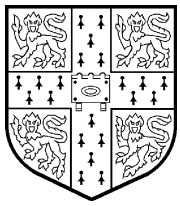
M Phil in Computer Speech and Language Processing

Department of Engineering, Cambridge University

Implementation of Frame Discrimination on a large task



Daniel Povey

April 15, 1999

Abstract

Most if not all speech recognition systems use Hidden Markov Models (HMM) to model the production of speech from sequences of phones or other basic units of speech. HMMs need to be trained, and this is done using speech utterances whose transcription is known. The most common method of training HMMs is known as Maximum Likelihood (ML) estimation. Recently, there has been interest in another class of HMM training methods, known generally as *discriminative* techniques. Among these, Maximum Mutual Information (MMI) estimation is probably the most popular. The performance gain of MMI as compared to ML decreases as the speech model becomes more complex and the model training data becomes insufficient [**?**]. Recently, a discriminative technique called Frame Discrimination (FD), related to MMI, has been developed by Kapadia [**?**], and shown on two small tasks (connected digit recognition and isolated letter recognition) to be more robust than MMI as the speech model becomes more complex. The purpose of this dissertation is to report the results of implementing FD on a medium-vocabulary (1,000 word) continuous speech recognition. An original optimisation, dubbed the Roadmap algorithm, whose function is similar to that of vector quantization (VQ), is developed in order to enable efficient re-estimation using FD, and a variant of the Extended Baum-Welch (EBW) equations is presented which seems to perform better than a previous approximation.

Contents

Chapter 1

Introduction to FD

1.1 Speech recognition, speech data and HMMs

For purposes of speech recognition, speech utterances can be discretized and processed to give a sequence of vectors

$$O = \mathbf{o}_1, \mathbf{o}_2, ..., \mathbf{o}_T \tag{1.1}$$

A HMM is a mathematical construct used to represent a logical unit of speech such as a phone or word, or a sequence of such units, which, given a certain set of parameters M, gives a likelihood value for a sequence of data O.

$$P(O|M) \tag{1.2}$$

More detail on HMM operation will be given later. In continuous speech recognition, HMMs representing phones can be concatenated to correspond to transcriptions of utterances; in this dissertation the symbol M will refer to the parametrisation of all phone HMMs for the set of phones used, and the concatenation of HMMs will be implicit, being governed by the transcription T of the utterance, so we represent the likelihood of the observed speech data given a transcription T as:

$$P(O|T, M) \tag{1.3}$$

In speech recognition the aim is to find the most likely transcription T, and this is done by factoring as:

$$P(T|O, M) = P(O|T, M)\frac{P(T)}{P(O)} \tag{1.4}$$

$$T_{\text{rec}} = \arg\max_T P(T|O, M) = \arg\max_T P(O|T, M)P(T) \tag{1.5}$$

$P(T)$ is given by the language model, which defines the probability of every possible transcription. $P(O)$ is irrelevant to finding the best transcription, and may be assumed to equal 1.

HMM parameter estimation

It has just been described in bare outline how HMMs can be used to decode speech, without going into the details of HMM operation above stating that they predict the probability of speech data given some particular transcription. The more difficult part conceptually, and the focus of this dissertation, is how to find the optimal HMM parametrisation M. It is not feasible to guess or manually estimate the parametrisation; this has to be deduced from training data. HMMs are

trained using sets of utterances paired with their transcriptions. The HMM parametrisation M is optimised using this training data, and the HMMs thus parametrised are then used to decode other speech– the "test data". It is assumed for purposes of the following discussion that the training examples and the utterances which are to be recognised are drawn from the same set. To justify this, the existence of an infinite set of possible utterances is hypothesised, which will be denoted as "the set of all utterances" although in practice it would be confined to a certain language and accent group. We then state that both the training and test data are drawn randomly from this same set of all utterances.

Maximum Likelihood estimation

The "canonical" method of estimating the parameters M from speech training data is by maximum likelihood estimation. This means we aim to choose that value of M which maximises the ML objective function, $F_{\mathrm{ML}}(M|O,T)$, which is the probability of the training data given the transcription. For I training utterances, this is given as follows:

$$\begin{aligned} F_{\mathrm{ML}}(M|O,T) &= \prod_{i=1}^{I} P(O_i|T_i,M) \\ &=_{\mathrm{def}} P(O|T,M) \end{aligned} \tag{1.6}$$

In ML, then, we maximise the probability of the observed speech data given the model. Assuming the speech training data (O,T) was actually produced by a HMM (this assumption is called "model correctness") and that all parametrisations M are equally likely, it is easy to show that the ML criterion $F_{\mathrm{ML}}(M|O,T)$ is proportional to the *a posteriori* probability of M being the model that really produced the speech:

$$\begin{aligned} P(M|O,T) &= P(O|T,M)P(M)/P(O) \\ &\propto P(O|T,M) \text{ if } P(M) \text{ is constant.} \end{aligned} \tag{1.7}$$

But in fact it is not possible to show that all values of M are equally likely, so the ML criterion cannot be shown to be correct in the finite training set case. In fact, even if all values of M were equally likely (an un-informative prior over M), the ML criterion could still pick a model other than the one which produced the data, if the training data were finite. This can informally be said to be due to the training examples being unrepresentative of the set of all utterances.

If there is infinite training data, and still assuming model correctness, then, without knowing the prior, ML can accurately estimate the most likely parametrisation M to have produced the speech. Informally this can be said to result from the fact that the conditional term drowns out the prior. We can be said to be optimising:

$$\log\ P(M) + \sum_i \log\ P(O_i|T_i,M) \tag{1.8}$$

For any model parametrisation M, there will be an expected value $\mathcal{E}(\log P(O|T,M))$ of the log probability of an utterance drawn from the set of all utterances. We can scale the objective function without affecting its maximum, as:

$$1/I(lgP(M) + \sum_i^I lgP(O_i|T_i,M))) \tag{1.9}$$

which clearly approaches $E(lgP(O_i^1|T_i^1,M))$ as $I \to \infty$, regardless of the prior. Once it is shown that the model which actually generated the data is the one with the highest value of $\mathcal{E}(P(O|T,M)$, it is proven that ML correctly estimates the parametrisation. This will not be attempted here.

Problems with ML

One of the conditions for ML being a "good" estimator is the assumption of model correctness: i.e, that the speech was produced from a HMM. This is not a valid assumption, because the speech was produced by a human being and not a Hidden Markov Model. In practice, what we want to ensure is that the speech recogniser is correct as much of the time as possible. If the training data set size approached infinity, we could ensure this by optimising the following function, which will be denoted as the 'classification error' function:

$$F_{\mathrm{CE}}(M|O,T) = \frac{1}{I}\sum_{i=1} I \left\{ \begin{array}{l} 1 \text{ if utterance i correctly recognised} \\ 0 \text{ if utterance i not correctly recognised} \end{array} \right\} \tag{1.10}$$

If the training data set approached infinity the value of $F_{\mathrm{CE}}(M|O,T)$ would approach its expected value, and we would be able to find the optimal model parametrisation by minimising that function. Unfortunately, this scheme is impracticable because the objective function is not differentiable w.r.t the parameters of M, and it is therefore very difficult to optimise M based on this criterion. There are, however, a number of objective functions which are related to this objective function in that they attempt to maximise the accuracy of the model in recognising speech. These are called *discriminative* objective functions.

Validity of different objective functions

An interesting question relating to these different objective functions is: how can a the classification error objective function be shown to be correct, while a different function, ML, has already been shown to be correct under a more restricted, but not disjoint, set of conditions? The trivial answer to this is that two different objective functions can be correct at the same time; they do not necessarily produce different results as the amount of training data approaches infinity. Presumably they would converge to the same point. More specifically: it has been proven that ML will find the HMM which generated the data, assuming that the data was indeed produced by a HMM. To prove that this is the same one as chosen by the "classification error" function would involve proving that the HMM which produced the data is the one which gives the lowest expected error rate during recognition. Again, this will not be attempted here.

1.2 Discriminative objective functions

Minimum Classification Error

Bing-Hwang Juang *et al* [**?**] present an objective function which is intended as a fairly approximation to the classification error function. It approaches the classification error function as certain parameters of the objective function approach infinity. The objective function is named Minimum Classification Error (MCE).

First the following function is defined:

$$d(O;M) = -\log P(O|T_{\mathrm{correct}}, M) + \mathrm{softmax}_{i\neq correct} \log P(O|T_i, M) \tag{1.11}$$

where

$$\mathrm{softmax}_{i=1}^{I} f(i) = \frac{1}{\nu}\log\left[\frac{1}{I-1}\sum_i exp(f(i)\nu)\right] \tag{1.12}$$

which approaches $\max_{i=1}^{I} f(i)$ as $\nu \to \infty$. As $\nu \to \infty$, the function $d(O;M)$ becomes a measure of whether or not the utterance is correctly classified by a recogniser, being negative when the utterance is correctly classified and positive otherwise. $d(O;M)$ is then embedded in a sigmoid function:

$$l(d) = \frac{1}{1 + exp(-\gamma d + \theta)} \tag{1.13}$$

which approaches a simple zero-one function as $\gamma \to \infty$. θ is normally set to zero, and $\gamma \geq 1$. The function $l(d(O; M))$ is then minimised. They report a 25% relative error reduction (for both words and strings) on the digit recognition model it was tested on, as compared to ML. In the models they used, inter-digit dependency was achieved by having as many left states in the model as there were left contexts, and as many right states as right contexts. I mention this because there is an interaction between success of discriminative versus ML estimation, and model complexity. The good performance of the technique was probably dependent upon the relative simplicity of the task it was applied to. This will be discussed later.

Maximum Mutual Information

A more widely used (or at least more widely discussed) discriminative objective function is the Maximum Mutual Information (MMI) objective function. This, in common with other discriminative techniques, aims to maximise the probability of the correct transcription at the expense of other transcriptions. Stated more exactly, it seeks to maximise the posterior probability, given the recogniser [which includes the language model, $P_{lm}(\cdot)$] of the correct string for each training utterance.

$$F_{\mathrm{MMI}}(M|O,T) = P(T|O,M) = \sum_i log \frac{P_\lambda(O(i)|w(i))\ P_{lm}(w(i))}{\sum_{\text{all w}} P(O(i)|w) P_{lm}(w)} \tag{1.14}$$

which can be expressed as:

$$\begin{aligned} D &= \sum_{i=1}^{I} log P_\lambda(w(i)|O(i)) \\ &= \sum_{I=1}^{I} log P_\lambda(O(i)|w(i)) + log P_{lm}(w(i)) - log P_\lambda(O(i)|R) \end{aligned} \tag{1.15}$$

where R is the recognition model, incorporating the language model probabilities as well as the HMM parameters. The HMM corresponding to $w(i)$, which in the case of large vocabulary continuous speech recognition (LVCSR) would be obtained by concatenating individual phone HMMs, is referred to as the numerator HMM, while R is the denominator HMM.

MMI derives its name from the fact that it is an information theoretic measure of the 'mutual information' between the recognise-r's output and our knowledge of what the speaker was saying: roughly speaking, how similar the two information sources are. In practice, it has some important advantages. An essential one is that it is differentiable. Another is that it takes all utterances into account on a roughly equal basis. This is in contrast to, for example, the minimum classification error approximation presented above, which only takes into account utterances which are near the boundary of being mis-recognised. One might think that it is an advantage to focus on utterances which are on this boundary. This might indeed be the case with infinite training data; however, with a limited training set it means that we are effectively ignoring many of the speech samples, reducing the effective size of the training set. This is a problem for generalisability, by which term I will denote the problem of inferring facts about utterances as a whole, from a limited sample of utterances.

Generalisability of Discriminative algorithms: motivation for FD

Discriminative optimisation algorithms in general suffer from lack of generalisability: they perform well on the training set but the difference between training set performance and test set

performance is greater than with maximum likelihood trained HMMs. This can be explained in terms of their objective functions, as follows: discriminative algorithms work by maximising the the probability of correct transcriptions and decreasing the probability of confusions: i.e, those incorrect transcriptions which tend to be generated by the recogniser. In order to work well on the test set, each possible type of confusion should be represented in the training set in proportion to the probability of its occurrence in the language as a whole. But this is unlikely to happen, because there are so many possible confusions but only a limited set of training data.

Consider, for example, a context-dependent phone-based Large Vocabulary Speech Recognition (LVCSR) system. For one context-dependent phone there will probably be several context-dependent phones with which it will confused in the language as a whole. But this particular context-dependent phone may only appear a handful of times in the training data, and it might well happen that on all those occasions the competing paths through the input (i.e, the incorrect transcriptions) are dominated by one particular path, perhaps even the correct one, leaving very few instances of confusions on which to train. Despite this, we still would ideally like several occurrences of our context-dependent phone mis-recognised in each of the several likely ways. This is clearly not going to happen. Kapadia [**?**] mentions this as a motivation for FD. He hypothesises that it might be profitable to increase the number of confused states– to alter the recognition model in such a way as to make more of the PDFs which correspond to incorrect transcriptions match the input, and use this altered model on the denominator of an MMI-like objective function. The hope is that we can alter the recognition model in such a way that more states are confused with the correct states for a given input, and these states will be roughly the same states which will be mistakenly assigned to those inputs in the language at large.

Frame Discrimination in its most general form

The general form of the FD objective function is an altered form of MMI, in which the denominator model is more general than the standard recognition model, in the sense of allowing a superset of the states which would be confused in MMI with the input, to be confused with it. This is denoted as follows:

$$F_{\mathrm{FD}}(\lambda) = \sum_{u=1}^{U} \log P_\lambda(O(u)|w(u)) + \log P^{lm}(w(u)) - \log P_\lambda(O(u)|N) \tag{1.16}$$

where N is the new denominator HMM.

