X}|\theta)$ is the probability of the observation $\mathbf{X}$ being generated by the model with parameters θ. $P(\mathbf{X})$ has not been included in Eq. (1.1) as it is a normalizing constant and does not affect the mode (θ_{MAP}) of the posterior distribution $P(\theta|\mathbf{X})$.

The MAP framework provides a way of incorporating prior information in the estimation process, which is useful when dealing with problems caused by limited training data. This prior information can be subjective or information obtained from non-task-specific data or models. MAP estimation has sometimes been referred to as Bayesian learning in the speech recognition literature. MAP estimation is an approximate Bayesian learning algorithm and will therefore not be referred to as Bayesian learning in this thesis.

When using MAP estimation for adaptation purposes, the non-task-specific information is encapsulated in the prior distribution. For example, in speaker adaptation, speaker independent data or models will be used to create the prior, along with any other subjective information. MAP starts from the seed model performance and converges asymptotically to task-specific performance as the amount of adaptation data increases. The usage of the MAP algorithm will be investigated in Chapter 4.

The following summarizes the advantages and disadvantages of Bayesian and transformation-based (Section 1.1.2) approaches,

- Bayesian adaptation requires relatively large amounts of adaptation data (compared to transformation-based methods).
- Transformation-based methods are typically much faster in that they require very small amounts of training data.
- Transformation-based adaptation has the advantage that it can be applied either directly to the features, or to the HMM parameters.
- A disadvantage of transformation-based methods is that they tend not to take full advantage of larger amounts of adaptation data.
- Transformation-based adaptation is usually text-dependent, whereas Bayesian adaptation is typically text-independent.

- Bayesian adaptation has good asymptotic properties: performance converges to speaker-dependent performance as the amount of adaptation speech increases.

1.1.2 Transformation-based adaptation

Transformation-based techniques estimate a transformation of seed model parameters, thereby creating the new task-specific model. The transformation is typically linear [33, 69], though non-linear transforms have also been used [1]. When using a linear transformation, we estimate the following transformation,

$$\mathbf{x} = \mathbf{A}\mathbf{y} + \mathbf{b}, \tag{1.2}$$

where $\mathbf{y}$ is the seed model parameter vector, $\mathbf{x}$ is the new adapted model parameters, $\mathbf{A}$ and $\mathbf{b}$ are the transformation matrix and offset vector.

The transformation will typically contain far fewer parameters compared to the model we are transforming. We will therefore be able to estimate a reasonably accurate transformation when very little data is available. Generally, when the adaptation data is limited, transformation-based adaptation can therefore efficiently transform all the HMM parameters using cluster-dependent transformations.

Given a seed HMM model ($\mathbf{y}$) with observation densities of the form

$$P(\mathbf{o}|s_t) = \sum_{i=1}^{M} c_s i \mathcal{N}(\mathbf{o}, \mu_{ig}, \mathbf{\Sigma}_{ig}), \tag{1.3}$$

where g is the index of the Gaussian codebook used by state s_t. The transformation

$$\mathbf{x}_t = \mathbf{A}_g \mathbf{y}_t + \mathbf{b}_g, \tag{1.4}$$

results in a adapted model with observation densities of the form

$$P(\mathbf{o}|s_t) = \sum_{i=1}^{M} c_s i \mathcal{N}(\mathbf{o}, \mathbf{A}_g \mu_{ig} + \mathbf{b}_g, \mathbf{A}_g \mathbf{\Sigma}_{ig} \mathbf{A}_g^T). \tag{1.5}$$

Only the parameters $\mathbf{A}_g$, $\mathbf{b}_g$, $g = 1 \ldots N_g$ need to be estimated, where N_g is the number of distinct transformations.

The transformation is typically estimated using the maximum likelihood criterion [33, 69], in which case it is known as maximum likelihood linear regression (MLLR). Recently the maximum *a-posterior* (MAP) criterion has been used to estimate the transformation parameters (MAPLR) [22], allowing the use of prior information in the estimation of the transformation. The minimum classification error criterion (MCE) has also been used [96, 95].

1.1.3 Hybrid adaptation algorithms

Combinations of Bayesian and transformation-based adaptation methods have been shown to combine some of the advantages of the two approaches. Hybrid algorithms using MLLR adaptation followed by MAP adaptation (MLLR-MAP) have been used with much success for speaker adaptation [32, 110] and cross-language adaptation [84]. The MAP adaptation algorithm, followed by the MCE discriminative training procedure (MAP-MCE) [74] has also been used to improve on the MAP and MCE procedures for speaker adaptation.

1.2 Training

Here, only the task-specific training data is available and one can therefore not use adaptation techniques to improve the performance of the resultant system. An example here would be training a speech recognizer for a new language, where only a small amount of data has been recorded and data from other languages is not available or other complications do not allow the use of adaptation.

1.2.1 Discriminative training

Conventional maximum likelihood (ML) estimation attempts to maximize the likelihood of the training data given the model parameters of the corresponding class. The models from other classes do not participate in the parameter estimation. By maximizing the likelihood of the correct model, but not minimizing the likelihood of other competing models, it cannot be guaranteed that the ML models will optimally discriminate against incorrect classes in recognition and therefore minimize recognition error. Several methods have been proposed to improve this by including discriminative information in the training criterion.

The maximum mutual information (MMI) criterion, which minimizes the class conditional entropy has been used to create a training procedure which is more discriminative [88, 87, 62, 113]. However, as is the case with the ML criterion, MMI does not necessarily minimize the classification error.

The aim of minimum classification error (MCE) training is to correctly discriminate the observations of an HMM for best recognition results and not to fit the distributions to the data. Discriminative training of hidden Markov models (HMMs) using MCE training has been used in several speech recognition tasks with much success. Tasks where MCE training has been used to improve recognition performance include: connected digit recognition [59, 23, 65, 108], the English "E"-set {b,c,d,e,g,p,t,v,z} [23], speaker adaptation [74] and continuous speech [65].

MCE is somewhat prone to overspecialization, especially when training data is limited. Overspecialization is characterized by good recognition performance for the data used during training, but poor recognition performance for independent testing data. MCE also tends to further emphasize any mismatch between the training and testing sets, resulting in a degradation in testing set performance after a maximum has been reached. Methods of reducing overspecialization in the minimum classification error algorithm will be investigated in Chapter 3.

1.2.2 Bayesian learning

The fundamental concept of Bayesian learning or analysis is that the plausibilities of alternative hypotheses are represented by probabilities, with inference being performed by evaluating these probabilities. The result of Bayesian learning is a probability distribution over model parameters that expresses our beliefs regarding how likely the different model parameter values are.

Given a vector $\mathbf{y} = (y_1, \ldots, y_n)$ of n observations, we have the conditional probability distribution $P(\mathbf{y}|\theta)$, which depends on the k parameters $\theta^T = (\theta_1, \ldots, \theta_k)$. To start the process of Bayesian learning we define a prior distribution $P(\theta)$ for the parameters θ. Using Bayes' rule, the conditional distribution of θ given the observed data (posterior distribution) is

$$P(\theta|\mathbf{y}) = \frac{P(\mathbf{y}|\theta)P(\theta)}{P(\mathbf{y})}. \tag{1.6}$$

The prior distribution is an important part of any Bayesian method, as it expresses our knowledge about the distributions prior to any data being observed. Using the prior distribution $P(\theta)$ and likelihood $P(\mathbf{y}|\theta)$ we can calculate the posterior distribution $P(\theta|\mathbf{y})$ which is then used to classify an unknown observation. Note that $P(\mathbf{y})$ is a normalization term and is usually ignored.

To classify an unknown observation, we integrate the predictions of the model with respect to the posterior distribution. This is typically a non-trivial task and the integral must either be numerically computed or simplified by approximating the posterior using some parametric form. An approximation which is often used assumes that the posterior is well approximated by a Normal distribution [72, 54, 58]. Assuming such a simple parametric form allows the integral to be easily computed. Such an approximation has the disadvantage that a complex multi-modal posterior distribution cannot be accurately approximated.

Markov chain Monte Carlo methods [36, 37, 38, 39, 26, 79, 111] can be used to numerically integrate the above model prediction with respect to the posterior distribution, and have been used for this purpose in the field of neural networks by Neal [82]. These methods make no assumption concerning the form of the posterior distribution.

The maximum *a-posteriori* (MAP) estimation method has been used extensively in speech recognition and can be considered an approximate Bayesian learning procedure if the posterior distribution is sufficiently peaked about its mode (assuming a single mode).

Bayesian learning allows us to use more complex models when little training data is available (as compared to point estimate techniques). The usage of Bayesian learning using a Markov chain Monte Carlo algorithm will be investigated in Chapter 5.

1.3 Problem statement

Conventional estimation algorithms (such as maximum likelihood estimation) rely on a reasonable amount of training data to give accurate parameter estimates. The accuracy of the parameter estimate is directly related to the recognition performance of the speech recognition system and is therefore of extreme importance. Overtraining, the phenomenon where training set performance is better than the performance for an independent testing set, is also more prevalent when training data is limited (it is easier for the model to become too specialized).

As mentioned, the collection of large speech databases is an expensive and time consuming task. As a result, it is not always possible to collect, segment and annotate large databases for every task, language and dialect. The sparse training data problem is therefore a real and important problem that must be addressed. This thesis therefore investigates and proposes several techniques which improve the recognition accuracy for sparse training data scenarios.

1.4 Organization of this thesis

This thesis is organized as follows.

In Chapter 2, the relevant hidden Markov model theory is reviewed. The hidden Markov model speech recognition system is also briefly described. The speech corpora used to experimentally evaluate the work in this thesis are described and relevant results from the literature are reported. Finally, overtraining is discussed in terms of the bias/variance problem.

Chapter 3 introduces minimum classification error (MCE) training. Overtraining is discussed within the MCE framework, and several modifications which limit overtraining are proposed. Various aspects of the MCE algorithm and the proposed modifications are discussed and experimentally evaluated.

Chapter 4 focuses on the application of Bayesian theory to adaptation in continuous speech recognition. The classical maximum *a-posteriori* (MAP) adaptation algorithm of Gauvain and Lee [45] is reviewed. An alternative gradient-based method of obtaining the MAP estimate, which does not use a parametric prior distribution, is introduced. A Bayesian inspired modification to the MCE training procedure is proposed. This method effectively tries to obtain the MAP point of the correct classification probability distribution of the parameters. Finally, the three different methods discussed in this chapter are experimentally compared.

In Chapter 5, Bayesian learning is introduced and an implementation for continuous speech recognition is discussed. Monte Carlo methods relevant to this work are introduced and discussed. The implementation of Bayesian learning within an HMM framework is described. The resultant system is tested for three situations where limited data is available for training purposes.

Finally, in Chapter 6, the discussions and results of previous chapters are summarized and conclusions are made. Suggestions for future research are also given.

1.5 Contributions of this thesis

The contributions of this thesis are,

1. New modifications to the MCE algorithm are proposed in Chapter 3. These modifications limit the effect of overtraining which is prevalent when using MCE training.
2. A new gradient-based MAP adaptation algorithm (GMAP) that does not make any assumptions concerning the form of the prior distribution is proposed in Chapter 4. This algorithm is shown to outperform the standard MAP approach of Gauvain and Lee [45] for the conditions tested.
3. A new MCE based MAP adaptation algorithm is proposed and tested in Chapter 4. This algorithm too is shown to work better than the standard MAP approach, as well as being better than standard MCE.
4. Bayesian learning is introduced. An original implementation of Bayesian learning for hidden Markov model speech recognition is introduced and discussed. This

is, to the author's knowledge, the first time that Bayesian learning using Markov chain Monte Carlo has been used for hidden Markov model speech recognition.

Chapter 2

Background

In this chapter the basic hidden Markov model (HMM) theory and notation is presented. The implementation details of the HMM software, as well as the speech datasets used in this work are also described. Finally, the bias/variance problem is discussed, relating model complexity to overtraining.

2.1 Hidden Markov models

This section documents the relevant HMM theory, as well as any implementation specific details. The configuration of the base system is also described.

The Hidden Markov model Toolkit for Speech Recognition (HMTSR) used in this thesis was developed by the author and Nieuwoudt [83] during their Ph.D. studies. The toolkit is used by several post-graduate students in the Speech Recognition group at the University of Pretoria.

It is not realistic to present a thorough review of hidden Markov modeling theory in this chapter; the reader is therefore referred to books such as that of Rabiner and Juang

[90] or their article "An introduction to Hidden Markov Models" [94, 92].

2.1.1 Feature extraction

Thirteen Mel-frequency cepstral coefficients (MFCCs), along with their first and second order differentials are used. The following describes the feature extraction process:

- A first order filter is used to pre-emphasize the speech signal [90, p. 112], to spectrally flatten the signal and to limit finite precision effects later in feature extraction. The filter transfer function used is

$$H(z) = 1 - a \cdot z^{-1}, \tag{2.1}$$

  where $a = 0.98$ was empirically found to work well.

- The preemphasized speech signal is blocked into frames [90, p. 113] of length 16ms, with overlap of 6ms. The frames are therefore 10ms apart. The number of samples (N_s) is determined by the block length (time) and the sampling rate. A sampling rate of 16kHz would therefore result in a block of 256 samples.

- A Hamming window [90, p. 114] is used to minimize signal discontinuities at the beginning and end of each frame. A Hamming window has the following form:

$$w(n) = 0.54 - 0.46cos(\frac{2\pi n}{N_s - 1}) \qquad 0 \leq n \leq N_s - 1. \tag{2.2}$$

- The power spectrum of the windowed frame is calculated using the fast Fourier transform.

- The power spectrum is filtered using N_f mel-spaced filters [90, p. 183-190]. The filters have triangular bandpass frequency responses. The number of filters is set using the following formula $N_f = round(0.0015 \cdot S_r)$, where S_r is the sampling rate. A sampling rate of 16kHz therefore results in 24 mel-spaced filters being used.

- The discrete cosine transform (DCT) of the natural logarithm of the 24 filter outputs is computed.
- The first 13 DCT coefficients are kept as the MFCCs.
- A second-order polynomial fitting [90, p. 194] of 5 consecutive MFCCs is used to estimate the first and second order derivatives of the MFCCs. This incorporates temporal information (external to the frame) into the features.

Note that the values in the above feature extraction process were determined empirically and will not necessarily perform best in all circumstances.

2.1.2 Continuous density hidden Markov models

A continuous density HMM is characterized by the following:

1. The number of states N. The individual states are labeled as $\{1, 2, \ldots, N\}$, and the state at time t is denoted as q_t.
2. The number of mixture components per state M. Figure 2.1 shows a continuous density HMM with 3 states ($N = 3$) and 4 Gaussian mixture components ($M = 4$).
3. The state-transition probability distribution $A = \{a_{ij}\}$, where a_{ij} is the probability of being in state j at time $t + 1$ after having been in state i at time t, i.e.
$$a_{ij} = P(q_{t+1} = j | q_t = i), \qquad 1 \leq i \leq N, 1 \leq j \leq N + 1. \tag{2.3}$$
Note that the probability of *leaving* an HMM from state i is given by $a_{i(N+1)}$.

