

EFFICIENT MLLR

Matthew Gibson
Gonville & Caius College
University Of Cambridge

Supervisor: Professor Phil Woodland

MPhil. Computer Speech, Text & Internet Technology

July 2004

Declaration

I Matthew Gibson of Gonville and Caius College, being a candidate for M.Phil in Computer Speech, Text and Internet Technology, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Where reference is made to the works of others, the extent to which the work has been used is indicated and duly acknowledged in the text and bibliography. This thesis has not been submitted in whole or in part for a degree at any other institution. This thesis does not exceed 15000 words in length, including footnotes and bibliography.

Signature

Matthew Gibson, *July 20, 2004*

Acknowledgements

I would like to thank my supervisor, Professor Phil Woodland, for his guidance and ideas throughout this project. Thanks also go to Andrew Naish-Guzman for clarity of thought with regard to general machine learning problems and for dealing patiently with many tedious L^AT_EX-related queries.

Abstract

The need for close to real time speech recognition has recently driven interest in fast LVCSR systems. Due to the time constraint, such systems often discard, where possible, sub-processes of the entire recognition process which demand relatively large amounts of computation and yield relatively small accuracy gains.

This report focusses on such speed-accuracy tradeoffs with regard to speaker adaptation. A variety of techniques are used to reduce the compute time of a baseline adaptation system which uses mean-only maximum likelihood linear regression (MLLR). Use of a mixture component-level Viterbi alignment to accumulate adaptation statistics and least squares linear regression transform estimation are compared to the baseline techniques in terms of speed and accuracy.

In the case of unsupervised adaptation, exploitation of word boundary information generated in an initial recognition pass is shown to further reduce adaptation time with no negative impact upon system accuracy. Again, in the unsupervised case, an illustration of how confidence scores can be used to simultaneously reduce adaptation time and improve accuracy is presented.

Table of Contents

Table of Contents	1
1 Introduction	3
2 Fast LVCSR Systems	5
2.1 Broadcast News Transcription	5
2.2 Conversational Telephone Speech	6
2.3 2003 Fast CU-HTK System	8
3 Speaker Adaptation Theory	11
3.1 Linear Transformations	11
3.2 Regression Class Trees	12
3.3 MLLR	13
3.4 LSLR	13
3.5 Embedded Baum-Welch Algorithm	14
4 Baseline System	17
4.1 Baseline System Description	17
4.2 Baseline Adaptation Subsystem	17
4.3 Baseline Evaluation	19
4.4 Prune Setting Impact	20
5 Efficient Adaptation	23
5.1 Viterbi Adaptation	23
5.1.1 Evaluation	24
5.2 Transform Estimation	25
5.2.1 MLLR Computational Cost	26
5.2.2 LSLR Computational Cost	28
5.2.3 Evaluation	31
5.2.4 Matrix Inversion Computation	32
5.3 Efficient Alignment	33
5.3.1 Constrained Alignment	33
5.3.2 Evaluation	34
5.3.3 Further Constrained Alignment	35
5.3.4 Evaluation	36
5.3.5 Beam Width And Alignment	37

	1
5.3.6 Evaluation	38
5.4 Confidence Scores And Adaptation	39
5.4.1 Evaluation	39
6 Conclusions and Further Work	43
6.1 Further Work	43
Bibliography	45

Chapter 1

Introduction

Large vocabulary continuous speech recognition (LVCSR) systems are often evaluated according to their accuracy with little consideration for recognition time. For some tasks recognition time is not a key issue. However, in the case of, for example, practical dialogue systems, recognition time close to realtime is mandatory.

Recently, evaluations such as those conducted by the National Institute of Standards and Technology (NIST) have encouraged researchers to focus on speed of recognition as well as accuracy. The Rich Transcription evaluations¹ evaluate systems separately according to recognition time. The following categories are used.

- Less than or equal to realtime.
- Less than or equal to ten times realtime ($10\times RT$).
- Unconstrained by time (unlimited compute).

This report describes an investigation into the efficiency of speaker adaptation techniques. Significant reductions in overall recognition time are shown to be possible with some compromise in recogniser accuracy.

Chapter 2 reviews recent work on fast ($10\times RT$) LVCSR systems, while Chapter 3 introduces speaker adaptation theory. The experimental baseline system is described and evaluated in Chapter 4 and Chapter 5 presents several efficiency refinements to baseline adaptation techniques. Conclusions and suggestions for further work are found in Chapter 6.

¹More information on NIST evaluations is available at <http://www.nist.gov/speech/tests/index.htm>.

Chapter 2

Fast LVCSR Systems

In this chapter recent work at Cambridge University Engineering Department (CUED) on fast ($10\times RT$) LVCSR systems is summarised.

The fast LVCSR systems developed at CUED are generally reduced and modified versions of the full (unlimited compute) systems. The reduced systems save computation whilst retaining as much accuracy as possible by omitting those parts of the recognition process which provide low accuracy gains at relatively high computational cost.

An interesting technique used in the 2003 CU-HTK system is to identify subsystems of the full system which complement each other well i.e. yield low word error rate (WER) when combined. The more complementary subsystems are retained for use in the fast system.

2.1 Broadcast News Transcription

The task of broadcast news transcription in less than $10\times RT$ was embraced in 1998 by CUED. Details of the system and its evaluation are found in [5]. The paper describes how the runtime of the recognition component of the CUED spoken document retrieval system ([6]) was reduced from approximately $300\times RT$ to $9.5\times RT$. The final recognition system executes the following stages, in the order given.

1. Segmentation
2. First pass decoding
3. Gender assignment
4. Segment clustering and MLLR transform estimation
5. Second pass decoding

The segmentation process is used to divide the input into categorised segments. Three categories are used: wideband speech, narrowband speech and music. The music segments are discarded from further processing.

First pass decoding uses two sets of pretrained HMM models, one for narrowband speech and one for broadband. Each model set uses tied-state crossword triphone models, with state output distributions comprising 16 Gaussian components. A trigram language model (LM) is incorporated into the search, which generates a lattice. This lattice is then rescored using a four-gram LM to produce a word transcription for each segment.

Gender assignment and second pass decoding use a different set of gender-dependent bandwidth-specific HMMs. These HMMs are used to determine the speaker gender by rescoreing the first pass transcription using both the male and female models. The most likely model determines the gender.

After the gender assignment, segments of the same gender and bandwidth are clustered using a procedure described in [7]. Maximum likelihood linear regression (MLLR) is used to estimate a transform for each cluster within each gender-dependent bandwidth-specific HMM set. The first pass transcription is used as the supervision for the adaptation.

The second decoding pass uses the adapted HMM sets to produce a lattice corresponding to each speech segment. A language model consisting of an interpolated four-gram and category trigram is then applied to the lattice to find the most likely transcription of each segment. The second pass decoding parameters are chosen in order to fulfill the $10\times RT$ requirement.

The processing time for each stage is given in Table 2.1. The **BNeval198** dataset is used.

Stage	Runtime ($\times RT$)
Segmentation	1.09
First pass decode	2.09
Gender assignment, segment clustering and transform generation	0.44
Second pass decode	5.86
Total	9.48

Table 2.1: Run Time: Broadcast News Transcription 98

When evaluated on the **BNeval198** dataset, the system yielded a WER of 16.1%. This is 2.3% absolute (16.6% relative) higher than the unlimited compute system described in [8].

2.2 Conversational Telephone Speech

In 2002, the CU-HTK system participated in the NIST evaluation for conversational telephone speech (CTS) transcription, otherwise known as the Hub-5 evaluation. Details of both the full (unlimited compute) and the $10\times RT$ system are found in [9]. Figure 2.1 illustrates the architecture of the $10\times RT$ CTS system.

The first decoding pass is used solely to enhance the second pass via estimation of CMN and CVN vectors and a VTLN warp factor. Maximum-likelihood (ML) estimated triphone models and a trigram LM are used at this stage.

The first pass output is used to resegment the manually segmented data. An original segment is split if more than 1 second of silence is detected. The new segmentation is used only for the purposes of estimation. Later passes use the original segmentation.

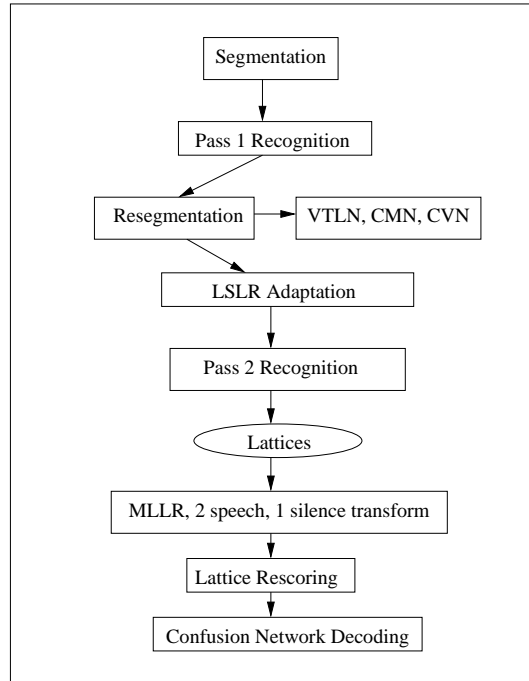


Figure 2.1: 2002 CU-HTK CTS System

The new segmentation is used in the estimation of CMN and CVN vectors and a VTLN warp factor. CMN and CVN reduce the time-invariant effects of an acoustic channel such as a telephone channel. ‘Side-based CMN’ assumes the channel effects are constant for the duration of an entire conversation, and is used since it has been shown to out-perform segment-based CMN ([15]). Side-based CMN, unlike segment-based CMN, does not suffer from unreliable estimations arising from short data segments. Similar arguments support the use of side-based CVN.

VTLN is a form of speaker normalisation detailed in [11]. The optimal frequency warp factor is found by using the first pass transcription and maximising the likelihood of the data over a range of warp factors.

The first pass transcription is also used to estimate a set of adaptation transforms using least squares linear regression (LSLR) discussed in Section 3.4). The second-pass models are adapted according to these LSLR-estimated transforms.

The second decoding pass uses the estimated VTLN warp factor to recode the audio data, and normalises the resulting feature vector using the estimated CMN and CVN vectors.

The models used for the second pass are triphones trained using the discriminative minimum phone error (MPE) criterion. An interpolation of the trigram LM derived from the acoustic training set and the corresponding LM based on the broadcast news (BN) corpus is used within the search.

The second pass decoding produces a lattice which is further rescored using an interpolated four-gram LM. This four-gram LM is an interpolation of a four-gram LM derived from the acoustic

training set, a four-gram LM based on the BN corpus, and a class-based trigram LM employing automatically identified classes.

The best transcription after the second decoding pass is used to estimate a transform set via MLLR. Two transforms are estimated for speech models and one for silence models.

The adapted model set is then used to acoustically rescore the lattice produced in the second pass.

CN decoding (detailed in [10]) involves transformation of the lattice into a sequence of confusion sets, and has been shown to successfully reduce WER. A confusion set contains competing word hypotheses and their corresponding posterior probabilities.

The run time for each of the above stages is detailed in Table 2.2. The `eva102` dataset is used, and the experiment is executed using an AMD Athlon XP 1900+ system.

Stage	Runtime (\times RT)
First Pass Decode + Resegmentation + VTLN	1.65
LSLR + Second Pass Decode	5.36
MLLR + Lattice Rescoring + CN	2.24
Total	9.25

Table 2.2: Run Time: Conversational Telephone Speech Transcription 02

The $10\times$ RT system yielded a WER of 27.2% on the `eva102` dataset, while the full system attained 23.9%, a relative gap of 13.8%. The full system ran in approximately $320\times$ RT.

2.3 2003 Fast CU-HTK System

For the 2003 NIST evaluation, CUED developed a $10\times$ RT system for both conversational telephone speech and broadcast news transcription. Both systems are detailed in [1]. Figure 2.2 shows the structure of the $10\times$ RT CTS system.