Asymptotic sub-optimality of FD

Kapadia points out that this objective function will most likely result in worse training set accuracy than MMI, because it is a less accurate measure of the confusions generated while recognising the training set. These confusions will be swamped out by new confusions allowed by the model N. This implies that FD is suboptimal as the amount of training data approaches infinity. Both ML and MMI, subject to different assumptions, can be shown to optimally estimate the HMM parameters as the amount of training data becomes infinite. But FD trades off asymptotic accuracy for performance on limited training sets

Actual denominator HMM used for FD

It has been stated that in FD a more general HMM than the recognition HMM is used for the denominator, but no examples of the possible nature of the HMM N have been presented. The model that is used by Kapadie [**?**], and which has been followed here, is a HMM with only one state, whose output distribution is all the outputs of other HMM states, summed together. This is a sum of all the Gaussian s in the system, each multiplied by their weights c_{jm} in the state PDFs. See Section **??** for an explanation of this notation.

Previous results

See Chapter ?? for previous results for the FD, MMI and and Minimum Classification Error objective functions.

Chapter 2

Re-estimation formulae

Whatever objective function is being used, there must be some means of changing the model parameters in order to maximise it. This is done iteratively: for none of these objective functions is there an analytical solution. The easiest objective function to maximise is ML. This is because there exist equations, known as the Baum-Welch equations (BW) which are guaranteed on each iteration to increase the objective function. What is more, the equations are not only guaranteed to make such an increase for very small step sizes (as is the case with the update equations used for discriminative techniques, known as the extended Baum-Welch equations, or EBW), but rather they bring about a substantial increase in output likelihood $P_\lambda(O|w)$ on each iteration.

2.1 HMMs

In order to present the BW equations, it is first necessary to describe the HMM as used for speech recognition in more detail. This will be done for the case where each utterance is described by a single HMM. In practice, we would concatenate many HMMs describing individual phones to produce a composite HMM for an entire utterance, but the presentation here is easily extended to such a case. In addition to showing the BW equations, a novel proof of their validity will be presented.

The speech data and HMMs

A HMM is designed to give a likelihood for an utterance. We will assume for purposes of this exposition that the HMM represents a transcription w, so that the likelihood of the speech data $\mathbf{O}$ given w is being calculated. As stated above, the speech data is split into time frames $t = 1...T$, so that:

$$\mathbf{O} = \mathbf{o}_1, \mathbf{o}_2, ..., \mathbf{o}_T$$

The $\mathbf{o}_t$ are vectors, which are supposed to represent the most important information about the speech sound at that time period. More detail than this is not necessary for the purposes of this discussion. A HMM predicts the likelihood of this sequence of time frames o_t, in the following way. It has a sequence of states $1..N$, and each state has a probability distribution $b_j(\mathbf{o})$ associated with it. The transition from state to state is also probabilistic, with probability a_{ij} for the transition from state i to state j. We can give the probability of the output for a sequence of states $X = x_1, x_2, ..., x_T$, as follows:

$$P(O|X, M) = a_{1x_1} b_{x_1}(\mathbf{o}_1) a_{x_1x_2} b_{x_2}(\mathbf{o}_2) a_{x_2x_3} b_{x_3}(\mathbf{o}_3) ... b_{x_T}(\mathbf{o}_T) a_{x_TN}$$

This follows the convention in speech recognition systems of having non-emitting states (states without a probability distribution) 1 and N, to facilitate the concatenation of models. It has

now been shown how to calculate the probability of the output given a state sequence X and the HMM M. However, we do not know X (That is why it is called a *hidden* Markov model). The probability $P(O|M)$ is given either by summing over all X, or by taking the most likely state sequence. For recognition tasks, it is usual to take the most likely state sequence. In fact, taking the most likely sequence during recognition is necessary in order to know the transcription because of the way continuous speech recognition works. This taking of the most probable sequence is known as the Viterbi approach. For HMM training, which is what we are concerned with here, it is more usual to take the sum over all possible state sequences. This is referred to as the Baum-Welch approach. We do not have to calculate this sum over all state sequences explicitly; which is fortunate, since the number of possible state sequences grows exponentially with the input length. There is an algorithm, called the forward-backward algorithm, which can calculate efficiently all the information we need to know while re-estimating HMM parameters.

State output likelihood functions

It has been mentioned that states have output probability distributions $b_j(\mathbf{o})$, but their form has not been described. In the system being discussed here (and in most if not all systems), they are mixtures of one or more Gaussian s. The functional form, in a system where we have M mixtures per state and input vectors of size n, is as follows:

$$\begin{aligned} b_j(\mathbf{o}) &= \sum_{m=1}^{M} c_{jm} b_{jm}(\mathbf{o}) \mathcal{N}(\mathbf{o}; (\mu), (\Sigma)) \\ &= \sum_{m=1}^{M} c_{jm} b_{jm}(\mathbf{o}) \frac{1}{\sqrt{(2\pi)^n |\Sigma_{jm}|}} e^{-\frac{1}{2}(\mathbf{o}-\mu_{jm})^T \Sigma_{jm}^{-1} (\mathbf{o}-\mu_{jm})} \end{aligned} \tag{2.1}$$

where μ_{jm} and Σ_{jm} are the means and variances of those Gaussian mixtures. c_{jm} in the above equation are the mixture weights; there is a constraint that $\sum_m c_{jm} = 1$ for all j.

Alignments

The alignment is the assignment of time frames to states, for an utterance O and a HMM parametrised as M. In the Viterbi alignment, where we are just taking the most probable path, the alignment may be said to consist of knowing which state produced the output at each time period from $1...T$. In the BW case (where we are summing over alignments), the alignment $L_j(t)$, which represents the probability that state j produced the output at t, may be defined as follows:

$$\begin{aligned} L_j(t) &= \sum_i \delta(x_t^{(i)} = j) P(X^{(i)}|O, M) \\ &= \sum_i \delta(x_t^{(i)} = j) \frac{P(O|X^{(i)}, M)}{\sum_{\hat{i}} P(O|X^{(\hat{i})}, M)} \end{aligned} \tag{2.2}$$

where $\delta(c) = 1$ when condition c is true, and zero otherwise. $L_j(t)$ for all j and t can be found more efficiently using the forward-backward algorithm, which I will not describe here. The forward-backward algorithm is linear in both the size of the HMM and the length of the input. The occupancy $L_j(t)$ for states has been described: we may require in addition the alignment for each mixture component of the Gaussian, i.e, the proportion of the output probability for which that mixture component was responsible. This is:

$$L_{jm}(t) = \sum_i \delta(x_t^{(i)} = j) \frac{P(O|X^{(i)}, M)}{\sum_{\hat{i}} P(O|X^{(\hat{i})}, M)} \frac{b_{jm}(\mathbf{o}_t)}{b_j(\mathbf{o}_t)} \tag{2.3}$$

2.2 The Baum-Welch re-estimation formulae

The Baum-Welch re-estimation formulae aim to maximise $P(O|M)$. They are very simple.

$$\hat{\mu}_{jm} = \frac{\sum_{t=1}^{T} L_{jm}(t)\mathbf{o}_t}{\sum_{t=1}^{T} L_j(t)} \tag{2.4}$$

$$\hat{\Sigma}_{jm} = \frac{\sum_{t=1}^{T} L_{jm}(t)(\mathbf{o}_t - \mu_{\mathbf{jm}})(\mathbf{o})_{\mathbf{t}} - \mu_{\mathbf{jm}})^{\mathbf{T}}}{\sum_{t=1}^{T} L_{jm}(t)} \tag{2.5}$$

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T} L_{jm}(t)}{\sum_{t=1}^{T} L_j(t)} \tag{2.6}$$

The re-estimation formulae for a_{ij} are usually given in terms of the backward and forward probabilities $\alpha_j(t)$ and $\beta_j(t)$, which relate the forwards-backwards algorithm and which I have not described here. They will be given here in a different form, more convenient for this proof:

$$\hat{a_{ij}} = \frac{\sum_{t=1}^{T-1} \sum_i \frac{P(O|X^{(i)})}{\sum_{\hat{i}} P(O|X^{(\hat{i})}, M)} \delta(\mathbf{x_t^{(i)}} = \mathbf{i}) \delta(\mathbf{x_{t+1}^{(i)}} = \mathbf{j})}{\sum_k \sum_{t=1}^{T-1} \sum_i \frac{P(O|X^{(i)})}{\sum_{\hat{i}} P(O|X^{(\hat{i})}, M)} \delta(\mathbf{x_t^{(i)}} = \mathbf{i}) \delta(\mathbf{x_{t+1}^{(i)}} = \mathbf{k})} \tag{2.7}$$

Proof of Baum-Welch re-estimation

A novel proof of Baum-Welch re-estimation is presented in Appendix **??**. It relies on transforming the sum of probabilities of possible state sequences $X^{(i)}$ to a weighted product (alternatively, a weighted sum of logarithms), and showing that an increase in the product results in an increase in the original sum.

Summary of Baum-Welch re-estimation

The Baum-Welch re-estimation formulae have been presented, and it has been proved that they result in an increase (or no change) in the objective function for ML. In practice, these formulae result in quite a substantial increase in the objective function, and four or five iterations are sometimes all that is necessary to train a HMM. Unfortunately, however, the Baum-Welch re-estimation formulae are not applicable to discriminative algorithms.

2.3 Extended Baum-Welch re-estimation formulae, and re-estimation in discriminative training.

The extended Baum-Welch re-estimation formulae are useful for training HMMs using discriminative criteria; they are used in the experiments reported here.. These equations contain smoothing constants which are supposed to be infinite for the behaviour of the algorithm to be theoretically guaranteed in any way, but in practice have to be set to small values. This means that the algorithms as used in practice are unproven; but they seem to work well. The reader is referred to [**?**] for a presentation of the proof of the validity of their proof. The equations are as follows:

$$\hat{\mu}_{jm} = \frac{\left\{\theta_{jm}^{num}(O) - \theta_{jm}^{den}(O)\right\} + D\mu_{jm}}{\left\{\lambda_{jm}^{num} - \lambda_{jm}^{den}\right\} + D} \tag{2.8}$$

$$\hat{\sigma}_{jm}^2 = \frac{\left\{\theta_{jm}^{num}(O^2) - \theta_{jm}^{den}(O^2)\right\} + D\left(\sigma_{jm}^2 + \mu_{jm}^2\right)}{\left\{\lambda_{jm}^{num} - \lambda_{jm}^{den}\right\} + D} - \hat{\mu}^2 \tag{2.9}$$

where the superscripts *num* and *den* represent the numerator and denominator HMM occupancies respectively. $\theta_{jm}(O)$ represents the sum of the output vectors (i.e, the training data) summed

according the occupancies of the mixture component (j,m), while $\theta_{jm}(O^2)$ is the squared input data summed similarly. λ_{jm} is the total occupancy of the mixture, summed over all time frames.

The re-estimation formula for the mixture weights, in its original form, is as follows:

$$\hat{c}_{jm} = \frac{c_{jm}\left\{\frac{\delta F(M)}{\delta c_{jm}} + C\right\}}{\sum_{\hat{m}} c_{j\hat{m}}\left\{\frac{\delta F(M)}{\delta c_{j\hat{m}}} + C\right\}} \tag{2.10}$$

where $F(M)$ is the value of the objective function, and the derivative $\frac{\delta F(M)}{\delta c_{jm}}$ is given as follows:

$$\frac{\delta F(M)}{\delta c_{jm}} = \frac{1}{c_{jm}}\left(\lambda_{jm}^{num} - \lambda_{jm}^{den}\right) \tag{2.11}$$

However, the value of the derivative $\frac{\delta F(M)}{\delta c_{jm}}$ can be quite large for small c_{jm}; one can see that it could easily outweigh the smoothing constant C if C were too small. In practice, C is set so that all mixture weights remain positive; so what happens is that these small mixture weights will give rise to a very large value of C, which in turn may not be appropriate for other states in which no mixture weights have small values. Merialdo [**?**] has suggested the following approximation: to $\frac{\delta F(M)}{\delta c_{jm}}$:

$$\frac{\delta F(M)}{\delta c_{jm}} \simeq \frac{\lambda_{jm}^{num}}{\sum_{\hat{m}} \lambda_{j,\hat{m}}^{num}} - \frac{\lambda_{jm}^{den}}{\sum_{\hat{m}} \lambda_{j,\hat{m}}^{den}} \tag{2.12}$$

In fact, the term 'approximation' is a misnomer because we already know $\frac{\delta F(M)}{\delta c_{jm}}$, and the above expression is clearly not near its value. The update equations thus altered to not even guarantee that the direction of motion of the new $\hat{c}_{jm}$ is correct; this can easily be verified using simple examples. The update equations have, however, been reported to work well in practice (Valtchev [**?**], Normandin [**?**], Kapdaia [**?**]). These update equations have been used for purposes of comparison with some new update equations for mixture weights, which are presented below.

Setting the constants C and D

C was set to the minimum value necessary to ensure that all updated weights remain positive. After Valtchev [**?**], D set was on a phone-by-phone basis to a constant I will call *DFACTOR* times the minimum value necessary to ensure that all variances remain positive. To obtain this value it is necessary to solve a quadratic equation for each mixture; the reader is referred to Valtchev [**?**] for details. In [**?**], *DFACTOR* was set to 2. Experiments reported here show that it may be necessary to set it higher (e.g, to 4) for the sake of recognition accuracy, especially where there are few mixtures.

Alternate weight update equations

Alternate update equations for the mixture weights in discriminative training are now described and tested. These equations are found to outperform Merialdo's approximation (described above), and so were used in the FD evaluation reported here. The motivation for these update equations is as follows.

Notice that the Baum-Welch update equations can be derived from the assumption that all occupancies remain constant as the parameters change. The proof is similar to part of the proof of the B-W equations presented above. This observation is not such a startling one, and it is not intrinsically useful to prove starting from a false assumptions a theorem which has already been

proven starting from correct assumptions. However, it will serve as the basis for update equations for use in discriminative re-estimation.