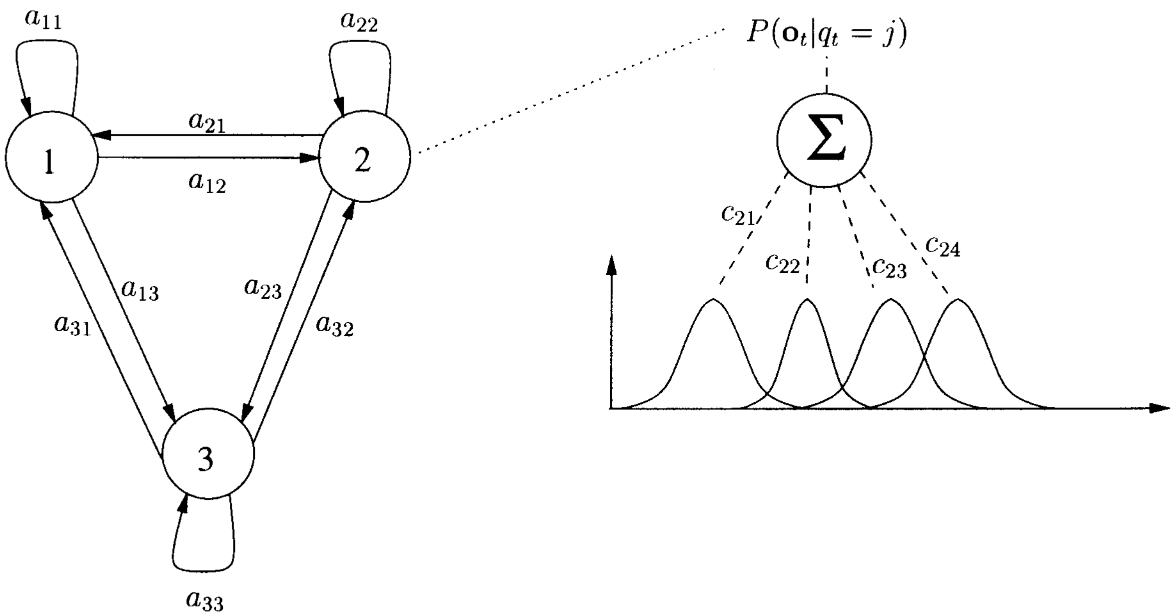


Figure 2.1: An $N = 3$ state $M = 4$ Gaussian mixture HMM.

4. The observation probability distribution $B = \{b_j(\mathbf{o}_t)\}$ for the observation $\mathbf{o}_t$, where

$$b_j(\mathbf{o}_t) = P(\mathbf{o}_t|q_t = j) \qquad 1 \leq j \leq N. \tag{2.4}$$

For a continuous density HMM the observation probability is represented as a finite mixture of the form

$$b_j(\mathbf{o}) = \sum_{k=1}^{M} c_{jk}\mathcal{N}(\mathbf{o}, \mu_{jk}, \mathbf{\Sigma}_{jk}). \tag{2.5}$$

5. Initial state probability distribution $\pi = \pi_i$, where

$$\pi_i = P(q_1 = i), \qquad 1 \leq i \leq N. \tag{2.6}$$

A complete specification of an HMM includes the parameters N, M, a_{ij}, π_i, and the mixture parameters c_{jk}, μ_{jk} and Σ_{jk}. The complete parameter set above of an HMM will be represented as

$$\theta = (A, B, \pi). \tag{2.7}$$

The MFCC features are largely uncorrelated. A diagonal covariance matrix (Σ_{jk}) is therefore used in this work, which greatly reduces the number of parameters that are used.

Left-to-right hidden Markov models limit transitions to forward transitions only, i.e. $a_{ij} = 0 \quad \forall\, j < i$. Left to right hidden Markov models with no skipping transitions are used in this work where $a_{ij} = 0 \quad \forall\, j \neq i$ and $j \neq (i+1)$.

The probability of an observation sequence $\mathbf{O} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$ given the model θ ($P(\mathbf{O}|\theta)$) can be obtained by summing over all possible state sequences $\mathbf{q} = (q_1 q_2 \ldots q_T)$,

$$P(\mathbf{O}|\theta) = \sum_{\text{all } \mathbf{q}} P(\mathbf{O}, \mathbf{q}|\theta) = \sum_{\text{all } \mathbf{q}} P(\mathbf{O}|\mathbf{q}, \theta) P(\mathbf{q}|\theta). \tag{2.8}$$

Assuming statistical independence of observations, we can write $P(\mathbf{O}, \mathbf{q}|\theta)$ as

$$\begin{aligned} P(\mathbf{O}, \mathbf{q}|\theta) &= \pi_{\mathbf{q}_0} b_{\mathbf{q}_1}(\mathbf{o}_1) \prod_{t=2}^{T} \Big(a_{\mathbf{q}_{t-1}\mathbf{q}_t} \cdot b_{\mathbf{q}_t}(\mathbf{o}_t) \Big) \\ &= \prod_{t=1}^{T} \Big(a_{\mathbf{q}_{t-1}\mathbf{q}_t} \cdot b_{\mathbf{q}_t}(\mathbf{o}_t) \Big), \end{aligned} \tag{2.9}$$

where $a_{\mathbf{q}_0\mathbf{q}_1}$ is taken to be $\pi_{\mathbf{q}_0}$.

The Viterbi algorithm is used to find the single best state sequence $\mathbf{q} = (q_1 q_2 \ldots q_T)$ and its probability $P(\mathbf{O}, \mathbf{q}|\theta)$, for a given observation sequence $\mathbf{O} = (\mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_T)$. The Forward-Backward procedure is used to determine the probability of an observation sequence given the model θ, i.e., $P(\mathbf{O}|\theta)$. Due to efficiency considerations, the best state sequence is used for recognition and some training procedures, as opposed to using all possible state sequences.

2.1.3 Training

The hidden Markov models are trained in at least three phases, namely

1. initialization,
2. segmental training
3. maximum likelihood training.

Embedded training is often used after the above training steps. A short discussion of the training procedures and related algorithms follows:

Initialization The HMM parameters are initialized as follows:

- The transition probabilities, initial probabilities, number of states and Gaussians per state are set by the user in a configuration file. In this work, the transition probabilities were always initialized as

$$a_{ij} = \begin{cases} 0.9 & j = i, \\ 0.1 & j = i+1, \\ 0 & \text{otherwise,} \end{cases} \tag{2.10}$$

  with the initial probabilities being initialized as

$$\pi_i = \begin{cases} 1 & i = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{2.11}$$

- The Gaussian mixture weights are initialized as $1/M$, for an M Gaussian mixture state.

- Labeled data is divided into equal sized segments, one segment for each state in the HMM. The speech segments are used to initialize the HMM mixture means and variances by using the segmental K-means algorithm to cluster the features into M clusters (M Gaussian mixture state), the means and variances of which are used to initialize the mixtures for the associated state.

Alternately, in the absence of labeled data, utterances are divided into equal sized blocks using the transcription given. Further training then proceeds using search-based training.

Maximum likelihood training The Baum-Welch method, also known as expectation maximization (EM) [28], is used to iteratively maximize the likelihood $P(\mathbf{O}|\theta)$.

To describe the procedure of segmental re-estimation of the parameters, we first define $\xi(i,j)$, the probability of being in state i at time t and state j at time $t+1$. We also define $\gamma_t(i)$ as the probability of being in state i at time t and $\gamma_t(j,k)$ as the probability of being in state j at time t with the kth mixture component accounting for $\mathbf{o}_t$.

The forward-backward algorithm [90, p. 334] is used to estimate the above probabilities given the current model. The parameter re-estimation formulas used are,

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.12}$$

$$c_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T} \sum_{m=1}^{M} \gamma_t(j,m)} \tag{2.13}$$

$$\mu_{jkl} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot \mathbf{o}_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{2.14}$$

$$\sigma_{jkl} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot (\mathbf{o}_t - \mu_{jkl})(\mathbf{o}_t - \mu_{jkl})'}{\sum_{t=1}^{T} \gamma_t(j,k)}. \tag{2.15}$$

Segmental training The forward-backward algorithm used in maximum likelihood training is relatively computationally expensive and slow. A segmental training step [61] is therefore used to quickly obtain a relatively good estimate of the model parameters (prior to using the EM/ML algorithm). The Viterbi algorithm is used to perform a forced-alignment, in which the best state sequence is obtained. The state alignment is then used to reestimate the model parameters. This is often called segmental training, as the observation sequence is segmented (state aligned), with the segments being used to reestimate the parameters of the associated states. It would, however, only be fair to mention that the forward-backward algorithm does have a stronger theoretical background.

The same re-estimation formulas as that used in the ML/EM algorithm (Eqs. (2.12) to (2.15)) are also used here. The probabilities $\xi_t(i,j)$, $\gamma_t(i)$ and $\gamma(j,k)$ are, however, estimated as follows

$$\xi_t(i,j) = \delta(\mathbf{q}_t - i)\delta(\mathbf{q}_{t+1} - j) \tag{2.16}$$

$$\gamma_t(i) = \delta(\mathbf{q}_t - i) \tag{2.17}$$

$$\gamma_t(j,k) = \delta(\mathbf{q}_t - j)\frac{c_{jk}\mathcal{N}(\mathbf{o}_t, \mu_{jk}, \Sigma_{jk})}{b_j(\mathbf{o}_t)} \tag{2.18}$$

where $\delta()$ is the Kronecker delta function. The Kronecker delta function is defined as

$$\delta(i) = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.19}$$

Embedded training It is often advantageous to use a search algorithm (see Section 2.1.5), not only to perform a state alignment, but also HMM alignment. Here a search algorithm is used along with the transcription of the speech units to automatically align (segment) the speech data (HMM alignment). We are therefore not relying on any manually created labels, but allowing the current model to determine where a label should begin and end. The search is limited to the transcription of the relevant utterance. Some optional HMM models are permitted inside the transcription, this is to allow models such as silence to be inserted (by the search), where they might not occur in the transcription. The re-estimation is performed in exactly the same way as done in the segmental training step. In some situations, where some or all of the data is not labeled, this step is essential. For a dataset which is not labeled at all, such as the TIDIGIT dataset, only the initialization step and embedded training is performed (the other training phases rely on labelled data).

Mixture splitting An alternative that has been found to work well is Gaussian splitting [100, 87]. Here, we initialize only one mixture component per state and then split the mixture component with the highest mixture component weight into two separate Gaussians. This is done at the end of each training iteration, until all allowed mixtures components have a non-zero mixture weight. The Gaussians are split in the direction of maximum variance. This is done by setting the mean and variance of the new Gaussian equal to the existing Gaussian and then moving the mean vectors a small distance away from each other in the direction of maximum variance. The weights of the two new Gaussians are set to be equal to one half that of the original Gaussian.

Figure 2.2 shows a two-dimensional example of a two-component Gaussian mixture where the component with the largest weight (0.7) is split into two new Gaussians, creating a mixture containing three Gaussians. Note that the mean and variance of the new Gaussians will tend to change considerably after the next training iteration. If, during training, a mixture weight were to become zero, the Gaussian associated with the largest weight will be split. This technique has been used for all experiments in this work.

2.1.4 Duration modeling

Hidden Markov models implicitly model state duration probability with a Geometric distribution, i.e.

$$p_i(d) = a_{ii}^{-1}(1 - a_{ii}), \tag{2.20}$$

which is the probability of d consecutive observations in state i. This exponential state

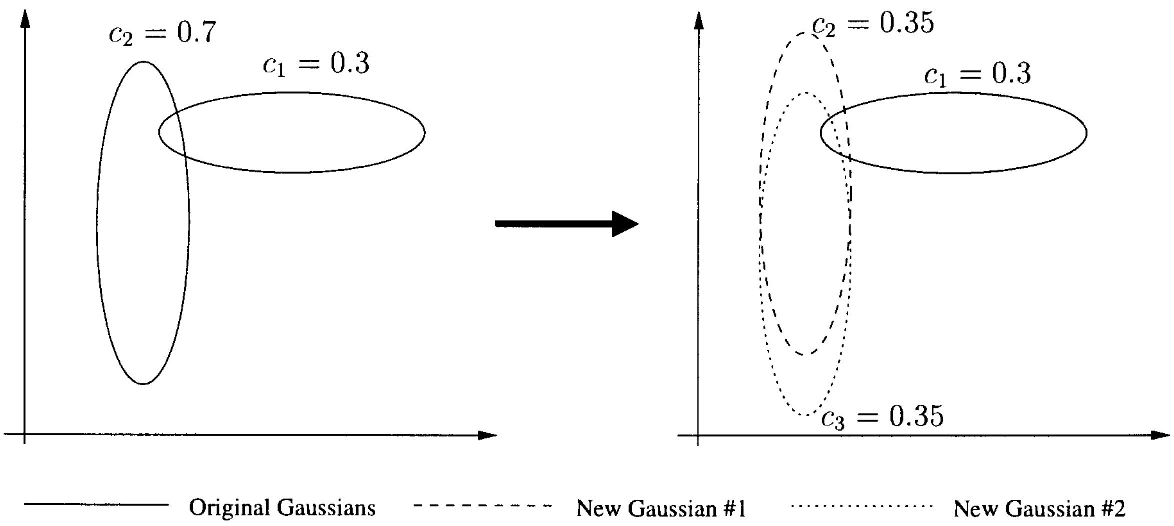


Figure 2.2: Two-dimensional example of mixture component splitting.

duration density is inappropriate for most systems. We would prefer to explicitly model the duration probability in some analytic form. Initial duration modeling approaches [90] assigned each state a discrete duration probability, which was incorporated into the search algorithm. These algorithms were, however, computationally expensive. Post-processing approaches [91, 93], which were more computationally efficient, added the duration contribution to the Viterbi metric after candidate paths had been identified. If the best path is not one of the candidate paths, then this method fails.

Burshtein [16] proposed a duration modeling approach, which adds the duration metric at each frame transition in the Viterbi algorithm.

The gamma distribution,

$$p(d) = \frac{\alpha^\beta}{\Gamma(\beta)} x^{\beta-1} e^{-\alpha x}, \quad 0 < d < \infty, \tag{2.21}$$

is often used to model the duration density, with $\alpha > 1$ and $\beta > 0$.

The modified Viterbi algorithm keeps track of the duration $d_s(t)$ of each state s at time t. Letting M_i denote the duration, at which the gamma distribution of state i is a maximum, then the duration penalty P_{ij} of making a transition from state i at time t to state j at time t+1 is

$$P_{ij} = \begin{cases} 0 & d_i(t) < M_i \text{ and } i = j \\ log(\frac{d_i(t+1)}{d_i(t)}) & d_i(t) \geq M_i \text{ and } i = j \\ log(d_i(t)) & d_i(t) < M_i \text{ and } i \neq j \\ log(M_i) & d_i(t) \geq M_i \text{ and } i \neq j. \end{cases} \tag{2.22}$$

Auto-transitions are therefore not penalized before the duration is M_i. After the duration M_i, we penalize gradually. Upon exit from the state, the overall duration metric is $log(d_i(t))$, which is as it should be. Word duration modeling is implemented in much the same way.

The duration model parameters are estimated using the Viterbi state-alignment. After determining the mean and variance of the duration from the state alignment, the parameters α and β are estimated for each state using Eqs. (A.14) and (A.15). A similar approach to duration modeling was also proposed by Du Preez [35]. Burshtein's approach has been implemented and is used, where indicated, in this thesis.

2.1.5 Search

Continuous speech recognition requires the segmentation and labeling of continuous speech. Fortunately, there are efficient methods of segmenting and labeling continuous speech at word and phoneme levels. Various (search) algorithms exist, each with advantages and disadvantages. However, only those algorithms implemented and used in this work will be described here.

Grammar networks and language models A grammar network/language model is used to determine which transitions may be taken, or what probability there is of taking a certain transition. A grammar is an explicit set of rules limiting which models/words/phonemes may follow others. The grammar is implemented as a finite state network (FSN), which is created from a text based grammar. Note that each FSN node is associated with a single HMM, but there may be multiple nodes associated with the same HMM.

Trellis search The trellis search algorithm [106] is a frame-synchronous implementation of the Viterbi search algorithm, which allows transitions between models or grammar nodes.

N-best search It is often necessary to obtain the N-best HMM sequences. The N-best strings are often used to obtain a confidence measure for a recognition output. MCE training uses the N-best strings to obtain a measure of misclassification. If the problem is simple enough, the N-best HMM sequences can be obtained by aligning all possible HMM sequences. This is, however, not feasible for most, if not all, speech recognition applications. As a result, efficient N-best search algorithms have been developed [106, 103]. The disadvantage, is that none of these algorithms are guaranteed to give the true N-best sequences (although most times there will only be small differences in probability and not the sequences).

