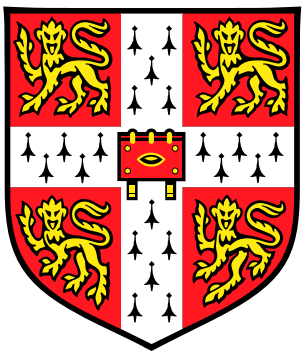
Generation and Combination of Complementary Systems for Automatic Speech Recognition

Catherine Breslin

Cambridge University Engineering Department
and
Darwin College
June 23, 2008



Dissertation submitted to the University of Cambridge
for the degree of Doctor of Philosophy

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration. It has not been submitted in whole or in part for a degree at any other university. Some of the work has been published previously in conference proceedings [15, 16, 17]. The length of this thesis including appendices, references, footnotes, tables and equations is approximately 56,000 words and contains 42 figures and 40 tables.

Summary

It has been found that using a combination of systems for large vocabulary continuous speech recognition (LVCSR) can outperform the use of a single system. For the combination to yield gains, the individual models must be complementary, i.e. they must make different errors.

Previous work in ASR has mainly relied on an ad-hoc approach to finding complementary systems. Multiple systems are built, and those that perform well in combination are selected. The multiple diverse systems can be built in many ways, including the use of different frontends, injecting randomness, altering the model topology or using different training algorithms. However, it is not guaranteed that independent systems will be complementary, and hence it is preferable to explicitly generate complementary systems. Generating complementary systems is a current research topic in ASR, and for machine learning in general.

In this thesis, two approaches for generating complementary systems for LVCSR are proposed. The first of these alters the decision tree algorithm for state clustering, and the second directly alters the standard ML and discriminative training algorithms. Both approaches make use of a data weighting over the training set, which allows the training data to be weighted to reflect the errors made by a number of previous systems. In this way, training can be focused directly on these errors. The complementary systems may correct errors made by previous systems, but can introduce new errors. However, if the systems make complementary errors, they should yield improvements when combined. The data weighting takes into account the errors made by multiple systems, and allows these algorithms to be embedded within an iterative framework for building multiple complementary systems. Thus, the approaches presented in this thesis differ from standard approaches to speech recognition in their aim of training an ensemble of speech recognisers which perform optimally when combined, not individually.

To address potential issues in decoding with the complementary systems, two modified combination schemes are proposed. They aim to better match the decoding and combination scheme with the complementary system training algorithm. The first of these uses the complementary system to rescore only segments of data where it is believed that the first system is incorrect. The second approach only combines the two systems when it is believed that the first is incorrect. Hence, both require accurate word error detection.

Experimental results on a broadcast news LVCSR task for three languages show that gains can be achieved by building multiple complementary systems with the directed tree approach. However, while the explicit training approach does build diverse systems, the current combination algorithm does not take advantage. Results also show that with an accurate word error detection algorithm, the combination of multiple complementary systems with the modified combination schemes could outperform the standard combination algorithm.

Keywords: speech recognition; hidden Markov models; acoustic modelling; complementary systems; system combination; discriminative training

Acknowledgements

First and foremost, I would like to thank my supervisor, Mark Gales, for his guidance and support throughout my time studying in Cambridge. Without his help, expertise and insight, this research would never have come to fruition. It has been a privilege to work with him.

Secondly, I would like to thank Toshiba Research Europe and the EPSRC for their generous funding, which has made it possible for me to complete this work. Also, I am indebted to those who have built and maintained the HTK toolkit.

I am grateful to many people in the Speech Research group for their help during my time here. Special thanks go to Anna Langley and Patrick Gosling for maintaining the computer systems at the Machine Intelligence Laboratory, and also to Andrew Liu, Khe Chai Sim and Kai Yu for their early help with setting up the broadcast news experiments, and their unending advice and assistance whenever I ran into problems.

There are many people who have made this group an interesting place to work. In no particular order, I'd like to thank Hank, Chris, Rogier, Andrew, Jamie, Frank, Graham, Kai and Khe Chai for their help and friendship. Particularly, I'd like to thank Mark, Hank and Kai for proofreading various parts of this thesis.

Finally, I must thank my friends outside of the department, who have always been there to distract me, and my family, who I don't visit often enough!

Acronyms

ASR	Automatic speech recognition
BN	Broadcast news
BW	Baum-Welch
CDF	Cumulative density function
CER	Character error rate
CMLLR	Constrained MLLR
CMN	Cepstral mean normalisation
CVN	Cepstral variance normalisation
DBN	Dynamic Bayesian network
EM	Expectation maximisation
FFT	Fast Fourier transform
FMLLR	Feature-space maximum likelihood linear regression
GMM	Gaussian mixture model
HLDA	Heteroscedastic linear discriminant analysis
HMM	Hidden Markov model
HTK	HMM toolkit
LVCSR	Large vocabulary continuous speech recognition
MAP	Maximum a posteriori
MFCC	Mel-frequency cepstral coefficients
ML	Maximum likelihood
MLLR	Maximum likelihood linear regression
MMI	Maximum mutual information
MPE	Minimum phone error
NIST	National Institute of Standards and Technology
PDF	Probability density function
PLP	Perceptual linear prediction
RM	Resource Management
SAT	Speaker adaptive training
SER	Sentence error rate
STC	Semi-tied covariances
VTLN	Vocal tract length normalisation
WER	Word error rate

Notation

These are the terms and notation used throughout this work.

Variables, Symbols and Operations

$\approx$	approximately equal to
$\propto$	proportional to
x	scalar quantity
$\hat{x}$	estimate of the true value of x
$\boldsymbol{x}$	vector of arbitrary dimensions
$\underset{x}{\text{argmax}}\, f(x)$	the value of x that maximises the value of $f(x)$
$\underset{x}{\text{argmin}}\, f(x)$	the value of x that minimises the value of $f(x)$
$\log(x)$	logarithm base e of x
$\exp(x)$	exponential of x
$E[f(x)]$	the expected value of $f(x)$, where x is a random variable
$f(x)\big\|_{\mu_0}$	evaluate function f(x) at the expansion point μ_0
$\sum_{n=1}^{N} a_n$	summation from $n = 1$ to N—that is, $a_1 + a_2 + \cdots + a_N$

Vectors and Matrices

$\mathcal{R}^d$	d-dimensional Euclidean space
$\boldsymbol{A}$	an arbitrary square matrix
$\boldsymbol{A}^\mathsf{T}$	transpose of matrix $\boldsymbol{A}$
trace$\{\boldsymbol{A}\}$	trace of matrix $\boldsymbol{A}$
diag$\{\boldsymbol{A}\}$	a diagonalised version of matrix $\boldsymbol{A}$
$\lvert\boldsymbol{A}\rvert$	determinant of matrix $\boldsymbol{A}$
$\boldsymbol{A}^{-1}$	inverse of matrix $\boldsymbol{A}$
a_{ij}	scalar value that is the element in row i and column j of $\boldsymbol{A}$

$\boldsymbol{I}$	identity matrix
$\boldsymbol{b}$	column vector
$\boldsymbol{a} \cdot \boldsymbol{b}$	dot product of $\boldsymbol{a}$ and $\boldsymbol{b}$ yielding a scalar
$h(t) * x(t)$	convolution operator—that is, $\int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau$

Observations

T	number of frames in a sequence of observations
t	time frame index
$\boldsymbol{o}_t$	speech observation vector at time t
$\mathcal{O}$	set of training data (sequence of speech observation vectors for ASR) $\{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T\}$
D	number of dimensions of full feature vector
d	dimension index
$\boldsymbol{C}$	discrete cosine transform matrix, with rows $\boldsymbol{c}_i$
$\boldsymbol{C}^{-1}$	inverse discrete cosine transform matrix

HMM Parameters

S	number of systems to be combined
s	index for the sth system
$\mathcal{M}$	set of current acoustic model parameters
$\mathcal{M}^{(s)}$	set of acoustic model parameters associated with system s
N	number of HMM states in the full acoustic model
θ_j	jth state of an HMM
$\boldsymbol{\Theta}$	set of P HMM states $\{\theta_1 \cdots \theta_P\}$
ψ_t	current state of an HMM at time t
$\boldsymbol{\psi}$	sequence of discrete speech states $\{\psi_1, \ldots, \psi_T\}$
$\boldsymbol{\Psi}$	set of all possible sequences $\boldsymbol{\psi}$
M	number of GMM components in the full acoustic model
m	index for the mth component of an GMM
c_{jm}	mixture weight associated with GMM component m of state j
$\boldsymbol{\mu}_{jm}$	mean of component m in state θ_j
$\boldsymbol{\Sigma}_{jm}$	variance of component m in state θ_j

Parameter Estimation

$\gamma_{jm}(t)$	posterior probability of component m in state θ_j at time t
$\mathcal{Q}(\mathcal{M}, \hat{\mathcal{M}})$	auxiliary function where component posteriors computed using parameter set $\mathcal{M}$ and output probabilities with $\hat{\mathcal{M}}$
$\mathcal{H}$	hypothesised transcription (sequence of words)
$\mathcal{H}^{(s)}$	hypothesised transcription output from system s
$\hat{\mathcal{H}}$	estimated 1-best hypothesis
$\mathcal{H}_{ref}$	reference transcription of data
K	number of words in a hypothesis
k	index to kth word in a hypothesis
$\mathcal{W}^{(k)}$	kth word in a hypothesis
$\hat{\mathcal{W}}^{(k)}$	kth word in estimated 1-best hypothesis
$\boldsymbol{\mathcal{W}}$	set of aligned words
$\mathcal{P}$	phone in a hypothesis
$\mathcal{L}(\mathcal{H}_1, \mathcal{H}_2)$	loss between two hypotheses, $\mathcal{H}_1$ and $\mathcal{H}_2$
$l(\mathcal{W}_1, \mathcal{W}_2)$	loss between two words, $\mathcal{W}_1$ and $\mathcal{W}_2$
λ_s	weight on system s
A + B	CNC combination of systems A and B

Probability and Distributions

$P(\cdot)$	probability mass function
$p(\cdot)$	probability density function
$p(x, y)$	joint probability density function—that is, the probability density of x and y
$p(x\|y)$	conditional probability density of x given y
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
$\mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	probability of vector $\boldsymbol{o}$ given a multivariate Gaussian distribution
$\delta(x)$	Dirac delta function: $\delta(x) = 0$ for $x \neq 0$ and $\int_{-\infty}^{\infty} \delta(x)dx = 1$
$\delta(i, j)$	Kronecker delta function, which equals 1 when $i = j$ and is 0 otherwise

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

Speech is a natural method of communication, and can potentially provide an intuitive user interface to machines. Applications range from speaker dependent tasks like desktop dictation and interaction with personal devices, such as mobile phones, through to speaker independent tasks such as automatic call centres, automatic subtitling and indexing videos for improved search.

For these applications to be practical, several different aspects of speech need to be considered, including speech-to-text, language understanding, dialogue systems, and text-to-speech. This thesis focuses on the first of these, the problem of automatic speech recognition (ASR).

1.1 Automatic Speech Recognition

Automatic speech recognition has been the subject of research for over fifty years, and has matured markedly during this time. This is due in part to the increase in available computing power, and in part to more sophisticated modelling techniques. The introduction of the HMM in the 1970s [8], and a statistical framework for ASR, has proven the most successful approach to date, and is the basis for current state-of-the-art speech recognisers.

Initially, the focus of automatic speech recognition was on isolated word recognition for small vocabularies, such as the task of digit recognition. With good performance on such tasks, the focus then shifted towards small and medium vocabulary continuous speech recognition. The performance on these has steadily improved and consequently, in recent years, research has begun to consider large vocabulary continuous speech recognition (LVCSR). Due to the expense of obtaining a large training database, real-life or *found data* is often used to train these systems. This is data that is readily available, such as broadcast news (BN) or broadcast conversation (BC). Performance on these tasks is still relatively poor, for example around 10%

character error rate (CER) on broadcast news Mandarin [48], which is significantly worse than the performance of human transcription on spontaneous speech [97].

ASR technology is now commercially deployed in a variety of applications. For example, Microsoft's Vista operating system allows the user to interact with the computer via speech recognition, while call centres like those at Cineworld and BA's flight information service also use a voice interface. Dictation software such as IBM's ViaVoice and Nuance's Dragon NaturallySpeaking have been available for many years, and can achieve good performance when the system has been trained for a particular user.

Recent LVCSR projects include the AGILE/GALE project[1] for speech-to-speech translation of broadcast news, Computers in the Human Interaction Loop (CHIL)[2] for incorporating computers into human interactions in a non-intrusive manner, and the TC-STAR project[3] where the overall aim is also speech-to-speech translation, focused on the European Parliamentary speeches.

1.2 Complementary Systems

This thesis concentrates on large vocabulary continuous speech recognition (LVCSR). A typical architecture for LVCSR is a multi-pass framework. Such a framework performs an initial decoding pass of the data to obtain a rough hypothesis, before refining the hypothesis in the final pass using complex models. It has been found that improved results can be obtained by using a combination of models in the final pass. A combination of models can only give improvements if they are complementary. That is, if they make different errors. This thesis is concerned with generating and combining complementary systems for large vocabulary recognition.

There are several reasons why a combination of systems can outperform a single system, and it is necessary to consider the limitations of using a single system for recognition. There are several ways to improve the performance of a single system. First, the underlying model parameters can be improved, perhaps by ignoring simple modelling assumptions, to better model the data. For speech recognition, more advanced models have had limited success, and the HMM based approach presented below in chapter 2 remains the most successful.

Second, the models can be made more complex by increasing the number of parameters in the system so they have more power to model the data. However, it is well known that increasing the number of parameters can led to overtraining and a lack of generalisation. To address this problem, more training data is normally added. Unfortunately, increasing the amount of training data is not always the most efficient way to improve results as it is expensive to accurately transcribe a large database, and the gains achieved from incorporating more training data become increasingly smaller [80].

Finally, improved training algorithms, such as discriminative training discussed in section 2.4.2.2, can be used to improve the underlying models. Again, sophisticated algorithms can lead to issues with overtraining, and there is a limit to how much improvement they can achieve in practice.

[29] suggests three theoretical reasons why an ensemble of classifiers may perform better than just one classifier alone:

[1] http://mi.eng.cam.ac.uk/research/projects/AGILE/
[2] http://chil.server.de/servlet/is/101/
[3] http://www.tc-star.org/

- Lack of training data
  - With limited training data, the estimated models will only be an approximation to the true underlying data distribution. The average of a number of different estimates may be closer to the true model than any of the individual estimates.
- Limitations of the optimisation algorithm
  - In practice, the optimisation algorithm is normally only able to find a local optimum for parameter estimates, except in very simple situations. Again, an average of multiple estimates may be closer to the global optimum than a single estimate.
- Representational issues
  - It may be the case that the model has limited capability and is unable to represent the true data distribution. A combination of multiple models may be able represent distributions which cannot be modelled by just one model.

For automatic speech recognition, these three factors are all limitations of the current approach. Large training databases are used for training complex models and optimum parameter estimation is an issue and an ongoing area of research. Also, as discussed in the following chapter, the HMM for speech recognition is not a correct model for speech and so does not accurately represent the underlying speech distribution. Thus, a combination of multiple speech recognisers may outperform a single speech recogniser.

This thesis presents two methods for building complementary systems for automatic speech recognition. The first method in chapter 6 alters the decision tree generation process and changes the parameter tying across HMM states. The algorithm biases the tree generation so states which are confusable are less likely to be clustered. By doing this, there is more information available to distinguish confusable states, though new errors may be introduced by clustering states which were not previously clustered. Thus, the new decision tree will make different errors to the original, and it is hoped they will be complementary. The parameter tying stage is chosen as the decision tree generation algorithm makes locally optimal decisions, and so small changes can yield very different decision trees.

The second method presented in this thesis for generating complementary systems focuses on explicitly altering the training algorithm to resolve errors. Currently, discriminative training focuses on portions of the training data where there are errors. Hence, it might be expected that discriminative training leads to complementary systems as it resolves errors made by the existing system. However, in doing this, the training algorithm is still required to keep a good representation of data which is well modelled, and so the ability to model poor data is limited. The algorithms presented in chapter 7 relax the requirementdevice that the training should keep a good representation of previously well modelled data. Now, the training algorithm may introduce new errors where there previously were none, but the training also aims to correct existing errors. Thus, the errors made by the two systems will be complementary. Chapter 7 presents modifications to the existing ML and discriminative training algorithms to achieve this.

Both approaches use a data weighting to reflect errors made by existing systems, to allow the training to focus on these errors. Chapter 5 presents the form of data weighting which is used for both the decision tree generation and the explicit complementary system training.