The first recognition pass, resegmentation, and normalisation stage (CMN, CVN and VTLN) are as described in Section 2.2. The only difference is that the first pass acoustic models are trained using the MPE criterion.

The second recognition pass uses MPE-trained triphones and the interpolated trigram LM described in Section 2.2. Before the second pass, MLLR adaptation is used to update the model parameters. One global transformation is estimated for all speech models.

The output of the second pass is a lattice which is then submitted to two separate models for acoustic rescoring. These models are carefully chosen from the following four model sets to optimise the WER after system combination.

- Speaker adaptive trained¹ models (SAT), using heteroscedastic linear discriminant analysis² (HLDA) for feature vector extraction

¹[16] describes speaker adaptive training and the benefits of its use.

²A discussion of HLDA is provided in [9].

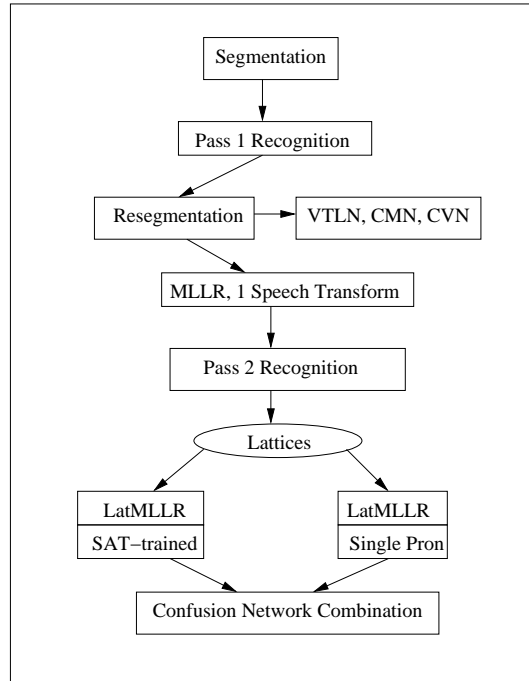


Figure 2.2: 2003 CU-HTK CTS System

- HLDA only
- Single pronunciation³ models, using HLDA
- Non-HLDA

All of the above models are trained using the MPE criterion and adapted using the output of the second pass and lattice MLLR ([12]). Multiple full mean transforms, multiple diagonal variance transforms and a single full variance transform are used in the adaptation.

Using confusion network combination, presented in [13], the SAT and the single pronunciation model set combination was found to yield the lowest WER from all the pairwise combinations of the above systems (see [1] for results). So these two systems are chosen for the the final stage of the $10\times$ RT system.

Table 2.3 shows the runtime breakdown of each stage using the *eval03* dataset. IBM x335 machines with 2.8GHz Intel Xeon CPUs, 512KB cache and 400MHz bus are used.

Using the *eval02* dataset, the $10\times$ RT system produced a WER of 23.3%, compared to the full ($190\times$ RT) system's 21.7%. The relative gap between the CTS full system WER and the fast system was reduced from 13.8% in 2002 to 6.9% in 2003.

³The advantages of a single pronunciation dictionary are presented in [9].

Stage	Runtime (\times RT)
Coding + Segmentation	0.068
First Pass Decode	0.890
VTLN	0.387
Second Pass Decode	3.178
Lattice MLLR (SAT)	1.422
Lattice Rescore (SAT) + CN	1.074
Lattice MLLR (single pronunciation)	1.200
Lattice Rescore (single pronunciation) + CN	0.919
System combination + Alignment	0.068
Total	9.207

Table 2.3: Run Time: Conversational Telephone Speech Transcription 03

Examination of Table 2.3 reveals that a significant portion of the overall runtime is spent performing adaptation. It is therefore certainly worthwhile investigating more efficient speaker adaptation techniques.

Chapter 3

Speaker Adaptation Theory

Speaker dependent (SD) speech recognition systems recognise the speaker(s) used to train the system. Speaker independent (SI) systems are designed to recognise any speaker. Given the same amount of training data, a SD system typically provides a WER two to three times lower than that yielded by a SI system.

Speaker-adapted SI systems use a relatively small amount of speaker data, the *adaptation data*, to reduce WER via an adaptation scheme. Adaptation can occur in either *static* mode or *dynamic* mode. In the case of static (or block) mode adaptation, the speaker presents some utterances to the unadapted system. The system collects these utterances, and when a sufficient amount of data is available, uses this to convert the system to a speaker-adapted system. The adapted system is then used for further processing of the speaker.

Using dynamic (or incremental) adaptation, the system repeats the adaptation process described above, continuously re-adapting the system as the user provides more utterances.

In addition to these different operational modes, adaptation may be a supervised or unsupervised process. In the unsupervised case, a transcription of the adaptation data must be provided. When using unsupervised adaptation the transcription of the user utterance is unknown. In this case the recogniser can be used to provide a transcription.

The choice of adaptation mode depends upon the application. For example, registration of a new user with a telephone banking dialogue system could use static, supervised adaptation. Contrastingly, realtime transcription of live sports commentary may use unsupervised, incremental adaptation.

3.1 Linear Transformations

The speaker adaptation methods used in this report are based on linear transformations of the model parameters of continuous density hidden Markov Models (CDHMMs) with multivariate Gaussian mixture state output distributions.

For the case of the mean of a Gaussian mixture component, this transformation is of the form

$$\hat{\boldsymbol{\mu}}_{jm} = \mathbf{A}^{(s)} \boldsymbol{\mu}_{jm} + \mathbf{b}^{(s)} \quad (3.1)$$

where $\hat{\boldsymbol{\mu}}_{jm}$ is the adapted mean, $\boldsymbol{\mu}_{jm}$ is the unadapted mean, $\mathbf{A}^{(s)}$ is an $n \times n$ matrix and $\mathbf{b}^{(s)}$ is an n -dimensional vector called the bias. Here n is the feature vector dimensionality, and the s superscript denotes a particular speaker.

Equation 3.1 can be reformulated as follows,

$$\hat{\boldsymbol{\mu}}_{jm} = \mathbf{W}^{(s)} \boldsymbol{\xi}_{jm} \quad (3.2)$$

where $\mathbf{W}^{(s)}$ is an $n \times (n + 1)$ matrix and $\boldsymbol{\xi}_{jm}$ is the extended mean vector of dimension $n + 1$.

The transformation matrix $\mathbf{W}^{(s)}$ may be estimated using either maximum likelihood linear regression (MLLR) or least squares linear regression (LSLR). To reduce the number of parameters to be estimated, the transformation matrix may be assumed to be of diagonal or block diagonal form. This is helpful in cases of limited adaptation data.

3.2 Regression Class Trees

To help overcome problems posed by limited adaptation data for the transform estimation, the Gaussian mixture components in the system may be grouped into non-intersecting sets $M^{(r)}$, and a transform $\mathbf{W}^{(sr)}$ applied to all components belonging to set $M^{(r)}$.

In the experiments which follow, components are grouped using a binary regression class tree, described in [3]. A regression class tree hierarchically clusters components which are acoustically similar into *regression classes*. Figure 3.1 depicts a binary regression class tree. *Base classes* (shown as squares in Figure 3.1) correspond to terminal nodes in the tree, and each component belongs to one and only one base class. During adaptation, a *node occupancy* is assigned to each node in

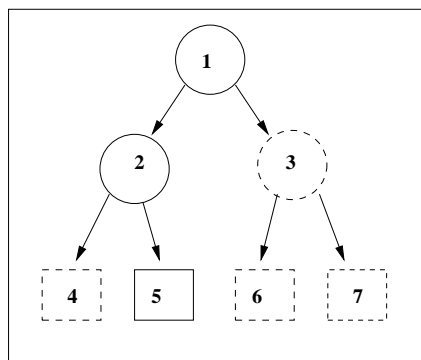


Figure 3.1: Binary Regression Class Tree

the regression tree, corresponding to the sum of the occupancy of the components contained within the node. Transforms are generated at the lowest node in the tree whose occupancy exceeds a certain threshold. In Figure 3.1, nodes with dotted borders represent those nodes with insufficient occupancy. So components belonging to base class 4 are transformed according to the transform generated at node 2, and components belonging to base classes 6 and 7 are transformed according to the transform generated at the root node 1.

Using regression trees, the generated transforms are sensitive to the amount and type of adaptation data.

3.3 MLLR

MLLR is presented in [2]. Using MLLR, $\mathbf{W}^{(sr)}$ is estimated to maximise the likelihood of the adaptation data. For mean adaptation, $\mathbf{W}^{(sr)}$ is estimated according to Equation 3.3.

$$\mathbf{W}^{(sr)} = \arg \max_{\mathbf{W}} \left\{ \sum_{(j,m) \in M^{(r)}} \sum_{t=1}^T L_{jm}(t) \log(\mathcal{N}(\mathbf{o}(t); \mathbf{W}\boldsymbol{\xi}_{jm}, \boldsymbol{\Sigma}_{jm})) \right\} \quad (3.3)$$

where $\mathbf{o}(t)$ is the observation at time t , $L_{jm}(t)$ is the occupancy of Gaussian mixture component m in state j , and $\mathcal{N}(\mathbf{o}(t); \mathbf{W}\boldsymbol{\xi}_{jm}, \boldsymbol{\Sigma}_{jm})$ is the probability of observation $\mathbf{o}(t)$ as determined by the Gaussian distribution with mean $\mathbf{W}\boldsymbol{\xi}_{jm}$ and covariance $\boldsymbol{\Sigma}_{jm}$.

The expectation maximisation technique can be used to estimate $\mathbf{W}^{(sr)}$. It is shown in [2] that, assuming that all covariance matrices in the HMM system are diagonal, a solution to Equation 3.3 is given by

$$\mathbf{w}'_i = [\mathbf{G}^{(i)}]^{-1} \mathbf{z}'_i \quad (3.4)$$

where \mathbf{w}_i is the i th row of $\mathbf{W}^{(sr)}$, \mathbf{z}_i is the i th row of \mathbf{Z} and the matrices \mathbf{Z} and $\mathbf{G}^{(i)}$ are defined in Equations 3.5 and 3.6.

$$\mathbf{Z} = \sum_{(j,m) \in M^{(r)}} \sum_{t=1}^T L_{jm}(t) \boldsymbol{\Sigma}_{jm}^{-1} \mathbf{o}(t) \boldsymbol{\xi}'_{jm} \quad (3.5)$$

$$g_{pq}^{(i)} = \sum_{(j,m) \in M^{(r)}} v_{ii}^{(jm)} d_{pq}^{(jm)} \quad (3.6)$$

In Equation 3.6, $g_{pq}^{(i)}$, $v_{ii}^{(jm)}$ and $d_{pq}^{(jm)}$ are the elements of the matrices $\mathbf{G}^{(i)}$, \mathbf{V}_{jm} and \mathbf{D}_{jm} respectively, where \mathbf{V}_{jm} and \mathbf{D}_{jm} are defined by Equations 3.7 and 3.8.

$$\mathbf{V}_{jm} = \left(\sum_{t=1}^T L_{jm}(t) \right) \boldsymbol{\Sigma}_{jm}^{-1} \quad (3.7)$$

$$\mathbf{D}_{jm} = \boldsymbol{\xi}_{jm} \boldsymbol{\xi}'_{jm} \quad (3.8)$$

3.4 LSLR

The LSLR training criterion, again, in the case of mean adaptation, is specified by Equation 3.9.

$$\mathbf{W}^{(sr)} = \arg \min_{\mathbf{W}} \left\{ \sum_{(j,m) \in M^{(r)}} \sum_{t=1}^T L_{jm}(t) (\mathbf{o}(t) - \mathbf{W}\boldsymbol{\xi}_{jm})' (\mathbf{o}(t) - \mathbf{W}\boldsymbol{\xi}_{jm}) \right\} \quad (3.9)$$

which has the solution

$$\mathbf{W}^{(sr)} = \left(\sum_{(j,m) \in M^{(r)}} \sum_{t=1}^T L_{jm}(t) \mathbf{o}(t) \xi'_{jm} \right) \left(\sum_{(j,m) \in M^{(r)}} \sum_{t=1}^T L_{jm}(t) \xi_{jm} \xi'_{jm} \right)^{-1} \quad (3.10)$$

It can be shown that LSLR is a special case of MLLR, where LSLR makes the additional assumption that all mixture components in the same regression class share the same covariance matrix.