The B-W equations for mixture weights can be proven starting from a less restrictive assumption than that the occupancies stay constant with changing model parameters. They can be proved assuming that:

1. The occupancies associated with a mixture of a state are only a function of the mixture weight.
2. The occupancy of a mixture does not decrease with increasing mixture weight
3. The occupancy of a mixture does not increase with decreasing mixture weight
4. The above two assumptions also hold for sets of mixtures: the summed occupancy of a set of mixtures does not change with opposite sign to its summed mixture weight.

The first assumption is ill-founded, but we will ignore that. Consider, an example where we have two mixtures (c_1, c_2) and the observed occupancies are both equal (at, say, 0.5). The initial occupancies are $0.1, 0.9$. For values $0.1 \leq \hat{c}_1 < 0.5, \quad 0.9 \geq \hat{c}_2 > 0.5$ we have:

$$\begin{aligned} \frac{\delta F}{\delta \hat{c}_1} - \frac{\delta F}{\delta c_2} &= \frac{1}{\hat{c}_1}\hat{\lambda}_1 - \frac{1}{\hat{c}_2}\hat{\lambda}_2 \\ \text{but} \quad \hat{\lambda}_1 &\geq 0.5, \text{since } \hat{c}_1 > c_1 \\ \text{and} \quad \hat{\lambda}_2 &\leq 0.5, \text{since } \hat{c}_2 < c_2 \\ \text{and} \quad \hat{c}_2 &> 0.5, \\ \hat{c}_1 &< 0.5 \\ \text{so} \quad \frac{\delta F}{\delta \hat{c}_1} - \frac{\delta F}{\delta \hat{c}_2} &> 0 \end{aligned}$$

This means that it is possible to gradually increase mixture weight $\hat{c}_1$ to 0.5 at the expense of $\hat{c}_2$, while knowing at each stage of this increase that we are guaranteed to be increasing the objective function (assuming our assumptions hold.) Notice, however, that in practice we would expect that increasing the value of c_1 would increase its occupancy, leading us to an optimum value > 0.5; but our assumption that the occupancy does not increase with increasing mixture values leads us to err on the side of safety, and not move the mixtures too far.

These same assumptions, when applied to discriminative re-estimation, do not allow us to make any change at all. The reason is that the bounds we have specified are now of the wrong sign to enable us to make any change in the mixture weights. In order to prove anything in the discriminative case, it is necessary to get opposite bounds on the denominator weights. We will assume:

1. The numerator and denominator occupancies associated with a mixture of a state are only a function of the mixture weight.
2. The numerator occupancy of a mixture does not decrease with increasing mixture weight
3. The numerator occupancy of a mixture does not increase with decreasing mixture weight
4. The denominator occupancy of a mixture does not decrease by a greater factor than the decrease of the mixture.
5. The denominator occupancy of a mixture does not increase by a greater factor than the increase of the mixture.

Say for example that we have initial occupancies $c_1^n = 0.1, c_2^n = 0.9$, and numerator and denominator occupancies are: $\lambda_1^n = 1.0, \lambda_1^n = 1.0, \lambda_1^d = 0.5, \lambda_2^d = 0.5$.

We want to increase c_1 and decrease c_2; this can be confirmed by calculating $\frac{\delta F}{\delta \hat{c}_1} - \frac{\delta F}{\delta \hat{c}_2}$ at the present mixture weights; we get: $\frac{1}{0.1} \cdot (1.0 - 0.5) - \frac{1}{0.9} \cdot (1.0 - 0.5) > 0$. In general, we have:

$$\frac{\delta F}{\delta c_1} - \frac{\delta F}{\delta c_2} = \frac{1}{\hat{c}_1}(\hat{\lambda}_1^n - \hat{\lambda}_1^d) - \frac{1}{\hat{c}_2}(\hat{\lambda}_2^n - \hat{\lambda}_2^d) \tag{2.13}$$

For $\hat{c}_1 > c_1$ and $\hat{c}_2 < c_2$, we can derive from our assumptions a minimum value for this differential:

$$\frac{\delta F}{\delta c_1} - \frac{\delta F}{\delta c_2} \geq \frac{1}{\hat{c}_1}(\lambda_1^n - \lambda_1^d \frac{\hat{c}_1}{c_1}) - \frac{1}{\hat{c}_2}(\lambda_2^n - \lambda_2^d \frac{\hat{c}_2}{c_2}) \tag{2.14}$$

$$\frac{\delta F}{\delta c_1} - \frac{\delta F}{\delta c_2} \geq \frac{1}{\hat{c}_1}(0.5 - 0.5\frac{\hat{c}_1}{0.1}) - \frac{1}{\hat{c}_2}(0.5 - 0.5\frac{\hat{c}_2}{0.9}) \tag{2.15}$$

$$\frac{\delta F}{\delta c_1} - \frac{\delta F}{\delta c_2} \geq \frac{1}{\hat{c}_1}(0.5 - 0.5\frac{\hat{c}_1}{0.1}) - \frac{1}{1-\hat{c}_1}(0.5 - 0.5\frac{1-\hat{c}_1}{0.9}) \tag{2.16}$$

This value remains positive until $\hat{c}_1 = 0.177, \hat{c}_2 = 0.822$. If the denominator occupancies are lower, at 0.1 each, the values can change to $\hat{c}_1 = 0.393, \hat{c}_2 = 0.607$ before this differential becomes positive. Thus it is demonstrated for the case of two mixtures how these limits on the occupancies can enable us to change the parameters while being sure that such change is justified. For more mixtures it would be difficult to use this same technique explicitly; however, it should be clear that to assume the occupancies remain at the given boundaries and solving $\frac{\delta F}{\delta c_1} - \frac{\delta F}{\delta c_2} = 0$ gives the correct result for the 2 mixture case; and by assumption 5 this can be generalised to M mixtures, by combining all but one mixture to make a composite mixture, and once the single mixture chosen is fixed, repeating the process for a mixture chosen from the remaining ones. But the process of finding the updated $\hat{c}_m$ is made easier, and is in any case equivalent, if ones assumes the numerator and denominator values to be fixed at the stated bounds, and integrating w.r.t $\hat{c}_m$f to give the value of the objective function as a function of the mixture weights. The objective function thus estimated is then minimised. Restating the bounds as equalities:

$$\hat{\lambda}_m^n = \lambda_m^n \tag{2.17}$$

$$\hat{\lambda}_m^d = \frac{\hat{c}_m}{c_m}\lambda_m^d \tag{2.18}$$

It follows from the optimisation criterion that $\frac{\delta F}{\delta \log(c_m)} = \lambda_m^n - \lambda_m^d$. So we have:

$$\begin{aligned} \frac{\delta F}{\delta \log(\hat{c}_m)} &= \lambda_m^n - \lambda_m^d \frac{\hat{c}_m}{c_m} \\ F &= C + \log \hat{c}_m \lambda_m^n - \frac{\lambda_m^d}{c_m}\hat{c}_m \\ (\hat{c}_1, ...\hat{c}_M)^{opt} &= \arg \max_{(\hat{c}_1,...\hat{c}_M)} \sum_{m=1}^{M} \log \hat{c}_m \lambda_m^n - \frac{\lambda_m^d}{c_m}\hat{c}_m \end{aligned} \tag{2.19}$$

subject to the constraint that $\sum_{m=1}^{M} c_m = 1$. The equation does not have an easy analytical solution, but it can be found using numerical algorithms. This is the approach I have adopted in my tests on the Resource Management corpus, after using a small task to compare this weight update equation with Merialdo's approximation (referred to above).

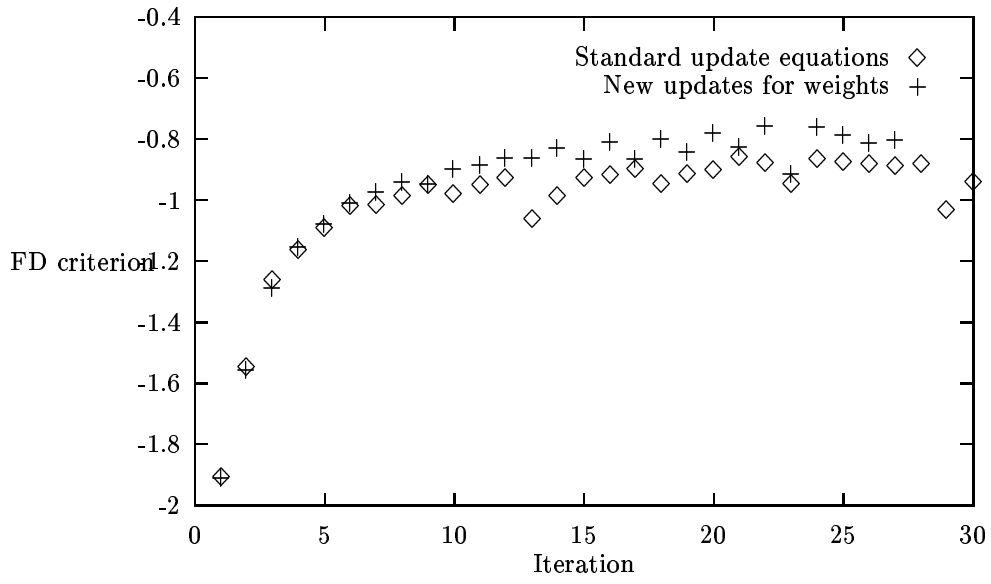


Figure 2.1: Plot of discriminative criterion against iteration, for a small task

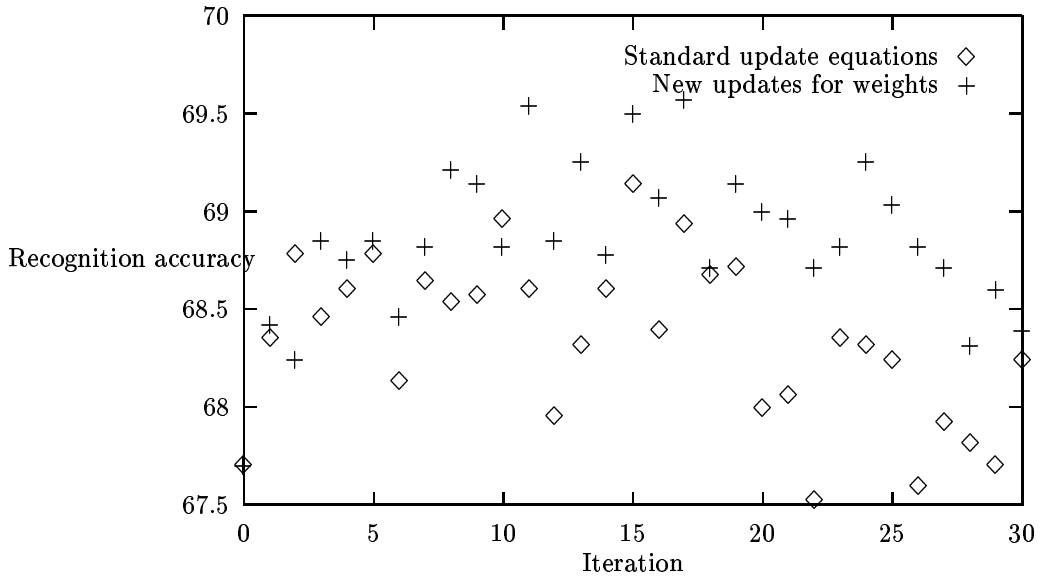


Figure 2.2: Plot of training set accuracy against iteration, for a small task

Chapter 3

Implementation

In the foregoing chapters, various objective functions have been described, including the FD function. The extended Baum-Welch update equations have been described, along with an alteration to them which is used here. Several iterations of updates using these equations are required before the objective function being optimised (in this case the FD criterion) begins to converge. In this section, the program (HDRest) which performs one of these updates is described.

Taking account of training files

The EBW equations contain occupancies λ^n_{jm} and λ^d_{jm}: the occupancies of individual mixtures in the numerator and denominator HMMs respectively, summed over time. They also refer to the quantities $\theta_{jm}(O)$ and $\theta_{jm}(O^2)$, which consist of the data values $\mathbf{o}_t$, and the squared data values, summed weighted by occupancy over all the input.

The reader will recall that the numerator HMM corresponds to a transcription of the training utterance $w(i)$. The denominator HMM is a single-state HMM whose PDF is all mixtures in the system added together. The denominator occupancies were conceptually very simple to compute; for each time frame, the occupancy for a mixture (j, m) is as follows:

$$\lambda_{jm} = \frac{c_{jm}\mathcal{N}(\mathbf{o}_t|\mu_{jm}, \sigma_{jm})}{\sum_{\hat{j}} \sum_{\hat{m}} c_{\hat{j}\hat{m}}\mathcal{N}(\mathbf{o}_t|\mu_{\hat{j}\hat{m}}, \sigma_{\hat{j}\hat{m}})} \tag{3.1}$$

The numerator and denominator occupancies λ and data sums θ have to be calculated for each training file; after all training files have been accounted for in this way, the update equations are applied and the trained HMMs are written to disk, ready either for testing or for another iteration of training.

Convergence of FD criterion on a small task

FD estimation was initially implemented naively and tested on a small subset of the Resource Management corpus, which had been altered to use a small number of phone class models. The models did not, represent phones, but classes of phones, e.g, consonants, liquids, etc. It was thus verified that the equations caused the FD criterion to increase, and to converge fairly quickly to a maximum. Refer back to Figure **??** for a plot of this convergence. This figure shows the convergence of the FD objective function both with Merialdo's [**?**] approximation to the weight updates, and the new approximation presented here. As stated previously, the new update equations were found to perform best, and have been used in further experiments.

Computation problem

To implement FD, it is necessary on each time frame of each input file to scan over all the Gaussians in the system, calculating the probability of the input vector $\mathbf{o}_t$ given that Gaussian. For the larger of the models which were tested on the complete Resource Management task, i.e, the 6 mixture model, there were 9500 mixtures in the system, each a 39-element vector. To calculate the value of each mixture would have taken days for each re-estimation of the model set, which was unacceptable, especially considering that an eventual aim of the project was to move onto even larger systems.