While the goal of ASR research is often to reduce the word error rate of the recogniser, a certain level of error may be acceptable in practice. For example, in a call centre application, the exact output from the recogniser is not important, provided the meaning can be extracted. In contrast, for automatic subtitling, errors are less tolerable. In this thesis, the goal is to reduce the word error rate of a combination of systems, rather than improve the individual system performance.

In this thesis, classifiers for both static and dynamic data are discussed. To avoid confusion between the two, only the former is referred to as a classifier. For classifying dynamic speech data, the HMM framework discussed in chapter 2 is used. A speech recognition system contains, among its components, a set of hidden Markov models representing acoustic units, typically phones. This distinction is made as techniques for generating and combining complementary classifiers for static data may not be applicable to dynamic data.

1.3 Thesis Organisation

This thesis is organised as follows. Chapter 2 introduces the standard HMM based approach to automatic speech recognition, and chapters 3 and 4 discuss existing methods for complementary system combination and generation, both for general applications and specifically for ASR. Chapter 5 discusses existing approaches to confidence scoring and data weighting, before presenting the form of data weighting which forms the basis of the two complementary system algorithms proposed in this thesis. Chapters 6 and 7 introduce the two new algorithms for building complementary systems. First, the directed decision tree (DDT) algorithm and a divergence measure for comparing decision trees are presented in chapter 6. Then, chapter 7 discusses modifications to ML and discriminative training for explicitly training complementary systems for ASR. Chapter 8 details the experimental setup for the tasks of broadcast news English, Mandarin and Arabic, which are used for experimental results. Chapters 9 and 10 present and discuss the experimental results obtained with the directed decision tree and the methods for explicitly training complementary systems. Next, chapter 11 discusses the problem of improved word error detection to improve the combination of complementary systems, while chapter 12 concludes and suggests directions for further work.

CHAPTER 2

HMM-based Statistical Speech Recognition

This chapter presents the theory behind a hidden Markov model (HMM) based automatic speech recognition system. An overview of automatic speech recognition as a statistical pattern recognition task is first given, followed by a discussion of the main elements of a large vocabulary speech recognition system including frontend processing, acoustic modelling, parameter estimation, language modelling, speaker adaptation, and decoding.

2.1 Statistical Framework for ASR

Speech recognition is a classic pattern recognition task where, given a speech signal, the task is to identify the most likely word sequence uttered. Speech is a dynamic signal; utterances can differ substantially in length, even if the same words are uttered by the same person. There are also a large number of potential word sequences, or hypotheses, which the system must search over to find the most likely. These properties make automatic speech recognition a complex task.

Formally, the task of a speech recognition system is to find the most probable hypothesis, $\hat{\mathcal{H}}$, given the uttered speech signal, $\mathcal{O}$

$$\hat{\mathcal{H}} = \underset{\mathcal{H}}{\operatorname{argmax}}\, P(\mathcal{H}|\mathcal{O}) \tag{2.1}$$

Using Bayes' rule, this can be formulated as

$$\begin{aligned} \hat{\mathcal{H}} &= \underset{\mathcal{H}}{\operatorname{argmax}} \frac{p(\mathcal{O}|\mathcal{H})P(\mathcal{H})}{p(\mathcal{O})} \\ &= \underset{\mathcal{H}}{\operatorname{argmax}}\, p(\mathcal{O}|\mathcal{H})P(\mathcal{H}) \end{aligned} \tag{2.2}$$

The denominator term $p(\mathcal{O})$ is dropped from the maximisation as it is not dependent on $\mathcal{H}$. In this framework, $p(\mathcal{O}|\mathcal{H})$ is obtained from the acoustic model and $P(\mathcal{H})$ from the language model. Figure 2.1 shows the typical components of a speech recognition system. The input speech is first converted to a series of feature vectors, or observations, $\mathcal{O} = \{\boldsymbol{o}_1 \cdots \boldsymbol{o}_t\}$, in a stage known as *frontend processing* or *feature extraction*. The features are then passed to the decoder, which searches for the most likely hypothesis, making use of the language and acoustic models. Optionally, speaker or environmental adaptation of the models can be performed which, for large vocabulary recognition, might make use of an initial output transcription. Acoustic models are trained at the sub-word level, typically at the phone level, and the dictionary contains a mapping from words to sub-word units. The system might output a single best hypothesis, a list of the N best hypotheses, or another representation of likely hypotheses such as a word lattice [121].

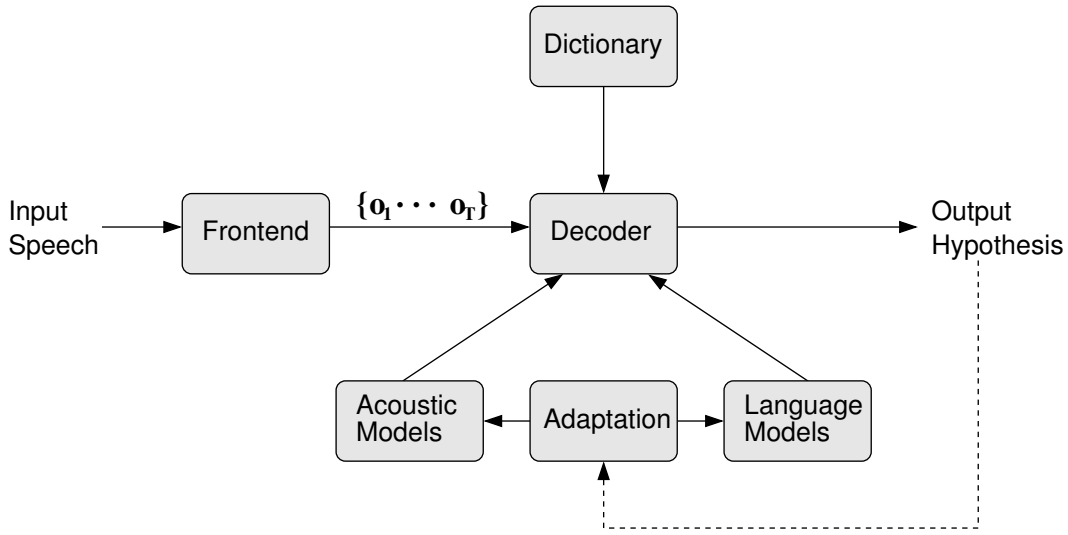


Figure 2.1: *Speech Recognition System Architecture*

The recogniser can only handle words for which a pronunciation exists, i.e. those words in the dictionary. A word to be recognised which is not in the dictionary is labelled *out of vocabulary (OOV)*. However, as the size of the dictionary increases, so does the complexity of the search, and so tasks are often categorised as small, medium or large vocabulary, depending on the size of the dictionary.

Both the acoustic and language models are trained on labelled training sets, and the system is evaluated on an independent test set. The aim in training is to model the underlying speech process such that the models generalise well to the unseen test data. A model is *overtrained* when it is overly biased towards the training set and fails to generalise to the test set. This can occur, for example, when the models contain too many parameters, or too many iterations of

training are performed. The converse, *undertraining*, occurs when the models are not complex enough to correctly model the speech, or have not been adequately trained.

This thesis concentrates on the acoustic modelling aspect of automatic speech recognition, particularly for training and decoding with multiple complementary acoustic models.

2.2 Frontend Processing

The first stage in a speech recognition system is to convert the input, a continuous speech signal, into an appropriate form that can be used for recognition. First, a representation of the speech signal is extracted as a series of feature vectors, $\mathcal{O} = \{\boldsymbol{o}_1 \cdots \boldsymbol{o}_t\}$. The extracted feature vectors should be compact and discriminatory. That is, they should contain all the information necessary for distinguishing words, and suppress the irrelevant information, while remaining small in dimension. The next step is to segment the signal into individual utterances, and often non-speech segments such as music, commercials and silence are removed. Then, optional speaker clustering and gender identification can be performed.

For this purpose, it is assumed that speech can be partitioned into a series of short quasi-stationary frames, and a single feature vector is extracted from each frame. Frames of data are overlapped, with a typical window size of 25ms and a frame shift of 10ms [163]. A Hamming filter is applied to each windowed frame of data to suppress discontinuities at the edge, before an FFT is performed to obtain the short-term spectrum. A pre-emphasis filter is normally applied to the short term spectrum to boost the energy at higher frequencies.

From this short-term spectrum, two popular representations for the speech signal can be extracted. These are mel frequency cepstral coefficients (MFCC) [27] and perceptual linear prediction (PLP) coefficients [66], both described below.

2.2.1 Feature Extraction

Figure 2.2 shows the process of obtaining MFCC [27]. These make use of the mel-scale, given by

$$f_{\mathsf{mel}} = 1127 \log\left(1 + \frac{f_{\mathsf{Hz}}}{700}\right) \tag{2.3}$$

This scale takes account of the fact that as frequency rises, the *perceived* pitch of the speech increases linearly at low frequencies but logarithmically at high frequencies. Filterbank coefficients are obtained by filtering the short-term spectrum with a series of N triangular band-pass filters, spaced according to the mel-scale. The logarithm of the amplitudes of each of the filters gives a vector of filterbank coefficients. These filterbank coefficients are highly correlated, and so a discrete cosine transform (DCT) is applied to transform the filterbank coefficients to MFCC

$$o_{td} = \sqrt{\frac{2}{N}} \sum_{i=1}^{N} \log(x_{tf_d}) \cos\left(\frac{\pi d}{N}(i - 0.5)\right) \tag{2.4}$$

where x_{tf_d} is the amplitude of filterbank d at time t, and N is the number of filterbank coefficients. o_{td} is the d^{th} cepstral coefficient at time t. Usually, the lower 12 cepstral coefficients

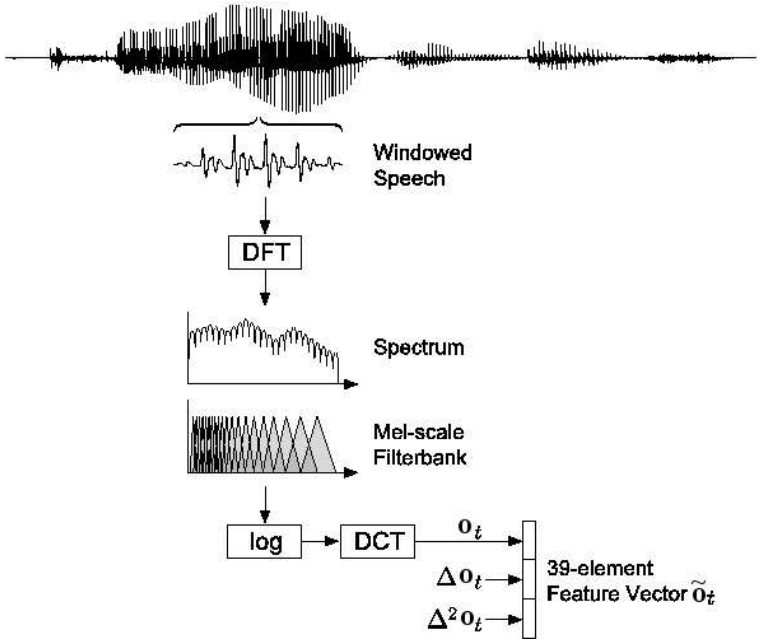


Figure 2.2: *MFCC extraction*

are used as the higher frequencies don't contain much useful information. The zeroth cepstral coefficient, or the log energy, is also incorporated to give a 13 dimensional feature vector.

Perceptual linear prediction (PLP) coefficients [66] make use of critical band filters, equal-loudness pre-emphasis, intensity-loudness warping and linear prediction. The frequency is warped according to the Bark scale

$$f_{\mathsf{bark}} = \log\left[\left(1 + \left(\frac{f_{\mathsf{Hz}}}{600}\right)^2\right)^{\frac{1}{2}} + \frac{f_{\mathsf{Hz}}}{600}\right] \tag{2.5}$$

In the HMM framework, observations are modelled as conditionally independent although there is some temporal correlation between observations. To better model this, it is desirable to incorporate information about the correlation between frames. This is often done by using dynamic coefficients [39]. The first-order dynamic, or delta, coefficients $\Delta\boldsymbol{o}_t$ are calculated using

$$\Delta\boldsymbol{o}_t = \frac{\sum_{\delta=1}^{\Delta}\delta(\boldsymbol{o}_{t+\delta} - \boldsymbol{o}_{t-\delta})}{2\sum_{\delta=1}^{\Delta}\delta^2} \tag{2.6}$$

Second order delta parameters are obtained in same way, replacing static by delta parameters. The final feature vector, $\tilde{\boldsymbol{o}}_t$ is then a concatenation of the static and dynamic parameters

$$\tilde{\boldsymbol{o}}_t = \begin{bmatrix} \boldsymbol{o}_t \\ \Delta\boldsymbol{o}_t \\ \Delta^2\boldsymbol{o}_t \end{bmatrix} \tag{2.7}$$

For a vector with 13 static features plus the first and second order deltas, the complete feature vector is 39 dimensional.

2.2.2 Feature Normalisation and Transformations

The MFCC and PLP features are speaker and environment dependent and, despite the discrete cosine transform, there are still some correlations between features [99]. A number of approaches have been proposed to address these issues.

One approach to speaker normalisation, vocal tract length normalisation (VTLN) [94] is commonly used to compensate for the fact that speakers have different vocal tract lengths. Formant frequencies in the short-term spectrum are shifted to account for the differing vocal tract lengths. VTLN is implemented as a linear warping of the frequency axis, where the warping factor is found empirically. The normalisation techniques cepstral mean and variance normalisation and Gaussianisation, along with feature transformations PCA, LDA and HLDA, are discussed in detail below.

2.2.2.1 Cepstral Mean and Variance Normalisation

Cepstral mean normalisation (CMN), or cepstral mean subtraction, converts the observed features so they have a mean of zero, which removes the bias that arises from fixed convolutional noise. This is done by subtracting the mean of the cepstral features from the observations to obtain a new feature vector $\tilde{\boldsymbol{o}}_t$

$$\tilde{\boldsymbol{o}}_t = \boldsymbol{o}_t - \frac{1}{T}\sum_{\tau=1}^{T} \boldsymbol{o}_\tau \tag{2.8}$$

To perform the normalisation, the average value of $\boldsymbol{o}_t$ needs to be calculated over T frames. For offline use it is trivial to calculate these using all available data. For online use however, it may be necessary to average over a smaller number of frames, perhaps just one utterance, otherwise the normalisation is delayed. Similarly, cepstral variance normalisation transforms the features so they have unity variance. Cepstral mean and variance normalisation are inexpensive to apply in practice, and so are commonly used in ASR.

2.2.2.2 Gaussianisation

CMN and CVN normalise the mean and variance of the date, i.e. the first and second order moments. Gaussianisation [23, 132] normalises the higher order moments transforming $\boldsymbol{o}_t$ via a non-linear function $\phi(.)$

$$\tilde{\boldsymbol{o}}_t = \phi(\boldsymbol{o}_t) \tag{2.9}$$

so $\tilde{\boldsymbol{o}}_t$, the transformed observation, is normally distributed with zero mean and identity variance

$$\tilde{\boldsymbol{o}}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{2.10}$$

This approach is motivated in part by the fact that the state output distributions for ASR are normally mixtures of Gaussians, and so are a better model of Gaussian input data.

To simplify the transform function estimation, it is assumed that the feature dimensions are independent. As some correlations between dimensions exist, the transformed feature vector is only approximately Gaussian. The transform is estimated to match the cumulative density function (CDF) of the actual data with that of a Gaussian distribution [132]. The input data CDF, $F(\boldsymbol{o}_t)$, can be approximated by a histogram [132] or a GMM [23]. The transformation for the d^{th} dimension of $\boldsymbol{o}$ is then

$$\tilde{o}_d = \Phi^{-1}\left(F(o_d)\right) \tag{2.11}$$

where Φ is the CDF of a Gaussian. Gaussianisation has successfully been used in large vocabulary recognition [98]. As it is assumed the feature dimensions are independent, it may be useful to use one of the feature transforms discussed next in section 2.2.2.3 to decorrelate the features before Gaussianisation is applied.

2.2.2.3 Feature Transforms

CMN, CVN, and Gaussianisation are feature normalisation techniques. An alternative class of algorithms transform the feature vector into an uncorrelated subspace to remove correlations while retaining discriminatory features. The linear transforms discussed in this section project the observation vector using a transform $\boldsymbol{A}$ to obtain a new observation, $\tilde{\boldsymbol{o}}_t$, given by

$$\tilde{\boldsymbol{o}}_t = \boldsymbol{A}\boldsymbol{o}_t \tag{2.12}$$

The most straightforward linear transform is Principal Component Analysis (PCA) [11], which decorrelates the global data covariance matrix using an Eigen-decomposition, $\boldsymbol{\Sigma}_g = \boldsymbol{A}^{\mathsf{T}}\boldsymbol{\Lambda}\boldsymbol{A}$, where $\boldsymbol{\Sigma}_g$ is the covariance of all observations $\{\boldsymbol{o}_1 \cdots \boldsymbol{o}_T\}$. This assumes that dimensions with large variance are more important, and hence PCA is not robust to scaling of feature dimensions.