3.5 Embedded Baum-Welch Algorithm

For both MLLR and LSLR estimation, the mixture component occupancies must be calculated. An extension of the Baum-Welch algorithm known as the *embedded Baum-Welch algorithm* is used for this purpose.

The algorithm uses as input a sequence of phone HMM models and corresponding utterance data. All word-boundary information is ignored. This is reasonable in the case of supervised adaptation, as often such detailed transcriptions are unavailable. As explained later, in Chapter 5, for the case of unsupervised adaptation, such word boundary information is often available, and its use can lead to more efficient adaptation.

The model sequence is used to construct a composite HMM via concatenation of the phone HMMs, and the forward-backward algorithm is executed on the composite HMM.

The forward-backward algorithm, in the case of a *single HMM model* is used to calculate mixture occupancies given a sequence of observations. It involves computation of a set of forward probabilities, $\alpha_j(t)$ and a set backward probabilities $\beta_j(t)$ defined in Equations 3.11 and 3.12.

For a sequence of T observations o_1, o_2, \dots, o_T , and corresponding sequence of states $x(1), x(2), \dots, x(T)$, for a time instance t , state j and a particular model set λ ,

$$\alpha_j(t) = p(\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t, x(t) = j | \lambda) \quad (3.11)$$

$$\beta_j(t) = p(\mathbf{o}_{t+1}, \mathbf{o}_{t+2}, \dots, \mathbf{o}_T | x(t) = j, \lambda) \quad (3.12)$$

The above probabilities can be computed recursively using Equations 3.13 and 3.15.

For $t = 1, 2, \dots, T$ and for $j = 2, 3, \dots, N - 1$,

$$\alpha_j(t) = b_j(\mathbf{o}_t) \left[\sum_{k=1}^{N-1} \alpha_k(t-1) a_{kj} \right] \quad (3.13)$$

and the forward probabilities are initialised according to Equation 3.14.

$$\alpha_j(0) = \begin{cases} 1 & \text{if } j = 1 \\ 0 & \text{Otherwise} \end{cases} \quad (3.14)$$

For $t = T - 1, T - 2, \dots, 1$, and for $j = 1, 2, \dots, N - 1$,

$$\beta_j(t) = \sum_{k=2}^{N-1} a_{jk} b_k(\mathbf{o}_{t+1}) \beta_k(t+1) \quad (3.15)$$

and the backward probabilities are initialised according to Equation 3.16.

$$\beta_j(T) = a_{jN} \quad (3.16)$$

Here N is the number of states in the model, a_{jk} is the transition probability from state j to state k and b_k is the output probability density function of state k .

It can easily be shown that

$$p(\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T | \lambda) = \sum_{k=2}^{N-1} \alpha_k(T) a_{kN} \quad (3.17)$$

and that

$$L_j(t) = p(x(t) = j | \mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T, \lambda) = \frac{\alpha_j(t) \beta_j(t)}{p(\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T | \lambda)} \quad (3.18)$$

With some additional computation, the mixture component occupancies can be calculated from the state-level occupancies.

$$L_{jm}(t) = p(x(t) = j, \text{mixture} = m | \mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T, \lambda) = \frac{L_j(t) c_{jm} b_{jm}(\mathbf{o}_t)}{b_j(\mathbf{o}_t)} \quad (3.19)$$

where c_{jm} is the mixture weight of component m in state j and b_{jm} is the probability distribution of mixture component m in state j .

The embedded Baum-Welch algorithm uses a composite HMM and so requires modified versions of the above forward-backward equations. The following formulae are quoted in [3].

Let $1, 2, \dots, Q$ be the ordered indices of the HMM models comprising the composite HMM.

For each model q the forward probability is initialised according to Equations 3.20, 3.21 and 3.22.

$$\alpha_1^{(q)}(1) = \begin{cases} 1 & \text{if } q = 1 \\ \alpha_1^{(q-1)}(1) a_{1N_{q-1}}^{(q-1)} & \text{Otherwise} \end{cases} \quad (3.20)$$

$$\alpha_j^{(q)}(1) = a_{1j}^{(q)} b_j^{(q)}(\mathbf{o}_1) \quad (3.21)$$

$$\alpha_{N_q}^{(q)}(1) = \sum_{i=2}^{N_q-1} \alpha_i^{(q)}(1) a_{iN_q}^{(q)} \quad (3.22)$$

where N_q is the number of states in model q and j is the index of an emitting state.

For $t = 2, \dots, T$ the forward probabilities are calculated recursively using Equations 3.23, 3.24 and 3.25.

$$\alpha_1^{(q)}(t) = \begin{cases} 0 & \text{if } q = 1 \\ \alpha_{N_{q-1}}^{(q-1)}(t-1) + \alpha_1^{(q-1)}(t) a_{1N_{q-1}}^{(q-1)} & \text{otherwise} \end{cases} \quad (3.23)$$

$$\alpha_j^{(q)}(t) = \left[\alpha_1^{(q)}(t) a_{1j}^{(q)} + \sum_{i=2}^{N_q-1} \alpha_i^{(q)}(t-1) a_{ij}^{(q)} \right] b_j^{(q)}(\mathbf{o}_t) \quad (3.24)$$

$$\alpha_{N_q}^{(q)}(t) = \sum_{i=2}^{N_q-1} \alpha_i^{(q)}(t) a_{iN_q}^{(q)} \quad (3.25)$$

The backward probabilities are initialised according to equations 3.26, 3.27 and 3.28.

$$\beta_{N_q}^{(q)}(T) = \begin{cases} 1 & \text{if } q = Q \\ \beta_{N_{q+1}}^{(q+1)}(T) a_{1N_{q+1}}^{(q+1)} & \text{Otherwise} \end{cases} \quad (3.26)$$

$$\beta_i^{(q)}(T) = a_{iN_q}^{(q)} \beta_{N_q}^{(q)}(T) \quad (3.27)$$

$$\beta_1^{(q)}(T) = \sum_{j=2}^{N_q-1} a_{1j}^{(q)} b_j^{(q)}(\mathbf{o}_T) \beta_j^{(q)}(T) \quad (3.28)$$

where again j is the index of an emitting state.

For $t = T-1, T-2, \dots, 1$, the backward probability is computed recursively according to Equations 3.29, 3.30 and 3.31.

$$\beta_{N_q}^{(q)}(t) = \begin{cases} 0 & \text{if } q = Q \\ \beta_1^{(q+1)}(t+1) + \beta_{N_{q+1}}^{(q+1)}(t) a_{1N_{q+1}}^{(q+1)} & \text{Otherwise} \end{cases} \quad (3.29)$$

$$\beta_i^{(q)}(t) = a_{iN_q}^{(q)} \beta_{N_q}^{(q)}(t) + \sum_{j=2}^{N_q-1} a_{ij}^{(q)} b_j^{(q)}(\mathbf{o}_{t+1}) \beta_j^{(q)}(t+1) \quad (3.30)$$

$$\beta_1^{(q)}(t) = \sum_{j=2}^{N_q-1} a_{1j}^{(q)} b_j^{(q)}(\mathbf{o}_t) \beta_j^{(q)}(t) \quad (3.31)$$

Note that for exit and entry HMM states, the forward and backward probabilities at time t represent the forward and backward probabilities at time $t - \Delta t$ and $t + \Delta t$, respectively, where Δt is a small time displacement.

Chapter 4

Baseline System

This chapter describes and evaluates the baseline recognition and adaptation systems.

4.1 Baseline System Description

The baseline recognition system is composed of state-clustered cross-word triphone HMM models. State output probability distributions are six-component Gaussian mixture models. The models were trained using the ARPA Resource Management (RM) training data.

The state clustering process is implemented using a phonetic decision tree, where the clustering is constrained such that states in the same cluster must share the same base phone model and share the same position within the triphone HMM.

The observation vector is 39-dimensional, containing the initial 12 Mel-frequency cepstral coefficients plus the signal energy. The first and second time derivatives of these static parameters are also included.

A non-probabilistic grammar listing all observed word pairings is provided with the RM data and used to define a bigram language model. The language model likelihoods are based only on the number of possible succeeding words for each word.

4.2 Baseline Adaptation Subsystem

Figure 4.1 is a high-level illustration of the CU-HTK adaptation subsystem. This is used as the baseline adaptation system. Adaptation data and a corresponding word-level transcription serve as input to the adaptation process. The output is a set of speaker-specific transformation matrices.

The first stage in the process is alignment, which receives some adaptation data and a corresponding word-level transcription and outputs a sequence of phone-level HMM models. This is the model sequence which best corresponds to the adaptation data. The preferred word pronunciations are chosen at this stage. A beam width (see Section 5.3.5) of 200 is used for the alignment step.

The model sequence is then used as input to the forward-backward algorithm, as well as the adaptation data. The forward-backward algorithm is essentially used to calculate the mixture component

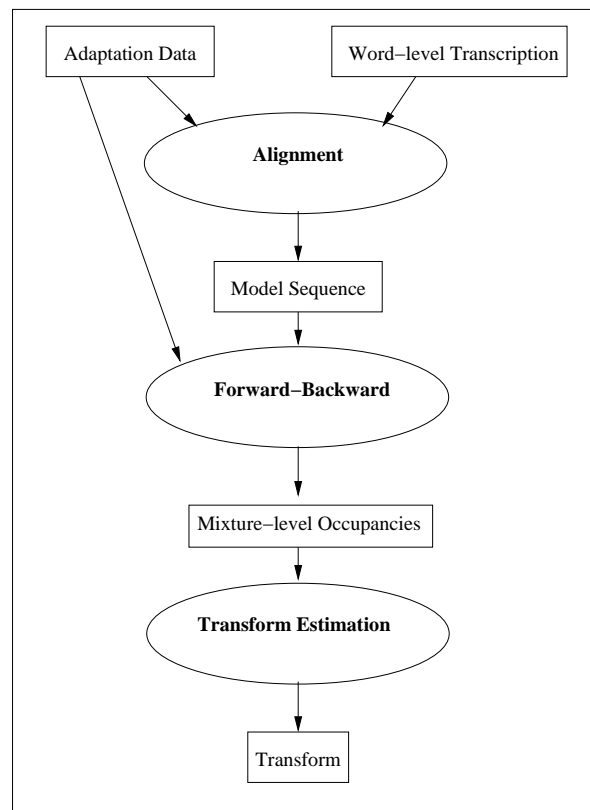


Figure 4.1: Baseline Adaptation System

occupancies ($L_{jm}(t)$) present in the linear regression formulae (Equations 3.3 and 3.9). Section 3.5 details the formulae used in the embedded Baum-Welch algorithm. Note that some useful running summations are also calculated during the forward-backward algorithm, namely $\sum_{t=1}^T L_{jm}(t)\mathbf{o}(t)$ and $\sum_{t=1}^T L_{jm}(t)$ for each mixture component.

The final stage, transform estimation, uses the mixture component occupancies to estimate the transformation matrix using the linear regression formula i.e. Equation 3.3 for the case of mean-only MLLR adaptation.

In the case of mean-only MLLR adaptation, it is assumed that the covariance matrices of all mixture components are diagonal, licensing the usage of the formulae quoted in Section 3.3. Matrix inversion is implemented using truncated singular value decomposition. This is necessary because the matrices $\mathbf{G}^{(i)}$ (see Equation 3.6) can be near-singular, so a stable technique is required to estimate the inverse matrices.

After transform estimation, the transformation is applied to the model set, and the adapted models used for future processing of the speaker.