Evaluation of speedup techniques

It was necessary to introduce optimisation schemes, which involved restricting the evaluation of Gaussian pdf's to a subset of the Gaussians in the system– a subset which is likely to contain the top few most important Gaussians for that data point, i.e, those which, when evaluated at the input value, give the highest probability. In measuring the success of these optimisation schemes, two measures are used in conjunction with each other: the percentage of Gaussians actually calculated and the average decrease in the log probability of the output. Due to not all the Gaussians being calculated each time, the probability assigned to the data is bound to decrease, but the aim is to make it decrease as little as possible while evaluating as low a proportion of the Gaussians in the system as possible.

Baseline

For some related results to compare with, the baseline results reported in Gales [**?**] are suggested. He reports a change in average likelihood of 0.021, with 35% of the Gaussians calculated. Those results are not quite comparable with those reported here, as they are based on a percentage of Gaussians calculated from lattice rescores, whereas these are based on a percentage of Gaussians calculated out of all the Gaussians in the system. He points out that less Gaussians would have been calculated in his system if it had been based on a recognition run (let alone on all the Gaussians in the model.) However, the order of the improvement eventually achieved is such as to probably outweigh these considerations. Also, some of the results obtained while investigating possibilities are from a system which does not differ significantly from conventional VQ, and so can serve as a baseline. These will be mentioned when the results of the optimisation are presented.

Intermediate results and introduction

Many different approaches were tried before the final technique was chosen. These are described in Appendix **??**. To reiterate the objective of these techniques: at each time frame, we would ideally like a list of the few Gaussians that best match the output vector $\mathbf{o}_t$. Most of the Gaussians in the system will have an occupancy very close to zero for any particular time frame. Therefore it is not necessary to take account of these, and it is best to avoid evaluating them.

3.1 Roadmap algorithm

The algorithm developed and used to find the best Gaussians is referred to here as the Roadmap algorithm. It involves first setting up a similarity relation between the Gaussians in the system, with each Gaussian being annotated with an ordered list of those most similar to it, the most similar first. Then the similarity relation is used to navigate among the Gaussians of the system, hopefully towards the one which best matches the input.

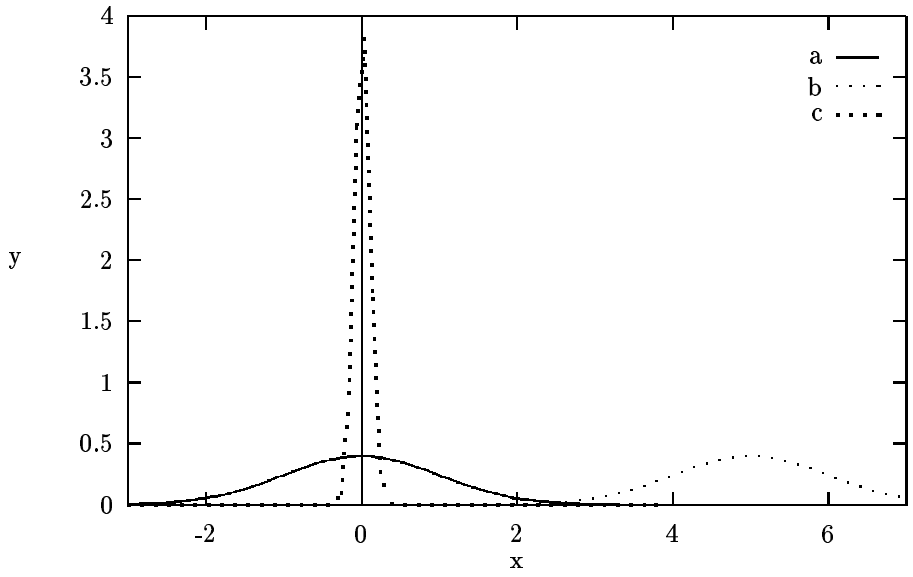


Figure 3.1: Divergence $(a, c) \simeq 50$, $(a, b) \simeq 25$

Problem with divergence

As mentioned in Appendix **??**, when a similar algorithm was tried using the divergence between two Gaussians as the distance measure, it was found to represent poorly the chance of two Gaussians both scoring highly on the same point. For instance, look at the Gaussians in Figure **??**. One would expect a and c to be more similar than a and b in terms of what data points they might match, but their divergence is greater.

New distance measure

It was decided that a distance measure more appropriate for this task was one which measured the overlap of Gaussians. Intuitively, overlap is the part of the graph that both Gaussians cover, being

$$\int_{x=-\infty}^{\infty} min(g_1(x), g_2(x))\delta x \tag{3.2}$$

This can be calculated with the help of a table giving the Gaussian integral, by first solving a quadratic equation to find the two points where the Gaussians overlap, and then looking up the appropriate integrals over those regions. There is a special case where the two means and variances are equal and there may be no unique solutions to the quadratic equation, but this can be handled approximately by perturbing one of the variances.

The "overlap" for a multidimensional diagonal-covariance Gaussian was calculated by multiplying the overlaps of all the dimensions together.

Calculating the overlap as described above proved too time-consuming, but an approximation was made, and its coefficients optimised using numerical routines. The following approximation was used for cases where $\sigma_2 \leq \sigma_1$; the Gaussians were swapped in the other case.

$$\sigma = \frac{\sigma_2}{\sigma_1} \tag{3.3}$$

$$\mu = \frac{\mu_2 - \mu_1}{\sigma_1} \tag{3.4}$$

$$\tag{3.5}$$

$$\begin{aligned} \log(\text{overlap}) \simeq & \frac{-0.0557}{\sigma} + 0.3137\sigma - 0.3292\mu^2 \\ & - 0.0037\frac{\mu^2}{\sigma} + 0.0429\mu^2\sigma - (0.3137 - 0.0557) \end{aligned} \tag{3.6}$$

Overlap-based distance measure

For two Gaussians $(\mu^{(1)}, \Sigma^{(1)})$ and $(\mu^{(2)}, \Sigma^{(2)})$, where the covariance matrices are diagonal, the distance measure is a sum over the dimensions as follows:

$$-\sum_i \log \ \textbf{overlap}((\mu_i^{(1)}, \Sigma_{ii}^{(1)}), (\mu_i^{(2)}, \Sigma_{ii}^{(2)})) \tag{3.7}$$

Setting up the similarity measure

An algorithm was developed to set up each Gaussian g with a set $S(g)$ of the n most similar Gaussians to it, where n was 20 in experiments reported here. This algorithm was tested by calculating the list of most similar Gaussians by brute force for a few examples, and seeing to what extent this coincided with the results of the algorithm; it passed this test perfectly. The algorithm is as follows; it essentially consists of testing on each iteration, for each Gaussian g, those Gaussians f s.t $\exists h, h \in S(g) \wedge f \in S(h)$, and if necessary adding f to $S(g)$. Also, on each iteration, random Gaussians are tested for possible inclusion in $S(g)$, to 'seed' the algorithm. For each g and for each $f \in S(g)$, a record is kept of the iteration at which f was added to $S(g)$; this is referred to as iter$(f|g)$, and is used to avoid calculating overlaps more than necessary. If $f \notin S(g)$, then iter$(f|g)$ becomes undefined.

1. For each g set $S(g) = \epsilon$
2. Set n to the number of similar Gaussians we want recorded for each.
3. Set iter $= 1$
4. For each Gaussian g:
    5. Set $T = \epsilon$. T is the set of Gaussians we know do not belong in $S(g)$ because they have been tested before.
    6. For each $f \in S(g)$ do:
        7. $T = T \cup \{f\}$
        8. If iter$(f|g) <$ iter $- 1$ then
            9. For each $h \in S(f)$ do:
                10. If iter$(h|f) <$ iter $- 1$ then
                    11. $T = T \cup \{h\}$
    12. $S'(g) = S(g)$. For each Gaussian $f \in S'(g)$ do:
        13. For each Gaussian $h \in S(f)$ do:
            14. If $h \notin T$ then:
                15. $T = T \cup \{h\}$
                16. If $g \neq h$ then $S(g) = S(g) \cup \{h\}, S(h) = S(h) \cup \{g\}$
                17. If $|S(g)| > n$ then remove the most distant element from g; likewise for $S(h)$.
    18. For 20 random Gaussians r in the system:
        19. If $r \notin T$ then $S(g) = S(g) \cup \{r\}$
        20. If $|S(g)| > n$ then remove from it the most distant element from g.

21. iter = iter + 1
22. Go to line 4 unless there are already K similar Gaussians noted for each Gaussian, and the summed distance over all Gaussian pairs recorded has changed by less than 0.02% on this iteration.

It was found that the "canonical" overlap formula gave slightly better results in terms of finding the best Gaussians for input vectors than did the approximation. Therefore, after the similarity measure was set up using the approximate formula described above, all the similarities were rescored using the canonical formula, all records of iterations $\text{iter}(f|g)$ when similarities were recorded were set to zero, and above algorithm (limited to two iterations) was performed again, starting as above from iteration 1 but using this time the "canonical" formula in place of the approximation.

Finishing the similarity measure

After initialising the similarity measure as described above, its symmetric closure was performed. Then "redundant" similarities were removed. "Redundant" was defined by analogy with a road map. No road will normally be built between town A and town B if there is already a fairly direct route via town C (unless the inhabitants of town C protest about the traffic– a scenario which is not likely to occur when C is a mathematical function.) The precise definition of redundancy was optimised empirically, to give:

$$\begin{aligned} &\textit{Redundant}(a,b) \textbf{ iff} \\ &\exists c, \delta(a,c) < 0.9\ \delta(a,b) \wedge \delta(c,b) < 0.9\ \delta(a,b) \wedge \delta(a,c) + \delta(c,b) < 1.7\ \delta(a,b) \end{aligned} \tag{3.8}$$

With these parameters, around 15% of the links in a system with 19,500 Gaussians in total and $n = 20$, were removed. Both of these adjustments to the similarity measure (symmetric closure, and removal of redundant links) were found to improve performance of the Roadmap algorithm in terms of finding the most important Gaussians of the input while calculating as few as possible.

The Roadmap algorithm

The algorithm used to find the most promising few Gaussians, which will be called the Roadmap algorithm, is similar to the one described in Appendix **??** to find the best Gaussians using the divergence similarity measure.

Firstly, the similarities are set up as described above, and those Gaussians f being recorded as similar to any Gaussian g are sorted in order of similarity to g.

Note that in the following description, for heap operations, when $Top(h)$ is called it is understood that the top element is removed from the heap h. Set notation is used to describe heap operations in a way which it is hoped is obvious.

BestPDFs($\mathbf{x}$, NPDFS)

Set heap h, which stores (integer, float) pairs with largest floating values at the top, as follows:

If this is the first iteration, then $h = \{(1, \mathcal{N}(\mathbf{x}|\mu_\mathbf{1}, \sigma_\mathbf{1})\})\}$

Otherwise set h to the integer id's of the 20 best pdfs for the last input value, paired with their output values for $\mathbf{x}$.

Set the $u = \epsilon$. (u is the set of those Gaussians which are used up: i.e, all their similar ones have been tested)

Set $(besti, bestf) = Top(h)$.

Set $bestn = 1$

While $|h| + |u| + 1 <$ NPDFS do:

If bestn exceeds the number of similar Gaussians recorded for $besti$, then:

Set $u = u \cup \{(besti, bestf)\}$

If $|h| > 0$ then set $(besti, bestf) = Top(h)$

Else set $besti$ to some random Gaussian id which has not yet been evaluated for this input value, and $bestf = \mathcal{N}(\mathbf{x}|\mu_{besti}, \sigma_{besti})\}$

Set $tempi$ to the id of the $bestn$'th most similar Gaussian to $besti$, and $tempf = \mathcal{N}(\mathbf{x}|\mu_{tempi}, \sigma_{tempi})\}$.

If $tempf > bestf$ (i.e, the new one is closer than the active one) then:

Set $h = h \cup \{(besti, bestf)\}, (besti, bestf) = (tempi, tempf), bestn = 1$

Else set $h = h \cup \{(tempi, tempf)\}$

Return $h \cup u \cup \{(besti, bestf)\}$

Not shown in the above algorithm description for simplicity, two features were added to take account of extra knowledge we have about the probability that the top few Gaussians have already been found. These two pieces of extra knowledge are:

1. When the top Gaussian is missed, the probability of the input given the top Gaussian actually found tends to be lower than average. Therefore, it seems that when the top Gaussian actually found has a lower than average input, there is more chance of the top Gaussian being missed.
2. If the identity of the top Gaussian changes, it is worthwile searching some more to see of the are any higher ones in the immediate vicinity (I.e, make sure the area of the top Gaussian has been properly explored.

The changes are as follows.

1. A record of the distribution of the maximum log probability m of the output for any Gaussian in the system is kept. This is assumed to be a Gaussian distribution, and is calculated from a weighted sum of m and m^2, the weight being a decaying value starting at 1 for the current time frame and decaying by 0.999 at each time frame going back in time. This is convenient for computation. This distribution is used to detect when the probability of the input given the best Gaussian so far is lower than average, and accordingly to test more Gaussians than one would normally test. The number tested is increased by 50% for each standard deviation below the mean, up to a maximum of 150%.
2. If the identity of the best Gaussian to match the input has changed less than 50 iterations ago, then iterate regardless of whether we have already tested the number of Gaussians (NPDFS) we were instructed to return.

Summary of Roadmap algorithm

It has been described how to set up the similarity measure between Gaussians for use in this algorithm, and how the list of likely best Gaussians is obtained for each time frame. This list of top Gaussians, along with the value of each when applied to the input, are then used by the part of the program which collects denominator statistics on each time frame.