PCA is an unsupervised technique, as it doesn't use class labels. Two further linear transforms, LDA and HLDA, are supervised and assume each Gaussian component in the HMM set is a separate 'class' to be discriminated.

Linear Discriminant Analysis (LDA) is a linear transform applied to the features which aims to project the features into an uncorrelated feature space while retaining discriminatory information. It minimises the within-class covariance $\boldsymbol{W}$ while maximising the between-class variance $\boldsymbol{B}$. The transform $\boldsymbol{A}$ is estimated to maximise

$$\hat{\boldsymbol{A}}_{LDA} = \underset{\boldsymbol{A}}{\operatorname{argmax}} \left\{ \frac{|\operatorname{diag}(\boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^{\mathsf{T}})|}{|\operatorname{diag}(\boldsymbol{A}\boldsymbol{W}\boldsymbol{A}^{\mathsf{T}})|} \right\} \tag{2.13}$$

This can be optimised using an Eigen-decomposition of the matrix $\boldsymbol{W}^{-1}\boldsymbol{B}$ where $\boldsymbol{B}$ is the between-class covariance and $\boldsymbol{W}$ is the average of the component variances, or within class covariance. LDA is commonly used in large vocabulary systems [100, 140].

HLDA [89] is an extension to LDA for heteroscedastic data, which relaxes the assumption of LDA that classes should have the same covariance. A global HLDA transform which does

not reduce the feature dimension is equivalent to a global semi-tied transform [43]. As for LDA, the HLDA transform is applied as in equation 2.12 while the parameters are estimated in an ML fashion [43], according to

$$\hat{\boldsymbol{A}}_{HLDA} = \underset{\boldsymbol{A}}{\operatorname{argmax}} \left\{ \sum_{t=1}^{T} \sum_{j=1}^{J} \gamma_j(t) \left(\log |\boldsymbol{A}|^2 - \log |\tilde{\boldsymbol{\Sigma}}_j| \right) \right\} \tag{2.14}$$

where $\tilde{\boldsymbol{\Sigma}}_j$ is the transformed state variance in the feature space defined by the transform $\boldsymbol{A}$ and $\gamma_j(t)$ is the state posterior, calculated by the forward-backward algorithm in section 2.3.2 below. HLDA has been successfully used in LVCSR systems [48, 62].

2.3 Hidden Markov Models

The Hidden Markov Model (HMM) was first introduced in the 1970s as a robust statistical model for time varying signals [8], and has since formed the basis of many ASR systems. An HMM is a finite state machine, where the entry to each state has an associated output distribution, $b(\boldsymbol{o})$. Entering a particular state, the HMM outputs an *observation* according to that state's output distribution, before transitioning to the next state according to a transition probability a. The underlying state sequence is hidden and only the output observations are observed. The HMM makes the following two assumptions:

- **Conditional independence:** an observation is conditionally independent of all other observations, given the state which generated it
- **First-order Markov assumption:** the probability of transitioning to a particular state is dependent only on the previous state

Although these assumptions are incorrect for speech, and so HMMs aren't a correct model for speech, they have proven the most successful model in practice.

2.3.1 HMM Topology for ASR

Typically, for ASR, each phone or other sub-word unit is modelled by a left-to-right 5 state HMM, as in figure 2.3. Three of the states are emitting, while the start and end states are not; this is to allow for easy model concatenation for continuous speech recognition. θ_j represents the j^{th} state of the HMM, and a hidden variable ψ_t is introduced to represent the current state of the HMM at time t. In figure 2.3, the hidden state sequence through the HMM which generates the observation sequence $\{\boldsymbol{o}_1 \cdots \boldsymbol{o}_6\}$ is $\{\theta_1, \theta_2, \theta_2, \theta_3, \theta_3, \theta_3, \theta_4, \theta_5\}$.

The dynamic Bayesian network (DBN) representation in figure 2.4 makes the two independence assumptions clear. The left-to-right arrows model the first-order Markov assumption, while the explicit dependence of the observation on just the current state is shown by the vertical arrows. The state sequence is hidden and discrete, while the observations are continuous and observed.

Each state output distribution $b_j(\boldsymbol{o})$ may take any form, and can be discrete or continuous. For continuous speech recognition, continuous state output distributions are normally used.

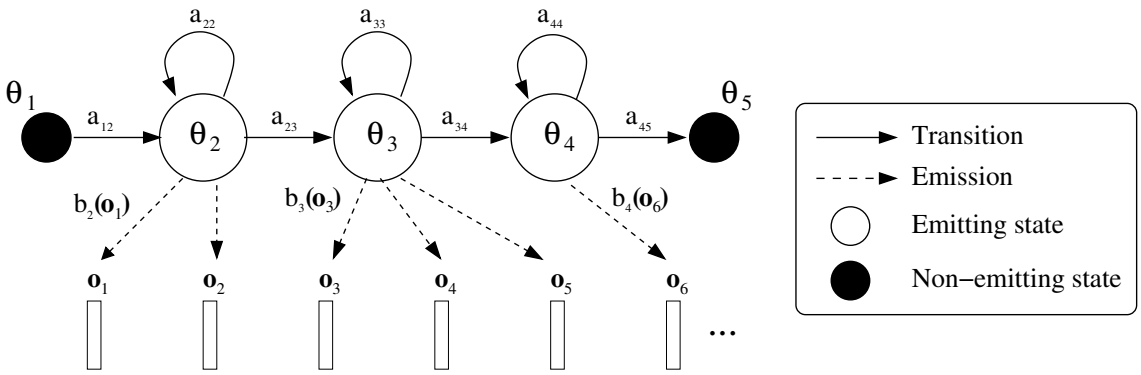


Figure 2.3: *5-state HMM with 3 emitting states*

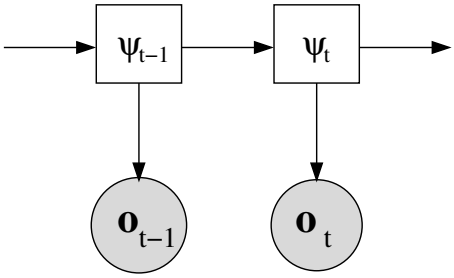


Figure 2.4: *HMM in figure 2.3 as a Dynamic Bayesian Network. The state sequence is hidden and observations are observed variables. The state sequence is discrete, and the output observations continuous.*

The most common form of output distribution is the Gaussian mixture model (GMM), given by

$$b_j(\boldsymbol{o}_t) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(\boldsymbol{o}_t; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}) \tag{2.15}$$

where $\mathcal{N}(.)$ is a Gaussian distribution

$$\mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{o}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{o}-\boldsymbol{\mu})\right) \tag{2.16}$$

The normalisation term is independent of $\boldsymbol{o}$ and can be precomputed and cached for efficiency. Other forms of output distribution have been used, for example mixtures of Laplacian distributions [61]. It is not necessary for all states to have the same form of output distribution, and in the particular case of GMM output distributions, it is not necessary for each state to have the same number of Gaussian components.

The parameters of the model, $\mathcal{M}$, are those of the state output distributions and the transition probabilities. For a GMM output distribution, the parameters are the means $\boldsymbol{\mu}_{jm}$, variances $\boldsymbol{\Sigma}_{jm}$ and component priors c_{jm} where j is the HMM state index and m is the GMM component. There is a sum-to-one constraint on the transition probabilities from each state and on the component priors, so $\sum_j a_{ij} = 1$ and $\sum_m c_{jm} = 1$.

For a d-dimensional feature vector, a full covariance matrix has $O(d^2)$ parameters. Due to the large number of states in an HMM system, the use of full covariance matrices is expensive in practice and requires a large amount of training data to robustly estimate the parameters. Hence, diagonal, or block-diagonal covariance matrices are often used. These can reduce the number of parameters to $O(d)$ but no longer model the correlations between features. Other techniques, such as semi-tied covariances [43] or SPAM (Subspace Precision and Mean) [5] can be used to reduce the number of parameters by sharing full covariance or precision matrices over a number of components. The former uses a component specific diagonal matrix and a shared full covariance, which may be global or tied over a number of components. The latter uses a global set of basis full covariance matrices, and a set of interpolation weights for each component.

2.3.2 Likelihood Calculation

The HMM is a generative model of speech, and, given a sequence of observations, can be used to obtain the acoustic likelihood score. The likelihood of a sequence of data, $\mathcal{O}$ given a transcription $\mathcal{H}$ is the sum of the probabilities for all possible state sequences $\boldsymbol{\Psi}$ through the transcription

$$p(\mathcal{O}|\mathcal{M}, \mathcal{H}) = \sum_{\boldsymbol{\psi}\in\boldsymbol{\Psi}} p(\mathcal{O}|\boldsymbol{\psi}, \mathcal{M}); \tag{2.17}$$

Applying the first order Markov and conditional independence assumptions of section 2.3 allows the likelihood to be approximated by

$$p(\mathcal{O}|\mathcal{M},\mathcal{H}) = \sum_{\boldsymbol{\psi}\in\boldsymbol{\Psi}} \prod_{t=1}^{T} P(\psi_t|\psi_{t-1}) p(\boldsymbol{o}_t|\psi_t); \tag{2.18}$$

It is infeasible to search explicitly over all possible state sequences and a recursive algorithm, the forward-backward algorithm [78], can instead be used. The *forward probability* $\alpha_j(t)$ is the sum of the likelihoods of all partial paths ending in state θ_j at time t, and can be calculated recursively

$$\begin{aligned} \alpha_j(t) &= p(\boldsymbol{o}_1 \cdots \boldsymbol{o}_t, \psi_t = \theta_j|\mathcal{M},\mathcal{H}) \\ &= \left[\sum_{i\leq j} \alpha_i(t-1) a_{ij}\right] b_j(\boldsymbol{o}_t) \end{aligned} \tag{2.19}$$

The $i \leq j$ condition enforces the strict left-to-right topology of the HMM. The forward probability is initialised at time $t = 0$ by

$$\begin{aligned} \alpha_1(0) &= 1 \\ \alpha_j(0) &= 0 \text{ for } j \neq 1 \end{aligned} \tag{2.20}$$

In the final state, N, at time T, the forward probability is

$$\alpha_N(T) = p(\mathcal{O}|\mathcal{M},\mathcal{H}) \tag{2.21}$$

Conversely, the *backward probability* $\beta_j(t)$ is the probability of seeing the observations $\boldsymbol{o}_{t+1} \cdots \boldsymbol{o}_T$, when in state θ_j at time t. That is, the sum of likelihoods of all partial paths beginning at state θ_j and ending in the final state of the HMM. $\beta_j(t)$ can also be defined recursively

$$\begin{aligned} \beta_j(t) &= p(\boldsymbol{o}_{t+1} \cdots \boldsymbol{o}_T|\psi_t = \theta_j,\mathcal{M},\mathcal{H}) \\ &= \left[\sum_{i\geq j} \beta_i(t+1) a_{ji} b_i(\boldsymbol{o}_{t+1})\right] \end{aligned} \tag{2.22}$$

The initial conditions in the final state are

$$\begin{aligned} \beta_N(T) &= 1 \\ \beta_j(T) &= a_{jN} \text{ for } j \neq N \end{aligned} \tag{2.23}$$

The algorithm terminates in the initial state of the HMM

$$\beta_1(0) = p(\mathcal{O}|\mathcal{M},\mathcal{H}) \tag{2.24}$$

The likelihood of being in a particular state at a particular time instance, $\gamma_j(t)$ can be expressed in terms of the forward and backward probabilities

$$\begin{aligned}\gamma_j(t) &= P(\psi_t = \theta_j|\mathcal{O},\mathcal{M},\mathcal{H}) \\ &= \frac{\alpha_j(t)\beta_j(t)}{p(\mathcal{O}|\mathcal{M},\mathcal{H})}\end{aligned} \tag{2.25}$$

The state posterior probabilities, $\gamma_j(t)$, form the basis of several algorithms for ASR, including the decision tree and HMM training algorithms below.

2.4 Training Hidden Markov Models

HMM training estimates a set of optimal model parameters $\hat{\mathcal{M}}$, with respect to the training data, $\mathcal{O}$, using some criterion, or objective function, $\mathcal{F}(\mathcal{M}|\mathcal{O})$

$$\hat{\mathcal{M}} = \underset{\mathcal{M}}{\text{argmax}}\, \mathcal{F}(\mathcal{M}|\mathcal{O}) \tag{2.26}$$

There are two common approaches to parameter estimation; maximum likelihood and discriminative training. Both methods are discussed below.

2.4.1 Maximum Likelihood

Maximum Likelihood (ML) training maximises the likelihood of the training data, given the reference transcription, i.e. $p(\mathcal{O}|\mathcal{H}_{ref},\mathcal{M})$, summed over the R training data utterances. Provided the likelihood never reaches zero, which is the case for Gaussian PDFs with non-zero variance, then the log likelihood can instead be maximised

$$\mathcal{F}_{ML}(\mathcal{M}) = \sum_{r=1}^{R} \log p(\mathcal{O}^{(r)}|\mathcal{H}_{ref}^{(r)},\mathcal{M}) \tag{2.27}$$

For clarity, the sum over all training data utterances is dropped. For the case of the HMM, the objective function may be expressed as a sum over all possible state sequences through the reference transcription $\boldsymbol{\Psi}_{ref}$

$$\mathcal{F}_{ML}(\mathcal{M}) = \left\{ \log \sum_{\boldsymbol{\psi}\in\boldsymbol{\Psi}_{ref}} p(\mathcal{O},\boldsymbol{\psi}|\mathcal{M}) \right\} \tag{2.28}$$

Direct optimisation of this equation is difficult, and so an auxiliary function $\mathcal{Q}$ is iteratively optimised using the Baum-Welch algorithm [8]. The auxiliary function is

$$\mathcal{Q}_{ML}(\hat{\mathcal{M}}^k,\hat{\mathcal{M}}^{k+1}) = \sum_{\boldsymbol{\psi}\in\boldsymbol{\Psi}_{ref}} P(\boldsymbol{\psi}|\mathcal{O},\hat{\mathcal{M}}^k,\mathcal{H}_{ref}) \log\left(p(\boldsymbol{\psi},\mathcal{O}|\hat{\mathcal{M}}^{k+1},\mathcal{H}_{ref})\right) \tag{2.29}$$

Finding a local maxima of the auxilary function, $\mathcal{Q}$, guarantees an increase in $\mathcal{F}_{ML}(\mathcal{M})$, and $\mathcal{Q}$ satisfies the inequality

$$\mathcal{F}_{ML}(\hat{\mathcal{M}}^{k+1}) - \mathcal{F}_{ML}(\hat{\mathcal{M}}^{k}) \geq \mathcal{Q}_{ML}(\hat{\mathcal{M}}^{k}, \hat{\mathcal{M}}^{k+1}) - \mathcal{Q}_{ML}(\hat{\mathcal{M}}^{k}, \hat{\mathcal{M}}^{k}) \tag{2.30}$$

where $\hat{\mathcal{M}}^k$ is the current parameter set at iteration k, and $\hat{\mathcal{M}}^{k+1}$ is the re-estimated set at iteration $k+1$. The Baum-Welch algorithm is a form of the Expectation-Maximisation (EM) algorithm [28]. It has two steps, and is described in figure 2.5. The first step, the E-step, computes the auxiliary function while the second M-step estimates the updated model parameters. The steps are alternated until convergence of the auxiliary function.