In the case of both supervised and unsupervised adaptation, a word-level transcription of the adaptation data is used as input to the adaptation process. The only difference is the source of the transcription. Supervised adaptation uses an externally-provided transcription while in the unsupervised case, the transcription is generated by a previous recognition pass.

The static and dynamic adaptation procedures differ only in that, in the static case, the transformation is estimated after all the adaptation data has been presented to the system, while, in the dynamic case, the system is incrementally updated as explained in Chapter 3.

Static, unsupervised MLLR mean-only adaptation is used in the baseline system.

The transformation matrix is assumed to be of block diagonal form, with three blocks each of size 13×13 , corresponding to the static, delta and delta-delta cepstral parameters of the feature vector.

A simple regression tree comprising two base classes was generated to cluster the Gaussian mixtures in the system. One base class is used for silence and noise models, and the other for speech models.

4.3 Baseline Evaluation

Some experiments were conducted to highlight the most time-consuming portions of the adaptation process.

Dataset	Alignment	Forward-backward	Transform Estimation
feb89	14791	82270	19927
feb91	14308	82079	20016
oct89	15284	88455	17121
sep92	15011	83391	21157

Table 4.1: Timing Results: Alignment, Forward-Backward & Transform Estimation

Table 4.1 gives timing information (in milliseconds) for the various stages for each RM evaluation dataset¹.

It is clear from the results of Table 4.1 that the forward-backward algorithm is the most time-consuming part of the entire adaptation process, accounting for over 69% of the total adaptation time in all cases. It should be noted, however, that the time taken for the forward-backward algorithm depends upon the prune threshold setting, while the alignment time depends upon the beam width setting. Sections 4.4 and 5.3.6 respectively explain the prune threshold and beam width setting and examine the effect of varying these parameters.

4.4 Prune Setting Impact

During the backward pass² of the forward-backward algorithm, at a particular time t , if a backward probability falls more than the pruning threshold below the maximum backward probability at time t , then the backward probability at that time frame/state is zeroed and disregarded in further calculations. During the subsequent forward pass, forward probabilities at particular time frame/states are only calculated if there exist corresponding non-zero backward probabilities.

If no path is found through the composite HMM after the backward pass (i.e the probability of the utterance is zero), the utterance is discarded. This can be caused by a low prune threshold setting, referred to as *over-pruning*.

Figure 4.2 shows how the total time spent executing the forward-backward algorithm varies with the prune threshold setting. The utterances of the **feb91** dataset are used. Figure 4.3 shows how the accuracy of the adapted recogniser varies with the prune threshold setting for several datasets. It is clear from Figure 4.2 that a low prune threshold setting is desirable to reduce the compute time of the forward-backward algorithm. However, very low prune thresholds can lead to much of the adaptation data being discarded due to over-pruning.

Figure 4.3 reveals that over-pruning at low thresholds leads to unreliable recogniser accuracy. This, in turn, is due to poor transform estimation due to small quantities of data. Decreasing the prune threshold setting is thus a simple way to achieve faster adaptation at the risk of losing reliable transform estimation. A prune threshold of 2000 was used for the baseline system.

¹Four different RM evaluation datasets are used throughout this report. Each dataset comprises 10 different speakers and 30 utterances (of variable length) per speaker.

²In the HTK implementation of the forward-backward algorithm, the backward pass precedes the forward pass.

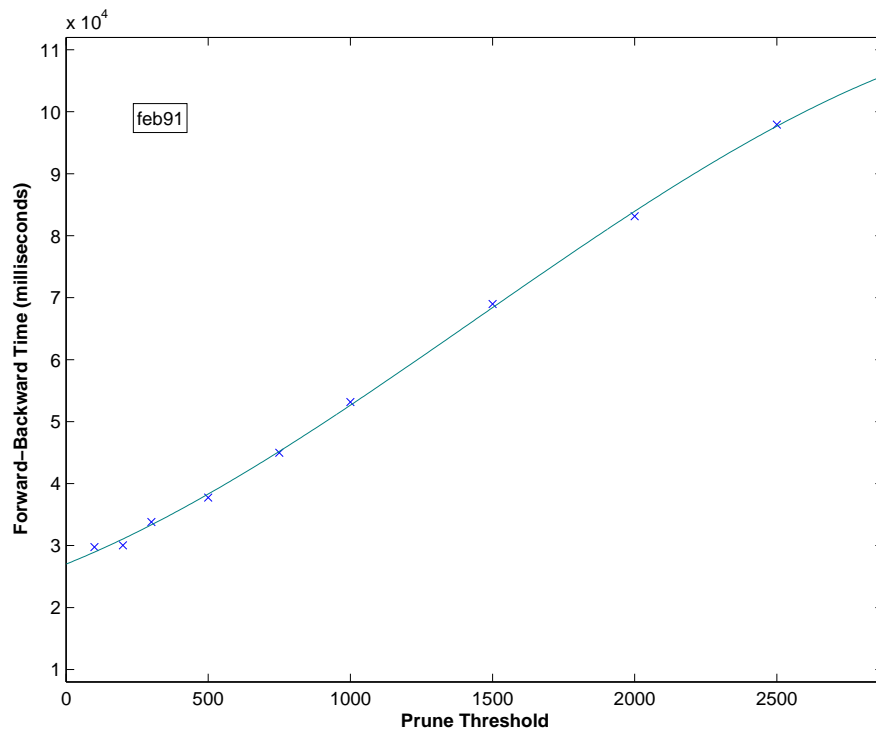


Figure 4.2: Prune Threshold Vs Forward-Backward Time

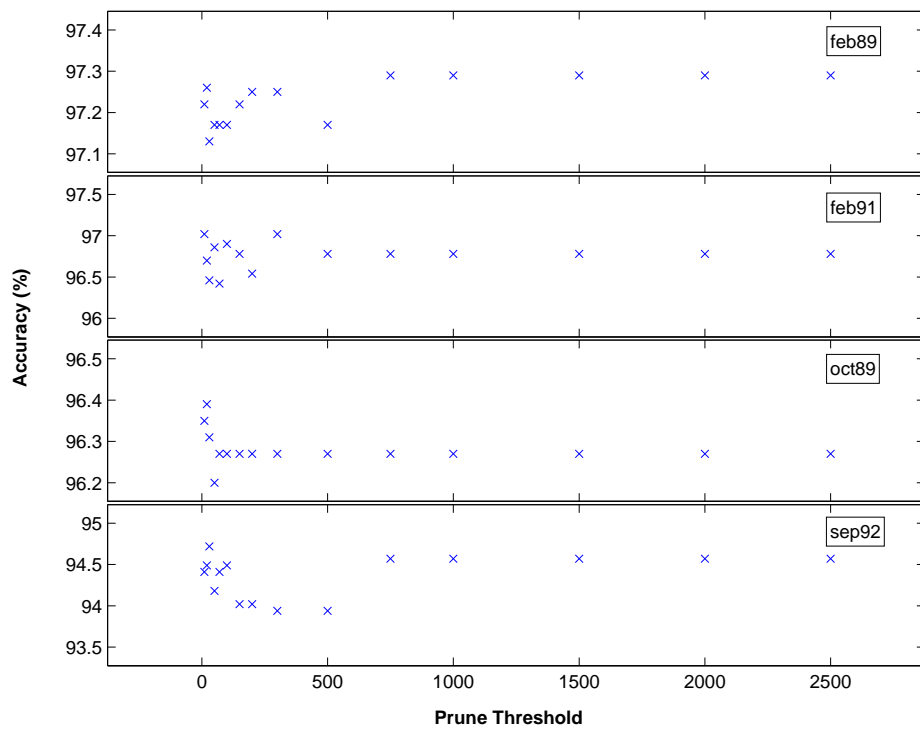


Figure 4.3: Prune Threshold Vs Accuracy

Chapter 5

Efficient Adaptation

This chapter introduces and evaluates some techniques designed to reduce the cost of all stages of the adaptation process.

Section 5.1 examines the effect of bypassing the forward-backward algorithm by using a Viterbi alignment while Section 5.2 investigates a technique to reduce the transform estimation computational cost. Both of these techniques apply equally well to unsupervised and supervised adaptation. The remainder of the chapter concerns only unsupervised adaptation. Section 5.3 demonstrates how the alignment process can be streamlined. Section 5.4 explains how confidence scores can be used to simultaneously reduce transform estimation computational cost and decrease WER.

5.1 Viterbi Adaptation

An alternative to Baum-Welch training is Viterbi training. Viterbi training uses Viterbi decoding (also referred to as Viterbi *alignment*) to align time frames with HMM states. The output of the decoding process is the most likely sequence of states - or *path* - through a composite HMM, given a sequence of observations. The time frame/state alignment is then used to estimate the HMM parameters.

Viterbi alignment may similarly be used for the purposes of adaptation. Given the most likely path, the adaptation transforms may be estimated by assuming that a particular state j and time frame t has an occupancy $L_j(t)$ of 1.0 if the state/time frame appears on the most likely path, and an occupancy of 0.0 otherwise. In the implemented alternative to the adaptation process, a mixture component-level Viterbi alignment (i.e. an alignment of time frames to mixture components) is used.

Figure 5.1 shows the adaptation process with the forward-backward algorithm replaced by Viterbi training. This process is referred to henceforth as *Viterbi adaptation*.

Viterbi decoding is implemented in the HTK using the token-passing algorithm described in [4]. Since the state-level Viterbi alignment is already computed by the alignment process, only the following refinements are required to produce a mixture-level alignment.

- In the forward pass of Viterbi decoding, record the most likely mixture component for each

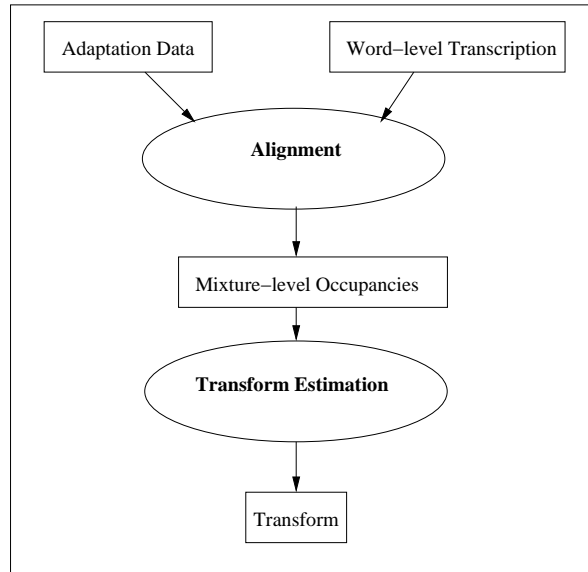


Figure 5.1: Viterbi Adaptation System

state/time frame combination.

- In the backward trace of the decoding, for each time frame, output the recorded mixture component in addition to the recorded state.

The most likely mixture component m within a state i at time t is found using

$$m = \arg \max_k \{c_{ik} b_{ik}(\mathbf{o}(t))\} \quad (5.1)$$

where c_{ik} is the mixture weight, and b_{ik} is the mixture output probability density function.

Now for each mixture component k , state i and time t , the product $c_{ik} b_{ik}(\mathbf{o}(t))$ is computed during the state-level output probability calculation. So the recovery of the most likely mixture involves no additional computation. Only some additional memory is required to record the most likely component for each state/time frame.

5.1.1 Evaluation

Table 5.1 gives timing information for each stage of the adaptation process for both the baseline system described in Section 4.2, and the Viterbi adaptation system. In both cases MLLR mean adaptation is used.

In the case of all datasets examined, Viterbi adaptation requires only 25% of the time necessary for the baseline adaptation process. It is clear from Table 5.1 that the majority of time saved by using this technique is due to the omission of the forward-backward stage.