3.2 Results of Roadmap algorithm

For a 9,500 mix system, it was possible to test an average of only 355 vectors per time frame (3.7%), while the log probability per frame decreased by an average of 0.005. Some suitable results to compare this to are the VQ results reported in Appendix **??**, for a two-level clustering VQ scheme with 256 cluster centres. The two-level clustering scheme has been shown, by the similarity of the average distance of a Gaussian to its cluster, to be very similar to a single-level VQ scheme, so it will do for a baseline. It was applied to exactly the same system. A decrease in log probability of 0.3 was noted while calculating 4% of the Gaussians. Compared to this, the Roadmap algorithm gives nearly two orders of magnitude improvement in accuracy, while evaluating approximately the same number of Gaussians. A result from a different parametrisation of the two-level VQ is 0.04 difference in accuracy with 16% of the vectors calculated (this was obtained using 1024 VQ centres). Compared to this, the Roadmap algorithm attains an 8-fold increase in accuracy in conjunction with a 5-fold decrease in computation.

Applicability of the Roadmap algorithm

This algorithm may not be suited for normal speech recognition tasks because it assumes that one is interested in all Gaussians in the system which match the input; whereas for speech recognition purposes one may instead want to know which Gaussians within a certain set of HMM states, i.e, those states that are within the pruning beam, match the input.

3.3 Application of the Roadmap algorithm to FD

As stated, this algorithm was used in the HMM re-estimation program (HDRest) which was developed in this project. Each time HDRest was run, i.e, each time a re-estimation of the HMM set according to the FD criterion and the altered EBW equations was performed, the similarity measure was set up and then used on each of the training files. Without this algorithm, it may not have been possible to test FD on this task, due to the huge computational effort. As it was, taking gathering the denominator statistics for the 6-mixture system took 3.4 times as long as gathering the numerator statistics (using the forward-backward algorithm), and 9.9 times as long on the 1 mixture system, where nearly as many Gaussians were calculated per time frame despite the lesser number in the system. It might have been possible to further decrease the number calcualted in the 1 mixture system without decreasing accuracy too much, but it was not attempted as the re-estimation was fast enough already.

Chapter 4

Results

4.1 Previous results

Quantitative results for FD

In [1], FD was shown to give better test set results than MMIE. The task was a connected-digit recognition task (ISOLET). Because it is not the same as the task being examined here, and because the performance of these systems is so dependent on model complexity and amount of training data, the quantitative results may not be directly comparable with those reported. But for the most complex system he tested, MLE gave a 96.15% recognition rate, MMI gave 96.47%, and FD 97.05%. This represents a 24% relative reduction in error rate for FD as compared to ML, and a 8% relative reduction in error rate for MMI as compared to ML. The relative reduction in error rate for the 1 mixture system was similar at 22%. FD was also tested on an isolated letter task, the CONNEX E-set, ranging from 47% relative increase in accuracy for the 1 mixture diagonal system to 23% for the least complex.

Qualitative results for FD

Kapadia tested FD alongside MMI, for varying model complexity. For all tests, both of these discriminative algorithms outperformed ML, although sometimes the margin was small. For the simplest ISOLET model, MMI performed better than FD. For all other ISOLET models and all the CONNEX E-set, FD performed better than MMI, with the margin increasing as model complexity increased. As compared to ML, the gain in performance from FD (measured as a relative change in error) stayed roughly constant as model complexity increased in the ISOLET case, and decreased in the CONNEX case. The gain in performance from MMI as compared to ML decreased in both cases. This strong dependence of MMI on model complexity has also been reported by Valtchev [**?**]. These results show that FD is successful in its aim, which is to retain generalisability when faced with limited training data and complex models.

Another result reported in [**?**] was that for each level of model complexity, Kapadia was able to train a model which outperformed all other models, by first training with FD and then with MMI. [Actually, all models are seeded with ML training.] This is inconvenient because it shows that the test results are not only a function of the objective function used: if this were the case, the models trained first with FD and then with MMI should give identical results to the MMI-trained models.

FD performed worse than MMI on the training set in all cases. In fact, MMI always scored 100% accuracy on the training set, which shows how good it is at adapting to data and indicates that if there were sufficient training data, MMI might be the training criterion of choice. For the simplest system tested, the relative decrease in training set accuracy was similar to the relative decrease in test set accuracy, indicating that FD generalises almost perfectly in that case. However, for more complex systems the relative gain from FD on the training set increased while the test set gain stayed roughly the same.

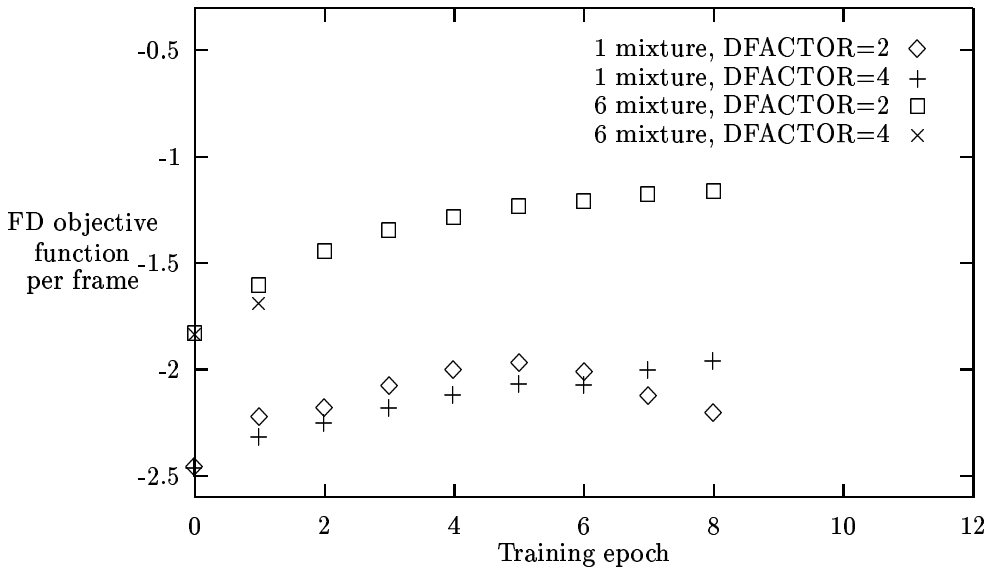


Figure 4.1: Plot of FD criterion against training epoch

Results for MMI

In Valtchev [**?**], a relative reduction in word error rate (WER) of 5-16% is reported. The magnitude of the reduction varies with the complexity of the model, with the most reduction reported for the model set with one mixture coomponent, decreasing to about 5% for the 12-mixture system.

Results for MCE

Bing-Hwang Juang *et al* [**?**] report a 25% error reduction for an isolated-letter recognition task, for MCE as compared to ML.

4.2 Results

System details

The results presented here are for tests on the Resource Management corpus, a 1,000 word task. State-clustered cross-word triphone HMMs were used, with mixtures of diagonal-covariance Gaussians. Results are reported for both 1 and 6 mixture systems. Feature vectors consisted of 39 elements, consisting of 12 mel-frequency cepstral coefficients and their first and second differentials. State clustering was used, decision trees being built for every state of every monophone HMM to create equivalence classes over sets of triphone HMM contexts. Then an iterative mixture splitting technique, as described by Young *et al.* [**?**], is performed, to find an optimal match between system complexity and available training data. Then a few iterations of MLE are performed, giving the baseline from which FD training is started.

4,000 short sentences of training data were used, and each test set consisted of 300 sentences. After doing the initial experiments on one test set, the most promising model sets were tested on another test set (the "validation test set"). For tests on the training data, a 300-sentence subset of the training data was used.

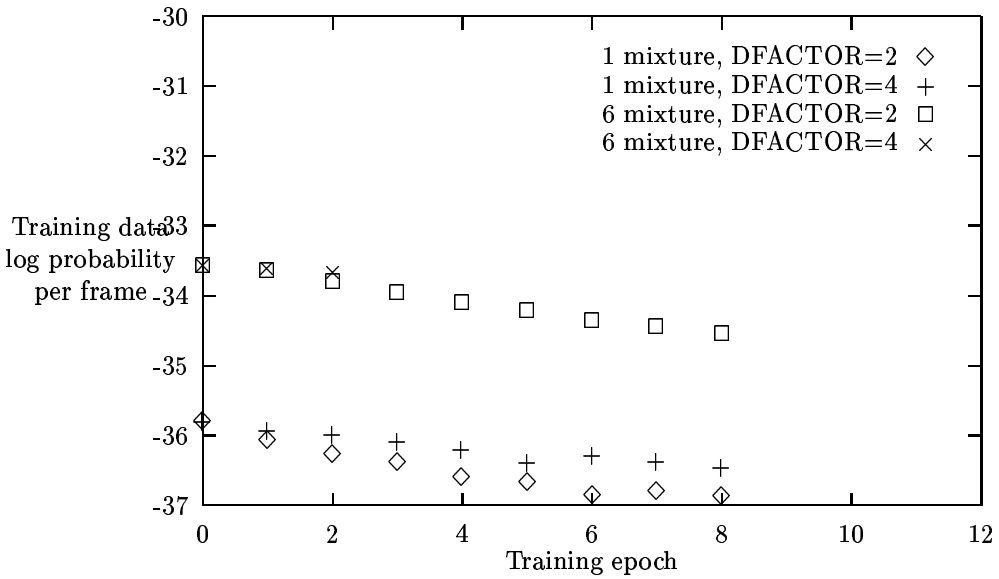


Figure 4.2: Plot of average log probability/frame of training data (ML criterion)

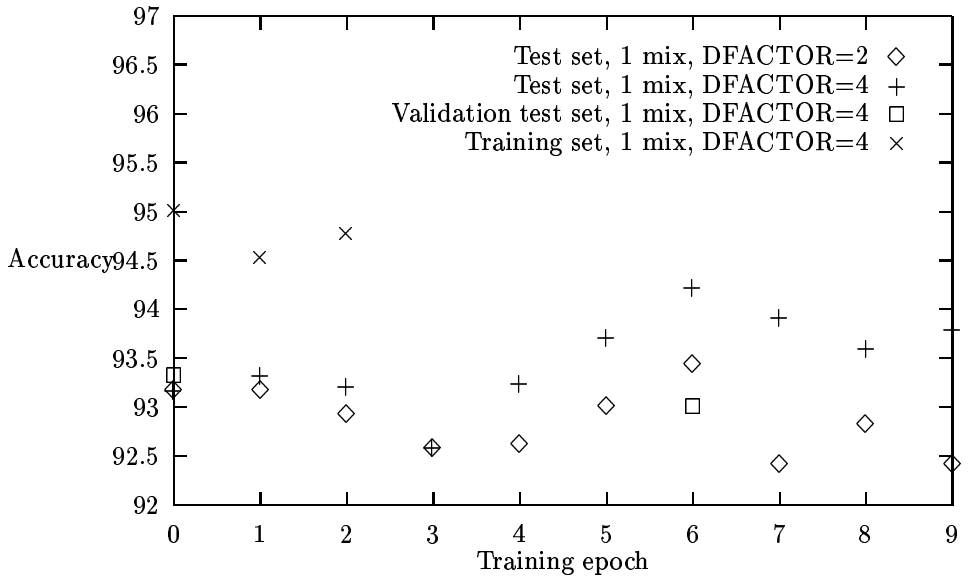


Figure 4.3: Plot of test set and training set word accuracy, for 1 mixture system

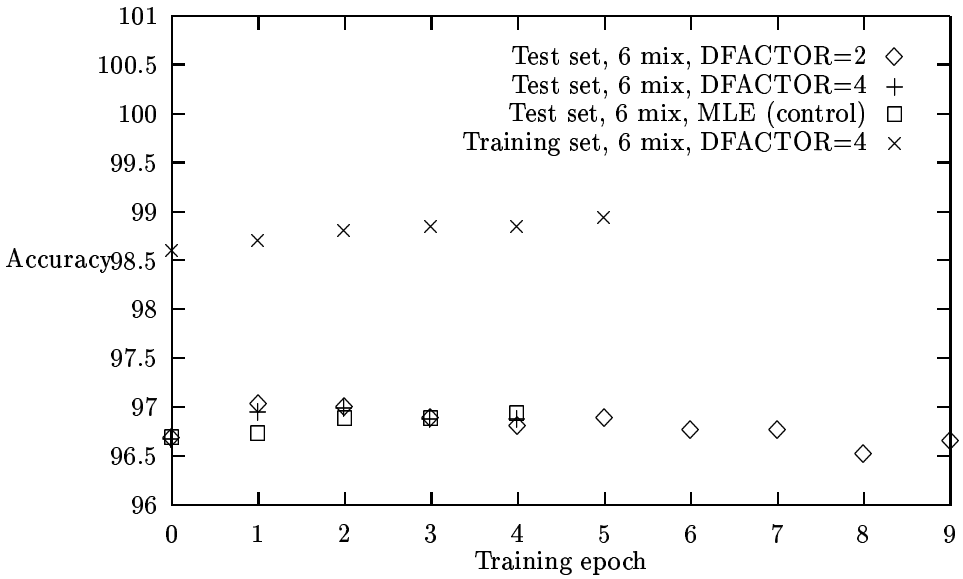


Figure 4.4: Plot of test set and training set word accuracy, for 6 mixture system

Plots

Figure **??** shows the FD criterion per frame increasing with iteration. It is interesting to note that the criterion does not increase indefinitely, but in the 1 mixture case where $DFACTOR = 4$, it starts to decrease again. It is difficult to say why this could be. The recognition results for the 1-mixture system with $DFACTOR = 4$ were poor. The reason why it is not so critical for the 6 mixture system (i.e, why that system seems more stable in terms of the FD criterion) may be that the constant D in the EBW equations is set to $DFACTOR$ times the highest value of D needed to make any individual variance positive. Since there are more mixtures per phone model in the 6 mixture system, the smoothing constant D will tend to be set higher in that case.