The N-best search implemented and used in this work is the tree-trellis algorithm proposed by Soong and Huang [106]. The search comprises two stages, namely

1. Modified Viterbi algorithm (Trellis search algorithm)

   The standard trellis search algorithm is modified, to store a partial path map. The partial path map contains the likelihood scores of all partial paths leading to every grammar node from all previous nodes.

2. A^* Tree search

   After the trellis search algorithm has been performed, a backward tree search is started from the terminal node. This part of the search is done time asynchronously. The search tree is implemented using the A^* algorithm [99, 86]. The A^* search restricts the search by using an admissible heuristic h, such that h never overestimates the cost to reach the goal. Here, we use the partial path map, which gives the exact cost.

2.2 Overtraining

Overspecialization (or overtraining) occurs in most training algorithms where a finite number of examples are available for training. If the training data set was perfectly representative of the test set (this would only truly occur when an infinite number of training examples were available), there would be no difference between training set and testing set performance. However, in practice data sets are limited in size and the test set performance tends to be worse than the training set performance. This is a result of the model becoming too specialized and not generalizing well.

Model complexity influences overtraining, especially when little training data is available. This phenomenon can be described in terms of the bias/variance problem.

2.2.1 The Bias/Variance Dilemma

Here, due to its less complex and easily interpreted nature, the bias/variance problem will be discussed within a regression framework. It is, however, just as relevant to classification problems.

The regression problem is to create a function $f(\mathbf{x}|\lambda)$ using a training set $\mathcal{D}$, with $\mathcal{D} = (x_1, y_1), \ldots, (x_n, y_n)$, where we wish to estimate $\mathbf{y}$ for an observation $\mathbf{x}$. Typically, the function f is fixed and we wish to merely estimate the function parameters λ. Given the data ($\mathcal{D}$), and $\mathbf{x}$, an appropriate measure of the suitability of $f(\cdot|\lambda)$ as a predictor of $\mathbf{y}$ is the mean-squared error (MSE), or

$$E[(\mathbf{y} - f(\mathbf{x};\mathcal{D}))^2|\mathbf{x},\mathcal{D}].$$

It can easily be shown [46] that

$$\begin{aligned} E[(\mathbf{y} - f(\mathbf{x};\mathcal{D}))^2|\mathbf{x},\mathcal{D}] &= E[(\mathbf{y} - E[\mathbf{y}|\mathbf{x}])^2|\mathbf{x},\mathcal{D}] \\ &+ (f(\mathbf{x};\mathcal{D}) - E[\mathbf{y}|\mathbf{x}])^2. \end{aligned} \tag{2.23}$$

The expectation $E[(\mathbf{y} - E[\mathbf{y}|\mathbf{x}])^2|\mathbf{x},\mathcal{D}]$ is not dependent on the data or on the estimator f. The distance $(f(\mathbf{x};\mathcal{D}) - E[\mathbf{y}|\mathbf{x}])^2$ therefore measures the effectiveness of f as a predictor of $\mathbf{y}$. The mean-squared error of f as an estimator of the regression $E[\mathbf{y}|\mathbf{x}]$ is

$$E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D}) - E[\mathbf{y}|\mathbf{x}])^2], \tag{2.24}$$

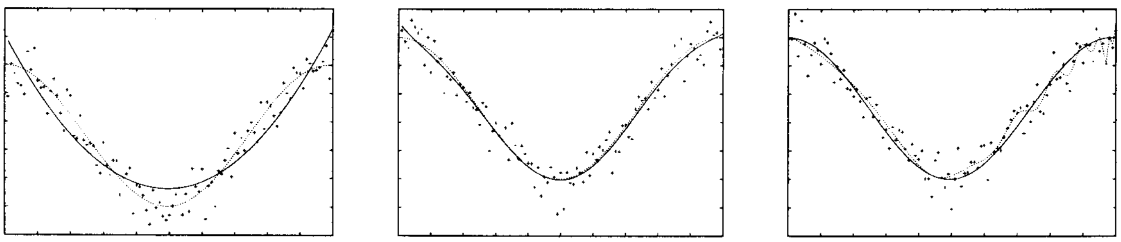
where $E_{\mathcal{D}}$ is the expectation with respect to the training set $\mathcal{D}$, i.e. the average over the ensemble of possible datasets $\mathcal{D}$. Equation (2.24) can be rewritten in terms of bias and variance [46],

$$\begin{aligned} E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D}) - E[\mathbf{y}|\mathbf{x}])^2] &= (E_{\mathcal{D}}[f(\mathbf{x};\mathcal{D})] - E[\mathbf{y}|\mathbf{x}])^2 \\ &+ E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[f(\mathbf{x};\mathcal{D})])^2]. \end{aligned} \tag{2.25}$$

The term $(E_{\mathcal{D}}[f(\mathbf{x};\mathcal{D})] - E[\mathbf{y}|\mathbf{x}])^2$ is the bias of the estimator and measures any systematic tendency for it to give the incorrect answer. An estimator (or classifier) is said to be biased when the bias term is non-zero. The term $E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[f(\mathbf{x};\mathcal{D})])^2]$ is the variance in the estimator error, and measures the sensitivity of the estimator to any randomness in the training examples.

The bias and variance of an estimator are typically affected by, among others, model type, model complexity and the parameter estimation algorithm. Unfortunately, reducing the bias typically increases the variance (and *vice versa*). Reducing the sum of the bias and variance (or mean-squared error of f), therefore generally requires a trade-off between their contributions.

The trade-off between bias and variance is usually optimized by varying the complexity of the model. This trade-off between bias and variance can be illustrated using a one-dimensional regression problem. Figure 2.3 shows one such example. In this case, a polynomial of degree n is fitted to the noisy data by minimizing the mean-squared error. Results using $n = 2$, $n = 5$ and $n = 50$ are shown.



$MSE = 0.0107$ (a) $MSE = 0.0005$ (b) $MSE = 0.0021$ (c)

Figure 2.3: One hundred observations of the raised cosine function $(0.5cos(10x/pi) + 1)$ plus noise. Noise has zero-mean Gaussian distribution, with standard deviation 0.1. The solid curve is the target function, and dotted curve the polynomial fit of degree n, with (a) $n = 2$, (b) $n = 5$ and (c) $n = 50$.

The polynomial fit in (a) is relatively poor, and is a result of the inability of the model (or function) to represent the underlying process. The solution in (a) has a large mean-squared error due mainly to the bias term. The polynomial fitted in (c) has a high degree n, and has started fitting the noise; here the variance term accounts for most of the mean-squared error. The polynomial fit shown in (b) is the result of a model which is complex enough to fit the true underlying process (low bias), yet simple enough such that the noise in the data is not modeled as well (low variance).

The optimal choice of model complexity will also vary with the amount of available training data. More training data will reduce the variance and so more complex models can be used. However, when there is relatively little data available, less complex models will be preferred.

The principles in the above discussion are incorporated in *Occam's Razor*. Simply stated, *Occam's Razor* is a principle that states that unnecessarily complex models should not be preferred to simpler ones. The bias/variance dilemma is often problematic when using single point estimates in sparse training data situations. The effects of the bias variance problem will be encountered in Chapter 4 where, for certain sparse data scenarios, less complex models are preferable. In Chapter 5 the usage of Bayesian learning is investigated. Bayesian learning reduces the variance of the estimate or solution and therefore results in more complex models being preferred over less complex ones in such sparse data scenarios.

2.3 Experimental procedure

The main goal of the experiments designed in this study is to investigate the relative effectiveness of the algorithms proposed in comparison with conventional algorithms. This work therefore attempts to keep as much in common for all other aspects of the speech models associated with both the conventional and the proposed techniques and to keep the recognizer structure as simple as possible. The basic HMM system described in Section 2.1 has been used throughout.

There is little point in presenting discrete phoneme recognition (phoneme classification) results as this can be misleading. A phoneme classifier that only chooses the most frequently observed phonemes will tend to perform well for phoneme classification. Such a classifier will, however, not necessarily work well for other tasks such as continuous word or phoneme recognition. Continuous phoneme recognition results are therefore reported throughout this thesis. The recognition accuracy of the system is reported, where accuracy is defined as:

$$Accuracy = \frac{Phones - Subs - Dels - Ins}{Phones}, \tag{2.26}$$

where $Phones$ refers to the number of phones in the correct transcription, $Subs$ the number of substitutions, $Dels$ the number of deletions and Ins the number of insertions. Error rates reported are simply $100\% - Accuracy$.

It is necessary to determine whether the recognition accuracy of a new system is better than that of an existing baseline or reference system. A test of significance can therefore be performed to determine whether this is probably true or not. We must therefore decide between the two hypotheses:

H_0: $P_n > P_b$,

H_1: $P_n <= P_b$,

where P_n is the recognition accuracy of the new system and P_b is the baseline or reference accuracy. A one-tailed test is used, since we are interested in determining whether the improvement in recognition accuracy is better than a reference system performance.

The maximum probability with which we would be willing to risk the error of rejecting a hypothesis when we should have accepted it is called the level of significance. If, for example, a level of significance of 0.01 were attained, then we would be 99% confident that we had made the correct decision in accepting the hypothesis.

Table 2.1 details the absolute improvements in recognition accuracy required to attain a 0.05 and 0.01 level of significance for the three speech databases used (described in Section 2.4). The baseline accuracy is assumed to be 50%, i.e. the worst case improvement required (a 70% baseline performance, for example, will require a smaller improvement to reach the same level of significance).

Table 2.1: Improvements in recognition accuracy required to attain a 0.05 and 0.01 level of significance for the three speech databases used. The baseline accuracy is assumed to be 50%. The number of phonemes in the relevant testing sets, used to determine the significance level, are also listed.

Database	Phonemes	Level of significance	
		0.05	0.01
TIMIT	59858	0.34%	0.48%
TIDIGITS	28330	0.49%	0.69%
SUNSpeech	9026	0.87%	1.23%

The improvements in accuracy required (Table 2.1) are typically smaller than one percent, but greater than 0.1%. Results will therefore be specified to one decimal place. In this work, improvements that are significant to a level of 0.01 will simply be referred to as *significant*.

2.4 Speech datasets

This section describes the speech corpora used in this thesis. Three different datasets are used, namely: TIMIT, TIDIGITS and SUNSpeech.

2.4.1 TIMIT

The TIMIT [4] corpus of read speech was designed to provide speech data for the acquisition of acoustic-phonetic knowledge and for the development and evaluation of automatic speech recognition systems. TIMIT contains a total of 6300 sentences, 10 sentences spoken by each of 630 speakers from 8 major dialect regions of the United States.

The sentences found in TIMIT consist of 2 dialect "shibboleth" sentences, 450 phonetically compact sentences, and 1890 phonetically diverse sentences. The dialect sentences (SA sentences) were meant to expose the dialectal variants of the speakers and were read by all 630 speakers. The phonetically-compact sentences were designed to provide a good coverage of pairs of phones, while the phonetically-diverse sentences (the SI sentences) were selected from existing text sources so as to add diversity in sentence types and phonetic contexts. Table 2.2 summarizes the sentences found in the TIMIT database.

Table 2.2: Summary of sentences found in TIMIT

Sentence type	Unique sentences	Number of speakers	Total	Sentences per speaker
Dialect (SA)	2	630	1260	2
Compact (SX)	450	7	3150	5
Diverse (SI)	1890	1	1890	3
Total	2342		6300	10

Following convention the standard TIMIT 39-phone set is used. The 63 phones found in the TIMIT database are reduced to 39 by combining models as done by Lee [68].

Training and testing sets

The TIMIT dataset is used in experiments in Chapters 3, 4 and 5. In Chapter 3 overtraining is investigated using the suggested training and testing sets. Chapters 4 and 5 however, present gender adaptation experiments and limited training data experiments using TIMIT.

A reduced female training set is used for this purpose, so as to create a scenario where there is limited training data for female speakers. The small female training set consists of speech data from two speakers from each of the eight dialect regions. No sentence text appears in both the training and test sets. A small gender independent training set will be required for testing the algorithms presented in Chapters 3 and 5. One such set has been created by randomly selecting two male and two female speakers from each of the eight dialect regions found in the TIMIT dataset.

Table 2.3 describes the different training and test sets used. Gender specific testing sets are used in Chapters 4 and 5.

Table 2.3: Description of TIMIT training and testing sets used

Description	Label	Number of speakers			Number of Sentences	Duration (minutes)
		Male	Female	Total		
Training sets:						
Full (standard)	T	326	136	462	4620	236.5
Male	T_M	326	0	326	3260	165.2
Female	T_F	0	136	136	1360	71.3
Female-small	T_{FS}	0	16	16	160	8.2
Small (gender indep.)	T_S	16	16	32	320	16.1
Testing set (standard)		112	56	168	1680	86.4

Use in literature

The TIMIT dataset has been used extensively in the speech recognition literature to experimentally test algorithms and hypotheses. The following summarizes a few such cases, so as to ensure that the reader has a fair idea of what performance can be expected for TIMIT. Only system configurations similar to that used here are reported.

Rathinavela and Deng [96] investigated the usage of state-dependent linear transforms of Mel-warped DFT features. The authors performed discrete (not continuous) phoneme recognition to test ML and MCE trained HMMs, as well as their "optimum-transformed HMM". Table 2.4 summarizes the results reported for simple left-to-right 3 state, 5 mixture HMM models.

Table 2.4: Summary of TIMIT results reported in [96]

Model/Training	Phonetic classification rate (%)
ML-HMM	59%
MCE-HMM	66%
MCE-THMM	69%

McDermott [75] used the TIMIT database to evaluate the effectiveness of MCE for continuous speech recognition. Table 2.5 summarizes the TIMIT results presented by McDermott in [75].

Table 2.5: Summary of TIMIT accuracy results reported by McDermott [75]

Number of mixtures	ML	MCE	ML+bigrams	MCE+bigrams
1	48.8		56.0	61.0
4	55.0	61.9	62.4	66.2
8	57.1	62.7	64.8	67.3
16	59.9	63.2	66.8	68.7

Yuk and Flanagan [119] investigate the use of neural network based adaptation methods applied to telephone speech recognition using TIMIT and NTIMIT [56]. Recognition accuracy of 62.2% is reported for TIMIT using their base system, a 3 state left-to-right mono-phone HMM with 30 Gaussian distributions per state.

Moreno and Stern [80] compared speech recognition accuracy for high quality recorded speech and speech as it appears over long-distance telephone lines. The performance of the CMU SPHINX system was compared for the TIMIT and NTIMIT [56] databases. Recognition accuracy of 52.7% was reported for the TIMIT test set.

2.4.2 TIDIGITS

The TIDIGITS [70] was designed and collected for the purpose of designing and evaluating algorithms for speaker-independent recognition of connected digit sequences. The corpus contains read utterances from 326 speakers (111 men, 114 women, 50 boys, and 51 girls) each speaking 77 digit sequences. The data was collected in a quiet environment and digitized at 20 kHz.

The digit sequences are made up of the digits: "zero", "oh", "one", "two", "three", "four", "five", "six", "seven", "eight", and "nine". The 77 digit sequences spoken by each of the speakers can be broken up as follows: 22 single-digit sequences (2 of each of the 11 digits) and 11 each of randomly generated 2,3,4,5 and 7-digit sequences.

Training and testing sets

The database is divided into two subsets, one to be used for algorithm design and the other for evaluation. The division yielded speaker independent training and testing sets, each containing half of the male and female speakers. The boys and girls test and training utterances are not used in this work.