Initialise $\hat{\mathcal{M}}^0$
While $\mathcal{Q}(\hat{\mathcal{M}}^k, \hat{\mathcal{M}}^{k+1}) - \mathcal{Q}(\hat{\mathcal{M}}^k, \hat{\mathcal{M}}^k) >$ threshold
E-step: compute $\mathcal{Q}(\hat{\mathcal{M}}^k, \hat{\mathcal{M}}^{k+1})$
M-step: estimate $\hat{\mathcal{M}}^{k+1} = \text{argmax}_{\mathcal{M}}\, \mathcal{Q}(\hat{\mathcal{M}}^k, \mathcal{M})$
k=k+1
End

Figure 2.5: *The EM algorithm*

The values of $\gamma_{jm}(t)$ are calculated using the forward-backward algorithm in section 2.3.2, treating Gaussian mixture components as hidden variables. The probability of being in state θ_i at time t and θ_j at time $t+1$, $\zeta_{ij}(t)$, is also calculated from the forward and backward probabilities

$$\begin{aligned} \zeta_{ij}(t) &= P(\psi_t = \theta_i, \psi_{t+1} = \theta_j | \mathcal{O}, \mathcal{M}) \\ &= \frac{\alpha_i(t) a_{ij} b_j(\boldsymbol{o}_{t+1}) \beta_j(t+1)}{\alpha_N(T)} \end{aligned} \tag{2.31}$$

From these, equation 2.29 becomes

$$\mathcal{Q}_{ML}(\hat{\mathcal{M}}^k, \hat{\mathcal{M}}^{k+1}) = \sum_{t=1}^{T} \left\{ \sum_{j=1}^{N} \sum_{m=1}^{M} \gamma_{jm}^k(t) \left[\log(c_{jm}^{k+1}) + \log \mathcal{N}(\boldsymbol{o}_t; \boldsymbol{\mu}_{jm}^{k+1}, \boldsymbol{\Sigma}_{jm}^{k+1}) \right] + \sum_{i=1}^{N} \sum_{j=1}^{N} \zeta_{ij}(t) \log a_{ij}^{k+1} \right\} \tag{2.32}$$

where the component priors, c_{jm}^k and transition probabilities, a_{ij}^k, should sum to one

$$\sum_{m=1}^{M} c_{jm}^k = 1 \;;\; \sum_{j=1}^{N} a_{ij}^k = 1 \tag{2.33}$$

Equating to zero and solving, the ML estimates for the transition probabilities $\hat{a}_{ij}$ and the GMM means $\hat{\boldsymbol{\mu}}_{jm}$, variances $\hat{\boldsymbol{\Sigma}}_{jm}$ and priors $\hat{c}_{jm}$ are

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} \zeta_{ij}(t)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \zeta_{ij}(t)} \tag{2.34}$$

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\sum_{t=1}^{T} \gamma_{jm}(t) \boldsymbol{o}_t}{\sum_{t=1}^{T} \gamma_{jm}(t)} \tag{2.35}$$

$$\hat{\boldsymbol{\Sigma}}_{jm} = \frac{\sum_{t=1}^{T} \gamma_{jm}(t) (\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_{jm})(\boldsymbol{o}_t - \hat{\boldsymbol{\mu}}_{jm})^{\mathsf{T}}}{\sum_{t=1}^{T} \gamma_{jm}(t)} \tag{2.36}$$

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T} \gamma_{jm}(t)}{\sum_{m=1}^{M} \sum_{t=1}^{T} \gamma_{jm}(t)} \tag{2.37}$$

In the Baum-Welch algorithm, the E-step of the EM algorithm is to compute the component occupation counts $\gamma_{jm}(t)$ and transition probabilities $\zeta_{ij}(t)$, and the M-step is to apply the update formulae 2.34-2.37. These two steps are repeated for a number of iterations until convergence although, in general, the EM algorithm only finds a local, and not a global, optimum.

2.4.2 Discriminative Training

ML training is optimal if the underlying acoustic models are correct, and if there is an infinite amount of training data [118]. Neither of these assumptions are correct for automatic speech recognition. HMMs aren't a correct model for speech, due to the independence assumptions stated in section 2.3, and the amount of training data is limited in practice. Additionally, the EM algorithm for ML training does not guarantee to find a global optimum for the model parameters. Discriminative training addresses these issues by explicitly optimising the model parameters with respect to the expected error rate, and allows the objective function in training to be matched with the evaluation criterion.

2.4.2.1 Maximum Mutual Information

Maximum Mutual Information (MMI) Estimation [6, 118, 149] maximises the posterior probability of the correct hypothesis, given a fixed language model. This is equivalent to maximising the mutual information between the models and the acoustic observation sequence. The MMI objective function for R training utterances is

$$\begin{aligned}
\mathcal{F}_{MMI}(\mathcal{M}) &= \sum_{r=1}^{R} \log P(\mathcal{H}_{ref}^{(r)} | \mathcal{O}^{(r)}, \mathcal{M}) \\
&= \sum_{r=1}^{R} \log \frac{p(\mathcal{O}^{(r)} | \mathcal{H}_{ref}^{(r)}, \mathcal{M}) P(\mathcal{H}_{ref}^{(r)} | \mathcal{M})}{\sum_{\mathcal{H}} p(\mathcal{O}^{(r)} | \mathcal{H}, \mathcal{M}) P(\mathcal{H} | \mathcal{M})} \\
&= \sum_{r=1}^{R} \left\{ \log p(\mathcal{O}^{(r)} | \mathcal{H}_{ref}^{(r)}, \mathcal{M}) P(\mathcal{H}_{ref}^{(r)} | \mathcal{M}) - \log \sum_{\mathcal{H}} p(\mathcal{O}^{(r)} | \mathcal{H}, \mathcal{M}) P(\mathcal{H} | \mathcal{M}) \right\}
\end{aligned} \tag{2.38}$$

where the sum in the denominator is over all possible word sequences.

In equation 2.39, the first, or numerator, term is simply the ML objective function. The second, denominator, term is identical to the first, except it includes a sum over all possible hypotheses. Auxiliary functions can be derived for both the numerator and denominator terms, $\mathcal{Q}_{ML}(\hat{\mathcal{M}}^k, \hat{\mathcal{M}}^{k+1})$ and $\mathcal{Q}_{den}(\hat{\mathcal{M}}^k, \hat{\mathcal{M}}^{k+1})$, where the only difference is in the transcription used to obtain the statistics. The possible competing hypotheses in the denominator term can be represented by an N-best list, or a word lattice. Due to the negation in the objective function, the extended Baum-Welch (EBW) algorithm discussed in section 2.4.2.4 is used for optimisation.

2.4.2.2 Minimum Bayes' Risk Training

MMI training assumes that all sentence level misclassifications have equal weight. That is, an utterance is either correct or incorrect. In practice, this is not ideal as some misclassifications can be more important than others. Minimum Bayes' risk (MBR) training allows for a more informative definition of utterance correctness. $\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref})$ is a loss function which defines the cost incurred when a system hypothesises the transcription $\mathcal{H}$ and the reference transcription is $\mathcal{H}_{ref}$. MBR training minimises the risk $\mathcal{R}$, or expected loss, of a classifier, and is approximated by the empirical loss

$$\begin{aligned} \mathcal{R} &= E\left[\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref})\right] \\ &= \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M})\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}) \end{aligned} \tag{2.39}$$

The model parameters are estimated to minimise the risk over the R utterances in the training set, and the objective function is

$$\mathcal{F}_{MBR}(\mathcal{M}) = \sum_{r=1}^{R} \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}^{(r)}, \mathcal{M})\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}^{(r)}) \tag{2.40}$$

For clarity, the sum over all training set utterances is dropped below.

Minimum Bayes' risk training is a general framework which encompasses many popular discriminative criteria via the loss function. An utterance level loss function [113] is straightforward to implement

$$\mathcal{L}_{1/0}(\mathcal{H}, \mathcal{H}_{ref}) = \begin{cases} 0 & \mathcal{H} = \mathcal{H}_{ref} \\ 1 & \mathcal{H} \neq \mathcal{H}_{ref} \end{cases} \tag{2.41}$$

For a single utterance, it can be seen that this loss function is closely related to MMI training above as it effectively maximises the posterior of the reference

$$\begin{aligned} \mathcal{F}_{MBR-1/0}(\mathcal{M}) &= \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M})\mathcal{L}_{1/0}(\mathcal{H}, \mathcal{H}_{ref}) \\ &= \sum_{\mathcal{H} \neq \mathcal{H}_{ref}} P(\mathcal{H}|\mathcal{O}, \mathcal{M}) \\ &= 1 - P(\mathcal{H}_{ref}|\mathcal{O}, \mathcal{M}) \end{aligned} \tag{2.42}$$

Alternatively, the Levenshtein distance [32] can be used. This matches the training and evaluation criteria, and the loss becomes a sum of word-level errors. Suppose the hypotheses $\mathcal{H}$ and $\mathcal{H}_{ref}$ are aligned into a set of word pairs $\{\mathcal{W}^{(k)}, \mathcal{W}_{ref}^{(k)}\}$, as discussed in section 2.7, then the sentence level loss is given by

$$\mathcal{L}_{lev}(\mathcal{H}, \mathcal{H}_{ref}) = \sum_{k=1}^{K} l_{lev}(\mathcal{W}^{(k)}, \mathcal{W}_{ref}^{(k)}) \tag{2.43}$$

The word level loss function for the Levenshtein distance, $l_{lev}(\mathcal{W}, \mathcal{W}_{ref})$, is given by

$$l_{lev}(\mathcal{W}, \mathcal{W}_{ref}) = \begin{cases} 0 & \mathcal{W} = \mathcal{W}_{ref} \\ 1 & \mathcal{W} \neq \mathcal{W}_{ref} \text{ i.e. insertion, deletion or substitution} \end{cases} \tag{2.44}$$

Another popular criterion, minimum phone error (MPE) training [122] fits within the MBR framework. Now the hypotheses $\mathcal{H}$ and $\mathcal{H}_{ref}$ are aligned into a set of phone pairs $\{\mathcal{P}^{(k)}, \mathcal{P}_{ref}^{(k)}\}$. The loss function is then calculated at the phone level

$$\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}) = \sum_{k=1}^{K} \min_{\mathcal{P}_{ref}} l(\mathcal{P}, \mathcal{P}_{ref}) \tag{2.45}$$

There may be multiple phone level pronunciations for the reference hypothesis, and the reference phone $\mathcal{P}_{ref}$ is chosen to minimise $l(\mathcal{P}, \mathcal{P}_{ref})$. This is known as *multiple pronunciation* MPE training. An alternative is to take the best single pronunciation of the reference given the current model parameters, and use this one pronunciation as the reference. This is known as *single pronunciation* MPE training. These two forms of training lead to systems which perform similarly, but give improvements upon combination [49], and thus can be used in addition to some other complementary training approaches to incorporate extra diversity.

Other discriminative criteria fall within the MBR framework. These include some variations on Minimum Classification Error (MCE) [101], Minimum Word Error (MWE) [65], and other discriminative criteria [53, 123].

MPE, and other discriminative criteria, have been optimised using the EBW algorithm discussed in section 2.4.2.4. The MPE criterion has also be used successfully to discriminatively train a feature transform [125] and hence obtain a set of discriminative features. This is known as fMPE.

2.4.2.3 Implementation Detail

Discriminative criteria typically involve a sum over all possible hypotheses, and so, in practice, some approximations are needed to allow their implementation. As mentioned above for the MMI objective function, it is impractical to consider a loss calculated over all competing hypotheses. Thus the objective function is normally restricted to consider just the most likely competing hypotheses, i.e. those in the evidence space $\mathcal{H}_e$. The objective function becomes

$$\mathcal{F}_{MBR}(\mathcal{M}) = \sum_{\mathcal{H} \in \mathcal{H}_e} P(\mathcal{H}|\mathcal{O}, \mathcal{M}) \mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}) \tag{2.46}$$

Care must be taken to ensure that $\mathcal{H}_e$ is a good representation of the most likely competing hypotheses so that the loss function can be calculated accurately. N-best lists [79], lattices [159] or pinched lattices [32] offer a convenient representation. These are discussed in more detail below, in section 2.7.

Secondly, the HTK implementation of MPE training [163] optimises the expected phone accuracy of the training data [122]. Thus the loss $\mathcal{L}$ is replaced with an accuracy function $\mathcal{A}$, which is a sum of phone accuracies

$$\mathcal{A}(\mathcal{H}, \mathcal{H}_{ref}) = \sum_{k=1}^{K} \max_{\mathcal{P}_{ref}} a(\mathcal{P}^{(k)}, \mathcal{P}_{ref}^{(k)}) \tag{2.47}$$

There may be multiple phone level pronunciations for the reference hypothesis, and the reference phone $\mathcal{P}_{ref}$ is chosen to maximise $\mathcal{A}$. The phone-level accuracy, $a(\mathcal{P}, \mathcal{P}_{ref})$ is given by

$$a(\mathcal{P}, \mathcal{P}_{ref}) = \begin{cases} 1 & \mathcal{P} = \mathcal{P}_{ref} \\ 0 & \mathcal{P} \neq \mathcal{P}_{ref} \\ -1 & \mathcal{P} \text{ is an insertion} \end{cases} \tag{2.48}$$

The third issue that occurs in practice is alignment of the multiple competing hypotheses in the evidence space to facilitate the loss calculation. With the exception of the 1/0 sentence error loss function, an alignment of words or phones to the reference is needed for every competing hypothesis in the evidence space. This is feasible for an N-best list, but becomes computationally expensive when the evidence space is represented by a lattice, as the number of competing hypotheses is large. Hence, other methods have been proposed to estimate the loss.

For MPE training, each arc in the lattice is marked with phone-level time stamps, and the following heuristic is used to estimate phone accuracy

$$a(\mathcal{P}, \mathcal{P}_{ref}) = \begin{cases} -1 + 2e(\mathcal{P}, \mathcal{P}_{ref}) \text{ if } \mathcal{P} = \mathcal{P}_{ref} \\ -1 + e(\mathcal{P}, \mathcal{P}_{ref}) \text{ if } \mathcal{P} \neq \mathcal{P}_{ref} \end{cases} \tag{2.49}$$

where $e(\mathcal{P}, \mathcal{P}_{ref})$ is the overlap between the two phones. From this approximation, a loss for each phone in the lattice can be found without explicitly having to perform the alignment.

An alternative approach is to restrict the evidence space to make the alignment simpler. For example, pinched lattices [32] and confusion networks [105] are more compact representations of multiple hypotheses are discussed in more detail in section 2.7.

2.4.2.4 Optimisation of Discriminative Criteria

The EM algorithm for optimising the ML criterion uses a *strong-sense* auxiliary function [122], $\mathcal{Q}_{ML}$, which has the same gradient as the original ML criterion around the current parameter estimates, and maximising the auxiliary function guarantees not to decrease the original criterion. However, in this thesis, a a *weak-sense* auxiliary function is optimised for discriminative training. A weak-sense auxiliary function relaxes the requirement of a strong-sense function, and only shares the same gradient as the original criterion around the current

parameter estimates. It does not guarantee to increase the original criterion [122]. For MMI training, a weak-sense auxiliary function is

$$\mathcal{Q}_{MMI}(\mathcal{M}^k, \mathcal{M}^{k+1}) = \mathcal{Q}_{num}(\mathcal{M}^k, \mathcal{M}^{k+1}) - \mathcal{Q}_{den}(\mathcal{M}^k, \mathcal{M}^{k+1}) + \mathcal{Q}_{sm}(\mathcal{M}^k, \mathcal{M}^{k+1}) \quad (2.50)$$

where $\mathcal{Q}$ is the ML auxiliary function from section 2.4.1. The three terms correspond to the numerator, $\mathcal{Q}_{num}$, and denominator, $\mathcal{Q}_{den}$, terms of the MMI objective function, and a smoothing term, $\mathcal{Q}_{sm}$, to ensure convergence. Optimisation of this auxiliary function is performed [6, 118, 122]. The statistics for the denominator term are calculated using all possible transcriptions, while the numerator statistics use just the reference transcription, so for the j^{th} state

$$\begin{aligned} \gamma_j^{num}(t) &= P(\psi_t = \theta_j | \mathcal{O}, \mathcal{M}, \mathcal{H}_{ref}) \\ \gamma_j^{den}(t) &= \sum_{\mathcal{H}} P(\psi_t = \theta_j | \mathcal{O}, \mathcal{M}, \mathcal{H}) \end{aligned} \quad (2.51)$$

The required numerator statistics for optimisation are

$$\begin{aligned} \Gamma_j^{num} &= \sum_{t=1}^{T} \gamma_j^{num}(t) \\ \Gamma_j^{num}(\mathcal{O}) &= \sum_{t=1}^{T} \gamma_j^{num}(t) \boldsymbol{o}_t \\ \Gamma_j^{num}(\mathcal{O}^2) &= \sum_{t=1}^{T} \gamma_j^{num}(t) \boldsymbol{o}_t \boldsymbol{o}_t^{\mathsf{T}} \end{aligned} \quad (2.52)$$

and the denominator statistics are

$$\begin{aligned} \Gamma_j^{den} &= \sum_{t=1}^{T} \gamma_j^{den}(t) \\ \Gamma_j^{den}(\mathcal{O}) &= \sum_{t=1}^{T} \gamma_j^{den}(t) \boldsymbol{o}_t \\ \Gamma_j^{den}(\mathcal{O}^2) &= \sum_{t=1}^{T} \gamma_j^{den}(t) \boldsymbol{o}_t \boldsymbol{o}_t^{\mathsf{T}} \end{aligned} \quad (2.53)$$

The update equations for the mean and variance are then

$$\hat{\boldsymbol{\mu}}_j = \frac{\Gamma_j^{num}(\mathcal{O}) - \Gamma_j^{den}(\mathcal{O}) + D_j \boldsymbol{\mu}_j}{\Gamma_j^{num} - \Gamma_j^{den} + D_j} \quad (2.54)$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\Gamma_j^{num}(\mathcal{O}^2) - \Gamma_j^{den}(\mathcal{O}^2) + D_j(\boldsymbol{\mu}_j \boldsymbol{\mu}_j^{\mathsf{T}} + \boldsymbol{\Sigma}_j)}{\Gamma_j^{num} - \Gamma_j^{den} + D_j} - \boldsymbol{\mu}_j \boldsymbol{\mu}_j^{\mathsf{T}} \quad (2.55)$$

The constants D_j arise from the smoothing term in the auxiliary function, and in the update equations are set to be large to ensure convergence. Typically, they are set at a Gaussian specific level, and depend on the state occupation probability [159].