	Baseline Adaptation				Viterbi Adaptation			
	feb89	feb91	oct89	sep92	feb89	feb91	oct89	sep92
Alignment	10325	10200	10816	10430	18546	18326	19408	18740
Forward-Backward	82270	82079	88455	83391	-	-	-	-
Transform Estimation	19927	20016	17121	21157	9776	9922	9417	10315
Total	112522	112295	116392	114978	28322	28248	28815	29055

Table 5.1: Timing: Baseline Vs Viterbi Adaptation

Some additional time is saved during transform estimation with Viterbi adaptation since only one mixture component per state is considered when estimating the transforms. The higher alignment times when using Viterbi adaptation are due to the following factors.

- Additional routines and memory allocation are necessary to produce a mixture component-level alignment where a phone-level alignment is produced in the baseline adaptation system.
- When using Viterbi adaptation some additional statistics are accumulated during the alignment phase, namely $\sum_{t=1}^T L_{jm}(t)\mathbf{o}(t)$ and $\sum_{t=1}^T L_{jm}(t)$ for each mixture component.

Table 5.2 compares the word accuracy of the baseline-adapted system and the Viterbi-adapted system.

Dataset	Unadapted	Baseline Adaptation	Viterbi Adaptation
feb89	96.60%	97.29%	97.26%
feb91	95.45%	96.78%	96.46%
oct89	95.45%	96.27%	96.31%
sep92	91.72%	94.57%	94.49%
Overall	94.81%	96.22%	96.13%

Table 5.2: Accuracy: Baseline Vs Viterbi Adaptation

Overall, the Viterbi adaptation system yields slightly lower accuracies (0.09% absolute) due to the more robust occupancy calculation using the forward-backward algorithm. This reaffirms the results of [2], where use of the forward-backward algorithm is shown to yield lower WER than use of Viterbi alignment.

5.2 Transform Estimation

One way to reduce the amount of computation required to estimate the transformation matrix $\mathbf{W}^{(sr)}$ is to replace the MLLR estimation technique with LSLR. In this section the computational cost of MLLR and LSLR are compared, and the different techniques are evaluated.

5.2.1 MLLR Computational Cost

Consider the statistics required for estimation of the transformation corresponding to one particular regression class $M(r)$. Suppose there are R components with non-zero occupancy in this class.

To simplify the cost calculation, it is assumed that no parameter tying scheme is used e.g. tied-mixture HMMs. Furthermore, it is assumed that the mixture components have diagonal covariance matrices and that a bias vector is used in conjunction with a full transformation matrix. The cost of estimation of a block matrix transformation can be derived in a similar manner.

Computation of \mathbf{Z}

To compute the matrix \mathbf{Z} of Equation 3.5, it is necessary to accumulate a weighted sum of observations for each mixture component $\sum_{t=1}^T L_{jm}(t)\mathbf{o}_t$. In the worst case this requires DT multiply-accumulate operations for each component, hence a total of RDT multiply-accumulate operations for all components in $M(r)$, where T is the total number of time frames of adaptation data, and D is the dimensionality of the observation vector.

Once the weighted sum of observations is calculated, and using the diagonal covariance assumption, computation of the matrix element z_{pq} is given by Equation 5.2.

$$z_{pq} = \sum_{(j,m) \in M(r)} \sigma_{pp}^{(jm)} \omega_p^{(jm)} \mu_q^{(jm)} \quad (5.2)$$

where z_{pq} , $\sigma_{pp}^{(jm)}$, $\omega_p^{(jm)}$ and $\mu_q^{(jm)}$ are the elements of the matrices \mathbf{Z} , Σ_{jm}^{-1} , $\sum_{t=1}^T L_{jm}(t)\mathbf{o}_t$ and ξ_{jm} respectively.

Since $\mu_{D+1}^{(jm)} = 1$, Equation 5.2 may be decomposed into Equation 5.3 and Equation 5.4.

$$z_{pq} = \sum_{(j,m) \in M(r)} \sigma_{pp}^{(jm)} \omega_p^{(jm)} \mu_q^{(jm)} \quad (5.3)$$

where $1 \leq p \leq D$ and $1 \leq q \leq D$.

$$z_{pq} = \sum_{(j,m) \in M(r)} \sigma_{pp}^{(jm)} \omega_p^{(jm)} \quad (5.4)$$

where $1 \leq p \leq D$ and $q = (D + 1)$.

To compute the the matrix \mathbf{Z} , firstly calculate $\sigma_{pp}^{(jm)} \omega_p^{(jm)}$ for each component, and for each value of the index p . Once This requires RD multiplications.

A further R multiply-accumulate operations are required to compute $\sum_{(j,m) \in M(r)} (\sigma_{pp}^{(jm)} \omega_p^{(jm)}) \mu_q^{(jm)}$ for a particular element index pq where $1 \leq p \leq D$ and $1 \leq q \leq D$. Since there are D^2 such elements, this section of matrix \mathbf{Z} requires RD^2 multiply-accumulate operations.

Computation of an element of the last column of \mathbf{Z} (Equation 5.4) requires a further R addition operations, and since there are D such elements, a total of RD addition operations are required.

Table 5.3 details the total cost of computation of \mathbf{Z} .

Operation Type	Operations
Multiplication	RD
Multiply-accumulate	$RDT + RD^2$
Addition/Subtraction	RD

Table 5.3: Cost of computation of \mathbf{Z} (Equation 3.5)

Computation of \mathbf{V}

Before computing the matrix $\mathbf{G}^{(i)}$ of Equation 3.6, the matrix \mathbf{V}_{jm} of Equation 3.7 is computed. To compute \mathbf{V}_{jm} , it is necessary to accumulate the sum of the occupancies for each mixture component, $\sum_{t=1}^T L_{jm}(t)$. This requires T addition operations per mixture component, a total of RT addition operations.

Additionally, assuming that covariance matrices are diagonal, and that their inverses are pre-computed, \mathbf{V}_{jm} requires a further D multiplication operations, where again, D is the dimensionality of the observation vector. Table 5.4 details the cost of calculating the matrix \mathbf{V}_{jm} for all mixture components in the regression class $M(r)$.

Operation Type	Operations
Multiplication	RD
Addition/Subtraction	RT

Table 5.4: Cost of computation of \mathbf{V}_{jm} (Equation 3.7) for R mixture components

Computation of \mathbf{G}

Computation of the matrix element $g_{pq}^{(i)}$ is given by Equation 5.5.

$$g_{pq}^{(i)} = \sum_{(j,m) \in M^{(r)}} \nu_{ii}^{(jm)} \mu_p^{(jm)} \mu_q^{(jm)} \quad (5.5)$$

where $g_{pq}^{(i)}$, $\nu_{ii}^{(jm)}$, and $\mu_q^{(jm)}$ are the elements of the matrices $\mathbf{G}^{(i)}$, \mathbf{V}_{jm} , and $\boldsymbol{\xi}_{jm}$ respectively.

Since $\mu_{D+1}^{(jm)} = 1$, Equation 5.5 may be decomposed into Equations 5.6, 5.7 and 5.8.

$$g_{pq}^{(i)} = \sum_{(j,m) \in M^{(r)}} \nu_{ii}^{(jm)} \mu_p^{(jm)} \mu_q^{(jm)} \quad (5.6)$$

where $1 \leq p \leq D$ and $1 \leq q \leq p$.

$$g_{pq}^{(i)} = \sum_{(j,m) \in M^{(r)}} \nu_{ii}^{(jm)} \mu_q^{(jm)} \quad (5.7)$$

where $p = (D + 1)$ and $1 \leq q \leq D$.

$$g_{pq}^{(i)} = \sum_{(j,m) \in M^{(r)}} \nu_{ii}^{(jm)} \quad (5.8)$$

where $p = (D + 1)$ and $q = (D + 1)$.

Note that since $\mathbf{G}^{(i)}$ is symmetric, only the lower triangle of the matrix need be computed.

Firstly calculate $\nu_{ii}^{(jm)} \mu_q^{(jm)}$ where $1 \leq q \leq D$, and for all components in the regression class. This is a total of RD multiplications.

Consider the section of matrix $\mathbf{G}^{(i)}$ described by Equation 5.6. A further R multiply-accumulate operations are required to compute $g_{pq}^{(i)}$ for a particular element index pq . Since Equation 5.6 applies to $D(D + 1)/2$ elements of matrix $\mathbf{G}^{(i)}$, this needs a total of $RD(D + 1)/2$ multiply-accumulate operations.

Equation 5.7 requires a further R addition operations, so a total of RD additions are required for all D elements satisfying the equation.

The remaining element of the matrix $\mathbf{G}^{(i)}$ is described by Equation 5.8 and requires R addition operations.

Table 5.5 details the total computational cost of the matrix $\mathbf{G}^{(i)}$.

Operation Type	Operations
Multiplication	RD
Multiply-accumulate	$RD(D + 1)/2$
Addition/Subtraction	$RD + R$

Table 5.5: Cost of computation of $\mathbf{G}^{(i)}$ (Equation 3.6)

Computation of \mathbf{W}

To compute the i th row of the transformation matrix $\mathbf{W}^{(sr)}$, it is necessary to compute $\mathbf{G}^{(i)}$, invert it, and then compute the matrix multiplication of Equation 3.4. The matrix multiplication for a single row entails $(D + 1)^2$ multiply-accumulate operations.

Table 5.6 details the total computational cost of the matrix $\mathbf{W}^{(sr)}$, including the cost of \mathbf{Z} , the cost of \mathbf{V}_{jm} for all regression classes, the cost of $\mathbf{G}^{(i)}$ for each row i , and the cost of the matrix inversion and matrix multiplication for each row.

Operation Type	Operations
Multiplication	$RD^2 + 2RD$
Multiply-accumulate	$RD^2(D + 1)/2 + RD^2 + RDT + D(D + 1)^2$
Addition/Subtraction	$RD^2 + 2RD + RT$
Matrix inversion	D

Table 5.6: Cost of MLLR computation of $\mathbf{W}^{(sr)}$ (Equation 3.4)

5.2.2 LSLR Computational Cost

Once again, for the purposes of this cost calculation, it is assumed that no HMM parameter-tying scheme is used.

Equation 3.10 gives the LSLR estimate of the transformation matrix. The computation of this estimation includes the computation of matrices \mathbf{X} and \mathbf{Y} , the inversion of \mathbf{Y} , and the matrix multiplication $\mathbf{X}\mathbf{Y}^{-1}$, where \mathbf{X} and \mathbf{Y} are given by Equations 5.9 and 5.10 respectively.

$$\mathbf{X} = \sum_{(j,m) \in M^{(r)}} \sum_{t=1}^T L_{jm}(t) \mathbf{o}(t) \xi'_{jm} \quad (5.9)$$

$$\mathbf{Y} = \sum_{(j,m) \in M^{(r)}} \sum_{t=1}^T L_{jm}(t) \xi_{jm} \xi'_{jm} \quad (5.10)$$

Computation of \mathbf{X}

Computation of \mathbf{X} requires the summation $\sum_{t=1}^T L_{jm}(t) \mathbf{o}(t)$ for each member of the regression class with non-zero occupancy, a worst-case total of RDT multiply-accumulate operations.

Computation of the matrix element x_{pq} is given by Equation 5.11.

$$x_{pq} = \sum_{(j,m) \in M^{(r)}} \omega_p^{(jm)} \mu_q^{(jm)} \quad (5.11)$$

where x_{pq} , $\omega_p^{(jm)}$ and $\mu_q^{(jm)}$ are the elements of the matrices \mathbf{X} , $\sum_{t=1}^T L_{jm}(t) \mathbf{o}_t$ and ξ_{jm} respectively.

Equation 5.11 may be rewritten as Equation 5.12 and Equation 5.13 since $\mu_{D+1}^{(jm)} = 1$.

$$x_{pq} = \sum_{(j,m) \in M^{(r)}} \omega_p^{(jm)} \mu_q^{(jm)} \quad (5.12)$$

where $1 \leq p \leq D$ and $1 \leq q \leq D$.

$$x_{pq} = \sum_{(j,m) \in M^{(r)}} \omega_p^{(jm)} \quad (5.13)$$

where $1 \leq p \leq D$ and $q = (D + 1)$.