As with TIMIT, the TIDIGIT dataset is used for gender adaptation and reduced data experiments in later chapters. It is for these experiments that reduced subsets are created for female speakers in the dataset. A reduced speaker set (T_{WS}) is created, using four randomly chosen female speakers, using all 77 digit sequences per speaker. The above set is further reduced by using only 7 digit sequences per speaker (2 single-digits and one each of 2,3,4,5 and 7 digit sequences). Table 2.6 presents the training and testing sets used for the TIDIGITS corpus.

Table 2.6: Training and testing sets used with the TIDIGIT database

Description	Label	Number of speakers			Digit	Duration
		Male	Female	Total	sequences	(minutes)
Training sets:						
Full (standard)	T	55	57	112	8624	253.4
Man	T_M	55	0	55	4235	121.9
Woman	T_W	0	57	57	4389	131.5
Woman-small	T_{WS}	0	5	5	385	10.1
Woman-very-small	T_{WVS}	0	5	5	35	55 s
Testing sets:						
Full		55	57	112	8623	254.4
Man		55	0	55	4235	123.1
Woman		0	57	57	4389	131.3

Use in literature

Normandin [87] proposed an approach for splitting Gaussian mixture components based on maximum mutual information estimation (MMIE) training. Experiments using the TIDIGITS dataset were conducted. Recognition word (digit) error rates of between 1.6% (1 mixture per state) and 1.0% (16 mixtures per state) were obtained for models with variable numbers of states. Utilizing their mixture splitting algorithm, however, resulted in a digit error rate of 0.71%.

Jiang *et al.* [57] investigated a new Bayesian predictive classification (BPC) approach for robust speech recognition where a mismatch between training and testing conditions occurred. TIDIGITS was used, along with other datasets, to test their new approximate BPC algorithm. Using a 10 state, 10 mixture per state continuous density HMM, digit error rates of 2.2% and 2.4% were reported for their baseline system and new BPC system respectively, when using the standard TIDIGITS dataset.

2.4.3 SUNSPEECH

The SUN Speech database was compiled by the Department of Electrical and Electronic Engineering of the University of Stellenbosch to contain phonetically labelled speech in both English and Afrikaans. The data was recorded in a controlled environment, with 12 bit resolution and a 16kHz sampling rate. Sixty sentences comprising four sentence sets were chosen to exhibit the diversity of phonemes in the two languages. Details of the number of speakers and the number of sentences spoken by each group of speakers are given in Table 2.7.

Table 2.7: Description of SUN Speech database: number of male and female speakers and total number of speakers for each sentence set

Language	Number of speakers			Sentence set	Number of sentences
	Male	Female	Total		
Afrikaans	24	16	40	1	10
	18	12	30	2	10
English	33	17	50	3	20
	22	4	26	4	20

A total of 59 phonetic categories, including both a *silence* and *unknown* category, were used to segment both the Afrikaans and the English speech. It was attempted to assign the labels phonetically, i.e. according to the sound produced, rather than phonologically assigning the labels, i.e. according to what was supposed to be said.

Training and testing sets

The SUN Speech database is used for cross-language adaptation experiments in this work. Subdivision of the dataset was therefore dictated by the requirements for cross-language adaptation.

A speaker independent (SI), sentence independent division of the Afrikaans data can be obtained by using data from the first sentence set for training and data from the second sentence set for testing. We however, need to create a smaller training subset, so as to recreate a scenario where extremely little Afrikaans training data is available.

The database is not entirely consistent in that some speakers, but not all, spoke sentences from more than one sentence set. Those speakers who spoke all of the Afrikaans sentences were used to create a reduced Afrikaans set. This has the added utility of creating a speaker-dependent (SD) test set; although this set is not used in this thesis. The reduced Afrikaans training set will be referred to as the "adaptation set", as it is used as such in many of the experiments.

The full English set is used for training purposes (it is not required for testing). Table 2.8 gives the details of the subdivision of the database into training and testing sets.

Table 2.8: Details of SUNSpeech training and testing sets used

Language	Description	Label	Number of Speakers			Sentence set	Duration (minutes)
			Male	Female	Total		
English	Training	E	55	21	76	3 and 4	135.5
Afrikaans	Training set	A	23	16	39	1	22
	Training subset	A_S	2	6	8	1	5.5
	SD test set		2	6	8	2	7.7
	SI test set		14	1	15	2	13

Use in literature

The usage of this database in the speech recognition literature is somewhat limited. Waardenburg *et al.* [115] investigated the isolated recognition of stop consonants using HMMs. More recently, Nieuwoudt and Botha [84, 85] investigated cross-language usage of acoustic information using this dataset, where continuous word recognition results were presented. To the authors knowledge, no continuous phoneme recognition results have been published for this dataset.

Chapter 3

Minimum classification error training

Although there are several learning algorithms (such as *maximum mutual information* and *minimum discriminative information*) which can be classified as discriminative training techniques, this chapter will exclusively describe Minimum classification error learning and several modifications thereof.

3.1 Introduction

Conventional maximum likelihood (ML) estimation attempts to maximize the likelihood of the training data given the model parameters of the corresponding class. The models from other classes do not participate in the parameter estimation. By maximizing the likelihood of the correct model, but not minimizing the likelihood of other competing models, it cannot be guaranteed that the ML models will optimally discriminate against incorrect classes in recognition. Maximum likelihood estimation can be problematic in situations where the distribution of the data to be recognized is significantly different from the distribution of the model, as noted in the literature [81, 14, 75].

A theory of error-corrective training for pattern classification was first proposed by Amari [2], where an adaptive procedure was developed and shown to converge to a local minimum of the classification error function.

Franco and Serralheiro [42] proposed a training procedure which aims explicitly at reducing the recognition error and increasing discrimination between classes. The algorithm is based on a criterion function which is the quadratic error between target state probabilities and the *a-posteriori* state probabilities given the training data. The target state probabilities are forced to be one for the correct state and zero for incorrect states. This criterion function was then used to adjust HMM parameters heuristically to improve the training set recognition rate.

Chen and Soong [19] introduced an N-best candidates based frame-level discriminative training algorithm based on an N-best tree-trellis algorithm [106]. A frame-level loss function was defined and minimized using the gradient descent method. The loss function was defined as a half-wave rectified log-likelihood difference between the correct and selected competing hypotheses. The loss function is accumulated over all training utterances. Their algorithm was tested using a connected Chinese digit recognition experiment and a large vocabulary isolated word experiment. Significant improvements over traditional ML were reported, with the string error rate being reduced from 17.0% to 10.8% for the connected digit experiment and the isolated word recognition error rate being reduced from 7.2% to 3.8%.

Maximum mutual information (MMI) A more formal, information theoretic approach based on the maximum mutual information criterion has been used to train HMM based speech recognition systems [88, 87, 62, 113]. The following description of MMI is a summary of that found in [75]. The MMI approach attempts to find the model parameters θ which minimize the conditional entropy $H_\theta(\mathbf{C}|\mathbf{X})$ of the random variable $\mathbf{C}$ (class) given the random variable $\mathbf{X}$ (data), i.e.

$$H_\theta(\mathbf{C}|\mathbf{X}) = -\sum_{c,x} P(\mathbf{C}=c, \mathbf{X}=x) log P_\theta(\mathbf{C}=c|\mathbf{X}=x), \tag{3.1}$$

which represents the uncertainty of $\mathbf{C}$ given that we have observed $\mathbf{X}$. The entropy of a discrete random variable $\mathbf{C}$, which is a measure of the uncertainty of $\mathbf{C}$, is defined as

$$H_\theta(\mathbf{C}) = -\sum_{c} P(\mathbf{C}=c) log P_\theta(\mathbf{C}=c). \tag{3.2}$$

The information provided by $\mathbf{X}$ about $\mathbf{C}$ can then be defined as

$$I_\theta(\mathbf{C};\mathbf{X}) = H_\theta(\mathbf{C}) - H_\theta(\mathbf{C}|\mathbf{X}). \tag{3.3}$$

Minimizing the conditional entropy, can therefore be accomplished by maximizing the mutual information between $\mathbf{C}$ and $\mathbf{X}$, $I_\theta(\mathbf{C};\mathbf{X})$, i.e.

$$I_\theta(\mathbf{C};\mathbf{X}) = \sum_{c,x} P(C=c, X=x) log \frac{P_\theta(C=c, X=x)}{P_\theta(C=c)P_\theta(X=x)}. \tag{3.4}$$

MMI maximizes the difference between $P_\theta(C=c, X=x)$ and $P_\theta(X=x) = \sum_c P_\theta(C = c, X = x)$. Unfortunately, maximizing the mutual information does not necessarily minimize the classification error.

Minimum classification error Juang and Katagiri [60] proposed a new formulation for the minimum classification error problem, together with a fundamental technique for designing classifiers that approach the objective of minimum classification error. The method was applied to multilayer neural networks, with significant improvements in performance over traditional training methods.

The minimum classification error training method, as introduced by Juang and Katagiri [60], has been used extensively in speech recognition. Applications thereof include training of neural networks [60, 104] and dynamic time warping (DTW) [64, 17, 76, 77] and hidden Markov models. The remainder of this section presents a concise literature survey of the usage of the MCE procedure to train HMM systems for speech recognition.

Chou *et al.* [23] introduced a segmental generalized probabilistic descent (GPD) training algorithm for HMM based speech recognizers using Viterbi decoding. Instead of using the forward-backward procedure, they proposed using the best state sequence obtained using the Viterbi algorithm. Instead of using a complicated constrained GPD algorithm, they apply segmental GPD to transformed HMM parameters, thereby ensuring that the HMM constraints are maintained. They reported results for both the E-set and TIDIGIT database. Significant improvements in phonetic classification from 76% to 88.3% were reported for the E-set problem when using a 10 state, 5 mixture HMM. Their results for the connected digit experiment (TIDIGITS) resulted in an improvement in continuous digit recognition rates from 98.7% to 98.8%. A more general and complete article was later published [59].

Chou *et al.* [24] later introduced a *minimum string error rate* training algorithm, based in the N-best string models. Here, the MCE criterion is applied at string level, with the goal of minimizing the string error rate in continuous and large vocabulary speech recognition tasks. The N most confusable strings are obtained by using the tree-trellis N-best search of Soong and Huang [106]. Their MCE algorithm was tested using the TIDIGIT database and the speaker independent portion of the DARPA naval resource management (RM) speech recognition task. An improvement in string error rate from 1.3% to 1.0% was reported for the TIDIGIT dataset, while a word error rate reduction of 17%-20% was observed when using the DARPA RM task.

McDermott [75] investigated the usage of string-level MCE. A second order optimization algorithm to minimize the string-level MCE criterion was described and found to be a reasonable alternative to the GPD algorithm. McDermott defined a more general MCE loss function which attempted to represent finer grained differences between the correct and incorrect strings, however, no significant advantage was found as a result of using this new loss function. A second loss function, also designed to reflect phoneme/word accuracy was proposed, but not evaluated. The TIMIT dataset was used to compare string-level MCE and the baseline ML-trained HMM systems, the results of which are summarized in Table 2.5.

Kwon and Un [65] proposed a new method of finding discriminative state weights recursively using the MCE algorithm. They relax the HMM constraints on state-weights, such that the sum of the mixture weights for an HMM sum to the number of states. The mixture weights for an individual state can therefore sum to a value greater or smaller than one. This results in what could be called state-weighting, where certain states have higher weights than others. The MCE algorithm was then used to estimate the weights. Experimental results showed that recognizers with phoneme-based and word-based state-weights achieved a 20% and 50% decrease in word error rate respectively for isolated word recognition, and a 5% decrease in error rate for continuous speech recognition.

Other applications of MCE within a speech recognition framework include speaker adaptation [74], keyword spotting [109], speaker identification [105] and feature extraction [9, 8].

The implementation of MCE discussed in this thesis is based on the work of Chou *et al.* [23, 59, 24]. The work presented in this chapter has been summarized in our ICSLP 2000 article [89].

3.2 Minimum classification error training

The aim of minimum classification error (MCE) training is to correctly discriminate the observations of an HMM for best recognition results and not to fit the distributions to the data.

This section will briefly describe and discuss the MCE algorithm; for a more detailed discussion the reader is referred to the original work of Juang and Katagiri [60] and Juang *et al.* [59].

3.2.1 Bayes risk

The following is a brief introduction to Bayes risk and Bayes decisions; for a more detailed text on this subject the reader is referred to the books of DeGroot [27] and Duda and Hart [40]. The optimal choice of answer for an inference problem is a $\theta \in \Theta$ which maximizes the expected utility [7],

$$\int_{\mathcal{X}} u(C(\mathbf{x};\theta)|\mathbf{x})p(\mathbf{x})d\mathbf{x}, \tag{3.5}$$

where $C(\mathbf{x};\theta)$ is the classifiers decision for the observation $\mathbf{x}$, u is a function attaching utilities to each consequence of a decision and $\mathcal{X}$ is the set containing all possible observations. Alternatively, we could work with a loss function $l(C(\mathbf{x})|\mathbf{x})$, where

$$l(C(\mathbf{x})|\mathbf{x}) = f(x) - u(C(\mathbf{x})|\mathbf{x}), \tag{3.6}$$

where f is an arbitrary, fixed function. The optimal solution is then the value of θ which maximizes the expected loss,

$$\int_{\mathcal{X}} l(C(\mathbf{x};\theta)|\mathbf{x})p(\mathbf{x})d\mathbf{x}. \tag{3.7}$$

The conditional loss $l(C_i|\mathbf{x})$, or the risk of classifying the observation $\mathbf{x}$ into class i can be defined as

$$l(C_i|\mathbf{x}) = \sum_{j=1}^{N} \lambda_{ij} P(C_j|\mathbf{x}), \tag{3.8}$$

where $P(C_j|\mathbf{x})$ is the *a-posteriori* probability of choosing the class j given the data $\mathbf{x}$, which can be easily obtained using Bayes' theorem if the class-conditional densities of the data are known. The value λ_{ij} is the cost of classifying a class i observation as class j. Typically, the costs used in the loss function are chosen to be the zero-one loss function or

$$\lambda_{ij} = \begin{cases} 0 & i = j \\ 1 & i \neq j, \end{cases} \tag{3.9}$$

which associates zero cost with correct classifications and unity cost for incorrect classifications. For this special case the conditional loss becomes

$$\begin{aligned} l(C_i|\mathbf{x}) &= \sum_{i \neq j} P(C_j|\mathbf{x}) \\ &= 1 - P(C_i|\mathbf{x}), \end{aligned} \tag{3.10}$$

which is the probability of an error in classification and the minimum risk classifier is the classifier which delivers the minimum classification error. The classifier which achieves minimum loss uses the following decision rule

$$C(\mathbf{x}) = C_i \quad \text{where} \quad i = \underset{j}{argmax} P(C_j|\mathbf{x}). \tag{3.11}$$

3.2.2 Optimization criterion

The error rate for a finite data set is a piecewise constant function of the classifier parameter θ and therefore a poor candidate for optimization using a numerical search. It is therefore necessary to define an optimization criterion which provides a reasonable estimate of the error probability.

For HMMs, the decision rule can be stated as follows,

$$C(\mathbf{x}) = C_i \quad \text{where} \quad i = \underset{j}{argmax} P_j(\mathbf{O}|\theta_j), \tag{3.12}$$

where $P_j(\mathbf{O}|\theta_j)$ is the log-likelihood of the input utterance or observation sequence $(\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n\})$ for the j-th model.

It is therefore necessary to express the operational decision rule (Eq. (3.12)) in a functional form. A class misclassification measure which attempts to emulate the decision rule is therefore defined [59],

$$d(\mathbf{O}) = -\ln[P_i(\mathbf{O}|\theta_i)] + \ln[\frac{1}{N}\sum_{j,j\neq i}^{N} e^{\ln[P_j(\mathbf{O}|\theta_j)]\eta}]^{1/\eta}, \tag{3.13}$$

where η is a positive number, and N is the number of N-best incorrect classes which are used in the misclassification measure. The value of η influences the behaviour of the right-hand term in Eq. (3.13); for $\eta = \infty$ the term becomes $max_{j,j\neq i} P_j(\mathbf{O}|\theta_j)$. A large η will result in the misclassification measure only incorporating the closest incorrect class, whereas a small η will include contributions from all of the incorrect classes. The misclassification measure is a continuous function of the classifier parameters. A value of $d(O) > 0$ implies a misclassification and $d(O) < 0$ a correct decision (for $\eta = \infty$).