For MPE training, a different auxiliary function must be optimised. This has the same update, but the required statistics are different.

$$\gamma_j^{mpe} = \left. \frac{\partial \mathcal{F}_{MPE}(\mathcal{M})}{\partial \log p(\mathcal{O}|\mathcal{M})} \right|_{\mathcal{M}=\mathcal{M}^k} \tag{2.56}$$

This quantity is the gradient of the MPE criterion with respect to the log likelihood of the data and can be found from phone marked lattices. The accuracy of each phone arc $\mathcal{P}$ is first calculated, and then a forward-backward pass is used to calculate the average phone accuracy of every sentence passing through a phone arc, $\mathcal{F}_{MPE}(\mathcal{M}|\mathcal{P})$ [122]. The MPE statistics can be rewritten as

$$\gamma_j^{mpe} = p(\mathcal{O}|\mathcal{P}) \left[a(\mathcal{P}, \mathcal{P}_{ref}) - \mathcal{F}_{MPE}(\mathcal{M}|\mathcal{P}) \right] \tag{2.57}$$

This form of the statistics allows MPE training to be carried out in the same framework as MMI training, where $\gamma_j^{mpe} = \gamma_j^{num} - \gamma_j^{den}$. MBR training with any loss function can also be implemented, by replacing the phone accuracy function with an appropriate loss.

2.4.2.5 Smoothing

One problem with discriminative training is the tendency to overtrain, and thus generalise poorly to unseen data. For this reason, smoothing to a prior model is often performed as part of the training, to prevent overtraining [124]. Smoothing with a static prior interpolates the parameter estimates with the parameters of a fixed, well-trained model. Smoothing with a dynamic prior interpolates the discriminative parameter estimates with a second estimate of model parameters. For MPE training, an ML or an MMI prior can be used to obtain the second set of estimates.

For the mean update, the discriminative estimate of the parameter is $\hat{\boldsymbol{\mu}}_j$, the smoothed estimate is $\tilde{\boldsymbol{\mu}}_j$, and the smoothing can be described as

$$\tilde{\boldsymbol{\mu}}_j = \frac{\gamma_j \hat{\boldsymbol{\mu}}_j + \tau \gamma_j^{pr} \boldsymbol{\mu}_j^{pr}}{\gamma_j + \tau \gamma_j^{pr}} \tag{2.58}$$

where γ_j is effective the state occupation count from the forward-backward algorithm

$$\gamma_j = \sum_{t=1}^{T} \gamma_j(t) \tag{2.59}$$

and $\boldsymbol{\mu}_j^{pr}$ and γ_j^{pr} are the prior mean estimate and occupation count respectively. In the case of a static prior, γ_j^{pr} is set to an arbitrary value. In the case of a dynamic prior, they are obtained from the second model parameter estimates.

A value τ controls the degree of smoothing in both the static and dynamic prior cases. When $\tau = 0$, there is no contribution from the prior model. As $\tau \to \infty$, the final parameter estimates approach those of the prior. Thus, a higher value of smoothing constrains the parameter estimates to be close to the prior.

2.4.3 Active and Unsupervised Training

As speech recognition systems increase in complexity, there is a need to use more training data to obtain good models. However, a large manual effort is needed to prepare and accurately transcribe a speech database, and it becomes difficult to guarantee that the training set is correctly labelled. Additionally, as the training database gets larger, adding more data leads to increasingly smaller gains in performance [80], and it has been seen that training on the entire set can give suboptimal performance [24, 129]. This could be due to overtraining, outliers, or wrongly labelled data. It is impractical to do an exhaustive search to find the optimal subset of training data, and so recent efforts have used *active training* [26] to automatically select an optimal subset of data for training.

Active training [26] is so called because it alters the learning algorithm from a passive one which has no control over its input to one which actively selects suitable portions of the training data. The aim is to select the most informative examples for training. That is, the examples which will give largest improvements in performance. There are many ways to decide what the most informative examples are in any training set, and the approach depends on the task.

Increasing the size of a training set often means including new examples which are similar to examples already existing in the training set. In this case, it is not expected that the new examples will improve the model performance. Thus, for improved performance, it is better to concentrate the training in regions of uncertainty [26]. For ASR, this means focusing the training on high error portions of the training set. If the correct labels, and hence the utterance error, is known, then focusing on high error segments for training can outperform using the entire data set, or just the low error portions [4, 81, 82].

It is relatively straightforward to obtain a large amount of untranscribed data, particularly for a task like broadcast news. This has led to *unsupervised training* where the system is trained on a large amount of unlabelled data [90, 153, 155]. Unsupervised training typically involves automatically transcribing the data and selecting those utterances where the recogniser is confident to add to the training set. Active and unsupervised training are closely linked as both are concerned with selecting suitable subsets of the training data.

For the case of unsupervised learning the correct labels and sentence error are unknown. Thus it is not appropriate to train on utterances with low confidence as the automatic transcription is likely to be wrong. There are two approaches to active training in an unsupervised fashion. The first approach relies on some human intervention, and selects low confidence regions of data to transcribe manually as these are more likely to improve the models [63, 128]. High confidence utterances are more likely to be correct, and the manual effort of transcription is unlikely to improve the models. It might also be useful to exclude very low confidence regions as these are likely to be outliers and hence not informative [129]. A second approach requires no human intervention, and selects high confidence portions of data to add to the training set, along with their automatic transcription [155].

An alternative use of active training is to build task dependent models by selecting appropriate utterances from a large database. For example, building infant and elderly specific [25], or foreign accent models [4] from a larger database of more general speech. In this scenario, the emphasis is on selecting data which is close to a small set of labelled task-dependent data, and is an alternative to model adaptation.

In previous work, active learning has been used with ML training, typically with a weighting at the utterance level. For word level active training, a method of weighting individual

words is needed. In [80, 83], the data is force-aligned and the observations for a particular word are weighted.

For broadcast news transcription, closed captions are normally available. These are normally not an exact transcription of the speech and are coarsely aligned with the signal. Lightly supervised training [90] makes use of these closed captions by aligning them with the automatic transcription, and discarding segments where the two disagree.

Active and unsupervised training require an accurate confidence measure, and a suitable method of applying a data weighting. These aspects are discussed further in chapter 5.

2.5 Language Modelling

The language model, $P(\mathcal{H})$, gives the prior probability of a word sequence during decoding [78]. A hypothesis is broken down into a series of words $\mathcal{H} = \{\mathcal{W}_1 \cdots \mathcal{W}_K\}$, and it is assumed that the language model probability can be written in terms of a word history

$$P(\mathcal{H}) = \prod_{k=1}^{K} P(\mathcal{W}_k | \mathcal{W}_{k-1}, \mathcal{W}_{k-2} \cdots \mathcal{W}_1) \tag{2.60}$$

A commonly used language model is the N-gram. An N-gram language model restricts this word history to the previous $(N-1)$ words

$$P(\mathcal{H}) \approx \prod_{k=1}^{K} P(\mathcal{W}_k | \mathcal{W}_{k-1}, \mathcal{W}_{k-2} \cdots \mathcal{W}_{k-N+1}) \tag{2.61}$$

N-gram language model probabilities are estimated by counting the occurrences of word sequences in a large corpus of text. To obtain good coverage, a large corpus is required. However, as the size of N increases, it is more likely that word sequences are not seen in the corpus and so have an unknown probability. This can be addressed by *discounting* or *backoff*. The former allocates a portion of the probability mass for unseen events, while the latter backs off to smaller values of N if a word sequence is not allocated a probability in the more complex language model.

N-gram language models are specific to the corpus on which they are trained. For a particular task, there may only be a limited amount of domain specific text, and so a domain specific language model can be interpolated with a more general background model.

2.6 Decoding

Decoding for automatic speech recognition is the process of finding the 'best' word sequence or hypothesis, given an observation sequence. As stated in section 2.1, decoding uses Bayes' decision rule to find the most likely hypothesis $\hat{\mathcal{H}}$ given the model parameters $\mathcal{M}$ and observation sequence $\mathcal{O}$. This can be written as

$$\hat{\mathcal{H}} = \underset{\mathcal{H}}{\operatorname{argmax}}\, p(\mathcal{O}|\mathcal{H}, \mathcal{M}) P(\mathcal{H}) \tag{2.62}$$

Two forms of decoding, Viterbi and minimum Bayes' risk, are discussed below.

2.6.1 Viterbi Decoding

Viterbi decoding [75] finds the most likely state sequence through an HMM, given the observation sequence $\mathcal{O}$. The most likely word sequence is recovered from the state sequence. This corresponds to finding the most likely sentence. The algorithm involves a search over all possible state sequences $\boldsymbol{\Psi}$ to find the most likely, $\hat{\boldsymbol{\psi}}$

$$\begin{aligned}\hat{\boldsymbol{\psi}} &= \underset{\boldsymbol{\psi}\in\boldsymbol{\Psi}}{\operatorname{argmax}}\, p(\mathcal{O},\boldsymbol{\psi}|\mathcal{M}) \\ &= \underset{\boldsymbol{\psi}\in\boldsymbol{\Psi}}{\operatorname{argmax}}\, p(\mathcal{O}|\boldsymbol{\psi},\mathcal{M})P(\boldsymbol{\psi}|\mathcal{M})\end{aligned} \tag{2.63}$$

This most likely state sequence is mapped to the most likely hypothesis. Using an HMM acoustic model, observations are conditionally independent given the state that generated them, hence

$$p(\mathcal{O}|\boldsymbol{\psi},\mathcal{M}) = \prod_{t=1}^{T} p(\boldsymbol{o}_t|\psi_t,\mathcal{M}) \tag{2.64}$$

Thus equation 2.63 becomes

$$\hat{\boldsymbol{\psi}} = \underset{\boldsymbol{\psi}\in\boldsymbol{\Psi}}{\operatorname{argmax}}\, P(\boldsymbol{\psi}|\mathcal{M}) \prod_{t=1}^{T} p(\boldsymbol{o}_t|\psi_t,\mathcal{M}) \tag{2.65}$$

It is impractical to perform an exhaustive search over all possible state sequences, and so a recursive form is used. $\phi_j(t)$ is defined as the likelihood of the best *partial path* at time t through the HMM to state ψ_t

$$\phi_j(t) = p(\boldsymbol{o}_1 \cdots \boldsymbol{o}_t, \psi_1 \cdots \psi_{t-1}|\mathcal{M}) \tag{2.66}$$

This can be calculated recursively as, due to the conditional independence assumption, $\phi_j(t)$ can be defined in terms of the likelihood of the partial path ending at the preceding state ψ_{t-1}

$$\phi_j(t) = a_{\psi_{t-1},j} b_j(\boldsymbol{o}_t)\phi_{\psi_{t-1}}(t-1) \tag{2.67}$$

The partial path at time t with maximum likelihood, $\phi_j^{(ML)}(t)$, can be found by a search over preceding states to find the best

$$\phi_j^{(ML)}(t) = b_j(\boldsymbol{o}_t) \max_{i\leq j} a_{ij}\phi_i^{(ML)}(t-1) \tag{2.68}$$

The initial conditions for this recursion are:

$$\begin{aligned} \phi_1^{(ML)}(0) &= 1 \\ \phi_j^{(ML)}(0) &= 0 \text{ for } j \neq 1 \end{aligned} \tag{2.69}$$

Unlike the forward probability in the forward-backward algorithm which is the likelihood of *all* paths entering a state, decoding approximates the probability by the *maximum likelihood* partial path to a particular state. In the final state N at time T, the maximum likelihood path is given by $\phi_N^{(ML)}(T)$.

For continuous speech recognition, HMMs model subword units and must be concatenated to form words and sentences. The token passing algorithm [163] is an implementation of the Viterbi algorithm for continuous speech recognition. Each state of each HMM contains one or more tokens which record the likelihood of the partial path up to and including that state, and the time frame at which the token entered that model. At each time step, all tokens are propagated in parallel according to the recursion in equation 2.68. The most likely token in the exit state of a model is propagated back to the beginning of all models, and the language model probabilities applied. At time T, the token with the highest likelihood can be used to trace back the most likely path, and hence recover the most likely word sequence. Tracing back the N most likely tokens at time T results in an N-best list of hypotheses, which can then be used for further processing.

Typically, there is a large difference in the dynamic range of the acoustic and language models due to modelling assumptions, and so an empirically determined grammar scale factor η is normally applied to the language model probabilities [75]. Also, the decoding algorithm has a tendency to insert short words. Short words are common in the corpora for language model training, and so tend to have an increased likelihood. To address this, a word insertion penalty ρ is often added to each path for every new word [75]. Due to numerical underflow, log probabilities are used during decoding, and so for an utterance of length L, the most likely word sequence maximises

$$\hat{\mathcal{H}} = \underset{\mathcal{H}}{\operatorname{argmax}} \{\log p(\mathcal{O}|\mathcal{M}, \mathcal{H}) + \eta \log P(\mathcal{H}) + \rho L\} \tag{2.70}$$

The final consideration when using the Viterbi algorithm is that of search efficiency. The use of complex acoustic and language models can increase the search space of the decoder and significantly slow down the algorithm. For example, separate tokens are needed for paths with distinct word histories, so a language model with a longer history can increase the number of tokens needed. To alleviate this problem, low probability tokens can be *pruned* as part of the decoding. Tokens with a low likelihood are deleted and those paths are not expanded further. Pruning decreases the computational cost of decoding, but introduces *search errors* where likely paths are pruned out before time T.

2.6.2 Minimum Bayes' Risk Decoding

The Viterbi algorithm of the previous section finds the most likely state sequence through the HMM. Thus, it finds the most likely sentence, and may be viewed as minimising the expected sentence error rate (SER). However, the most common evaluation metric for speech

recognition systems is the word error rate (WER). This mismatch leads to suboptimal performance in terms of word error rate, and may be addressed by minimum Bayes' risk decoding [57, 105, 141, 158] which allows the evaluation metric to be included as part of the decoding algorithm. More specifically, MBR decoding aims to find the best word sequence using

$$\hat{\mathcal{H}} = \underset{\tilde{\mathcal{H}}}{\operatorname{argmin}} \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M}) \mathcal{L}(\mathcal{H}, \tilde{\mathcal{H}}) \tag{2.71}$$

Equation 2.71 is comparable to equation 2.46 for MBR training. The difference in decoding is that the reference hypothesis, $\mathcal{H}_{ref}$, is unknown.

The loss function, $\mathcal{L}(\mathcal{H}, \tilde{\mathcal{H}})$ can be matched to the evaluation measure of interest. For example, as for MBR training, the Levenshtein distance can be used in MBR decoding to match the decoding and evaluation criteria [57]. Or if the task is more specific, such as named entity extraction, a more suitable error measure and loss function may be the F-measure [57] which balances precision and recall.

As written above, the MBR criterion is too computationally expensive to implement directly as it involves a search over all possible word sequences to find the true minimum. Thus, the search is normally restricted to a subset of hypotheses. The hypothesis space $\mathcal{H}_h$ is the space over which the decoder searches for the best hypothesis, and may be different from the evidence space $\mathcal{H}_e$ which represents the most likely hypotheses. Thus, the best hypothesis is approximated by

$$\hat{\mathcal{H}} = \underset{\tilde{\mathcal{H}} \in \mathcal{H}_h}{\operatorname{argmin}} \sum_{\mathcal{H} \in \mathcal{H}_e} P(\mathcal{H}|\mathcal{O}, \mathcal{M}) \mathcal{L}(\mathcal{H}, \tilde{\mathcal{H}}) \tag{2.72}$$

Even when the hypothesis spaces are represented by recognition lattices, there may still be too many hypotheses to perform a full search. Hence advanced search algorithms such as A* search may be employed [57]. However, it is usually the case that the lattice is compressed in some way to reduce the search space. For example, pinched lattices [32], N-best lists, [58, 141], and confusion networks [105] have a considerably smaller search space than recognition lattices.

A popular approach to word error minimisation is to select the word sequence with the highest sum of posterior probabilities [105, 156]. This can be done over an N-best list [141] or a lattice. For word-error minimisation over lattices, a forward-backward pass over the lattice is first done to obtain word posterior probabilities for each of the arcs in the lattice. This posterior probability takes into account the language and acoustic score, and thus makes further processing straightforward. [156] searches directly in the lattice for this best path, while [105] first converts the lattice to a confusion network and thus makes the search for the best word sequence trivial. This latter approach is described in more detail below.