To calculate the section of matrix \mathbf{X} described by Equation 5.12 requires RD^2 multiply-accumulate operations. Calculation of the last column of matrix \mathbf{X} (Equation 5.13) requires RD addition operations.

Table 5.7 summarises the computational cost of the matrix \mathbf{X} .

Operation Type	Operations
Multiply-accumulate	$RD^2 + RDT$
Addition/Subtraction	RD

Table 5.7: Cost of computation of \mathbf{X} (Equation 5.9)

Computation of \mathbf{Y}

Computation of \mathbf{Y} requires the summation $\sum_{t=1}^T L_{jm}(t)$ for R components of the regression class, a worst-case total of RT addition operations.

Computation of the matrix element y_{pq} is given by Equation 5.14.

$$y_{pq} = \sum_{(j,m) \in M^{(r)}} \rho^{(jm)} \mu_p^{(jm)} \mu_q^{(jm)} \quad (5.14)$$

where y_{pq} , and $\mu_q^{(jm)}$ are the elements of the matrices \mathbf{Y} and ξ_{jm} respectively, and $\rho^{(jm)} = \sum_{t=1}^T L_{jm}(t)$.

Since $\mu_{D+1}^{(jm)} = 1$, Equation 5.14 may be rewritten as Equations 5.15, 5.16 and 5.17.

$$y_{pq} = \sum_{(j,m) \in M^{(r)}} \rho^{(jm)} \mu_p^{(jm)} \mu_q^{(jm)} \quad (5.15)$$

where $1 \leq p \leq D$ and $1 \leq q \leq p$.

$$y_{pq} = \sum_{(j,m) \in M^{(r)}} \rho^{(jm)} \mu_q^{(jm)} \quad (5.16)$$

where $1 \leq q \leq D$ and $p = (D + 1)$.

$$y_{pq} = \sum_{(j,m) \in M^{(r)}} \rho^{(jm)} \quad (5.17)$$

where $p = (D + 1)$ and $q = (D + 1)$.

Note that \mathbf{Y} is symmetric, so only the lower triangle is computed.

Calculation of \mathbf{Y} firstly requires RD multiplication operations to compute $\rho^{(jm)} \mu_q^{(jm)}$ for each component in the regression class and for each index q where $1 \leq q \leq D$.

To calculate the section of matrix \mathbf{Y} described by Equation 5.15 needs a further a further $RD(D + 1)/2$ multiply-accumulate operations.

Calculation of the section of matrix \mathbf{Y} described by Equation 5.16 requires an additional RD addition operations.

Calculation of the bottom-right element of matrix \mathbf{Y} described by Equation 5.17 requires R addition operations.

Table 5.8 displays the total computational cost of the matrix \mathbf{Y} .

Operation Type	Operations
Multiplication	RD
Multiply-accumulate	$RD(D + 1)/2$
Addition/Subtraction	$RD + RT + R$

Table 5.8: Cost of computation of \mathbf{Y} (Equation 5.10)

Computation of \mathbf{W}

The additional operations required to calculate the transformation matrix $\mathbf{W}^{(sr)}$ are

- inversion of matrix \mathbf{Y}
- matrix multiplication $\mathbf{X}\mathbf{Y}^{-1}$

The matrix multiplication requires $D(D+1)^2$ multiply-accumulate operations. Table 5.9 displays the total computational cost of the matrix $\mathbf{W}^{(sr)}$.

Operation Type	Operations
Multiplication	RD
Multiply-accumulate	$RD(D+1)/2 + RD^2 + RDT + D(D+1)^2$
Addition/Subtraction	$RT + 2RD + R$
Matrix inversion	1

Table 5.9: Cost of LSLR computation of $\mathbf{W}^{(sr)}$ (Equation 3.10)

Table 5.10 compares the computational cost of MLLR estimation and LSLR estimation of the transformation matrix for a single regression class. It is evident that MLLR estimation requires more operations of all types.

Operation Type	Operations	
	MLLR	LSLR
Multiplication	$RD^2 + 2RD$	RD
Multiply-accumulate	$RD^2(D+1)/2 + RD^2 + RDT + D(D+1)^2$	$RD(D+1)/2 + RD^2 + RDT + D(D+1)^2$
Addition/Subtraction	$RD^2 + 2RD + RT$	$RT + 2RD + R$
Matrix inversion	D	1

Table 5.10: Cost of MLLR Vs LSLR Estimation

5.2.3 Evaluation

Table 5.11 displays the time taken to estimate the transformations (in milliseconds) using both MLLR and LSLR for the RM evaluation datasets. The Viterbi adaptation system is used in both cases. The mixture components are split into two regression classes - one for silence models and one for speech. A significant reduction in transform estimation time in the case of LSLR is evident, as predicted by the cost comparison of Section 5.2.2. For all datasets, LSLR transform estimation requires just less than 20% of the time demanded by the MLLR technique.

Table 5.12 compares the accuracy of the Viterbi-adapted MLLR and LSLR systems. The MLLR technique yields slightly higher accuracies (0.12% absolute). This is due to the false assumption that all mixture components of the same regression class share the same covariance matrix when using LSLR. This has been shown to be the case in [2]. The same paper also shows that the performance

Dataset	Transform Estimation Time	
	MLLR	LSLR
feb89	9776	1947
feb91	9922	1972
oct89	9417	1868
sep92	10315	2054

Table 5.11: MLLR Vs LSLR Transform Estimation Times

Dataset	Unadapted	LSLR Estimation	MLLR Estimation
feb89	96.6%	97.22%	97.26%
feb91	95.45%	96.94%	96.46%
oct89	95.45%	96.16%	96.31%
sep92	91.72%	93.75%	94.49%
Overall	94.81%	96.01%	96.13%

Table 5.12: Accuracy: MLLR Vs LSLR Estimation

gap between MLLR and LSLR increases as the number of regression classes increases, highlighting the importance of usage of the covariance matrix information in transform estimation.

5.2.4 Matrix Inversion Computation

Notice that no decomposition of the matrix inversion operation into more basic units of computation was detailed in the preceding sections. The technique used by the HTK to invert matrices is an extension of singular value decomposition (SVD) known as truncated SVD. Truncated SVD is used because it is a stable technique for the inversion of near-singular matrices.

The SVD algorithm, however, is difficult to cost in terms of elementary operations since it uses a procedure with an indefinite number of iterations. It is clear, however, from Table 5.10 that in the case of both MLLR and LSLR, as the amount of adaptation data (T) increases, and the amount of components in the regression class with non-zero occupancy (R) increases, the matrix inversion computation will occupy a less significant portion of the entire transform computation.

Table 5.13 compares the total time spent on the MLLR transform estimation to the total time spent on matrix inversion during the MLLR estimation for each speaker in the **feb89** dataset.

The processing time of the matrix inversion varies only slightly from speaker to speaker. This is because the inversion computation does not directly relate to the amount of adaptation data processed or the amount of regression class components participating in the transformations. Matrix inversion demands only a small portion the overall transform estimation time, so efficient inversion routines are not explored in this report.

Speaker	MLLR Transform Estimation Time	Total Matrix Inversion Time
cmh1_8	932	70
dm10_1	606	64
dwa0_5	1195	70
esg0_4	1177	69
gaw0_7	1100	69
gmb0_5	1076	71
hlm0_2	1204	69
jd0_6	617	63
kls0_1	647	64
lms0_1	1265	69

Table 5.13: MLLR Transform Estimation Times Vs Total Matrix Inversion Time

5.3 Efficient Alignment

Techniques have been successfully implemented to significantly diminish the overall adaptation time via reduction of the transform estimation time and use of mixture component-level Viterbi alignment in place of the forward-backward algorithm. Table 5.14 shows how the transform estimation time compares with the time required to align the data when both LSLR and Viterbi adaptation are deployed. The system is otherwise identical to the baseline system of Section 4.2.

Dataset	Alignment	LSLR Transform Estimation
feb89	18502	1947
feb91	18294	1972
oct89	19376	1868
sep92	18701	2054

Table 5.14: Viterbi Alignment And LSLR Transform Estimation Times

For all datasets, the alignment time accounts for over 90% of the entire adaptation time when using Viterbi adaptation with LSLR transform estimation. In this section an attempt to reduce the alignment time is presented and evaluated.

5.3.1 Constrained Alignment

As explained in Section 5.1, in the Viterbi adaptation system, a word-level transcription of the speech data is used as input to the alignment process. This process aligns mixture components to time frames, and passes the alignment to the transform estimation procedure. In the case of unsupervised adaptation, the word-level transcription is produced by a previous recognition pass. Word boundary information, i.e. word start and end times, is also provided in this transcription, but is ignored by the alignment process.

This word boundary information can, however, be used effectively, by constraining the alignment to pursue only those paths which respect the word end times.

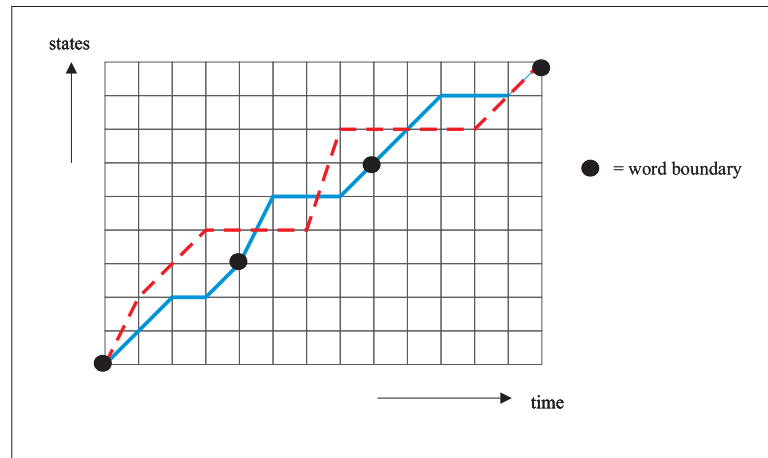


Figure 5.2: Constrained Alignment

The solid and dashed lines in Figure 5.2 represent two possible state sequences through a composite HMM. The bold dots represent word boundaries. When using unconstrained alignment both paths are potentially the best path through the composite HMM. However, when using constrained alignment only the path represented by the solid line is considered as a possible best path. This is because the dashed path makes transitions between words at times other than the specified word boundary times. All such paths which fail to respect the word boundaries are disregarded. Thus the number of paths pursued, and potentially the alignment time, is reduced.

The technical implementation of constrained alignment involves the following alterations to the token-passing algorithm.

- External token propagation from the HMM models of one word to the following word occurs only at the time frame corresponding to the word end time.
- At any particular time frame t , only the HMM models of one word are active. To enforce this, all of the HMM models of a word are deactivated immediately after token propagation to the following word.

5.3.2 Evaluation

The constrained alignment technique is now compared to the unconstrained version in terms of adaptation time and accuracy of the adapted system.

LSLR transform estimation and Viterbi adaptation are used, the system being otherwise identical to the baseline system described in Section 4.1. Note that a beam width of 200 is used during alignment.

Dataset	Unconstrained	Constrained	Saving
feb89	18502	15080	16%
feb91	18294	14750	20%
oct89	19376	15831	19%
sep92	18701	15122	19%
Total	74843	60783	19%

Table 5.15: Timing: Unconstrained Vs Constrained Alignment

Table 5.15 shows how the constrained alignment time compares with the unconstrained version. With the RM evaluation data, the constrained alignment technique reduces the total alignment time by 19%. Table 5.16 reveals that the use of constrained alignment comes at no expense of accuracy of adaptation.

Dataset	Unadapted	Unconstrained Alignment	Constrained Alignment
feb89	96.6%	97.22%	97.22%
feb91	95.45%	96.94%	96.94%
oct89	95.45%	96.16%	96.16%
sep92	91.72%	93.75%	93.75%

Table 5.16: Accuracy: Constrained Vs Unconstrained Alignment

Identical accuracies are obtained since identical alignments (and hence transformations) are produced by the constrained and unconstrained systems. This, in turn, is because the same best path has been identified in the first recognition pass, which generated the word-boundary information.