The complete loss function is then defined in terms of the misclassification measure using a zero-one loss function,

$$l_i(\mathbf{O};\theta) = l(d(\mathbf{O})). \tag{3.14}$$

It is worth noting, that using $\eta = 1$ and loss function $l(d) = d$ [101] results in an optimization criterion very close to that of MMI (Eq. (3.4)). The zero-one loss function can be any continuous zero-one function, but is typically the following sigmoid function

$$l(d) = \frac{1}{1+e^{-\gamma d+\lambda}}, \tag{3.15}$$

where λ is typically set to zero (or slightly smaller than zero). A loss value less than 0.5 indicates a misclassification has occurred (assuming $\lambda = 0$).

For a given observation, the classifier performance can be written as

$$l(\mathbf{O};\theta) = \sum_{i=1}^{N} l_i(\mathbf{O};\theta) I(X \in C_i), \tag{3.16}$$

where $I(\cdot)$ is the indicator function.

3.2.3 Optimization methods

As mentioned in Section 3.2.1, the optimal solution for an inference problem is that which minimizes the expected loss. For a classification problem involving N classes, the expected loss is defined as

$$L(\theta) = E_{\mathbf{O}}[l(\mathbf{O};\theta)] = \sum_{i=1}^{N} \int_{\mathbf{O}\in C_i} l_i(\mathbf{O};\theta) p(\mathbf{O}) d\mathbf{O}. \tag{3.17}$$

The generalized probabilistic descent (GPD) algorithm [60] is used to minimize the expected loss. The GPD algorithm is given by:

$$\theta_{t+1} = \theta_t - \epsilon_t U_t \nabla l(\mathbf{O};\theta)|_{\theta=\theta_t} \tag{3.18}$$

where U_t is a positive definite matrix, ϵ_t is the learning rate or step size of the adaptation, and θ_t is the model parameters at time step t. It can be shown [60] that the expected loss converges to a local minimum when using the GPD algorithm and the following conditions are satisfied

$$\sum_{t=1}^{\infty} \epsilon_t \to \infty, \quad \text{and} \tag{3.19}$$

$$\sum_{t=1}^{\infty} \epsilon_t^2 < \infty. \tag{3.20}$$

Using the class conditional likelihood function $P_i(\mathbf{O}|\theta_i)$, i.e.

$$P_i(\mathbf{O}|\theta_i) = \sum_{\text{all } \mathbf{q}} P_i(\mathbf{O}, \mathbf{q}|\theta_i), \tag{3.21}$$

where $\mathbf{q}$ is a given state sequence, requires the use of the relatively computationally expensive forward-backward procedure [90, p. 334]. We can, however, also use the maximum of the joint observation-state probability, i.e.

$$P_i(\mathbf{O}|\theta_i) \approx \max_{\mathbf{q}}[P_i(\mathbf{O}, \mathbf{q}|\theta_i)]. \tag{3.22}$$

Since the best state sequence is segmented using the Viterbi algorithm and this segmented sequence is used in the calculation of $P_i(\mathbf{O}|\theta_i)$, this instance of the MCE algorithm based on Eq. (3.22) is often referred to as segmental MCE.

3.2.4 Parameter transformation

The GPD algorithm is an unconstrained optimization technique and given that certain constraints must be maintained for HMMs, some modifications are required. Instead of using a complicated constrained GPD algorithm, Chou *et al.* [23] applied GPD to transformed HMM parameters. The parameter transformations ensure that there are no constraints in the transformed space where the updates occur. The following HMM constraints should be maintained,

1. $\sum_j a_{ij} = 1$ and $a_{ij} \geq 0$,
2. $\sum_k c_{jk} = 1$ and $c_{jk} \geq 0$, and
3. $\sigma_{jkl} \geq 0$.

The parameter transformations given in Table 3.1 are therefore used before and after parameter adaptation (Eq. 3.18).

Table 3.1: Parameter transformations used in MCE.

Parameter	Transformed parameter	Forward transform	Reverse transform	
a_{ij}	$\tilde{a}_{ij}$	$\tilde{a}_{ij} = \ln(a_{ij})$	$a_{ij} = \frac{e^{\tilde{a}_{ij}}}{\sum_j e^{\tilde{a}_{ij}}}$	Transition probabilities
c_{jk}	$\tilde{c}_{jk}$	$\tilde{c}_{jk} = \ln(c_{jk})$	$c_{jk} = \frac{e^{\tilde{c}_{jk}}}{\sum_k e^{\tilde{c}_{jk}}}$	Mixture weights
μ_{jkl}	$\tilde{\mu}_{jkl}$	$\tilde{\mu}_{jkl} = \frac{\mu_{jkl}}{\sigma_{jkl}}$	$\mu_{jkl} = \sigma_{jkl}\tilde{\mu}_{jkl}$	Gaussian mean
σ_{jkl}	$\tilde{\sigma}_{jkl}$	$\tilde{\sigma}_{jkl} = \ln(\sigma_{jkl})$	$\sigma_{jkl} = e^{\tilde{\sigma}_{jkl}}$	Gaussian std. dev.

The sensitivity of the mean parameter update is determined by the size of the associated variance. The positive definite matrix U_t should therefore be chosen carefully, so as to compensate for this sensitivity. Chou *et al.* [23] used a diagonal matrix, where the diagonal elements were equal to the variances (for the mean parameter update). This is equivalent to using the mean parameter transformation in Table 3.1 with the matrix

U_t equal to the identity matrix and has been used throughout. Under these conditions, GPD reverts to the simpler gradient descent algorithm (for all parameters), where

$$\theta_{t+1} = \theta_t - \epsilon_t \nabla l(\mathbf{O}; \theta)|_{\theta=\theta_t}. \tag{3.23}$$

3.2.5 Parameter adaptation

It can be easily verified that the partial derivative of $l_i(d_i)$ (Eq. (3.15)) with respect to the misclassification measure (d_i) is

$$\frac{\partial l_i}{\partial d_i} = \gamma d_i(1 - d_i). \tag{3.24}$$

Calculation of the parameter update defined in Eq. (3.18) or (3.23) requires the gradient function $\nabla l(\mathbf{O}; \theta)$ or $\frac{\partial l(\mathbf{O};\theta)}{\partial \theta}$. In what follows, the gradients and parameter updates for the four parameter types (mean, variance, mixture weights and transition probabilities) are derived.

Gaussian mixture means

According to Eq. (3.23) the parameter update for the mean vectors of HMM i is as follows:

$$\tilde{\mu}^{(i)}_{jkl}(n+1) = \tilde{\mu}^{(i)}_{jkl}(n) - \epsilon \frac{\partial l_i(\mathbf{O}_n; \theta_n)}{\partial \tilde{\mu}^{(i)}_{jkl}}. \tag{3.25}$$

The partial derivative of $l_i(\mathbf{O}_n; \theta_n)$ with respect to $\tilde{\mu}^{(i)}_{jkl}$ can be obtained using the chain rule

$$\frac{\partial l_i(\mathbf{O}_n;\theta_n)}{\partial \tilde{\mu}_{jkl}^{(i)}} = \frac{\partial l_i}{\partial d_i}\frac{\partial d_i}{\partial \tilde{\mu}_{jkl}^{(i)}}, \tag{3.26}$$

where $\frac{\partial l_i}{\partial d_i}$ can be calculated using Eq. (3.24), and from Eq. (3.13) we get

$$\frac{\partial d_i}{\partial \tilde{\mu}_{jkl}^{(i)}} = \begin{cases} -\frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{\mu}_{jkl}^{(i)}} & \text{correct class} \\ \frac{e^{\eta f_i(\mathbf{O};\theta_n)}}{\sum_{j,j\neq i} e^{\eta f_j(\mathbf{O};\theta_n)}} \frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{\mu}_{jkl}^{(i)}} & \text{incorrect class,} \end{cases} \tag{3.27}$$

where $f_i(\mathbf{O};\theta_n) = \ln P_i(\mathbf{O}|\theta_n)$. From Eq. (2.9) we have

$$\frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{\mu}_{jkl}^{(i)}} = \sum_{t=1}^{T} \delta(\bar{q}_t - j)\frac{\partial \ln(b_j^{(i)}(\mathbf{o}_t))}{\partial \tilde{\mu}_{jkl}^{(i)}}, \tag{3.28}$$

where $\delta()$ is the Kronecker delta function, and

$$\begin{aligned} \frac{\partial \ln(b_j^{(i)}(\mathbf{o}_t))}{\partial \tilde{\mu}_{jkl}^{(i)}} =& c_{jk}^{(i)}(2\pi)^{-d/2}|\Sigma_{jk}^{(i)}|^{-1/2}(b_j^{(i)}(\mathbf{o}_t))^{-1}\left(\frac{\mathbf{o}_{tl}}{\sigma_{jkl}^{(i)}} - \tilde{\mu}_{jkl}^{(i)}\right) \\ & e^{-\frac{1}{2}\sum_{l=1}^{D}\left(\frac{o_{tl}}{\sigma_{jkl}^{(i)}}-\tilde{\mu}_{jkl}^{(i)}\right)^2} \\ =& \frac{c_{jk}\mathcal{N}(\mathbf{o}_t;\mu_{jk},\Sigma_{jk})}{b_j^{(i)}(\mathbf{o}_t)}\cdot\left(\frac{o_{tl}-\mu_{jkl}}{\sigma_{jkl}}\right). \end{aligned} \tag{3.29}$$

Having updated the transformed mean using Eq. (3.25), the correct mean can be found using the inverse transformation $\mu_{jkl}^{(i)}(n+1) = \tilde{\mu}_{jkl}^{(i)}(n)\sigma_{jkl}^{(i)}(n+1)$.

Gaussian mixture variance

The Gaussian mixture variance update is

$$\tilde{\sigma}_{jkl}^{(i)}(n+1) = \tilde{\sigma}_{jkl}^{(i)}(n) - \epsilon \frac{\partial l_i(\mathbf{O}_n;\theta_n)}{\partial \tilde{\sigma}_{jkl}^{(i)}}, \tag{3.30}$$

where

$$\frac{\partial l_i(\mathbf{O}_n;\theta_n)}{\partial \tilde{\sigma}_{jkl}^{(i)}} = \frac{\partial l_i}{\partial d_i} \frac{\partial d_i}{\partial \tilde{\sigma}_{jkl}^{(i)}}. \tag{3.31}$$

The partial derivative $\frac{\partial l_i}{\partial d_i}$ is obtained using Eq. (3.24) and again from Eq. (3.13) we have

$$\frac{\partial d_i}{\partial \tilde{\sigma}_{jkl}^{(i)}} = \begin{cases} -\frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{\sigma}_{jkl}^{(i)}} & \text{correct class} \\ -\frac{e^{\eta f_i(\mathbf{O};\theta_n)}}{\sum_{j,j\neq i} e^{\eta f_j(\mathbf{O};\theta_n)}} \frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{\sigma}_{jkl}^{(i)}} & \text{incorrect class,} \end{cases} \tag{3.32}$$

where

$$\frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{\sigma}_{jkl}^{(i)}} = \sum_{t=1}^{T} \delta(\bar{q}_t - j) \frac{\partial \ln(b_j^{(i)}(\mathbf{o}_t))}{\partial \tilde{\sigma}_{jkl}^{(i)}} \tag{3.33}$$

$$\frac{\partial \ln(b_j^{(i)}(\mathbf{o}_t))}{\partial \tilde{\sigma}_{jkl}^{(i)}} = \frac{c_{jk}\mathcal{N}(\mathbf{o}_t;\mu_{jk},\Sigma_{jk})}{b_j^{(i)}(\mathbf{o}_t)} \cdot \left[\left(\frac{o_{tl} - \mu_{jkl}^{(i)}}{\sigma_{jkl}^{(i)}} \right)^2 - 1 \right]. \tag{3.34}$$

Finally, we use

$$\sigma_{jkl}^{(i)}(n+1) = e^{\tilde{\sigma}_{jkl}(n+1)}. \tag{3.35}$$

Gaussian mixture weights

The MCE update for the Gaussian mixture weights is

$$\tilde{c}_{jk}^{(i)}(n+1) = \tilde{c}_{jk}^{(i)}(n) - \epsilon \frac{\partial l_i(\mathbf{O}_n;\theta_n)}{\partial \tilde{c}_{jk}^{(i)}}, \tag{3.36}$$

where

$$\frac{\partial l_i(\mathbf{O}_n;\theta_n)}{\partial \tilde{c}_{jk}^{(i)}} = \frac{\partial l_i}{\partial d_i}\frac{\partial d_i}{\partial \tilde{c}_{jk}^{(i)}}. \tag{3.37}$$

From Eq. (3.13), we have

$$\frac{\partial d_i}{\partial \tilde{c}_{jk}^{(i)}} = \begin{cases} -\frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{c}_{jk}^{(i)}} & \text{correct class} \\ \frac{e^{\eta f_i(\mathbf{O};\theta_n)}}{\sum_{j,j\neq i} e^{\eta f_j(\mathbf{O};\theta_n)}} \frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{c}_{jk}^{(i)}} & \text{incorrect class,} \end{cases} \tag{3.38}$$

where

$$\frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{c}_{jk}^{(i)}} = \sum_{t=1}^{T} \delta(\bar{q}_t - j) \frac{\partial \ln(b_j^{(i)}(\mathbf{o}_t))}{\partial \tilde{c}_{jk}^{(i)}}, \tag{3.39}$$

$$\frac{\partial \ln(b_j^{(i)}(\mathbf{o}_t))}{\partial \tilde{c}_{jk}^{(i)}} = c_{jk}^{(i)}\left[\frac{\mathcal{N}(\mathbf{o}_t;\mu_{jk},\Sigma_{jk})}{b_j^{(i)}(\mathbf{o}_t)} - 1\right]. \tag{3.40}$$

Finally, we use

$$c_{jk}^{(i)}(n+1) = \frac{e^{\tilde{c}_{jk}(n+1)}}{\sum_k e^{\tilde{c}_{jk}(n+1)}}. \tag{3.41}$$

Transition probabilities

The update for the transition probabilities is

$$\tilde{a}_{ij}^{(i)}(n+1) = \tilde{a}_{ij}^{(i)}(n) - \epsilon \frac{\partial l_i(\mathbf{O}_n;\theta_n)}{\partial \tilde{a}_{ij}^{(i)}}, \tag{3.42}$$

where

$$\frac{\partial l_i(\mathbf{O}_n;\theta_n)}{\partial \tilde{a}_{ij}^{(i)}} = \frac{\partial l_i}{\partial d_i}\frac{\partial d_i}{\partial \tilde{a}_{ij}^{(i)}}. \tag{3.43}$$

From Eq. (3.13), we get

$$\frac{\partial d_i}{\partial \tilde{a}_{ij}^{(i)}} = \begin{cases} -\frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{a}_{ij}^{(i)}} & \text{correct class} \\ \frac{e^{\eta f_i(\mathbf{O};\theta_n)}}{\sum_{j,j\neq i} e^{\eta f_j(\mathbf{O};\theta_n)}} \frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{a}_{ij}^{(i)}} & \text{incorrect class.} \end{cases} \tag{3.44}$$

From Eq. 2.9,

$$\frac{\partial f(\mathbf{O};\theta_n)}{\partial \tilde{a}_{ij}^{(i)}} = \sum_{t=1}^{T}\sum_{s} \delta(\bar{q}_{t-1} - i)\delta(\bar{q}_t - s)\Big[\delta(j-s) - a_{ij}\Big]. \tag{3.45}$$

Finally, the reverse transformation is applied,

$$a_{ij}^{(i)}(n+1) = \frac{e^{\tilde{a}_{ij}(n+1)}}{\sum_k e^{\tilde{a}_{ij}(n+1)}}. \tag{3.46}$$

3.3 Embedded MCE

In the derivation of the MCE algorithm in the previous sections, it was assumed that the whole training observation $\mathbf{O}$ must be one of N classes. MCE can, however, be applied at the level of various speech units, such as phonemes, words and sentences. Here, a search is used to automatically segment (state and HMM alignment) and label the utterance for usage within the MCE framework. More specifically, the correct alignment and the N-best incorrect alignments are required, for which an N-best search is used.