Figure **??** shows the average training set log probability per frame (which is the ML criterion) decreasing with iteration. Since the models had already been trained with ML, the ML criterion was presumably at an optimum and it was bound to decrease.

Figure **??** shows the accuracy for the 1 mixture system varying with training epoch. The reader will note that in the 1 mixture system, with $DFACTOR = 2$, FD training results in a decrease in recognition accuracy for almost all iterations. This shows that the FD criterion, which was increasing quite happily for most of those iterations, is not a good guide to test set accuracy. What may have happened is that insufficient smoothing caused the means of the Gaussians to jitter randomly about their optimum, introducing random variation into the recognized speech. The set with $DFACTOR = 2$ seems to be performing better on test set accuracy. However, testing the best point with a validation test set was not able to confirm the error rate decrease (it increased for the validation test set), and the training set accuracy seems to be decreasing, so it is not possible to say that FD has increased the accuracy at all. Certainly if there is no increase in training set accuracy one cannot hope for an increase in test set accuracy.

The greatest increase in accuracy for the 6 mixture system was 10%, for the first iteration of MLE with $DFACTOR = 2$. Testing with the training set, for the set of iterations with $DFACTOR = 2$, shows a steady increase in accuracy, to a maximum of 24.3%. This is promising, and consistent with the results of Kapadia, which show rather more training set gain than test set gain. It is difficult to say why the results were better for the 6 mixture system; normally one expects discriminative techniques to do better on smaller systems. The previous results of Kapadia [**?**] do not show any similar problem with 1-mixture systems.

The reader will note that a "MLE control" is included in the results. This was included in case the initial models had not been trained to their optimum using MLE, to verify that any increase in accuracy could not have been more simply obtained by extra MLE training. The maximum gain from the extra MLE was 8%, leaving only 2% advantage to MLE. A disadvantage for FD is that the best result was not on the last iteration and therefore the best had to be arbitrarily picked, which tends to overestimate the gain, whereas the best ML result was on the last iteration. However, further tests will have to be done to see whether FD performs better if training is started from a model set which has converged to the MLE criterion.

Overall, it was disappointing to note that FD did not always decrease the error rate. Further work with the optimisation technique (the Roadmap algorithm) parametrised differently needs to be done, to see whether this could be an artifact of the optimisation.

Possible reason for failure

The tasks Kapadia [**?**] used to test FD were artificial tasks, where the frequency in the test and training data of all the speech units (digits, letters) was probably about the same. This may have been a necessary condition for the success of FD, for the following reason: in FD the denominator occupancy for a state of a HMM takes no account of how many times that HMM appears in the training data, but only of the acoustic properties of its PDF. This means that those states that appear often in the training data will have their occupancies dominated by the numerator, and states that appear infrequently will have their occupancies dominated by the denominator. However, this effect may not have been seen in Kapadia's task, if all training set frequencies were approximately equal. Various options have been added to the re-estimation program to compensate for this; however, there has not yet been time to test these on the RM corpus.

4.3 Combination of output

Part of the original aim of this project was to test whether it was possible to combine the output of HMMs trained with a discriminative and a non-discriminative technique, to produce transcriptions of higher accuracy than either of the originals. There was no time for this. However, a rough evaluation is attempted here of the feasibility of such an approach. The results reported here are based on the idea of an "oracle"- when the two different model sets produce different transcriptions for a sentence, and one of them is correct, the oracle can tell us which it is.

Figure **??** shows how the oracle sentence accuracy changes with iteration in the 6 mixture system, based on combination with the original (MLE) 6-mix system. The baseline, at iteration 0, is the original combined with itself, which is identical to no combination. The "1 mix MLE" data point shows the results of combining it with the 1-mixture MLE system. This shows that complex discriminative techniques may not be necessary to achieve performance gains from combination: combining it with a very simple 1-mixture model is almost as good. The maximum gain in accuracy here is 24.5%, and there is a 16% gain in accuracy from combining with the 1-mixture model.

Figure **??** shows oracle sentence accuracy for the 1 mixture system. The maximum relative gain in accuracy here is 23%, quite similar to the gain in the 6 mixture system. Notice that the oracle accuracy is higher in this case where $DFACTOR = 2$, i.e, where there is less smoothing. This makes sense because the systems for which the re-estimation formulae are less smoothed will naturally be more different from the original (MLE) system than those with a high smoothing constant.

Validity of oracle as a measure of potential accuracy

The idea of an oracle which can choose the correct one out of two possible transcriptions is not unreasonable for the case where the speech system is a front-end to a human-computer interface. Presumably not all sentences would have meaning to the interface, and when the outputs of the two systems differed it would usually be possible to pick the correct one based on meaning, or lack

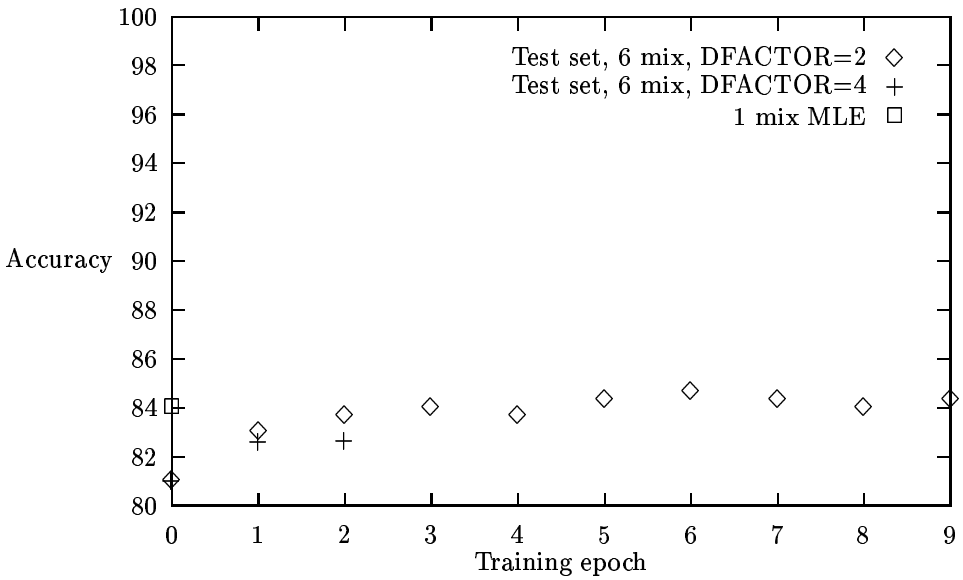


Figure 4.5: Oracle sentence accuracy, for these models combined with the 6-mix MLE models

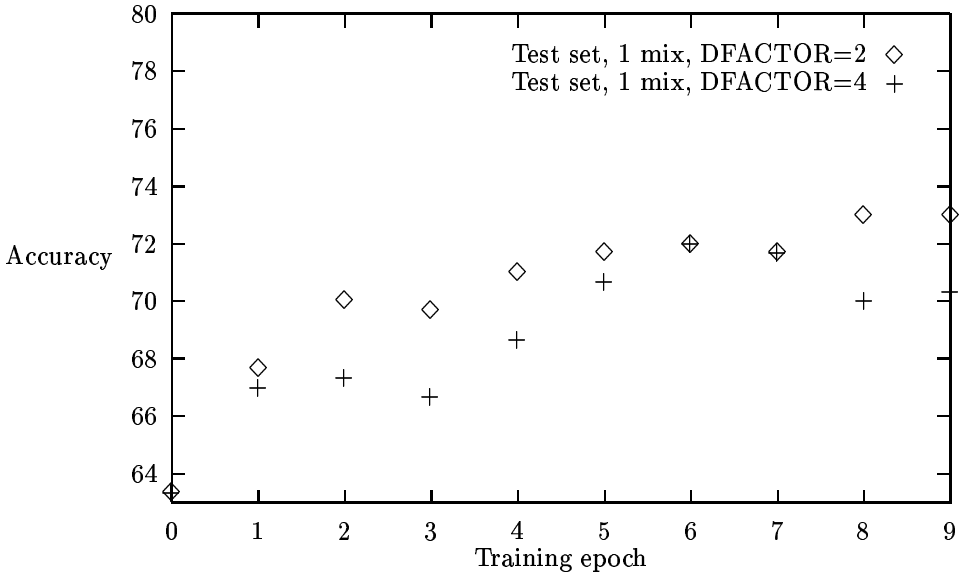


Figure 4.6: Oracle sentence accuracy, for these models combined with the 1-mix MLE models

of it. However, this is somewhat messy because it makes the interface between the two modules more complex, and the language system becomes dependent upon what it is being used for. The "oracle" performance does, however, give an upper bound on the possible performance of the system if some form of combination is used. The reason why error rates have been given on the sentence level and not the word level is the difficulty of designing and implementing a scheme to give the oracle error rate on the word level.

Previous work on combination

Fiscus [?] has developed a system, called ROVER (Recogniser output voting error reduction), to combine the outputs of various systems and hopefully get a lower error rate than any of the original systems. By combining the output of five of the submissions to the LVCSR 97 Hub 5E Benchmark Test, he was able to get a 12.5% relative reduction in word error rate as compared to the best of the original systems which were combined. Unfortunately, the scheme only works with at least three sets of outputs, for the same reason that voting makes little sense with only two people. But it is conceivable that a similar system could be made to work with only two systems by using a wide coverage grammar or another form of language model to break ties. Consider, for example, these errors which have been produced by one of the model sets tested here. In each case, the system output follows the correct transcription. These are a random sample of errors, and have not been selected according to any criterion.

```
SR159:EDIT THE BISMARK SEA AREA THREAT
SR159:EDIT THE BISMARK SEA AREA THREATS
ST0078:ARE ENGLAND AND FOX IN WELLINGTON
ST0078:ARE ENGLAND THAN FOX IN WELLINGTON
SR460:EDIT THE ALERT INVOLVING CITRUS
SR460:EDIT THE EIGHTH OF ALERTS INVOLVING CITRUS
ST1844:WHAT IS THE TOTAL NUMBER OF CRUISERS IN PACFLT
ST1844:WHAT IS THE TOTAL NUMBER OF CRUISERS BE IN PACFLT
SR143:WHEN WILL WHIPPLE CHOP FROM PACFLT TO LANTFLT
SR143:WHEN WILL WHIPPLE CHOP FROM PACFLT THE LANTFLT
ST1839:DO WE HAVE A SUB IN WESTPAC WITH TEST DEPTH LESS THAN ONE THOUSAND FEET
ST1839:DO WE HAVE A SUB IN WESTPAC WITH TEST DEPTH LESS THAN ONE THOUSAND FEET IN
ST1473:GET A LIST OF PAC AREA ALERTS
ST1473:GET A LIST THE PAC AREA ALERTS
ST0062:IS SCHENECTADY AS FAST AS SHASTA
ST0062:IS SCHENECTADY HAS SHASTA AS SHASTA
ST1143:MAKE THE LETTERS ONE SIZE LARGER
ST1143:MAKE LETTERS ONE SIZE LARGER
ST2220:WILL MONTICELLO CHOP TO ATLANTIC FLEET BY SIX HUNDRED HOURS ZULU TUESDAY
ST2220:WILL MONTICELLO CHOPPED TO ATLANTIC FLEET BY SIX HUNDRED HOURS ZULU TUESDAY
```

The question is: can these errors be distinguished from the correct transcriptions based on language information? If so, the given "oracle" performance on a sentence level can be acheived. The files for which, in my estimation, the system output would be rejected by a wide-coverage parser are ST0078, SR1844, SR143, ST1473, ST0062 and ST2220; whereas the outputs for SR159, SR460, ST1839 and ST1143 would not be rejected. This is based on purely grammatical constraints: i.e, the author's guess at what the result would be of applying a fairly wide-coverage grammar to these examples. Notice that this is not the same as the language model used in the recogniser, because such language models are normally n-gram, and would probably be insufficient for these purposes. Based on this small sample, in which 60% of the errors could be detected and the rest of the ties would have to be broken randomly, we could expect an improvement of $23\% * 0.6 = 14\%$ relative decrease in sentence error rates. This is based on both the 1 mixture and 6 mixture systems, in which the relative decrease in sentence error resulting from sentence level combination using an

oraclee was about 23% in both cases. However, it might be possible to get even higher decreases in error rate by combining the outputs of the two systems using lattices (as in [?]) to produce either a list of possibilities or a lattice representing them, and testing the results using the wide-coverage parser. Something to bear in mind, however, is that if the errors generated by the system make too much sense, the system becomes less useful because its errors are more difficult to detect by humans.

4.4 Conclusions and further work

This paper has described an implementation of Frame Discrimination, a discriminative HMM estimation technique, for a fairly large vocabulary continuous speech recognition system (1,000 words). An algorithm has been developed which makes it possible to efficiently apply FD to large tasks. The performance of the system has been evaluated.

In the 1-mixture system an increase in training set error rate was registered, which indicates that FD may actually be worse than ML in this case. For the 6-mixture system the training set error rate seemed to decrease steadily by 24%, with the greatest relative decrease in test set accuracy being 10%. However, an 8% decrease was obtained by more iterations of ML, so further tests have to be done to see whether FD does better starting from initial models which have converged to a ML optimum. For comparison, Valtchev [?] reports a 5%-16% relative WER decrease for MMI on a larger task. Initial results seem to show that FD does not match these increases, although further tests will have to be done to see whether this is due to the optimisation used.

The possibility of combining the outputs of systems trained with the ML and FD criteria was mentioned, and a rough upper bound on the relative decrease in sentence error was given at 24.5%. It was conjectured that a 16% decrease in sentence error could be obtained using a wide-coverage parser to decide between alternatives. But it was pointed out that it may not be necessary to use FD for this, since comparable gains were shown to be possible by combining systems with different numbers of mixture components.