2.6.2.1 Confusion Network Decoding

Confusion network (CN) decoding is a form of MBR decoding which aims to minimise word error rate. A confusion network [104, 105] is derived from a word lattice by merging overlapping links that correspond to the same word, and clustering links that correspond to different words into confusion sets. An example lattice and confusion network are shown in figure 2.6. Multiple lattice arcs for overlapping words are clustered and merged, such as the lattice arcs

for the words 'IN' in the example. Words representing confusions are aligned into a single confusion set. For example, the words 'BUT' and 'IN' are aligned, as they are phonetically similar and overlap in time.

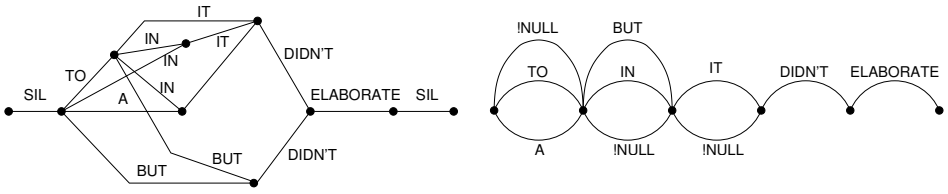


Figure 2.6: *CN generation from a lattice*

To build a CN, it is necessary to obtain a complete alignment of all lattice arcs. It is infeasible to do this directly as there are a large number of hypotheses, and hence an iterative approach is used to cluster links and carry out the alignment. The algorithm is based on heuristics which take into account lattice topology, time stamps and phonetic information.

First, lattice arc posteriors are calculated in a forward-backward pass and low posterior links are pruned out as, in practice, these can affect the alignment. In the first stage of CN generation, lattice arcs with the same word are merged based on their overlap. The arcs are iteratively merged, where at each step the two sets of arcs with the highest similarity are merged. The similarity for merging two clusters of lattice arcs, $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$, is

$$\text{sim}(\boldsymbol{A}_1, \boldsymbol{A}_2) = \max_{\mathcal{W}_1 \in \boldsymbol{A}_1, \mathcal{W}_2 \in \boldsymbol{A}_2} \text{e}(\mathcal{W}_1, \mathcal{W}_2) P(\mathcal{W}_1|\boldsymbol{A}_1) P(\mathcal{W}_2|\boldsymbol{A}_2) \tag{2.73}$$

where $\mathcal{W}_n$ is a word from the cluster $\boldsymbol{A}_n$, $P(\mathcal{W}|\boldsymbol{A})$ is the posterior of a word $\mathcal{W}$ in the cluster $\boldsymbol{A}$, and the overlap between two word arcs, $\text{e}(\mathcal{W}_1, \mathcal{W}_2)$, is normalised by the word length. In this step, links for the same word are merged and so their posteriors are added.

The next step is to cluster sets of arcs with different words together, to give confusion sets. Again, the process is an iterative one where the two clusters with the highest similarity are aligned, until there are no more overlapping clusters. Now, the similarity function is the expected phonetic similarity, $\text{sim}_p(.)$, between two sets of lattice arcs

$$\begin{aligned} \text{sim}(\boldsymbol{A}_1, \boldsymbol{A}_2) &= \text{E}\left[\text{sim}_p(\boldsymbol{A}_1, \boldsymbol{A}_2)\right] \\ &= \operatorname*{avg}_{\mathcal{W}_1 \in \boldsymbol{A}_1, \mathcal{W}_2 \in \boldsymbol{A}_2} \text{sim}_p(\mathcal{W}_1, \mathcal{W}_2) P(\mathcal{W}_1|\boldsymbol{A}_1) P(\mathcal{W}_2|\boldsymbol{A}_2) \end{aligned} \tag{2.74}$$

The resulting graph is known as a confusion network and has a linear structure. Competing hypotheses in the confusion network are potential confusions made by the recogniser, and words in a confusion set are annotated with posterior probabilities. The total posterior for a confusion set can be less than 1, and so a !NULL arc is added to that confusion set to account for the remaining probability mass. Thus, confusion networks can introduce deletions.

The confusion network preserves the order of words in the original lattice, but allows hypotheses to exist which were not present in the original lattice. The final hypothesis is obtained by selecting, for each confusion set, the word which has the highest posterior.

There is a bias in the posteriors due to the incomplete representation of the hypothesis space and the accuracy of the posterior probabilities obtained from the particular model set.

Depending on the task, this bias may need to be accounted for, [34, 67]. A fast CN generation algorithm has also been proposed [160].

2.7 Aligning Multiple Hypotheses

Sections 2.4.2.2 and 2.6.2 described training and decoding algorithms which require the use of multiple hypotheses. Chapter 3 describes approaches to combining multiple hypotheses. Thus, it is often necessary in LVCSR to align multiple hypotheses both against each other or against a reference transcription.

This section first discusses the representations of multiple hypotheses. Then, the simple case of string alignment is discussed, before its extension to aligning multiple hypotheses.

2.7.1 N-best Lists and Word Lattices

In the token passing algorithm of section 2.6.1, it is possible to trace back the N most likely paths through the HMM and obtain a list of the N most likely hypotheses. However, the hypotheses in N-best lists tend not to differ significantly, and a more compact representation of multiple hypotheses is a word lattice [121].

Word lattices are directed graphs where arcs correspond to words, each with a start and end time. Multiple hypotheses can share word arcs, hence reducing the memory requirements for storing multiple hypotheses. Arcs are typically annotated with acoustic and language model probabilities. Word lattices are useful as they provide a compact representation of the most likely hypotheses, and can easily be used for further processing.

One useful step is to convert the acoustic and language model scores in the lattice into posterior probabilities [157]. This can be done using a forward-backward pass, as discussed in section 2.3.2, but at the word level rather than the frame level.

2.7.2 Levenshtein Alignment

The alignment of two strings can be done using a dynamic programming algorithm. The Levenshtein, or edit, distance [96] is the number of insertions, substitutions and deletions required to transform one string into another. For example, in figure 2.7, to transform string S1 into string S0 requires one deletion (‘*a*’), one insertion (‘*it*’), and one substitution (‘*didn't*’ for ‘*would*’). The optimal alignment of two strings is that which minimises the Levenhshtein distance.

S0:	!NULL	but	it	didn't	elaborate
S1:	a	but	!NULL	would	elaborate

Figure 2.7: *Levenshtein alignment of two transcriptions, S0 and S1*

Calculation of the optimal alignment between two strings is done using dynamic programming. The algorithm may associate different penalty scores with insertions, deletions and substitutions, and !NULL arcs are added to hypotheses to handle the insertions and deletions, as in figure 2.7 to align the words ‘*a*’ and ‘*it*’.

The alignment of two or more strings gives a set of word correspondences. After alignment, each string consists of a set of K words, $\mathcal{H} = \{\mathcal{W}^{(1)} \cdots \mathcal{W}^{(K)}\}$, which may include !NULL arcs to enable the alignment. The k^{th} words in each hypothesis are aligned, to give a set of aligned words $\boldsymbol{\mathcal{W}} = \{\mathcal{W}^{(1,k)} \cdots \mathcal{W}^{(S,k)}\}$ from hypotheses $\mathcal{H}^{(1)}$ to $\mathcal{H}^{(S)}$.

2.7.3 Aligning Multiple Hypotheses

Alignment of multiple hypotheses is used throughout this thesis, for example, in minimum Bayes' risk training and decoding in sections 2.4.2.2 and 2.6.2 above. Thus, the problem of aligning multiple hypotheses must be considered.

One algorithm which aligns a small number of hypotheses is ROVER, discussed in section 3.3.1.2 below. In this algorithm, the strings are iteratively aligned using the dynamic alignment multiple times [36]. Thus, if there are S systems yielding S hypotheses, the dynamic programming alignment is performed $S-1$ times. To align a small number of confusion networks the same process is performed, but the substitution cost in the dynamic alignment is altered [34]. For the ROVER alignment, the cost of aligning two words is 0 if they are identical, and 1 if they are different. For CNC, the probability that two sets of words, $\boldsymbol{\mathcal{W}}^{(1)}$ and $\boldsymbol{\mathcal{W}}^{(2)}$ are identical, $P(\boldsymbol{\mathcal{W}}^{(1)} = \boldsymbol{\mathcal{W}}^{(2)}|\mathcal{M}^{(1)}, \mathcal{M}^{(2)}, \mathcal{O})$, can be calculated as

$$P(\boldsymbol{\mathcal{W}}^{(1)} = \boldsymbol{\mathcal{W}}^{(2)}|\mathcal{M}^{(1)}, \mathcal{M}^{(2)}, \mathcal{O}) = \sum_{\mathcal{W} \in \boldsymbol{\mathcal{W}}} \lambda P(\mathcal{W}|\mathcal{M}^{(1)}, \mathcal{O}) + (1-\lambda) P(\mathcal{W}|\mathcal{M}^{(2)}, \mathcal{O}) \quad (2.75)$$

where $\boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{W}}^{(1)} \cap \boldsymbol{\mathcal{W}}^{(2)}$ is the intersection of the two sets of words, and λ is a weight on the first system. Then, the cost of aligning the two sets of words is $1 - P(\boldsymbol{\mathcal{W}}^{(1)} = \boldsymbol{\mathcal{W}}^{(2)}|\mathcal{M}^{(1)}, \mathcal{M}^{(2)}, \mathcal{O})$. Thus, sets of words which are identical have an alignment cost of 0, and as the similarity between the sets of words decreases, the cost increases to 1. The cost of insertion and deletion remains the same.

The dynamic programming alignment is suitable if there are a small number of hypotheses to be aligned. However, when there are many thousands of hypotheses, such as in a word lattice, it becomes impractical to explicitly perform all the alignments. Hence, an approximate alignment can be used instead.

Confusion network generation, discussed above in section 2.6.2.1 is one approach to generating an approximate alignment of multiple hypotheses. The input lattices contain timestamps, and CN generation uses heuristics based on both a phonetic similarity measure and the overlap between words. This similarity measure is used to decide whether two words are competing, and should be aligned or not.

A second approach is to somehow compact the multiple hypotheses to make alignment easier. For example, pinched lattices [32] are a more compact representation of likely competing hypotheses. Lattices are 'pinched' at regions of high confidence, removing the low confidence competing words, and thus segmented into smaller chunks where it is much less computationally expensive to perform the multiple dynamic alignments.

2.8 Decision Trees for Parameter Tying

It is widely known that the realisation of a phoneme depends on its context, and hence it is desirable to model this context dependency as part of an ASR system. However, modelling

context leads to a dramatic increase in the number of model parameters to train, and also the memory needed to store the models. For example, in an ASR system with 50 phones, modelling triphone context, i.e. the preceding and following phones, results in 50^3 models. Obtaining enough training data to adequately model these is difficult, even with thousands of hours of data, as there are many contexts which appear infrequently or not at all. For these reasons, it is normal to share parameters between components of an ASR system, hence reducing the total number of parameters to train. The clustering can be done at many levels, for example at the phone (HMM) level [7], by clustering state output distributions [119] or covariance matrices [84].

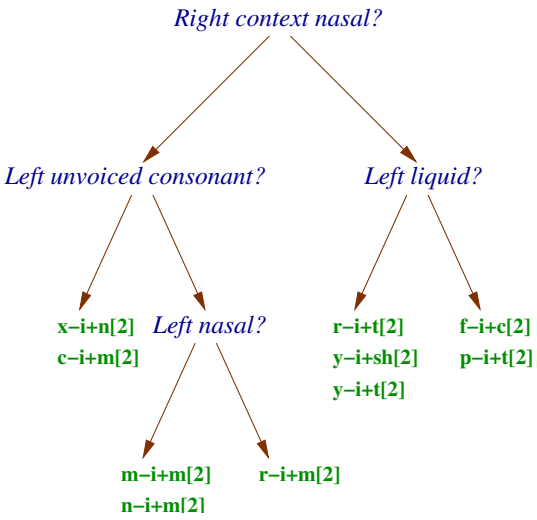


Figure 2.8: *Example decision tree to cluster second state of phone i*

In this thesis, decision trees are used to cluster states of HMMs for parameter tying. Decision trees contain questions at their nodes, and states are clustered at their leaves. The questions typically concern triphone context, but any form of knowledge can be incorporated. An example decision tree is shown in figure 2.8, to cluster the triphone contexts of the second state of phone i. In this figure, the notation a-i+b[2] denotes the second state of the triphone for the vowel i, where the previous context is a, and the following context is b. There are five clusters in the tree, defined by the four questions at the nodes of the tree. For example, the triphone contexts x-i+n and c-i+m are clustered together, and thus share an output distribution. One tree is built for each state of each phone, to cluster the corresponding triphone contexts. Thus, for a system with 50 phones and 3 emitting states per HMM, 150 decision trees are built.

Decision trees are widely used as they provide an elegant way to cluster unseen contexts, they allow expert knowledge to be incorporated via the questions, and their size can be dependent on the amount of training data available. Decision tree clustering is a top-down clustering algorithm which maximises the likelihood of the data. As the decision tree is grown, the states clustered at each node are split according to the question which gives the best local increase in data likelihood. This is continued recursively until the total data likelihood falls below a threshold. Additionally, when splitting a node, it is necessary that there are enough examples in the new clusters to accurately estimate the parameters. Hence, there is a second threshold on minimum occupancy which must be satisfied.

In order to perform this clustering, it is assumed that the alignment of observations to states is not altered by the clustering. This simplifies the calculation of the data log-likelihood. Due to this assumption, the effect of the transition probabilities can also be ignored as they would only be significant if the state alignments changed. The log-likelihood of the data is assumed to be the average of the state log-likelihoods, weighted by their state occupancy. If the state output distributions are Gaussian, the total likelihood of the data, L, being generated by a set of A tied states, $\boldsymbol{\Theta} = \{\theta_1 \cdots \theta_A\}$, is approximated by

$$\begin{aligned} L_{\boldsymbol{\Theta}} &= \sum_{t=1}^{T}\sum_{j=1}^{A} \gamma_j(t) \log \mathcal{N}(\boldsymbol{o}_t; \boldsymbol{\mu}_{\boldsymbol{\Theta}}, \boldsymbol{\Sigma}_{\boldsymbol{\Theta}}) \\ &= \sum_{j=1}^{A}\sum_{t=1}^{T} -\frac{1}{2}\left(\log\left[(2\pi)^D|\boldsymbol{\Sigma}_{\boldsymbol{\Theta}}|\right] + D\right)\gamma_j(t) \\ &= -\frac{1}{2}\left(\log\left[(2\pi)^D|\boldsymbol{\Sigma}_{\Theta}|\right] + D\right)\sum_{t=1}^{T}\sum_{j=1}^{A}\gamma_j(t) \\ &= -\frac{1}{2}\gamma_{\boldsymbol{\Theta}}\left(\log\left[(2\pi)^D|\boldsymbol{\Sigma}_{\Theta}|\right] + D\right) \end{aligned} \tag{2.76}$$

where it is assumed that all the states in $\boldsymbol{\Theta}$ are tied, and so share a common mean $\boldsymbol{\mu}_{\boldsymbol{\Theta}}$ and variance $\boldsymbol{\Sigma}_{\boldsymbol{\Theta}}$, and D is the dimensionality of the data [119]. The cluster parameters can be found from

$$\gamma_{\boldsymbol{\Theta}} = \sum_{t=1}^{T}\gamma_{\boldsymbol{\Theta}}(t) = \sum_{t=1}^{T}\sum_{j=1}^{A}\gamma_j(t) \tag{2.77}$$

$$\boldsymbol{\mu}_{\boldsymbol{\Theta}} = \frac{\sum_{t=1}^{T}\gamma_{\boldsymbol{\Theta}}(t)\boldsymbol{o}_t}{\gamma_{\boldsymbol{\Theta}}} \tag{2.78}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{\Theta}} = \frac{\sum_{t=1}^{T}\gamma_{\boldsymbol{\Theta}}(\boldsymbol{o}_t - \boldsymbol{\mu}_{\boldsymbol{\Theta}})(\boldsymbol{o}_t - \boldsymbol{\mu}_{\boldsymbol{\Theta}})^{\mathsf{T}}}{\gamma_{\boldsymbol{\Theta}}} \tag{2.79}$$

To obtain the initial state alignments, a forward-backward pass of the data is performed with a well-trained system. The sufficient statistics needed to calculate the quantities in equations 2.77 to 2.79 are the means $\boldsymbol{\mu}_j$, variances $\boldsymbol{\Sigma}_j$ and occupation counts γ_j of all states in the system.