The application of MCE to strings of phoneme or word units is known as string-level MCE or embedded MCE [59, 78, 75, 24]. Here, the observation $\mathbf{O}$ is a concatenated string of observations belonging to different classes. In this situation, MCE is used to minimize the string error rate and not the individual phoneme or word error rates. Note that although the phoneme or word error rates are not directly minimized, minimizing the string error rate will tend to decrease the phoneme and word error rates.

The MCE procedure remains the same, except that in this case the correct string and N-best strings are required, as opposed to the single correct and N-best incorrect acoustic units. The N-best search proposed by Soong and Huang [106] to generate the N-best alignments and subsequently used by Chen and Soong [18, 19] and McDermott [75] for string-level MCE was implemented and used in our work.

Figure 3.1 shows a state occupancy diagram for a simple example where embedded MCE is used. Here, the observation is of the word "boot", containing the phonetic units b, uw and t, with silence on either side. The state occupancy diagram for the correct string is given at the top, with the state occupancy diagram of the most confusable incorrect string below it. Let us assume, for the purpose of this example, that we are only working with the correct string and the single best incorrect string. State occupation is indicated with thick black or grey lines. A grey line is used to indicate when an error in state occupancy occurs.

When state occupancy is correct for both the correct and best incorrect string (black lines in incorrect string state occupancy in Figure 3.1), the gradient of Eq. (3.13) with respect to the parameters will be zero (gradients due to correct and incorrect strings are equal, but of opposite sign). The models d and aw in the incorrect string are a substitution and insertion respectively and the state alignments associated with these units are therefore incorrect. Although the units *uw* and *t* are in the correct position, the state alignment is not correct and the state alignments displayed in grey will result in the associated state being penalized. Therefore, incorrect state occupancy will result in the model associated with the correct alignment being reinforced and the model associated with the incorrect class penalized according to the update functions defined earlier.

3.4 Discussion and Experiments

The following sections will discuss and experimentally validate a number of issues related to the use and implementation of the MCE algorithm. They are:

- batch-mode versus online optimization,
- smoothness of the loss function,
- the need for a zero-one loss function, and
- overtraining in MCE and several solutions.

The TIMIT dataset, described in Section 2.4.1, were used for this purpose. Two training sets were used, namely the standard TIMIT training set T and the small gender independent training set T_S, which were described in Section 2.4.1. The relevant training and testing set details are reproduced in Table 3.2 for convenience. The basic configuration of the system is as described in Section 2.1. A 3 state, 5 mixture HMM is used to model each of the 39 phonetic units in the TIMIT dataset.

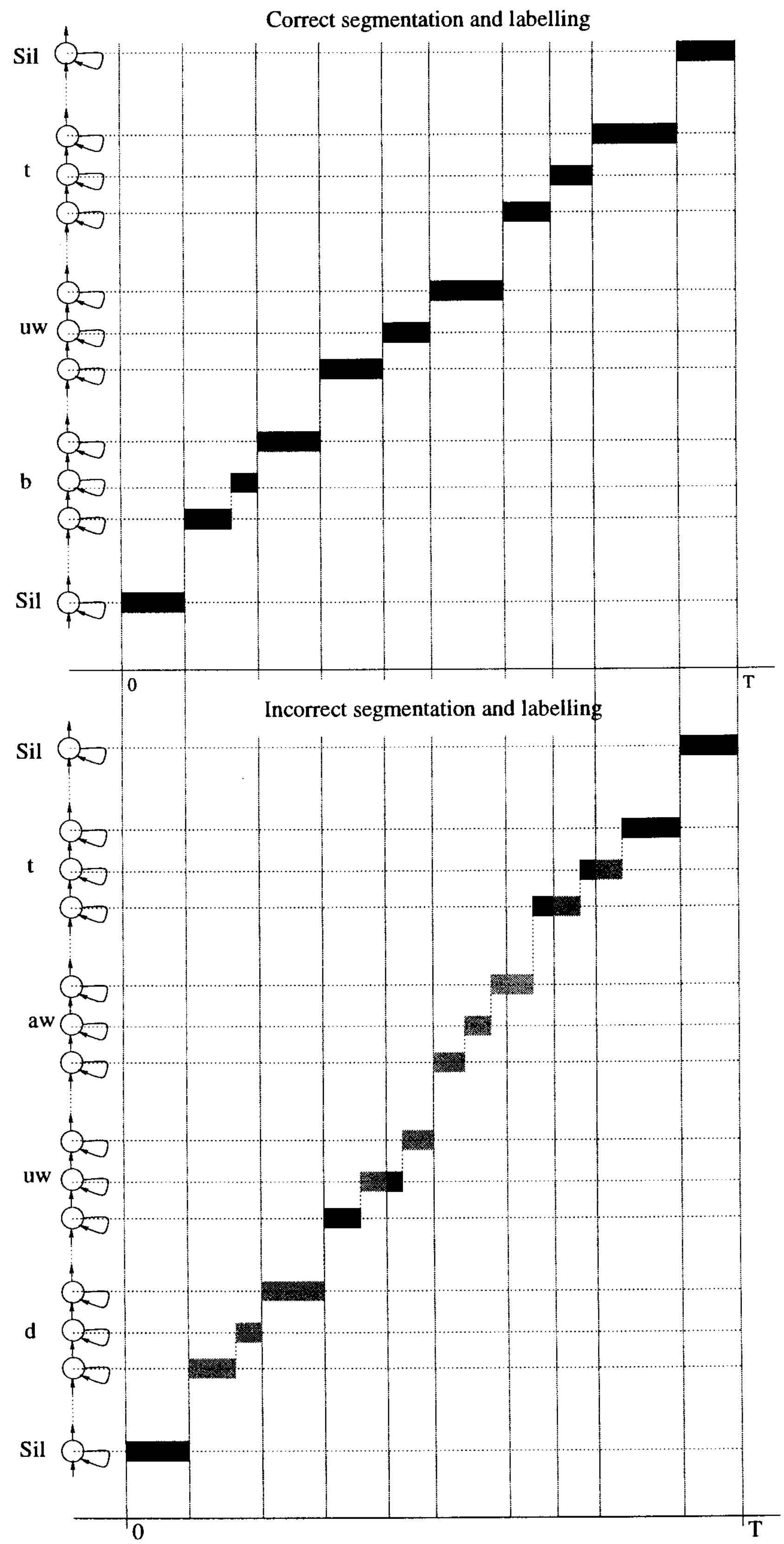


Figure 3.1: State occupancy diagram of the correct and an incorrect string

Table 3.2: Description of TIMIT training and testing sets used.

Description	Label	Number of speakers			Number of	Duration
		Male	Female	Total	sentences	(minutes)
Training sets:						
Full (standard)	T	326	136	462	4620	236.5
Small (gender indep.)	T_S	16	16	32	320	16.1
Testing set (standard)		112	56	168	1680	86.4

3.4.1 Batch-mode versus online optimization

Online (stochastic) optimization algorithms use a single training example to determine the parameter update (Eq. (3.18) in this case) for each training example. The deterministic (batch-mode) descent algorithm computes the combined gradient for all the training examples in the training set, which is then used in the parameter update.

The online descent algorithm can perform better in situations where there is redundancy in the training data. The number of passes through the data to find a local minimum is therefore often less for online than for batch-mode optimization. It is, however, necessary to randomly select examples without replacement when using the online algorithm. If not done, oscillatory and non-optimal behaviour might be observed in the optimization process, as a result of data set structure.

Figure 3.2 presents the performance of online and batch-mode MCE using the standard TIMIT training and testing sets. The learning rates for both algorithms were set just below the point at which they became unstable. The online descent algorithm is considerably faster in terms of training time, as opposed to deterministic (batch) gradient descent. Maximum training and testing set performance is better when using online optimization – batch-mode optimization has found a less optimal local minimum. Online descent is therefore preferred and is used throughout.

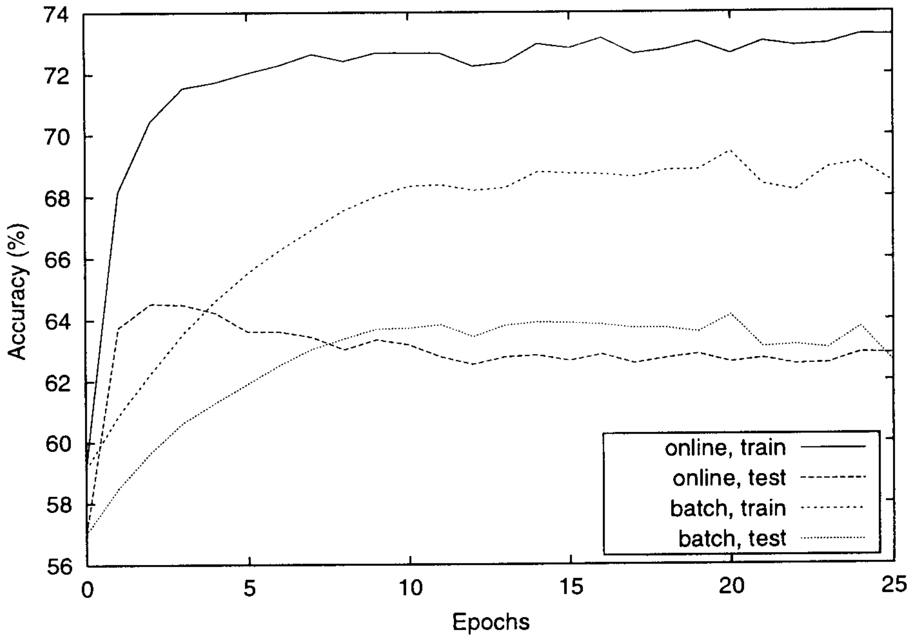


Figure 3.2: Training and testing set performance for standard string-level MCE using the batch and online optimization algorithms

3.4.2 Smoothness of the loss function

The smoothness of the zero-one loss function (Eq. (3.15)) is somewhat critical to the performance of the MCE algorithm. A zero-one loss function which is reasonably sharp, limits the effect of individual examples on the loss and allows only those examples on or near the decision boundaries to affect the parameter update. Outliers will therefore effectively not be included in the gradient calculation, thereby promoting robustness. A smoother loss function, is less likely to be caught in local minima. This effect will be further discussed in Section 3.4.3.

McDermott [75] stated that too much smoothing would result in a discrepancy between the function being optimized and the target, minimum classification error. I, however, do not believe that this is the case, as an extremely smooth sigmoid loss function is linear in the region of interest and so the misclassification measure, which is designed to emulate the decision rule, is therefore directly minimized. It is, however, true that a discrepancy results between the function being optimized and the target when the zero-one loss function is relatively sharp. Here, the function being optimized can be interpreted as "*minimum classification error for only those utterances which have misclassification measures close to zero*".

Figure 3.3 shows a histogram of the misclassification measure values for the utterances in the TIMIT training set, when using a 3 state 5 mixture HMM. All of the misclassification measure values are greater than zero, which implies that none of the utterances were correctly classified (only strictly true when $\eta = \infty$). The sigmoid loss function (Eq. (3.15)) output for $\gamma = 0.1$, $\gamma = 0.01$ and $\gamma = 0.001$ are also shown. It is important that the derivative of the loss function must be significant for a large part of the training data. The loss function with $\gamma = 0.1$ is too sharp and only has a reasonable derivative for small misclassification measure values, which will result in only a few of the utterances influencing the update. One would, therefore, not expect the MCE algorithm to work well for $\gamma = 0.1$. The loss function using $\gamma = 0.01$ is mostly linear for a reasonable part of the training data, and has a significant derivative for a large part of the training data. Using $\gamma = 0.001$ results in a loss function which is effectively linear for all of the training utterances. The optimal value for γ will therefore, probably lie somewhere around 0.01.

Figure 3.4 shows the TIMIT test set accuracy versus the sigmoid loss function parameter γ for both the full training set T and the small training set T_S [1]. As expected, $\gamma = 0.1$ results in a sigmoid function which is too sharp and the accuracy attained (58.3% for T) is therefore considerably worse than that attained using smaller values of γ. Peak accuracy of 64.7% results when $\gamma = 0.01$ is used for the training set T. The MCE algorithm is considerably less effective when the small training set is used (T_S), resulting in a peak testing set performance of only 53.8%.

[1]Note that phone recognition accuracy results (Eq. (2.26)) are reported throughout this chapter

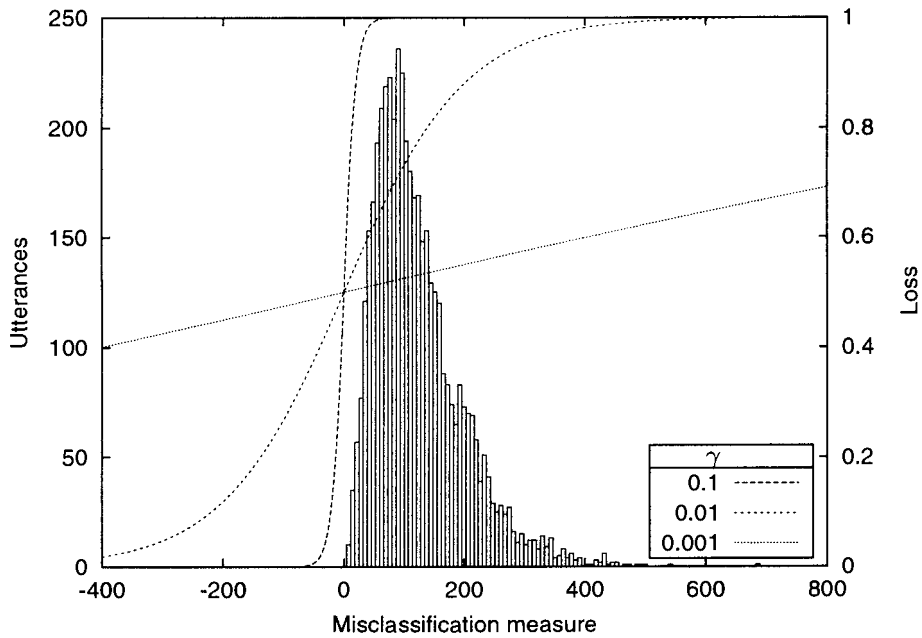


Figure 3.3: Histogram of the misclassification measure before embedded MCE training for the TIMIT training set. The sigmoid loss functions with $\gamma = 0.1$, $\gamma = 0.01$ and $\gamma = 0.001$ are plotted.

3.4.3 Need for a zero-one loss function

The question must be asked, "Why is a smoothed zero-one loss function needed?". The main reason why one would introduce a sigmoid function is to promote stability in the training process and to ensure that the resultant gradient function (and therefore the parameter update) is finite and continuous.

Figure 3.5 shows an example of a simple two class problem where a sigmoid loss function could result in the best minimum not being attained. Here we assume that the two classes can each be modeled by a single Gaussian distribution. Let us also assume that we know the variances are fixed and equal to one. The ML estimates of the means are (0.02, 4.23) for $+$'s and (0.25, 0.09) for $\times$'s. The theoretical minimum error classification boundary is then as shown by line (a) in Figure 3.5.