Since Viterbi alignment is of course performed within the first recognition pass, one could attempt to estimate transformations directly at this stage. In theory, this is a reasonable suggestion. In practice, however, a significant amount of extra memory is required to record mixture component-level information for each time frame and active state in the first pass.

5.3.3 Further Constrained Alignment

The results of Section 5.3.2 are encouraging. It is, however, possible to improve further upon adaptation time whilst retaining identical recogniser accuracy.

The problem is to decode a composite HMM using Viterbi decoding. In the general case, for each word in the composite HMM, the word end time is unknown. This is, however, not the case when using constrained alignment. For each word, the word end time T is known beforehand. So, for each word, the decoded path must include the final state of the final HMM model at time T .

Consider the forward pass of the token-passing algorithm at time t . If constrained alignment is in use, only one word is active at time t , so consider only the HMMs comprising this word. It is only necessary to consider tokens which can reach the final state of the final HMM (the word end state) at time T , since the ‘winning’ token must certainly do so. Therefore any token which must visit more than $T - t$ emitting states before reaching the word end state can be disregarded.

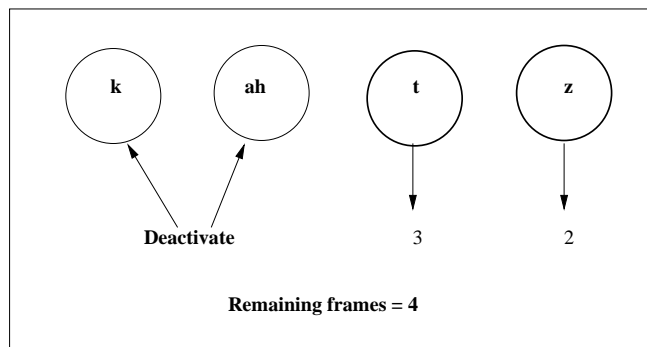


Figure 5.3: ‘Look Ahead’ Mechanism

In practice, it is simpler to deactivate entire HMM models which contain no possible winning token. The minimum number of emitting states between one model and the word end state can be calculated by summing the *minimum duration* of all of the succeeding models. If this sum is greater than $T - t$, the model is deactivated.

This technique is illustrated in Figure 5.3. The model for the word ‘cats’ is shown. Suppose the minimum duration for the phones ‘t’ and ‘z’ are 3 and 2 respectively. Suppose further that the number of remaining frames is 4. Then the phone models ‘k’ and ‘ah’ can be safely deactivated because they contain no token which can be propagated to the end state of the phone ‘z’ within the time remaining.

This further constraint on alignment will be referred to as the ‘look-ahead’ feature.

5.3.4 Evaluation

Table 5.17 shows the impact of the ‘look-ahead’ feature described in Section 5.3.3. Alignment times using only constrained alignment are compared with alignment times when using constrained alignment with ‘look-ahead’ enabled.

Dataset	Constrained	Constrained + ‘Look-Ahead’	Saving
feb89	15080	14731	2.3%
feb91	14750	14490	1.8%
oct89	15831	15419	2.6%
sep92	15122	14717	2.7%
Total	60783	59357	2.3%

Table 5.17: Alignment Times: ‘Look-Ahead’ Feature Impact

With the ARPA RM evaluation data, use of the look-ahead mechanism offers, on average, a saving of slightly over 2% on constrained alignment time. This is a relatively minor saving, but importantly the technique has no impact on the accuracy of the adaptation.

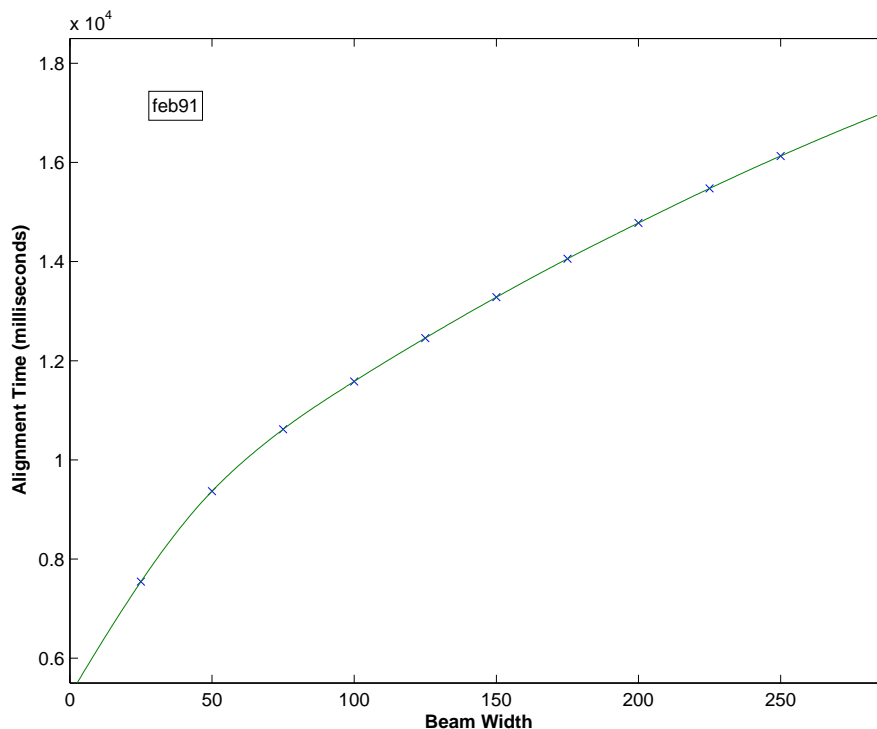


Figure 5.4: Beam Width Vs Alignment Time

5.3.5 Beam Width And Alignment

It is interesting to investigate if further additional restrictions upon the alignment process have a significant impact on the accuracy of the adapted system.

With the HTK, several pruning techniques may be used to restrict the number of paths pursued during the Viterbi alignment.

One such technique is known as the *beam search*. This search records the token with the highest probability in each HMM model at each time frame, the model's 'best' token. The global best token is also noted. The beam search subsequently deactivates all models for which the best token probability falls below a user-specified threshold of the global best token probability. This threshold is called the *beam width*. The token passing step is executed for each time frame, but ignores all inactive models. An HMM model is reactivated when a token is propagated into its initial state.

Beam search thus restricts the number of simultaneously active models, and hence the amount of computation required at each time frame. Another way to restrict the number of simultaneously active models is simply to provide a maximum number which must not be exceeded. This may be used alone, or in addition to a beam width.

With beam search enabled, it is possible that a sub-optimal alignment of an adaptation utterance will not be found, a consequence of pruning the best path at some stage. It is also possible that

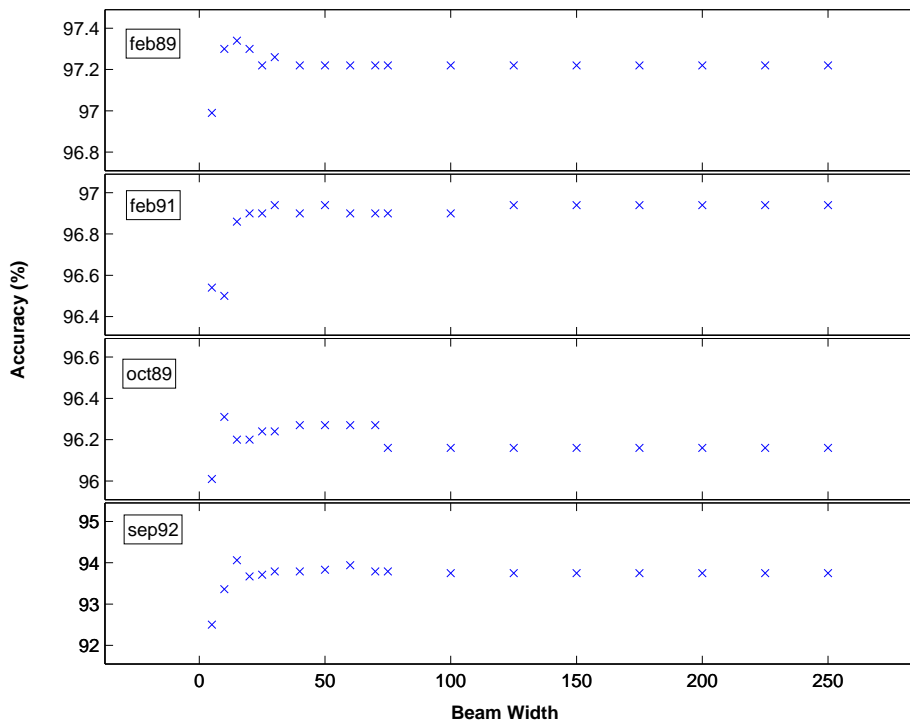


Figure 5.5: Beam Width Vs Accuracy

no alignment will be found, in which case the utterance is discarded from subsequent adaptation stages.

5.3.6 Evaluation

Figure 5.4 shows how the total alignment time varies with the beam width setting for the **feb91** dataset and Figure 5.5 displays the relationship between the accuracy of the adapted system and the beam width setting during the alignment phase. Constrained alignment is used with the look-ahead mechanism. It is clear from Figure 5.4 that as small a beam width as possible is desirable for low alignment times. However, Figure 5.5 reveals that very low beam widths can lead to unreliable transform estimation. This, in turn, is due to many adaptation utterances being discarded during alignment. Fewer adaptation utterances are used to calculate the transforms, which, consequently, are poorly estimated.

In the case of the ARPA RM evaluation data, an alignment beam width of 125 can be used with no impact on the accuracy of the adapted system. Table 5.18 shows how the efficient adaptation times compare with the baseline adaptation times. The efficient system uses constrained Viterbi alignment with ‘look-ahead’ and a beam width of 125 in conjunction with LSLR transform estimation.

	Baseline Adaptation				Efficient Adaptation			
	feb89	feb91	oct89	sep92	feb89	feb91	oct89	sep92
Alignment	10325	10200	10816	10430	12408	12223	12963	12367
Forward-Backward	82270	82079	88455	83391	-	-	-	-
Transform Estimation	19927	20016	17121	21157	1947	1970	1864	2050
Total	112522	112295	116392	114978	14355	14193	14827	14417

Table 5.18: Timing: Baseline Vs Efficient Adaptation

Overall, the efficient adaptation time is less than 13% of the baseline adaptation time, and yields an absolute accuracy 0.21% lower than the baseline system (96.22% versus 96.01%). This is a relative WER increase of 5.6%.

5.4 Confidence Scores And Adaptation

When using unsupervised adaptation, the first recognition pass can be used not only to generate a word-level transcription, but also to associate a confidence score with each word. This confidence score is a weighted sum of scores associated with the word, including its acoustic, language and pronunciation log-likelihoods.

Exploitation of confidence scores to improve the accuracy of unsupervised adaptation has been successfully demonstrated, for example in [14]. It is, however, interesting to investigate if the use of confidence scores can also reduce adaptation time. By ignoring data segments which have been assigned low scores during the first recognition pass, the quantity of adaptation data is reduced. This will lead to faster transform estimation, and possibly improved accuracy of the transform estimation.

When processing an utterance, the first pass transcription is used to calculate a *normalised confidence score* for each word. This is equal to the confidence score divided by the duration of the word. Both the duration and the confidence score are present in the transcription. If the normalised confidence score falls below a predefined *confidence limit*, the data segment associated with the word is discarded from the transform estimation procedure.

5.4.1 Evaluation

The adaptation subsystem used for evaluation uses LSLR transform estimation and employs constrained Viterbi alignment with ‘look-ahead’ and a beam width of 75.

The relationship between the confidence limit setting and the transform estimation time is illustrated in Figure 5.6. As the confidence limit increases, more data is discarded, the transform estimation accumulates fewer statistics and so requires less time.