As splitting a cluster of states is assumed to have a local effect only, i.e. no effect on other clusters, only the local change in log likelihood must be maximised to grow the tree. The change in likelihood, δL when splitting a parent cluster, $\boldsymbol{\Theta}$, into a set of descendant nodes, $\boldsymbol{D}$, is

$$\begin{aligned} \delta L &= -\sum_{\boldsymbol{d}\in\boldsymbol{D}}\left\{\frac{1}{2}\log|\Sigma_{\boldsymbol{d}}|\sum_{t=1}^{T}\gamma_{\boldsymbol{d}}(t)\right\} + \frac{1}{2}\log|\Sigma_{\boldsymbol{\Theta}}|\sum_{t=1}^{T}\gamma_{\boldsymbol{\Theta}}(t) \\ &= -\sum_{\boldsymbol{d}\in\boldsymbol{D}}\left\{\frac{1}{2}\gamma_{\boldsymbol{d}}\log|\Sigma_{\boldsymbol{d}}|\right\} + \frac{1}{2}\gamma_{\boldsymbol{\Theta}}\log|\Sigma_{\boldsymbol{\Theta}}| \end{aligned} \tag{2.80}$$

where $\gamma_{\boldsymbol{d}}$ is the total occupancy of the descendant cluster. For a binary tree, $\boldsymbol{D}$ contains just two descendants. Constructing a decision tree is thus a top-down clustering process, with three stages:

1. **Statistics**
   - Obtain statistics for all *seen triphone* contexts using the forward-backward algorithm
2. **Question Selection**
   - Recursively build the tree by selecting the question which gives the change in highest data likelihood, provided the occupancy count of each new cluster is above a threshold
3. **Stopping criterion**
   - Stop building the tree when the data likelihood falls below a threshold

The effect of the algorithm is to cluster states which are close in acoustic space, as these are likely to be well modelled by a shared distribution. A disadvantage of parameter sharing is that only contextual and linguistic information is available to distinguish clustered states, as they share an output distribution. This is not ideal if clustered states are confusable and lead to errors. The decision tree algorithm does not consider whether states are confusable or not when performing the clustering, and so it is possible that confusable states will be clustered.

As the decision tree algorithm is locally optimal, slightly altering any stage of the process can lead to very different decision trees being built. For this reason, the decision tree algorithm is a good stage to focus on for complementary system generation. A further advantage of altering the decision tree algorithm is that no changes need to be made to the HMM training algorithm. However, the decision tree generation stage typically occurs early in the process of building a speech recognition system, and so it is time-consuming to build and evaluate many systems with different decision trees.

2.9 Adaptation

The feature transforms and normalisation schemes discussed in section 2.2.2 partially address speaker and environment variations in the input. However, these are global approaches and therefore limited in addressing differences caused, for example, by differing speakers or environments. Adaptation techniques have been proposed which address this mismatch. Although adaptation is discussed here in the context of speakers, the techniques can also be applied to task and environment adaptation. Adaptation can either be supervised, where the transcription of the adaptation data is known, or unsupervised, where an initial transcription must first be found.

Adaptation techniques typically transform an initial, speaker independent, model into a speaker dependent model. This can be done by interpolating models, transforming the features or model parameters, or by further training. Normally, there is much less speaker-dependent data available for adaptation than there is in the training set used to train the

initial speaker independent models. Hence, adaptation techniques must work well with limited data.

MLLR is the most common technique for adaptation [42, 46, 95]. MLLR linearly transforms the mean [95] and/or variance [42] of a model to better represent a particular speaker.

Transforms are normally tied across a number of model components, using a regression class tree [41]. Regression class trees tie components of HMMs which are close in acoustic space, in order to apply a single transform over a number of components. This has some similarity to decision trees for parameter tying, but typically there are far fewer classes for adaptation.

The adapted mean $\tilde{\boldsymbol{\mu}}_j$ is a transform of the speaker independent mean $\boldsymbol{\mu}_j$

$$\tilde{\boldsymbol{\mu}}_j = \boldsymbol{H}_{r_j}\boldsymbol{\mu}_j + \boldsymbol{g}_{r_j} \tag{2.81}$$

where $\boldsymbol{H}_{r_j}$ is an $n \times n$ transform associated with state θ_j, and $\boldsymbol{g}_{r_j}$ is an $n \times 1$ bias. r_j refers to the regression class which state θ_j belongs to, over which transforms are shared. The variance transform, an $n \times n$ transform, $\boldsymbol{J}_{r_j}$, is given by

$$\tilde{\boldsymbol{\Sigma}}_j = \boldsymbol{J}_{r_j}\boldsymbol{\Sigma}_j\boldsymbol{J}_{r_j}^{\mathsf{T}} \tag{2.82}$$

When the mean and variance transforms are restricted to be identical, the transform can be applied directly to the features and is known as CMLLR [42].

The transform parameters, $\mathcal{T}$, are the transforms $\boldsymbol{H}_{r_j}$, $\boldsymbol{g}_{r_j}$ and $\boldsymbol{J}_{r_j}$, and are estimated using maximum likelihood and the EM algorithm. $\tilde{\boldsymbol{\mu}}_j$ and $\tilde{\boldsymbol{\Sigma}}_j$ of equations 2.81 and 2.82 can be substituted into the auxiliary function in equation 2.32, and the transform parameters $\hat{\mathcal{T}}$ can be estimated by optimising

$$\mathcal{Q}_{MLLR}(\hat{\mathcal{T}}^k, \hat{\mathcal{T}}^{k+1}) = \sum_{t=1}^{T}\left\{\sum_{j=1}^{N}\gamma_j^k(t)\left[\log\mathcal{N}(\boldsymbol{o}_t; \hat{\boldsymbol{\mu}}_j^{k+1}, \hat{\boldsymbol{\Sigma}}_j^{k+1})\right]\right\} \tag{2.83}$$

The above approach requires an initial speaker independent model which is typically trained on multiple speakers, and so captures both the inter-speaker and intra-speaker variability. MLLR can be used as part of training, interleaving transform and model parameter estimates, to train a speaker-independent model which captures just the intra-speaker variability and a transform set which captures the inter-speaker variability. The resulting model set should be more compact, i.e. have smaller variances, than the original speaker independent model. This is known as speaker adaptive training (SAT) [3].

Another speaker adaptation approach is MAP adaptation [50] which performs an update of the model parameters, interpolating between a prior estimate of model parameters and the ML update from the adaptation data. The prior estimate is often the speaker independent model parameters, and the influence of the prior can be controlled via a parameter τ. When a small amount of adaptation data is available, the update tends towards the prior. When a large amount of data is available, the update tends towards the ML estimate. This form of adaptation is closely related to the parameter smoothing for discriminative training, in section 2.4.2.5. MAP adaptation tends to perform worse than MLLR on small amounts of adaptation data, but outperform MLLR on large amounts of data.

2.10 Summary

This chapter has summarised the current state-of-the-art for large vocabulary automatic speech recognition. The statistical approach to ASR was first presented, and then each of the modules was discussed in more detail. These include frontend processing, language modelling, acoustic modelling, and decoding. Additionally, several techniques were discussed that are an important part of a large vocabulary recogniser, including the use of multiple hypotheses, feature normalisation and transformations, parameter tying, discriminative training, speaker adaptation and minimum Bayes' risk decoding.

Chapter 3
System Combination

When the final hypothesis is to be obtained from multiple systems, the method of system combination is an important consideration. This chapter first introduces the multi-pass framework for large vocabulary continuous speech recognition used at Cambridge. Then, existing methods for efficiently combining multiple classifiers and their application to ASR are discussed. Different forms of system combination are discussed in this chapter, where the combination is performed at the hypothesis level, by combining distributions during decoding, or where the combination is implicitly performed. Methods for training the corresponding models are presented in the following chapter. Finally, an ideal form of combination is discussed.

3.1 Multi-pass Combination Framework for LVCSR

In recent years, as computational power has increased, there has been an interest in large vocabulary continuous speech recognition (LVCSR) on *found data*. That is, real-life data which is plentiful and easy to collect, though not necessarily labelled. For example, television news broadcasts, meetings and lectures all provide a convenient and plentiful source of real-life unlabelled speech data. Using such sources together with the unsupervised training techniques discussed in section 2.4.3 avoids the cost of collecting and transcribing large amounts of clean data, but the nature of the data introduces additional complexities.

Typically, data of this sort can contain an unknown number of different speakers, with different languages, noise sources and non-speech sounds. It is usual to automatically segment the input speech into single utterances and remove non-speech. Speaker clustering can also be performed to identify individual speakers. Additionally, the sound quality can vary significantly throughout the database, and often the speech is spontaneous and ungrammatical,

so complex acoustic and language models are needed for decoding. Typically, 4 and 5-gram language models are used, and acoustic models may be discriminatively trained with speaker adaptation and normalisation techniques such as VTLN, HLDA and CMN/CVN.

Examples of recent LVCSR tasks are: broadcast news transcription in English [86], Arabic [1, 49, 107, 140] and Mandarin [48, 76, 137], lecture transcription [56], meeting transcription [62, 74], conversational telephone speech [47] and transcription of European Parliamentary speeches [100, 126].

The direct use of complex models can prove too slow to be useful in practice due to the increased search effort required. Hence, it is important to design a system that can efficiently make use of complex models for decoding. Typically, a multi-pass architecture is used for LVCSR [35, 51, 114], where simple models are used in initial passes to refine the hypothesis space, and complex models act on this reduced hypothesis space. Word lattices, as discussed in section 2.7.1, are often used to represent the refined search space as they provide a compact representation of competing hypotheses.

A multi-pass framework is necessary if unsupervised speaker adaptation, discussed in section 2.9, is performed. First an approximate hypothesis is needed in order to perform the adaptation, and this is obtained from an initial model. The final hypothesis is then obtained by decoding with the adapted models.

In developing the models for use in such a framework, it is normal to train a variety of systems to find the best performance. Experiments have shown that rather than just using one system, gains can be obtained from combining multiple systems [114]. Hence, multi-pass architectures often make use of multiple models in the final pass, which are combined in some way.

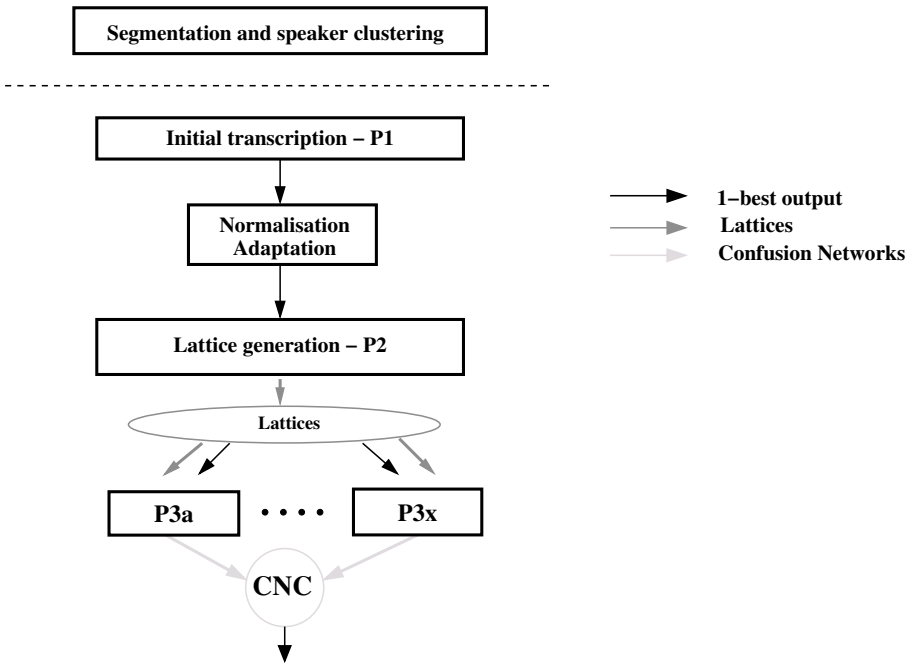


Figure 3.1: *A multi-pass combination framework for ASR*

An example of a multi-pass combination framework for ASR decoding is shown in figure 3.1. This framework is used for broadcast news transcription at Cambridge. First, utterance

segmentation and speaker clustering are performed. Then, there are three passes for decoding; the first, P1, obtains an initial transcription for adaptation and normalisation, the second pass, P2, performs the adaptation and generates lattices, while the third pass, P3a to P3x, rescores these lattices using multiple models, and combines their outputs together using confusion network combination (CNC), detailed in section 3.3.1.3 below.

It is only sensible to use multiple models in the final pass if they are complementary. That is, if they make different errors. An ad-hoc approach to generating complementary systems is to train a number of different systems and select the ones which combine to give the best results. It is not possible to predict which systems are complementary based on individual performance alone, so many possible combinations must be performed to find the best. Also, it is not guaranteed that any independently trained systems will in fact be complementary and, as the number of individual systems increases, it becomes increasingly time-consuming to evaluate the different combinations. Experiments have shown that gains are normally only seen if the systems being combined have comparable individual performance [48]. In previous work, examples of this ad-hoc approach have used individual systems with different

- segmentations [48, 126]
- frontends, i.e. features and normalisation [62, 73, 76, 98, 100, 110, 137, 143]
- covariance modelling [98]
- training algorithms [49, 100, 137]
- decision trees [74, 126]
- microphones [74]
- phone sets [143]
- dictionaries [48]
- adaptation [62, 98]

In figure 3.1, the hypotheses estimated by systems P3a to P3x are explicitly combined using confusion network combination, although any of the hypothesis combination schemes described in section 3.3.1 could instead be used. The framework also allows for an implicit combination approach, as discussed in section 3.3.4 below, where different systems are used in the individual passes, P1 to P3. The remainder of this chapter considers existing approaches to system combination, and chapter 4 discusses existing methods for generating complementary systems that could be used in the final pass.

3.2 General Combination Methods

Before discussing combination methods for ASR, a simple multiclass classification task is first considered. A labelled set of N pairs, $\{\boldsymbol{o}_i, \mathcal{H}_i\}$, form the data set, where there are V classes and thus $\mathcal{H}_i \in \{C_1 \cdots C_V\}$. For the observation $\boldsymbol{o}$, an ensemble of S classifiers, $\mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}$, make S decisions about the class which the observation belongs to. From these S hypotheses, the best hypothesis, $\hat{\mathcal{H}}$, must be selected. An example of this kind of task is

phone classification, where an observation is classified according to the phone which generated that observation. Combination methods for this simpler task are discussed below. There are two levels at which the combination can be performed. First, the individual hypotheses can be combined, as in voting and posterior combination. Second, the individual model distributions can be combined, as is done for mixtures and products of experts.

This static task is simple and the algorithms described below are not easily applied directly to LVCSR. However, they form the basis of several combination algorithms for ASR, discussed below in section 3.3.

3.2.1 Majority and Weighted Voting

Suppose each classifier, $\mathcal{M}^{(s)}$, makes a hard decision about which class an observation belongs to, $\mathcal{H}^{(s)}$. The simplest way of combining independent classifier outputs is to implement a majority voting scheme. Such a scheme counts how many classifiers allocated the observation to each class, and selects the class which has the highest frequency of occurrence. Thus, to select the best hypothesis, $\hat{\mathcal{H}}$, the decision rule has the form

$$\hat{\mathcal{H}} = \underset{\mathcal{H}\in\{C_1\cdots C_V\}}{\operatorname{argmax}} \left\{ \sum_{s=1}^{S} \delta(\mathcal{H}^{(s)}, \mathcal{H}) \right\} \tag{3.1}$$

where $\delta(\mathcal{H}^{(s)}, \mathcal{H}) = 1$ when $\mathcal{H} = \mathcal{H}^{(s)}$ and 0 otherwise.

There are a number of problems with this simple voting process. It is common for the voting to result in ties, in which case, without further information, an arbitrary choice must be made. With a large number of classes, it is less likely that two classifiers will yield the same hypothesis and ties are more likely. Also, poor classifiers are given the same weight as good classifiers and so will have the same influence on the final choice of class. To overcome these problems, a weighted majority voting scheme can be used. Here, each of the S classifiers is assigned a weight, λ_s. These weights should be adjusted to reflect the performance of the classifiers, and typically sum to 1. The weighted voting scheme selects the final hypothesis as

$$\hat{\mathcal{H}} = \underset{\mathcal{H}\in\{C_1\cdots C_V\}}{\operatorname{argmax}} \left\{ \sum_{s=1}^{S} \lambda_s \delta(\mathcal{H}^{(s)}, \mathcal{H}) \right\} \tag{3.2}$$

Although this form of voting is not appropriate for continuous speech recognition tasks, it may be of use for phone, or digit, recognition style tasks where the number of classes is small.