If the sigmoid function is quite sharp, then the derivative for the "outliers" in the

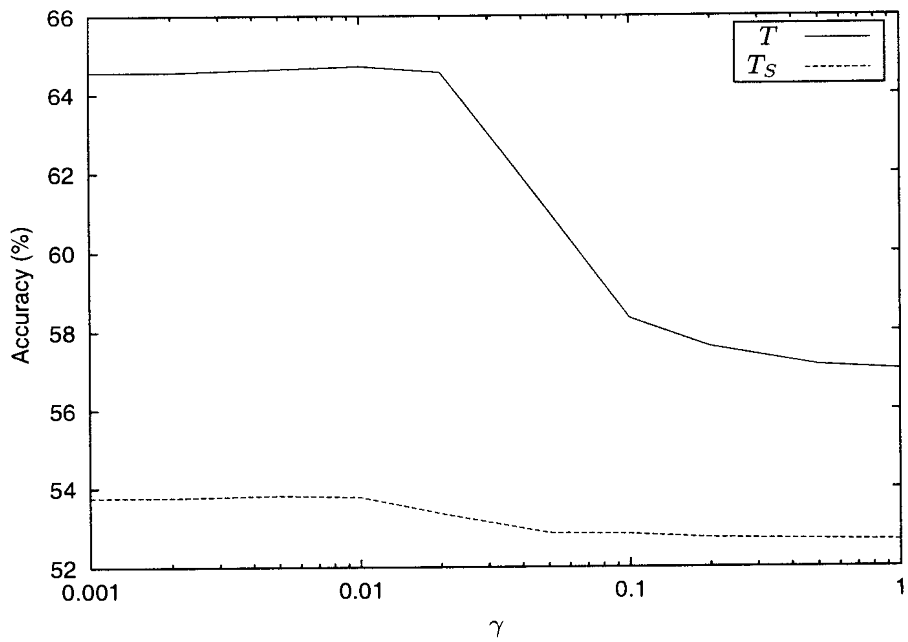


Figure 3.4: TIMIT test set accuracy versus γ for the training set T and T_S.

top-right corner will be close to zero and the class boundary (b), which has a lower classification error, will not be reached using MCE. This situation occurs because the true distributions are not Gaussian as we assumed they were. It is, therefore, unfortunate that by using a (relatively sharp) zero-one loss function we are introducing additional local minima. It is these local minima which can also in certain situations save the algorithm from overtraining. This could be the case if there is excessive noise in the training set, or if the data has been poorly labelled. Whether decision boundary (b) is truly better than (a) can only be determined by using an independent test set.

The results of the previous section (3.4.2), where small values of γ resulted in optimal performance, support the above discussion. Table 3.3 gives the results for the MCE algorithm with and without using a sigmoid loss function with $\gamma = 0.01$. There is no significant difference between the two algorithms, with the 0.1% difference for the large training set T due more to rounding than any algorithmic difference.

This result is important for modifications and algorithms proposed later in this and other chapters, where the algorithm or modification cannot be mathematically justified

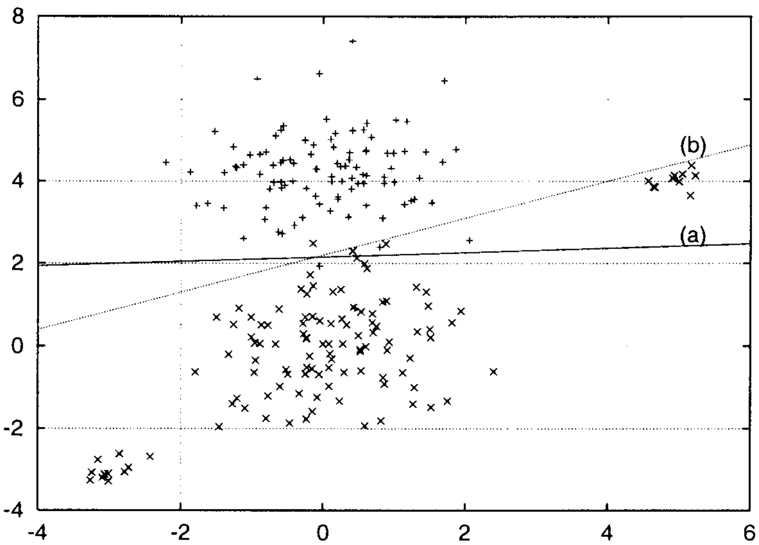


Figure 3.5: Example of where a sharp sigmoid does not shift the ML class boundary (a) to boundary (b), with lower classification error.

Table 3.3: Comparison of MCE accuracy results when using a sigmoid loss function and no sigmoid loss function

Training set	sigmoid $\gamma = 0.01$	no sigmoid
T	64.7	64.6
T_S	53.8	53.8

when using a sigmoid loss function. If there is a concern over the stability of the algorithm, a sigmoid loss function with a small value of γ can be used, so that the loss function is linear for a reasonable part of the training data. We can then ignore the sigmoid loss function when incorporating such modifications in the loss function or optimization criterion.

3.4.4 Overtraining in MCE

Minimum classification error (MCE) training is somewhat prone to overspecialization. This section investigates various techniques which improve performance and generalization of the MCE algorithm.

In Section 2.2 overtraining was discussed from a model complexity perspective. Here, it is the parameter estimation algorithm which results in further overspecialization. It is also an unfortunate tendency of MCE and other discriminative training algorithms to result in a decrease in testing set performance after maximum performance has been attained (in terms of training time).

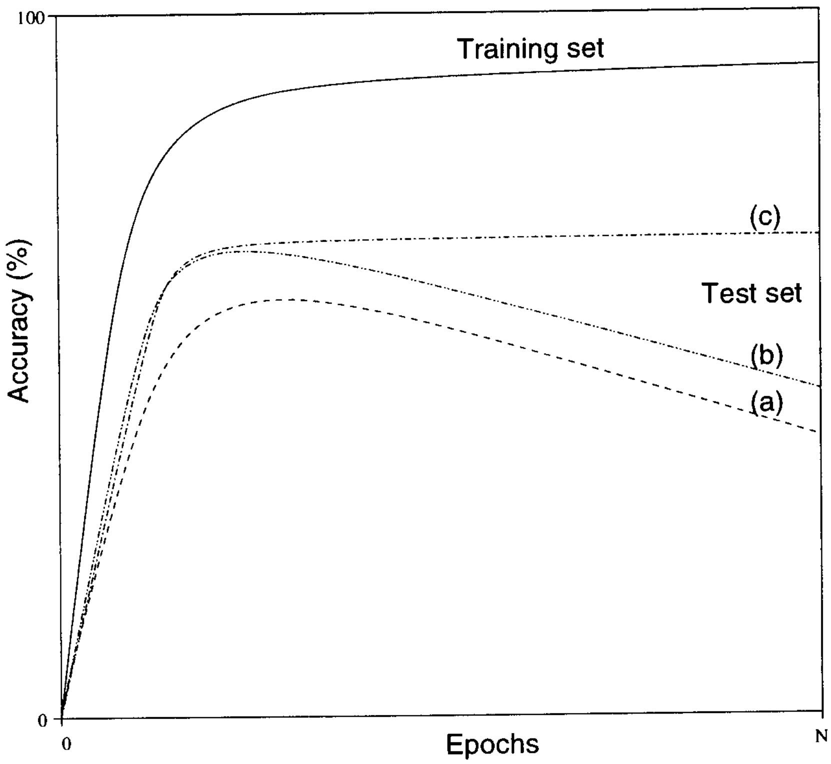


Figure 3.6: Comparison of typical (a) and preferred (b and c) testing set performance

Figure 3.6 shows the typical form of results obtained for the training and testing sets (a). Limiting specialization of the classifier would result in reducing the difference between training and testing set error rates. We would, for example, prefer result (b) in Figure 3.6 to result (a). We also wish to limit the degradation in performance after maximum performance has been reached. An algorithm which has the characteristics of (c) in Figure 3.6 would be advantageous in that a cross-validation set would not be required to choose the best model in an unbiased way.

Regularization techniques are often used to improve generalization. In regularization, a penalty term $F(\theta)$ which is called a regularizer is added to the original objective function, creating a new objective function, i.e.

$$\tilde{l}(\theta; \mathbf{O}) = l(\theta; \mathbf{O}) + \zeta F(\theta). \tag{3.47}$$

The regularizer conveys *a-priori* knowledge about the process which is to be learned. Shimodaira *et al.* [104] presented a method to prevent over-fitting and improve the generalization performance of the MCE algorithm when applied to neural networks. A simple version of the Tikhonov regularizer [11] was used for this purpose in [104]. This regularizer requires that the second order derivative of the likelihood with respect to the model parameters be computed. That is, however, not a simple task for hidden Markov models and was not incorporated in our work. The following sections discuss the two approaches that were followed to reduce overtraining.

Penalizing large variances

It can be expected that overspecialization would result in the variances of certain of the Gaussian mixtures becoming very small. To reduce overfitting, a penalty term proportional to the sum of the square (or power) of the inverse of the variances (precisions) of the Gaussians of the HMM states is therefore proposed. This is as expressed in Eq. (3.48), and is added to the loss function of MCE (Eq. (3.15)).

$$\alpha \sum_{\sigma} (1 + \rho \sigma_{jkl}^2)^{-\xi} \tag{3.48}$$

Figure 3.7 shows the derivative of this penalty term with respect to the variance (σ_{jkl}^2). The gradient is negative and becomes large for very small variances. A negative gradient used in the parameter update (Eq. (3.30)) will result in the variance becoming

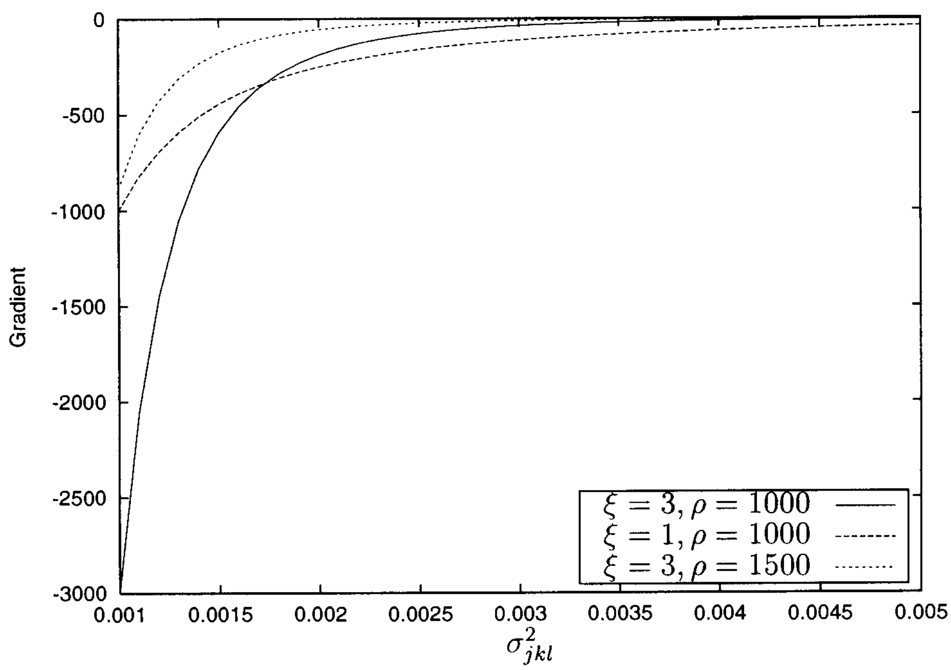


Figure 3.7: Gradient resulting from the variance penalty term.

larger. Note that if there were no other gradients resulting from the standard MCE update, the variances would continue to grow larger.

The values α, ρ and ξ are constants that are empirically determined. As seen in Figure 3.7, the constants ρ and ξ determine the form of the penalty term. The constant α is a scaling factor and is used to increase the contribution of the penalty term independently of the variance value. One has been added to $\rho\sigma^2_{jkl}$ to ensure that the gradient is finite for $\sigma_{jkl} = 0$.

This results in what could be called "precision decay", thereby ensuring that variances do not become too small. This has the indirect consequence of reducing overfitting. It was empirically found that $\xi = 3$ produced the best results. Figure 3.8 shows the performance of the MCE algorithm using the variance penalty term (MCE+VPEN) versus the parameters α and ρ when using the small training set T_S. Peak performance of 54.8% is attained when using $\alpha = 1000$ and $\rho = 1000$ (versus 53.8% without the penalty term).

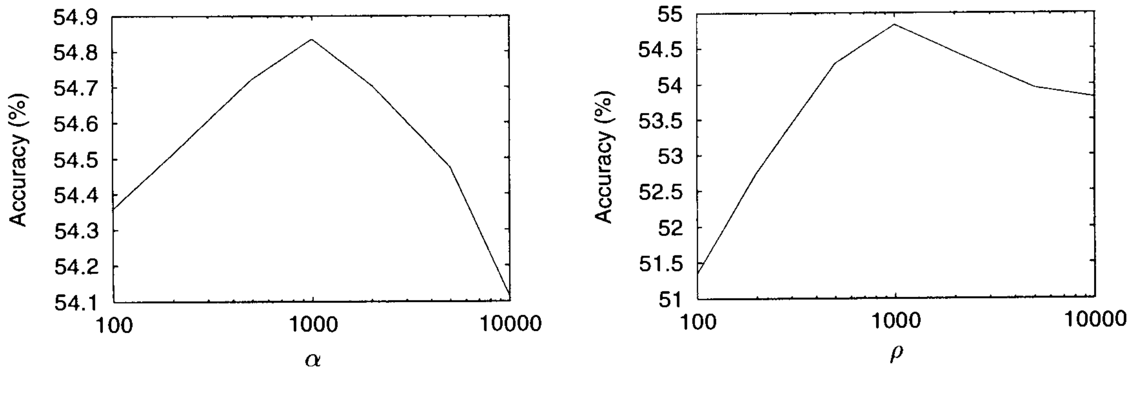


Figure 3.8: Testing set performance for the modified MCE with variance penalty (MCE+VPEN) using the small training set T_S, (a) versus α for $\rho = 1000$ and (b) versus ρ for $\alpha = 1000$

Figure 3.9 shows the advantage of using the above penalty term. Here, the parameter values $\xi = 3$, $\rho = 1000$ and $\alpha = 1000$ have been used, with no sigmoid loss function. The results are similar when using a sigmoid loss function. The test set results are better (2.1% relative improvement in error rate), while the training set performance has decreased.

Table 3.4 presents the results of using a sigmoid function versus not using it when using the variance penalty term with the MCE algorithm. Interestingly, a sigmoid function works better when the full training set is used, while the MCE algorithm without a sigmoid loss function performs better when the small training set is used. Note that the optimal values for parameters α and ρ differ for each configuration and were empirically determined for each case.