Bibliography

[1] Sadik Kapadia, *Discriminative Training of Hidden Markov Models*, Ph.D. thesis, Engineering Dept., Cambridge University, 1998

[2] V. Valtchev, J. J. Odell, P. C. Woodland and S. J. Young, *MMIE training of large vocabulary speech recognition systems*, Speech Communication 22(1997) pp 303-314.

[3] A. P. Dempster, N. M. Laird and D. B. Rubi, *Maximum Likelihood from Incomplete Data via the EM Algorithm* Journal of the Royal Statistical Society vol 39 pp 1-88, 1977

[4] Bing-Hwang Juang, Wu Chou and Chin-Hui Lee, *Minimum Classification Error Rate Methods for Speech Recognition*, IEEE Transactions on Speech and Audio Processing Vol. 5 No. 3, May 1997

[5] M. J. F. Gales, K. M. Knill and S. J. Young, *State-based Gaussian Selection in Large Vocabulary Continuous Speech Recognition using HMMs*, Cambridge University Engineering Dept.

[6] Y. Normandin, *Hidden Markov Models, maximum mutual information estimation, and the speech recognition problem.* Ph.D. Thesis, 1993, Dept. of Electrical Engineering, McGill Univeristy, Montreal.

[7] S. Kapadia, *MMI training for continuous parameter recognition of the TIMIT database.* Proc. Internat. Conf. Acoust. Speech Signal Processing. Minneapolis, pp 491-494.

[8] S. J. Young, J. J. Odell, P. C. Wodland, 1994. *Tree-based state tying for high accuracy acoustic modelling.* Proc. Human Language Technology Workshop. Morgan Kaufmann, Plainsboro, NJ, pp. 307-312

[9] B. Merialdo, 1988. *Phonetic recognition using hidden Markov models and maximum mutual information training.* Proc. Internat. Conv. Acoust. Speech Signal Processing, New York. Vol. 1, pp. 111-114

[10] The CRC concise enyclopedia of mathematics, `http://www.astro.virginia.edu/~eww6n/math/`

[11] J. G. Fiscus, *Recogniser output voting error reduction*

[12] P. S. Gopalakrishnan, D. Kanevsky, A. Nadas, D. Nahamoo, *An Inequality for Rational Functions with Applications to Some Statistical Estimation Problems*, IEEE Trans. on Information Theory V.37, No. 1, pp 107-113, Jan 1991

Appendix A

Proof of Baum-Welch re-estimation

It is demonstrated that the BW formulae will result in an increase in the likelihood function, or leave it unchanged. This is a slightly different proof from that normally employed to validate the Baum-Welch equations, which may be found in Dempster [?], and one which I find more intuitive. First the general problem of maximising a sum will be addressed:

$$F(A_1, ..., A_I) = \sum_i \hat{A}_i \tag{A.1}$$

starting with initial values $A_i > 0$. It will be demonstrated that if we increase the value of the weighted product:

$$\prod_i \hat{A}_i^{A_i}$$

then the value of the sum $\sum_i \hat{A}_i$ will be increased. If the value of the product is increased, then:

$$\prod_i \hat{A}_i^{A_i} > \prod_i A_i^{A_i} \tag{A.2}$$

$$\sum_i A_i \log(\hat{A}_i) > \sum_i A_i \log(A_i) \tag{A.3}$$

$$\sum_i A_i \log\left(\frac{\hat{A}_i}{A_i}\right) > 0 \tag{A.4}$$

We want to show that this implies an increase in the initial sum, or:

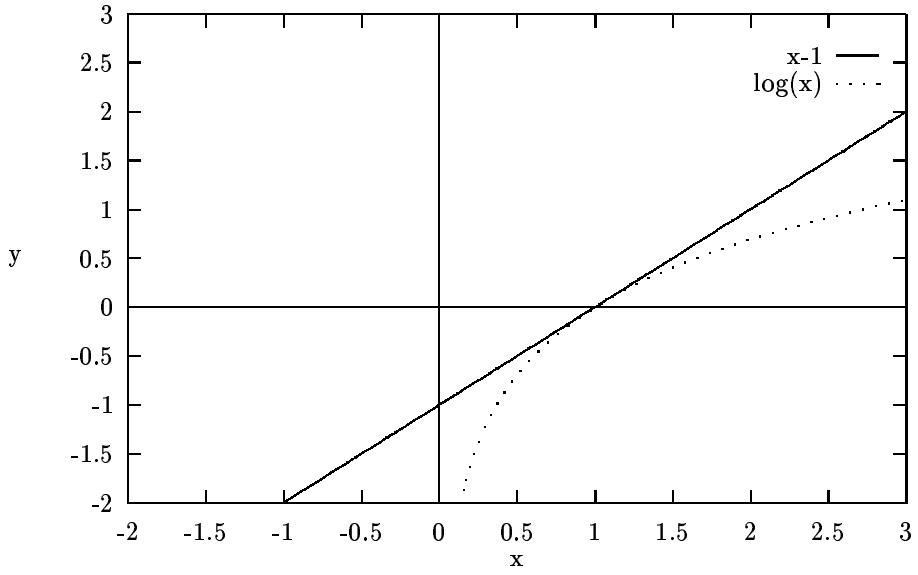
$$\left\{\sum_i \hat{A}_i - A_i\right\} > 0 \tag{A.5}$$

To show this, it is sufficient to show that:

$$\sum_i \hat{A}_i - A_i \geq \sum_i A_i \log\left(\frac{\hat{A}_i}{A_i}\right) \tag{A.6}$$

Rearranging:

$$\sum_i A_i \left(\frac{\hat{A}_i}{A_i} - 1\right) \geq \sum_i A_i \log\left(\frac{\hat{A}_i}{A_i}\right) \tag{A.7}$$



But $x - 1 \geq \log(x)$. (See Figure **??**) So, assuming of course that the A_i are positive, the proof holds, and increasing the value of the product will result in an increase in the sum.

In the application of this to the BW formulae, the A_i correspond to probabilities of the output given different transcriptions. $P(O|M)$, which we are trying to maximise, corresponds to a sum over all transcriptions.

$$F_{\mathrm{ML}}(M|O) = P(O|M) = \sum_i P(O|X^{(i)}, M) \tag{A.8}$$

Applying the above result, we choose instead to maximise the weighted product:

$$\prod_i P(O|X^{(i)}, \hat{M})^{P(O|X^{(i)}, M)} \tag{A.9}$$

which is equivalent to maximising:

$$F'(M) =_{\mathrm{def}} \prod_i P(O|X^{(i)}, \hat{M})^{\frac{P(O|X^{(i)}, M)}{P(O|M)}} \tag{A.10}$$

or alternatively:

$$\log F'(M) = \sum_i \frac{P(O|X^{(i)}, M)}{P(O|M)} \log(P(O|X^{(i)}, \hat{M})) \tag{A.11}$$

If we increase the value of $F'(M)$, we are guaranteed to increase the value of $F(M)$. The Baum-Welch equations maximise the value of the product $F'(M)$.

Transition probability updates

Demonstrating this first for the transition probabilities: we are considering the probabilities a_{ij} for a fixed i, noting the constraint that $\sum_j a_{ij} = 1$. The following equation results from taking all

terms in $\hat{a}_{ij}$ out of $F'(M)$ and treating the rest as a constant K.

$$\log F'(M) = K + \sum_j \log \hat{a}_{ij} \sum_{t=1}^{T-1} \sum_n \frac{P(O|X^{(n)})}{P(O)} \delta(x_t^{(n)} = i)\delta(x_{t+1}^{(n)} = j) \tag{A.12}$$

The values of a_{ij} which maximise value this sum, subject to the constraint that $\sum_j a_{ij} = 1$, are found by the method of Lagrangian multipliers.

Lagrangian multipliers

This explanation of Lagrangian multipliers has been filled out from the brief explanation in [**?**]

This technique is used to find an extremum in $f(x_1, x_2, ..., x_n)$ subject to the constraint that $g(x_1, x_2, ..., x_n) = C$. Restated, we want to find values $x_1, x_2, ...x_n$ within the surface $g(x_1, x_2, ..., x_n) = C$ such that for any direction of motion $(dx_1, dx_2, ..., dx_n)$ within the given surface defined by $g(\cdot) = C$, there will be no change in $f(\cdot)$. Restating these two requirements:

$$g(x_1, x_2, ..., x_n) = C, \text{and} \tag{A.13}$$

$$\begin{aligned} \forall(dx_1, dx_2, ...dx_n), \frac{\delta g}{\delta x_1}dx_1 + \frac{\delta g}{\delta x_2}dx_2, ... = 0 \quad \rightarrow \\ \frac{\delta f}{\delta x_1}dx_1 + \frac{\delta f}{\delta x_2}dx_2, ... = 0 \end{aligned} \tag{A.14}$$

This second requirement means that the plane defined by $\frac{\delta g}{\delta x_1}dx_1 + \frac{\delta g}{\delta x_2}dx_2, ... = 0$ and the plane defined by $\frac{\delta f}{\delta x_1}dx_1 + \frac{\delta f}{\delta x_2}dx_2, ... = 0$ must be the same plane, and therefore that the vectors $(\frac{\delta g}{\delta x_1}, \frac{\delta g}{\delta x_2}dx_2, ...)$ and $(\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}dx_2, ...)$ which define those planes must be collinear. This collinearity may be stated formally as:

$$\exists\lambda, \quad \frac{\delta f}{\delta x_1} + \lambda\frac{\delta g}{\delta x_1} = 0 \ \wedge \ \frac{\delta f}{\delta x_2} + \lambda\frac{\delta g}{\delta x_2} = 0, \ ... \tag{A.15}$$

The term λ is known as the *Lagrangian multiplier.*

Using Lagrangian multipliers to find the optimal transition probabilities

Equation **??** may be stated more simply as:

$$\log F'(M) = K + \sum_j k_j \log a_{ij} \tag{A.16}$$

Finding an extremum in $F'(M)$ subject to the constraint that $\sum_j a_{ij} = 1$ may be restated as:

$$\sum_j \hat{a}_{ij} = 1, \text{and} \tag{A.17}$$

$$\exists\lambda\forall j \ \frac{k_j}{\hat{a}_{ij}} + \lambda = 0 \tag{A.18}$$

The solution may be found at:

$$\hat{a}_{ij} = \frac{k_j}{\sum_j k_j} \tag{A.19}$$

$$\lambda = \sum_j k_{ij} \tag{A.20}$$

This results in the Baum-Welch re-estimation equations for a_{ij} as stated in Equation **??** namely:

$$\hat{a_{ij}} = \frac{\sum_{t=1}^{T-1} \sum_{n} \frac{P(O|X^{(n)})}{P(O)} \delta(\mathbf{x_t^{(n)}} = \mathbf{i}) \delta(\mathbf{x_{t+1}^{(n)}} = \mathbf{j})}{\sum_{k} \sum_{t=1}^{T-1} \sum_{n} \frac{P(O|X^{(n)})}{P(O)} \delta(\mathbf{x_t^{(n)}} = \mathbf{i}) \delta(\mathbf{x_{t+1}^{(n)}} = \mathbf{k})} \tag{A.21}$$

Updates of mixture components are analogous to transition probability updates (in the sense that there is a sum to one constraint), and will not be presented separately.

Gaussian updates

To repeat equation **??**: the function $F'(M)$, which we are trying to maximise, reads as follows:

$$logF'(M) = \sum_{i} \frac{P(O|X^{(i)}, M)}{P(O|M)} \log(P(O|X^{(i)}, \hat{M})) \tag{A.22}$$

In order to derive the formulae for the Gaussian updates, it will be assumed that the transcriptions $X^{(i)}$ specify the mixture component used as well. This convention has not been chosen in proving the transition update equations because it makes it more complex, and does not affect the validity of the proof. For the present purposes of proving the Gaussian updates, each $x_t^{(i)}$ is an ordered pair (i, m), specifying the state and mixture components.

Taking the terms in $b_{jm}(\mathbf{o}_t)$, we have:

$$\log F'(M) = K + \sum_{i} \frac{P(O|X^{(i)}, M)}{P(O|M)} \sum_{t=1}^{T} + \log(b_{jm}(\mathbf{o}_t)) \delta(x_t^{(i)} = (j, m)) \tag{A.23}$$

This may be restated in terms of alignments as follows:

$$\begin{aligned} \log F'(M) &= K + \sum_{t=1}^{T} L_{jm}(t) \log(b_{jm}(\mathbf{o}_t)) \\ &= K + \sum_{t=1}^{T} L_{jm}(t) \log\left(\frac{1}{\sqrt{(2\pi)^n |\Sigma_{jm}|}}\right) - \frac{1}{2}(\mathbf{o}_t - \mu_{jm})^T \Sigma_{jm}^{-1} (\mathbf{o}_t - \mu_{jm}) \end{aligned} \tag{A.24}$$

where n is the dimension of the vectors. The update equations will now be proven for one dimension; this can trivially be extended to n dimensions if the covariance matrix is diagonal (as is the case in this system.) Stated as simply as possible, we are maximising:

$$\sum_{t} k_t \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2}\frac{(\mu - o_t)^2}{\sigma^2} \tag{A.25}$$

Differentiating by μ, the minimum is at:

$$\frac{\delta F}{\delta \hat{\mu}} = \sum_{t} k_t \cdot -\frac{(\hat{\mu} - o_t)}{\sigma^2} = 0 \tag{A.26}$$

$$\hat{\mu} = \frac{\sum_t k_t o_t}{\sum_t k_t} \tag{A.27}$$

Differentiating by σ, the minimum is at:

$$\sum_{t} k_t \left\{ -\frac{1}{\sigma} + \frac{(\hat{\mu} - o_t)^2}{\sigma^3} \right\} = 0 \tag{A.28}$$

$$\sum_{t} k_t \sigma^2 = \sum_{t} k_t (\hat{mu} - o_t)^2 \tag{A.29}$$

$$\sigma^2 = \frac{\sum_t k_t (\hat{mu} - o_t)^2}{\sum_t k_t} \tag{A.30}$$

Replacing the k_t with $L_{jm}(t)$, what we have is the Baum-Welch equations for one dimension.