Figure 5.7 plots the accuracy of the adapted system against the confidence limit setting for all datasets. The horizontal lines display the accuracy yielded when no confidence limit is used. At low confidence limits, the majority of the adaptation data is used and the adaptation transform is similar to the one generated when no confidence limit is employed. As the limit increases, a small

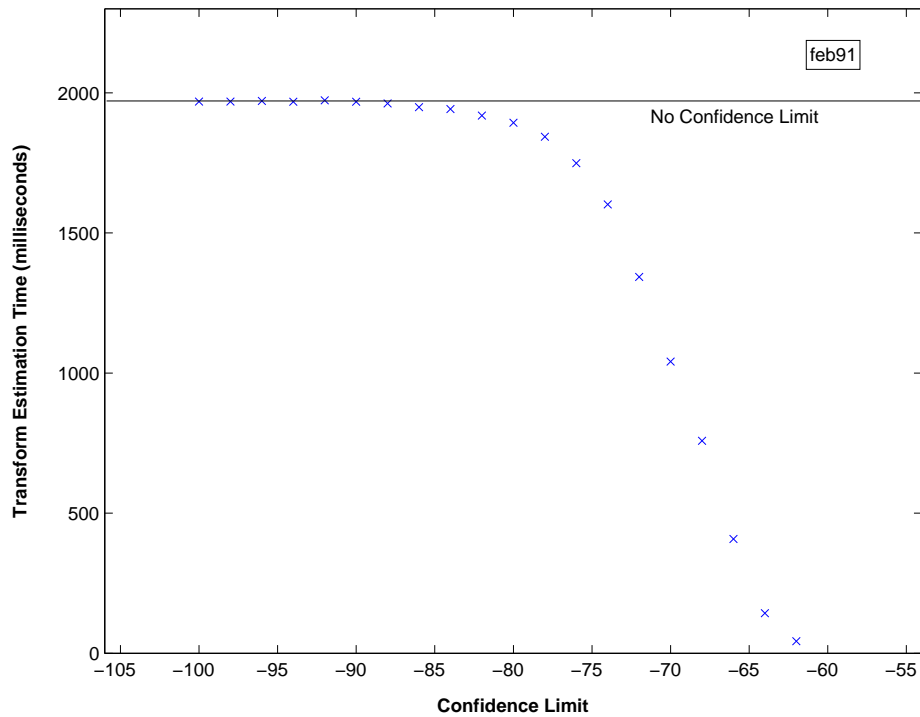


Figure 5.6: Confidence Limit Vs Transform Estimation Time

amount of low-confidence data is discarded, while enough data remains to yield a robust transform estimation, and improved accuracy is observed. Inclusion of the low-confidence data clearly has a degrading effect upon the transform estimation. However, high confidence limits result in less reliable adaptation. This is a consequence of poor transform estimation which, in turn, is due to small amounts of adaptation data being used at high confidence limit settings.

It is clear that an optimal setting exists for the confidence limit which both increases transform estimation time and increases accuracy of the adapted system. A technique for automatic estimation of this optimal setting is beyond the scope of this report, and is mentioned only as a possibility for future work.

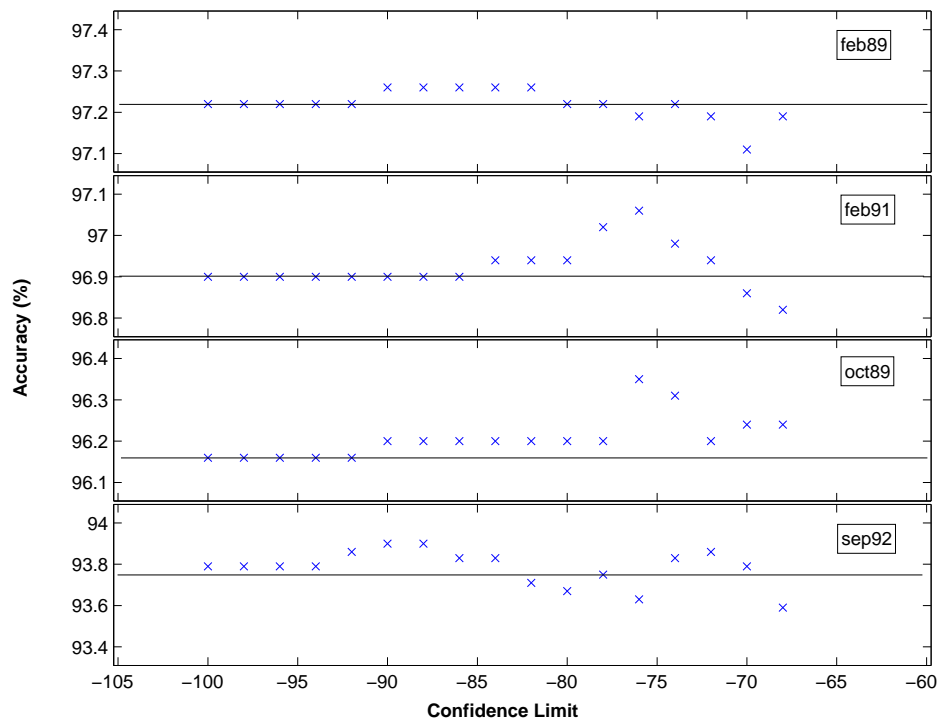


Figure 5.7: Confidence Limit Vs Accuracy

Chapter 6

Conclusions and Further Work

The CU-HTK adaptation system has been decomposed into three main stages: alignment, forward-backward algorithm and transform estimation. Each of these stages have been examined in terms of their contribution to the overall adaptation time.

It has been shown that a component-level Viterbi alignment can be effectively used to replace the forward-backward algorithm, offering a reduction of 75% the total baseline adaptation time with an observed relative increase of 2.4% (0.09% absolute) in WER.

Use of LSLR in place of the more general MLLR transform estimation technique displays impressive transform estimation time savings (a reduction of more than 80% over the baseline MLLR method) at the expense of a relative increase of 3.2% in WER (0.12% absolute).

In the case of unsupervised adaptation, use of word boundary times from the first recognition pass has been shown to save 19% on the baseline alignment time with no impact on WER. Careful use of the beam width setting can significantly lower alignment time with no negative impact on WER, while use of the ‘look-ahead’ feature offers relatively minor computation savings, again, without affecting WER.

Combining all of the above techniques, a reduction in the baseline adaptation time of 87% is possible, at the expense of a relative WER increase of 5.6% when using unsupervised adaptation and the RM evaluation data.

It has additionally been demonstrated how use of first pass confidence scores can simultaneously reduce the transform estimation time and improve the accuracy of the adapted system.

6.1 Further Work

Savings on alignment time during unsupervised adaptation are due to the use of word boundary times produced by a previous recognition pass. Extending this idea, one could record phone-level boundary times during the initial recognition pass. This information would be subsequently used to constrain the alignment. The time saved during alignment may be greater than the extra time required to record phone-level information during recognition, but this is certainly not clear and requires further investigation.

Use of the beam search technique is interesting when seeking to minimise alignment times. The optimal beam width saves as much time as possible whilst retaining enough adaptation data to yield reliable estimates of the adaptation transforms. In the experiments described in this report, the beam width was fixed manually.

With the HTK, one can also specify upper and lower beam width limits, as well as a beam width increment. If this configuration is used, the alignment process will firstly attempt to align the data using the lower beam width. In the case of failed alignment, the beam width is incremented by the specified amount and the alignment is retried. This process is repeated until either the upper beam width limit is attained or the alignment succeeds. Future work could investigate how these settings can be used to further reduce alignment time whilst retaining robust transform estimations.

A more sophisticated system might attempt to automatically set the beam width parameters discussed above, perhaps on a per-utterance basis. In the case of unsupervised adaptation, information from the initial recognition pass might serve to indicate use of a higher or lower beam width for a particular adaptation utterance. Further investigation is required to identify features of the data which indicate the need for higher beam width settings.

Another possibility for further work is to examine the relative importance of each mixture component in the system. It is feasible that some mixture components, when adapted, have a higher impact upon the recogniser accuracy than others. If this is indeed the case, an efficient adaptation system could use only those mixture components with a relatively high impact. An interesting experiment would observe adaptation speed and accuracy using various subsets of all mixture components.

How use of a confidence limit can reduce the transform estimation time has been demonstrated. However, it is also possible to use this limit to reduce alignment time. If the data associated with a low-confidence word is never used in transform estimation, there is no need to produce an alignment for this word. Further work might explore how much computation can be saved in bypassing the alignment of low-confidence words.

Bibliography

- [1] G. Evermann & P.C. Woodland, 'Design of Fast LVCSR Systems', *Proceedings, ASRU*, St. Thomas, U.S. Virgin Islands. 2003.
- [2] C.J. Leggetter & P.C. Woodland, 'Maximum Likelihood Linear Regression for Speaker Adaptation of Continuous Density Hidden Markov Models', *Computer Speech and Language*, Vol 9 pp.171-185, 1995.
- [3] S.J. Young et al, 'The HTK Book for HTK version 3.2.1', 2003.
- [4] S.J. Young, N.H. Russell & J.H.S. Thornton 'Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems', *Cambridge University*, 1989
- [5] J.J. Odell, P.C. Woodland, & T. Hain 'The CUHTK-Entropic 10xRT broadcast news transcription system', *Proceedings, DARPA Broadcast News Workshop*, pp. 271-275, Herndon, VA, USA, 1999.
- [6] S.E. Johnson, P. Jourlin, G.L. Moore, K. Sparck-Jones & P. C. Woodland 'The Cambridge University Spoken Document Retrieval System' *Proceedings ICASSP*, pp.49-52, Phoenix, 1999.
- [7] S.E. Johnson & P.C. Woodland 'Speaker Clustering Using Direct Maximisation of the MLLR-Adapted Likelihood' *Proceedings ICSLP*, pp.1775-1779, Sydney, 1999.
- [8] P.C. Woodland, T. Hain, G.L. Moore, T.R. Niesler, A. Tuerk, D. Povey & E.W.D. Whittaker, 'The 1998 HTK Broadcast News Transcription System: Development and Results' *Proceedings, DARPA Broadcast News Transcription and Understanding Workshop*, March 1999.
- [9] T. Hain, P.C. Woodland, G. Evermann, M.J.F. Gales, X. Liu, G.L. Moore, D. Povey & L. Wang, 'Automatic Transcription of Conversational Telephone Speech - Development of the CUHTK 2002 System', *Cambridge University Engineering Department Technical Report, CUED/F-INFENG/TR-465*, 2003.
- [10] L. Mangu, E. Brill & A. Stolcke, 'Finding Consensus Among Words: Lattice-Based Word Error Minimization' *Proceedings, Eurospeech* pp. 495-498, Budapest, 1999.
- [11] L. Lee & R.C. Rose, 'Speaker Normalization using Efficient Frequency Warping Procedures' *Proceedings ICASSP*, pp. 353-356, 1996.

- [12] L.F. Uebel & P. C. Woodland ‘Speaker Adaptation Using Lattice-based MLLR’ *Proceedings ISCA ITRW Adaptation Methods for Speech Recognition*, Sophia-Antopolis, 2001.
- [13] G. Evermann & P.C. Woodland, ‘Posterior Probability Decoding, Confidence Estimation and System Combination’ *Proceedings, Speech Transcription Workshop*, College Park, MD, 2000.
- [14] M. Pitz, F. Wessel & H. Ney, ‘Improved MLLR Speaker Adaptation using Confidence Measures for Conversational Speech Recognition’ *Proceedings, 6th International Conference on Spoken Language Processing*, pp. IV-548 - IV-551, Beijing, October 2000.
- [15] T. Hain & P. C. Woodland, ‘CU-HTK Acoustic modelling experiments’ *Proceedings, NIST Hub5 Workshop*, Linthicum Heights, MD, 1998.
- [16] T. Anastasakos, J. McDonough, R. Schwartz, & J. Makhoul, ‘A compact model for speaker-adaptive training’ *Proceedings, ICSLP*, pp. 11371140, 1996.