Table 3.4: Accuracy results for the MCE algorithm (sigmoid versus no sigmoid) when using the variance penalty term

Training set	sigmoid $\gamma = 0.01$	no sigmoid
T	65.4	64.7
T_S	53.8	54.8

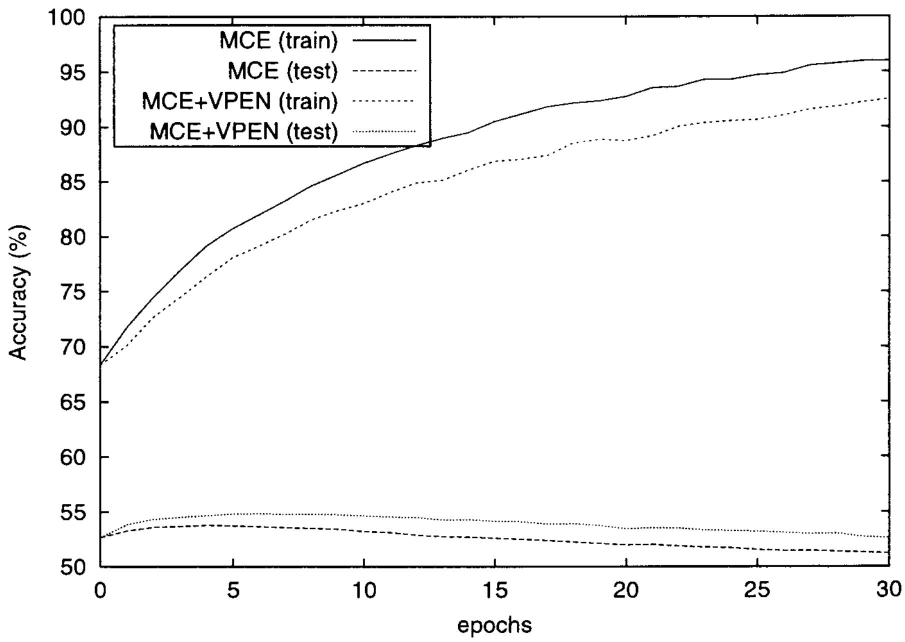


Figure 3.9: Training and testing set performance for standard string MCE and modified MCE with variance penalty (MCE+VPEN), no sigmoid loss

Weighted likelihood term

One potential problem with string-level MCE is that parts of the training string that do not result in errors are effectively ignored and therefore do not reinforce the associated parameters. Focusing solely on errors will result in overspecialization. Adding a weighted likelihood term (of the correct class) to the MCE loss function may therefore reduce overtraining. This will tend to reinforce correct substrings, while still penalizing errors. The misclassification measure in Eq. (3.13) then becomes

$$d(\mathbf{O}) = -(1+\kappa)\ln P_i(\mathbf{O};\theta_i) + \ln[\frac{1}{N}\sum_{j,j\neq i}^{N} e^{\eta \ln P_j(\mathbf{O};\theta_j)}]^{1/\eta}, \tag{3.49}$$

where κ is the weighting of the additional likelihood term, $P_i(\mathbf{O};\theta_i)$. This modification has been chosen so as to increase the gradient resulting from correct substrings by a factor κ. However, when using a sigmoid loss function, no modification (to misclassification measure or loss function) can be made that will increase the gradient associated with correct substrings by a uniform factor κ, while not affecting that associated with incorrect substrings. Such a modification can therefore not be mathematically justified when using a smoothed zero-one loss function. It can, however, be implemented as a simple heuristic where the gradient for the correct class is simply multiplied by a weighting factor $(1 + \kappa)$.

Figure 3.10 presents the results for the MCE algorithm using the weighted likelihood term (MCE+WL) for the small training set (T_S). Significantly, using a sigmoid function does not work nearly as well as the algorithm without a sigmoid loss function when using the weighted likelihood term. Peak performance of 55.0% is attained when using a weight $\kappa = 1.5$, for the algorithm without a sigmoid loss function (versus 53.8% with $\kappa = 0$).

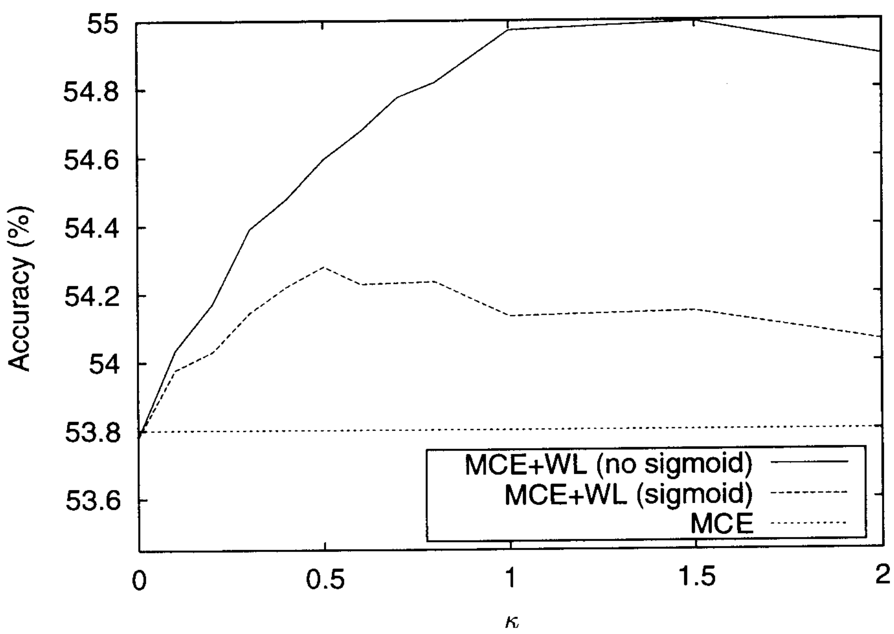


Figure 3.10: Testing set performance for MCE with the weighted likelihood term (MCE+WL) plotted versus κ.

Figure 3.11 shows the performance of the MCE algorithm with and without using the weighted likelihood term versus the number of training epochs, for the small training set. A sigmoid function is not used. Maximum testing set performance is better when using the weighted likelihood term, but more significantly, it stops degradation of performance after peaking (preferred result (c) in Figure 3.6). Results when using a sigmoid loss function are similar with a slightly worse error rate being achieved.

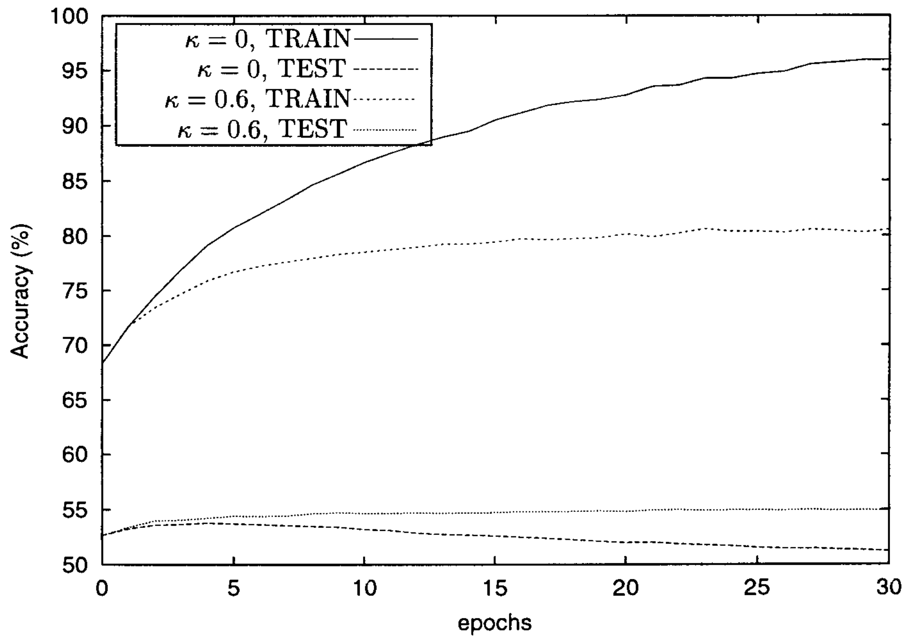


Figure 3.11: Training and testing set performance for standard string MCE and modified MCE with the weighted likelihood term (MCE+WL), sigmoid loss function not used

Table 3.5 presents the results when using the weighted likelihood term with the MCE algorithm for both the small and full training datasets. Here, in conjunction with the weighted likelihood term, using no sigmoid function performs best for both datasets. This is due to the fact that the implementation thereof for the algorithm using a sigmoid loss function is not mathematically justified and is merely a heuristic implementation thereof.

Table 3.5: Accuracy results for the MCE algorithms using the weighted likelihood term (MCE+WL)

Training set	sigmoid $\gamma = 0.01$	no sigmoid
T	64.7	65.1
T_S	54.3	55.0

Word MCE

Presenting arbitrarily long strings to the string-level MCE algorithm is not optimal. An error at a specific point in time will potentially result in the incorrect segmentation at that time (not just incorrect labelling), and such an error in segmentation will therefore influence the recognition and segmentation of subsequent acoustic units. The usage of a language model will also tend to result in an error at any given point resulting in further errors later in the utterance. Errors occurring earlier during recognition of a string therefore influence the recognition for the rest of the string. Our confidence in the accuracy of segmentation and classification after an error has occurred will therefore tend to be low. As the N-best string outputs from the recognizer are used as discriminative training examples, the number of incorrect strings are limited. Most of these "incorrect" strings differ only in a few places, resulting in only a few potential errors being addressed during discriminative training.

To improve the above, presenting smaller word-based strings to the string-level MCE algorithm is investigated. This is particularly appropriate when training speech recognizers on speech databases which have long sentences. This, however, requires that one has a dataset which is also labeled at word level.

A sentence would therefore be presented to the string-level MCE algorithm word by word in isolation. The N-best hypotheses (string of phones) would therefore be generated for each word individually (using the relevant part of the utterance) and used to determine the MCE gradients and updates. This is as opposed to the standard string-level MCE algorithm where the N-best hypotheses are generated for the entire sentence.

Figure 3.12 compares results for sentence- and word-based MCE for the small training set, plotted versus the number of training epochs. A sigmoid loss function is not used. The improvement in performance is marked, resulting in a 7.4% relative reduction in error rate. Another advantage here is that the usage of word-based string MCE limits degradation of performance after a maximum is reached.

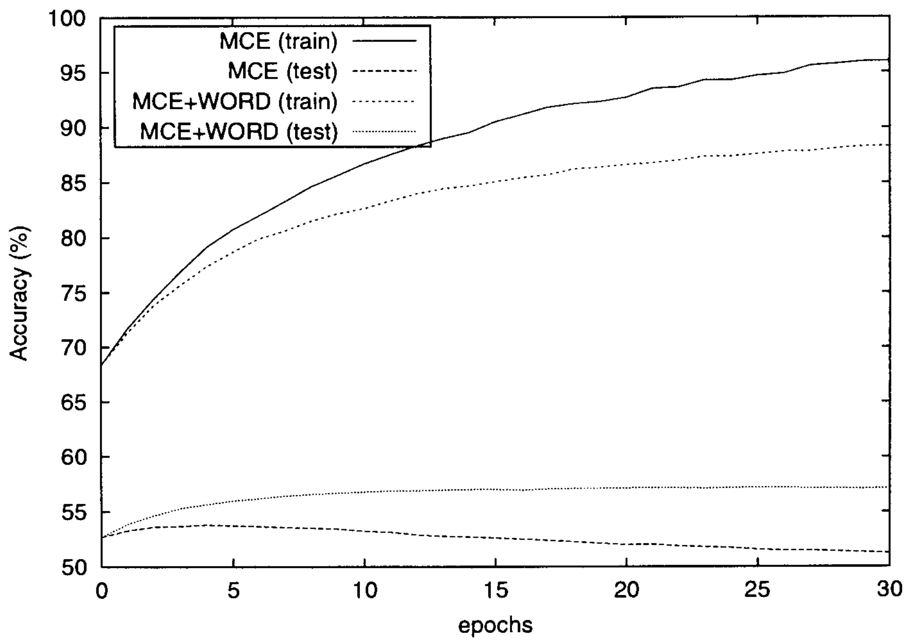


Figure 3.12: Training and testing set performance for standard string MCE and word-string MCE when using the small training set T_S (no sigmoid loss)

The results for both small and full training sets are presented in Table 3.6. Here, using a sigmoid loss function proves to be marginally better (and probably not significant) when the small training set (T_S) is used. This is, however, not the case when the full training set (T) is used, where using a sigmoid loss function produced a recognition accuracy which is marginally worse than that attained when not using a sigmoid loss function. Improvement in error rates relative to that obtained by standard MCE are significant (accuracy of 66.7% versus 64.7%), irrespective of whether a sigmoid loss function is used or not.

3.4.5 Summary and discussion of results

Table 3.7 gives a summary of the results when using the different modifications proposed for MCE in this chapter. The full training set (T) is used. MCE training alone produces a 17.7% *relative* reduction in error rate over baseline maximum likelihood

Table 3.6: Accuracy results for the word-based string MCE algorithms (MCE+WORD). Relative improvement in error rate compared to standard MCE are given in brackets

Training set	sigmoid $\gamma = 0.01$	no sigmoid
T	66.3 (5.1%)	66.7 (5.9%)
T_S	57.9 (8.9%)	57.2 (7.4%)

(ML) training. However, employing the modifications results in a relative reduction in error of up to 23.3% being attained over ML training.

Table 3.7: Summary of phoneme error rate results for MCE and modifications on the full training set T

Training method	Phoneme error rate %	
Baseline (ML)	43.0	
	sigmoid	no sigmoid
MCE	35.5	35.4
MCE+VPEN	34.6	35.3
MCE+WL	35.3	34.9
MCE+WORD	33.7	33.3
MCE+WORD+VPEN	33.3	**33.0**
MCE+WORD+WL	33.7	33.3

Table 3.8 gives a summary of the results when using the different modifications and the small training set (T_S) is used. Here, standard string-level MCE only results in a 2.5% relative reduction in error rate over baseline maximum likelihood (ML). However, employing the proposed modifications results in a relative reduction in error rate of up to 12.2% being attained. The modifications have more of an effect when less training data is available and overtraining is more prevalent.

Tables 3.7 and 3.8 provide results obtained when combining the word-based string MCE algorithm and the other modifications (penalty and weighted likelihood). Unfortu-

Table 3.8: Summary of phoneme error rate results for MCE and modifications on the small training set T_S

Training method	Phoneme error rate %	
Baseline (ML)	47.4	
	sigmoid	no sigmoid
MCE	46.2	46.2
MCE+VPEN	46.2	45.2
MCE+WL	45.7	45.0
MCE+WORD	42.1	42.8
MCE+WORD+VPEN	**41.6**	42.2
MCE+WORD+WL	42.1	42.8

nately, the weighted likelihood term fails to improve upon the performance of the word-based string-level MCE algorithm for either of the two datasets (MCE+WORD+WL versus MCE+WORD in the tables). This indicates that the effect of the two modifications is similar, which can be seen in the results presented for the individual procedures earlier. Both, for example, reduce degradation in performance after maximum testing set performance is reached. Limited improvements in performance were obtained when the variance penalty term was combined with the word-based string-level MCE algorithm.

Significant improvements in performance on the testing sets are obtained using the modifications to MCE as proposed. The modifications proposed are relatively simple to implement and limit overspecialization to some degree. The additional computational expense resulting from the use of the proposed modifications is very small and is not measurable. Although variation in performance did result from the use or non-use of a sigmoid function, there is little evidence to suggest that any one of the two possibilities is a better choice, with the resultant variation in performance generally being relatively small compared to the improvements in error rate due to the modifications.

3.5 Summary

This chapter described the MCE algorithm and its usage within a continuous speech recognition framework. The algorithm performs gradient descent on a criterion function which is a close approximation of the classification error. For the TIMIT database, usage of the MCE criterion resulted in significant gains in performance over the standard ML training procedure.

The effect of a smoothed zero-one loss function was discussed and experimentally determined for the TIMIT dataset; where it was found that small values of the sigmoid parameter γ performed best. Furthermore, the need for a zero-one loss function was questioned and the conclusion was reached that there is little evidence that there is an advantage or disadvantage to using a smoothed zero-one loss function. This result was used later in modifying the MCE criterion (adding a weighted likelihood term), where the modification could not be mathematically justified when a non-linear loss function was used.

Overtraining within the MCE framework was discussed and three modifications were proposed. The first modification attempted to stop the mixture variances from becoming very small, which results when little data is available. The second modification added a weighted likelihood term to the MCE criterion, thereby reinforcing correct substrings, as well as improving discrimination for incorrect substrings. Finally, a word-based string-level MCE algorithm was proposed, in which smaller word-based substrings were used, instead of the the entire string. Significant gains in performance resulted when using these modifications with the TIMIT database.