Appendix B

Intermediate results

Speeding up Vector Quantization

It was clear from the start that a very substantial improvement was required, and if VQ was to be used a very large number of VQ regions would have been needed. Unfortunately, the standard clustering algorithm proved to slow for this. The standard VQ clustering technique is the Linde-Buzo-Gray algorithm [?]. It is designed to minimise the average distortion per Gaussian:

$$\delta_{avg} = \frac{1}{M} \sum_{m=1}^{M} \left\{ \min \phi = 1^{\Phi} \delta(\mu_m, \mathbf{c}_\phi) \right\} \tag{B.1}$$

The distance measure δ used is between two points, or means. It is a weighted Euclidean, based on the average covariances of Gaussians in the system.

$$\delta(\mu_i, \mu_j) = \sum_{k=1}^{K} \frac{1}{\sigma^2(k)} \{ w(k)(\mu_i(k) - \mu_j(k)) \} \tag{B.2}$$

The algorithm, briefly, goes as as follows:

1. Start out with just one cluster, at the centre of all the Gaussians, with all Gaussians assigned to it.
2. While there are less than Φ clusters:
   3. Split the largest cluster, i.e, the cluster with the largest distortion, i.e, largest $\sum_{m \in \phi} \delta(\mu_m, \mathbf{c}_\phi)$, by producing two cluster centres a small distance away from $\mathbf{c}_\phi$.
   4. Iteratively assign the Gaussians to clusters, and find the clusters centres (as the average of the Gaussians assigned to them), until there is no more change.

The problem is that assigning Gaussians to clusters takes $O(M)$ time, and is performed $O(\Phi)$ times, which can become a problem. To solve this, various modifications were made:

1. In step 3, splitting not the largest cluster but a number of the largest clusters.
2. Although it is not shown in the algorithm above, it is sometimes necessary to split clusters to replace clusters which have too few Gaussians assigned to them; in this case also, the small clusters are split all at once rather than one by one.
3. In step 4, we do not wait for the cluster centres to converge unless it is the last of the outer iterations (i.e, the number of cluster centres was complete.)

These modifications did not significantly increase the average cost per Gaussian: see Figure **??**.

Even with this improvement, the very large number of clusters required could not be acheived in a short space of time. Also, as the number of clusters increased, the time taken to work out

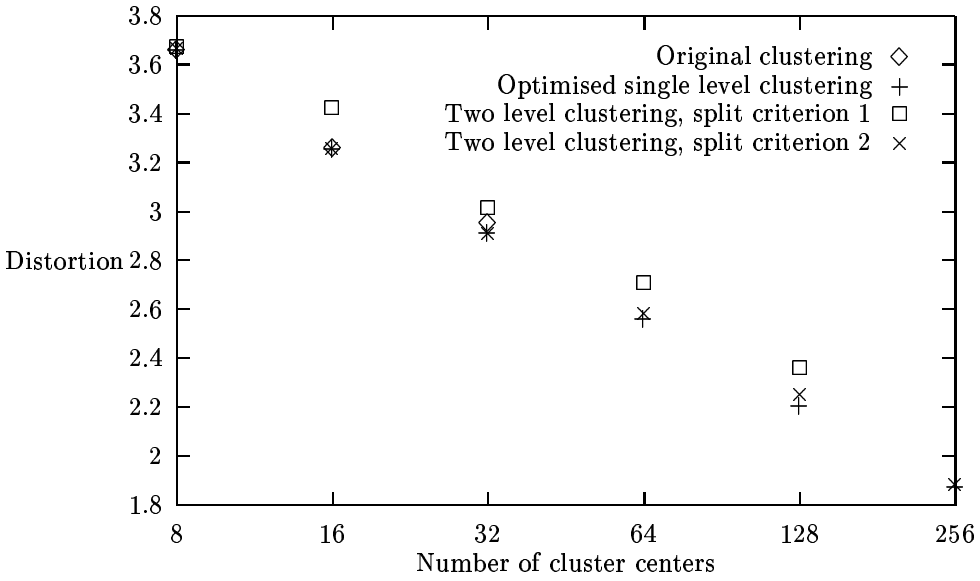


Figure B.1: Average distortion per Gaussian decreasing with cluster size: 1000 mix system

the best cluster centre at each time frame became important. To solve this, a two-level clustering algorithm was developed, in which first the Gaussians were split into a small number of clusters, and then those clusters were clustered. Rather than there being a fixed number of sub-clusters for each top level cluster, the number of sub-clusters was fixed globally, and the number of sub-clusters for each top level cluster was allowed to vary in order to minimise the total divergence.

In the clustering algorithm, two variations on the cluster splitting scheme were tried. In scheme 1, the clusters with the largest average deviation were split; in scheme 2, the clusters with the largest summed deviation were split. For the single-level clustering, it made very little difference and accordingly only scheme 1 is plotted; however, for the two-level case scheme 2 was markedly better, so both are shown. For further experiments, scheme 2 was used.

Figure **??** shows that clustering on two levels does decrease somewhat the average distortion per cluster. However, it has the advantages of decreasing the computation time, both while initialising the VQ codebook and while working out the VQ index for each data frame.

Time taken to cluster

The reason for some of these optimisations is to decrease the time taken to cluster. Accordingly, the time taken to perform clustering is presented. At 32 clusters, the optimised single level clustering took 36% as long as normal clustering, while the two-level clustering (also optimised) took 30% as long. The un-optimised clustering in this case took one minute. The speedup was much greater for larger numbers of clusters.

Results of VQ

Two-level clustering, with 256 clusters at the bottom level, decreased the average log probability of the output by 0.3 when calculating 4% of the Gaussians. Increasing the number of clusters to 1024 without changing the percentage of Gaussians changed resulted in an average log probability decrease of 0.26. It seems that the gain in performance, measured this way, from increasing the number of clusters, is very slight. But it was of the same order as the decrease in average distance: $\frac{3.486}{2.82} \simeq \frac{0.3}{0.26}$. This may show that average distance per cluster centre is a good guide to the decrease in log probability when the clustering is used in practice.

Interpretation of results: failure

The results above are not directly comparable with the baseline results reported earlier, because there was more probability loss with less VQs tested, so one cannot say which was better or worse. However, the 256 cluster set tested can itself be used as a baseline because of the similarity with single-level VQ. Two level clustering has been shown (by the average distance measure) to be about the same as single level clustering, and was only a means to being able to increase the number of clusters. The fact that increasing the number of clusters to 1024 did not result in much of an improvement in performance– at least not the kind of improvement I needed– means that two-level clustering is not a great success, because it is a means to very large numbers of clusters, and very large numbers of clusters have been found not to be very much more effective than smaller numbers. Note that the number of Gaussians in this system was 9000, so 1024 VQ centres is quite a substantial proportion of the total, and yet it is still not much better than 256. I felt that the reason might be that the variances held useful information as well.

Tuning VQ

Various tunings to the basic VQ scheme were tried. The behaviour initially was to assign a fixed number of VQ centers to each Gaussian. It was noticed that Gaussians with a higher value of $|\Sigma|$ (I.e, a higher variance in most dimensions) were more often missed out, so to compensate for this more VQ indices were assigned to those Gaussians with a higher variance. This resulted in a minor improvement.

Pruning using most significant element of input vector

A quite substantial improvement in speed was afforded by the following scheme: for each VQ index parameters were stored which enabled one to calculate the significance of each particular input vector in distinguishing between them. The significance of and input value $\mathbf{o}(i) = x$ was defined as the variance of the logs of the output values of the Gaussian in that dimension. This is a 4th-order polynomial in $\mathbf{o}(i)$. At each time frame, these were calculated using stored coefficients, and the 5 most significant elements of the 39-element vector were used to select the most promising Gaussians before calculating the rest of them. Half of Gaussians assigned to the VQ index were calculated in full. This was not perfect pruning: the most significant Gaussian was sometimes missed. I noticed in these cases that the Gaussian missed was often one of those which was closest to the vq centre. Therefore I added to the output probability for each Gaussian based on the most significant elements of the input a number indicative of how close to VQ center that Gaussian was. This improved results. Eventually I was able to prune away two thirds of the Gaussians using the 5 most significant elements of the input, and half of the remaining ones using the next 5 most significant elements, without losing any of the most significant Gaussians that I tested. This means that the technique, when tuned correctly, can be very successful.

This technique did not make its way into the final system, and the description here of it is sketchy partly for that reason. It was not included in the final system because of the extra coding effort that would have been required, and because it is really orthogonal to the nature any vector quantization or related technique that one may use, and therefore there was nothing to be learned from transferring it.

This is a technique which is applicable to speech recognition tasks, and therefore may deserve attention.

Gaussian clustering

After finding that standard VQ, as the number of clusters increased, did not give the performance required, it was decided to try clustering Gaussians not only by their means, but by their variances as well. The distance measure used was the divergence, whose functional form in one dimension is:

$$\frac{1}{2}\left\{\frac{\sigma_1^2}{\sigma_2^2}+\frac{\sigma_2^2}{\sigma_1^2}-2+\left(\frac{1}{\sigma_1^2}\frac{1}{\sigma_2^2}\right)(m_1-m_2)^2\right\} \tag{B.3}$$

This was used to cluster the vectors in exactly the same way as the means (as described above), the "average" of a number of Gaussians being the Gaussian parametrisation that minimised the summed divergence between it and the others. This was found numerically. Notice that the property of transitivity ($\delta(a,c) = \delta(a,b) + \delta(a,c)$), which the divergence does not possess, is not required in clustering. There was no reason why it could not be used in the clustering algorithm.

The clustering algorithm was extended from two levels to many levels, the objective being to cluster the Gaussians in something roughly like a tree. Clustering of the Gaussians in the system proceeded from the top down, each level being used to help with the clustering at the level below, in the most time-consuming part of the clustering algorithm which is the assignment of Gaussians to cluster centers. In finding the best cluster center at the currently lowest level to which to assign a particular Gaussian, the higher levels of the tree are used for pruning. To describe this in detail would unnecessarily lenghten this thesis, especially considering that this technique did not make it to the final system.

The use of it was as follows: At each level of the tree, starting from the top, a fixed number of the most promising Gaussians were selected based on their parents on the next level up, and how well they did. Then value of the input vector given all of these most promising Gaussians was calculated, and the top few kept in order to prune on the next level down. As I have said, there is no need to describe this in detail. I will give some typical results: 0.02 decrease in average log likelihood, with 10% of Gaussians per time being calculated. This figure includes the Gaussians in the clustering tree, which in this technique make quite a significant proportion of the effort.

Similarity-based technique

It was decided that too much effort was being expended calculating the value of the input for Gaussians which were not part of the system. A technique was developed whereby each Gaussian in the model set was annotated with a list of the n most similar other Gaussians in the model set. Then, an algorithm roughly as follows was exectuted:

1. Start off with one Gaussian, picked using a scaled-down version of the previous algorithm
2. While less than the specified Gaussians have been tested:
3. Test the closest Gaussian to the nearest Gaussian to the input vector out of those Gaussians that have not yet had all their close Gaussians tested.

This algorithm worked quite well, but I do not report the results because it is superseded by the algorithm I eventually used. After tuning this algorithm in various ways and exploring its behavour with a debugger, I came to the conclusion that the problem with it was the distance measure. There was one particular Gaussian that kept being missed out, while the closest match to the input was often another particular mix in the same state, which I assume was quite close to the missed-out one. However, the divergence measure placed them extremely far apart. Upon examining the Gaussians' parameters, I discovered that in one particular dimension they had very different variances, differing by perhaps a factor of 100. This seemed to be the reason for the very high divergence between them.

Examining Figure ?? shows why the divergence is a bad measure for this purpose. Intuitively one would expect Gaussians a and c to be closer together than a and b in the sense of being more likely to give a high probability for many of the same input vectors. However, the divergence between a and c, at about 50, is higher than the divergence between a and b, which is about 25. This is why a new divergence measure was developed.

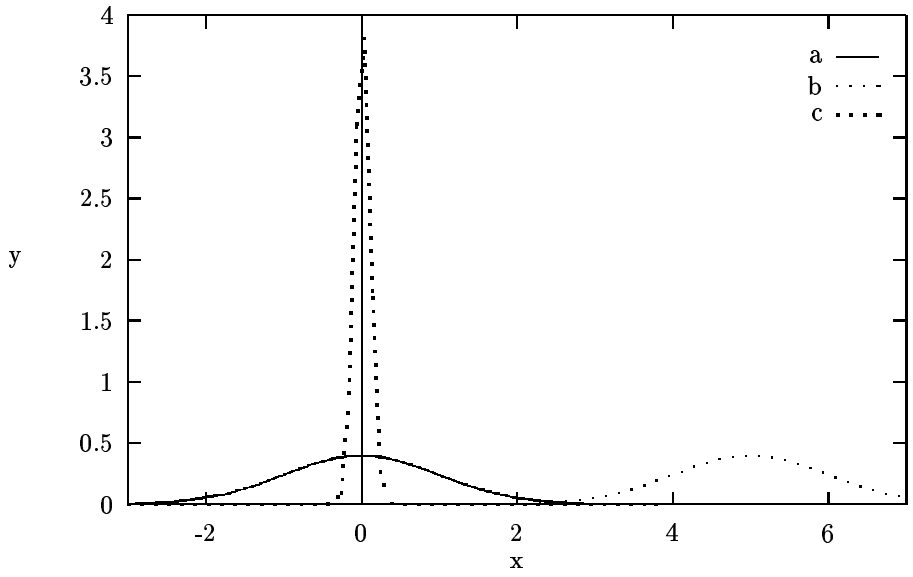


Figure B.2: Divergence $\delta(a,c) \simeq 50,\ \delta(a,b) \simeq 25$