3.2.2 Posterior Combination

The voting schemes in the previous section assume that individual classifiers make hard decisions about class membership, i.e. the effective posterior probability of a particular hypothesis is either 1 or 0. However, rather than simply output the 1-best hypothesis, many classifiers output posterior probabilities or confidence in class membership, $P(\mathcal{H}|\boldsymbol{o}, \mathcal{M}^{(s)})$. It is possible to modify the simple majority voting scheme in the previous section to reflect the use of posteriors. Now the decision is based on

$$\hat{\mathcal{H}} = \underset{\mathcal{H}\in\{C_1\cdots C_V\}}{\operatorname{argmax}} \left\{ \sum_{s=1}^{S} \lambda_s P(\mathcal{H}|\boldsymbol{o}, \mathcal{M}^{(s)}) \right\} \tag{3.3}$$

where again a weight, λ_s, has been introduced to allow the performance of the individual classifiers to be reflected and the posterior probability of class membership $P(\mathcal{H}|\boldsymbol{o}, \mathcal{M}^{(s)})$ replaces the hard decision $\delta(\mathcal{H}^{(s)}, \mathcal{H})$ in the previous section. A standard scheme that makes use of this form of weighted voting is AdaBoost [37], discussed in more detail in section 4.1.2.

3.2.3 Mixtures and Products of Experts

Majority voting and posterior combination are schemes for selecting the best from a set of hypotheses. Alternatively, individual model distributions can be combined to obtain a single score for each class, for use in further processing stages. Mixtures [11] and products of experts [70] are commonly used for combining scores, where each expert is a probability distribution.

The likelihood of an observation given all models, $p(\boldsymbol{o}|\mathcal{H}, \mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)})$, can be a weighted sum of likelihoods from a set of experts

$$p(\boldsymbol{o}|\mathcal{H}, \mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}) = \sum_{s=1}^{S} \lambda_s p(\boldsymbol{o}|\mathcal{H}, \mathcal{M}^{(s)}) \tag{3.4}$$

or a weighted product of likelihoods from the experts

$$p(\boldsymbol{o}|\mathcal{H}, \mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}) = \frac{1}{Z} \prod_{s=1}^{S} \lambda_s p(\boldsymbol{o}|\mathcal{H}, \mathcal{M}^{(s)}) \tag{3.5}$$

where

$$Z = \int_{\mathcal{R}^D} \prod_{s=1}^{S} \lambda_s p(\boldsymbol{o}|\mathcal{H}, \mathcal{M}^{(s)}) d\boldsymbol{o} \tag{3.6}$$

The normalisation term Z ensures that the result is a valid probability distribution. In practice, this term can only be calculated analytically for a few forms of expert, and must be approximated in other cases. In contrast, it is trivial to normalise a mixture of experts by enforcing a sum-to-one constraint on the weights, $\sum_{s=1}^{S} \lambda_s = 1$.

One commonly used mixture for automatic speech recognition is the Gaussian Mixture Model, previously described in section 2.3.1 as the HMM state output distribution

$$p(\boldsymbol{o}) = \sum_{m=1}^{M} c_m \mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m) \tag{3.7}$$

where there are M Gaussian experts in the mixture, and c_m is the weight associated with each Gaussian. c_m replaces the classifier weight, λ_s, for a Gaussian mixture model as it is

possible to use mixtures or products of GMMs, particularly when combining multiple HMM states. A product of Gaussians can also be used

$$p(\boldsymbol{o}) = \frac{1}{Z} \prod_{m=1}^{M} \mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m) \tag{3.8}$$

As the product of multiple Gaussians is also Gaussian, the normalisation term Z can be calculated analytically [45], and so equation 3.8 becomes

$$p(\boldsymbol{o}) = \mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3.9}$$

where

$$\boldsymbol{\mu} = \boldsymbol{\Sigma} \left(\sum_{m=1}^{M} \boldsymbol{\Sigma}_m^{-1} \boldsymbol{\mu}_m \right) \tag{3.10}$$

$$\boldsymbol{\Sigma} = \left(\sum_{m=1}^{M} \boldsymbol{\Sigma}_m^{-1} \right)^{-1} \tag{3.11}$$

The mixture and product frameworks for Gaussians can be extended to handle mixtures and products of GMMs, as the normalisation quantities can be calculated analytically. Mixtures and products of Gaussians allow the modelling of more complex distributions, while still retaining the analytic properties of Gaussian distributions. A mixture of experts typically yields a high likelihood where one or more of the components have a high likelihood, and thus a mixture model tends to be broader than the individual components alone. A product of experts, on the other hand, typically yields a high likelihood only where all the components have a high likelihood. Thus, products model sharper distributions than their individual components.

3.3 Combination for Automatic Speech Recognition

An ASR system typically outputs a multiple word hypothesis or a lattice, often with associated confidence scores. Hence, due to the large number of potential output hypotheses, the methods discussed above are not appropriate for combining the output hypotheses of ASR systems.

In this section, existing methods for combining speech recognisers are discussed. These fall into three classes - hypothesis, or posterior, combination; distributional, or likelihood, combination; and implicit combination. In the first scheme, the data is decoded independently using multiple systems, and posteriors are combined. In the second, likelihoods from multiple systems are combined as part of the decoding process. In the final scheme, the output from one system is used, or refined, by a second system.

3.3.1 Hypothesis Combination Schemes

A straightforward way to combine multiple ASR systems is to decode the data independently using each system, and somehow combine the output hypotheses. This can be done by directly combining the output hypothesis spaces, in the form of lattices or N-best lists, or by first aligning the most likely hypotheses and then selecting the best. ROVER and CNC are word-level examples of the latter approach, presented in sections 3.3.1.2 and 3.3.1.3, although with an appropriate alignment stage they could be applied at other levels.

Combination of hypotheses is often done within the minimum Bayes' risk decoding framework, although sections 3.3.1.5 and 3.3.1.6 discuss two more general forms of posterior combination.

The schemes discussed in this section are based on posterior combination, and hence a reliable confidence measure or posterior probability is needed to perform the combination, as in section 5.1. Additionally, the combination can be done at any level, but in practice is often done at the word or utterance level.

3.3.1.1 Minimum Bayes' Risk Decoding

The minimum Bayes' risk (MBR) decoding scheme presented in section 2.6.2 also provides a framework for model combination. MBR decoding uses

$$\hat{\mathcal{H}} = \underset{\tilde{\mathcal{H}}}{\operatorname{argmin}} \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M}) \mathcal{L}(\mathcal{H}, \tilde{\mathcal{H}}) \tag{3.12}$$

Although MBR decoding was originally applied to N-best lists [141] and recognition lattices from just one system, the the values of $P(\mathcal{H}|\mathcal{O}, \mathcal{M})$ and $\mathcal{L}(\mathcal{H}, \tilde{\mathcal{H}})$ in equation 3.12 may be calculated from a single recogniser, or from a combination of multiple recognisers. Hence, MBR decoding can be used as a method for combining multiple systems.

ROVER and CNC are two popular word-level combination schemes which fall into this framework, and are discussed in the sections below. They rely on a word-level alignment of multiple hypotheses, as discussed in section 2.7. An alternative to aligning hypotheses is to directly integrate the multiple output hypothesis spaces, e.g. lattices, before rescoring and searching for the best hypothesis. The lattices from multiple systems can be combined, and arcs with the same time stamps and words merged to give a single lattice [21]. From this lattice, the best path is found. This approach has the advantage that timing information from the lattices is not lost, as happens with the word level alignment schemes. Acoustic and language model scores are not easily merged, so a straightforward approach is to first calculate a posterior probability for each lattice arc, as in section 2.7.1, and merge corresponding arcs by adding the posteriors together.

The best, or most probable, path can be found directly using posteriors from the merged lattice. Alternatively the arcs can be rescored and the best path found using a different function. For example, the time frame word error (fWER) [21, 71, 72] or the expected phone accuracy [21]. The fWER criterion is a smoothed estimate of the word error

$$fWER(\mathcal{H}, \tilde{\mathcal{H}}) = \sum_{\mathcal{W} \in \mathcal{H}} \frac{\sum_{t=t_{start}}^{t_{end}} 1 - \delta_t(\mathcal{W}, \tilde{\mathcal{H}})}{1 + \alpha(t_{end} - t_{start} - 1)} \tag{3.13}$$

t_{start} and t_{end} are the start and end times of word $\mathcal{W}$. $\delta_t(\mathcal{W}, \tilde{\mathcal{H}})$ is the Kronecker delta function between the word $\mathcal{W}$ and hypothesis $\tilde{\mathcal{H}}$ at time t. fWER correlates with word error rate, and so finding the path with the minimum fWER score is expected to minimise word error rate.

A second approach is to use scores from the hypothesis space of one system to drive the search of another [93, 120].

These approaches are an extension of MBR decoding where the decoding is performed over an integrated hypothesis space. As such, they perform the combination at the utterance level, and do not allow for hypotheses that did not exist in the original hypothesis spaces.

3.3.1.2 ROVER

The ROVER algorithm [36] was devised at NIST to allow voting methods to be used for word-level system combination within large vocabulary speech recognition tasks. This scheme makes use of the one-best hypothesis from a set of speech recognisers, with optional confidence associated for each output. ROVER first breaks each 1-best hypothesis, $\mathcal{H}^{(s)}$, into its constituent words:

$$\mathcal{H}^{(s)} = \left\{ \mathcal{W}_1^{(s)}, \ldots, \mathcal{W}_{K^{(s)}}^{(s)} \right\} \tag{3.14}$$

where $K^{(s)}$ is the length of the hypothesis from system S. The S hypotheses are aligned at the word level, as discussed in section 2.7, and !NULL links can be added to align hypotheses of differing lengths. Voting can now be done at the word level, which significantly reduces the number of classes over voting at the sentence level.

Thus, ROVER has two distinct stages. First, the algorithm begins by iteratively aligning the hypotheses from the component systems to create a Word Transition Network (WTN). The alignment algorithm is similar to the one used to score recognition results. It is worth noting that iteratively combining system outputs does not necessarily guarantee an optimal WTN and the final WTN is sensitive to the order of combination. Once the WTN has been created, a simple voting module is used to decide upon the best word from a set of correspondences in a similar fashion to that described in the majority voting scheme in section 3.2.1. If there is a tie between words, and no further information is available, a random choice must be made. This algorithm is shown in figure 3.2, where four systems $S0 \cdots S3$ are aligned. For each of the five sets of aligned words, the most frequent is chosen yielding a final hypothesis *'but it didn't elaborate'*. Due to the word level combination, this final hypothesis does not necessarily exist in the hypotheses that are combined.

S0:	to	but	it	didn't	elaborate
S1:	a	but	did	not	elaborate
S2:	!NULL	but	!NULL	didn't	elaborate
S3:	!NULL	in	it	didn't	elaborate
ROVER:	!NULL	but	it	didn't	elaborate

Figure 3.2: *ROVER Combination of four systems, S0-S3, without confidence scores*

To alleviate the problem of ties between words, it is possible to incorporate confidence scores or word posteriors from the outputs into the voting process, to make a more refined

choice. One possible voting form applies the following classification for each set of confusions in the WTN

$$\hat{\mathcal{W}} = \underset{\mathcal{W}}{\text{argmax}} \left\{ \beta \frac{N(\mathcal{W})}{S} + (1-\beta) P(\mathcal{W}|\mathcal{O}, \mathcal{M}^{(s)}) \right\} \tag{3.15}$$

where β is a tuning parameter, $N(\mathcal{W})$ is the number of times that word $\mathcal{W}$ occurs and $P(\mathcal{W}|\mathcal{O}, \mathcal{M}^{(s)})$ is the posterior probability of word $\mathcal{W}$. This form of voting is an interpolation between the majority voting scheme of equation 3.1 and an unweighted posterior combination scheme, similar to that in equation 3.3.

ROVER combination for two systems reduces to picking the word with the highest confidence where the two disagree. Hence, if the recogniser confidence scores are not reliable, then ROVER between two systems often does not perform well. For example, if one system consistently overestimates the confidence scores, then that system is likely to be chosen when there is a disagreement between the two systems. The final performance is likely to be similar to the single system alone. It is possible to alleviate this problem somewhat by mapping the confidence scores to more representational values. As more systems are combined, ROVER becomes more robust to the confidence scores.

One limitation of ROVER is that only the 1-best hypothesis from each component system is used. N-best ROVER [59] uses an N-best list from each recogniser in place of the 1-best to better model the hypothesis space. Other refinements have used machine learning techniques in place of voting to select the best word [68, 168] and incorporated other constraints into the voting, such as language model information [136]. Machine learning techniques are discussed in more detail in section 5.1.3, in the context of confidence scoring.

ROVER is typically performed at the word-level, as this matches the word-level error rate which is used as an evaluation metric for speech recognition systems. Additionally, performing ROVER at a finer granularity, such as phone, frame, or state, may lead to problems in mapping the final best hypothesis at this level back to a valid word sequence.

3.3.1.3 Confusion Network Combination

Rather than use the 1-best output from each system, CNC begins with lattice outputs and first converts these to confusion networks, as in section 2.6.2.1. The confusion networks for all systems are aligned, as discussed in section 2.7, and the best word for each segment is chosen. The word-level voting scheme for confusion network combination uses the posterior combination scheme of equation 3.3:

$$\hat{\mathcal{W}} = \underset{\mathcal{W}}{\text{argmax}} \frac{1}{S} \sum_{s=1}^{S} \lambda_s P(\mathcal{W}|\mathcal{O}, \mathcal{M}^{(s)}) \tag{3.16}$$

where the set of possible classes in equation 3.3 is replaced by the set of possible words. Thus, like ROVER, CNC is a word-level combination scheme, although it makes use of multiple hypotheses from each component system in the form of lattices. With an appropriate alignment stage, CNC could be applied at any level, but is normally applied at the word level.

Unweighted confusion network combination of two systems, S0+S1, is shown in figure 3.3. The final hypothesis obtained is *'but it didn't elaborate'*. As for ROVER, the final hypothesis can differ from the existing hypotheses of the systems being combined.

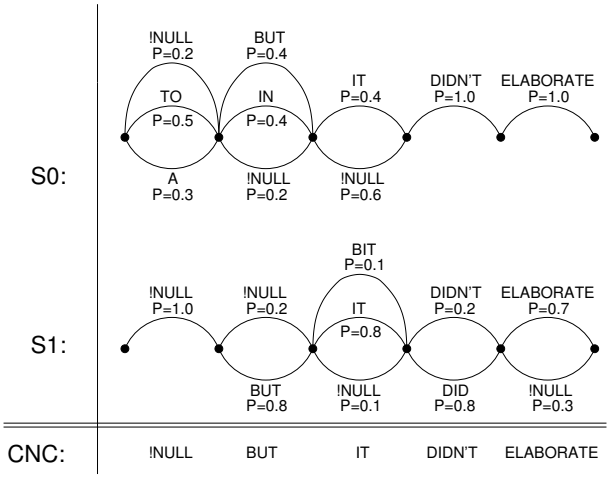


Figure 3.3: *Confusion Network Combination of two systems*

Like ROVER, CNC relies on accurate confidence scores and can perform poorly when these aren't reliable. However, it tends to be more robust than ROVER to poor confidence scores when there are few systems as multiple lattice words are used in combination.

Both ROVER and CNC may be viewed as schemes for generalising the majority voting and the posterior combination approaches to continuous speech recognition. Neither of the schemes are guaranteed to be optimal. In both cases, the alignment step, i.e. the generation of the WTN or confusion network, is not necessarily optimal. The order in which systems are combined is important for both ROVER and CNC, and previous experiments have suggested that systems should be combined in increasing order of word error rate [71]. Both ROVER and CNC align hypotheses at the word level and so the alignments of words to frames become unreliable. A further post-processing step is required to obtain accurate alignments from the final hypothesis.

Despite these limitations, ROVER and CNC are the dominant form of system combination for state-of-the-art speech recognition systems. However, it has been found that ROVER and CNC normally only perform well when the systems to be combined have comparable error rates [142], which limits the types of system that can be combined in practice. Word-level confusion network combination is used for combining multiple systems throughout this thesis, and also forms the basis of the new combination techniques investigated in chapter 11.

3.3.1.4 Weighted Combination

ROVER and CNC use a combination of the confidence scores output by multiple recognisers to determine the best hypothesis. However, poor confidence scores can skew the results obtained from combination. Hence it is useful to use information about the reliability of the classifiers when doing the combination, and weight each hypothesis accordingly. Weights can be based on classifier performance, or estimated on a held-out data set.

An alternative approach is to derive classifier weights using Bayesian decision theory, for example BAYCOM [131]. Here, a number of classifiers are assumed to output both a hypothesis and a feature vector $\boldsymbol{x}$ which may be multidimensional. In total, the S systems

are assumed to output N unique hypotheses. Then, assuming the individual model hypotheses are independent of each other, the best hypothesis is chosen as

$$\begin{aligned}\hat{\mathcal{H}} &= \underset{\mathcal{H}}{\operatorname{argmax}}\, P(\mathcal{H}|\mathcal{O}, \mathcal{H}_1 \cdots \mathcal{H}_S, \boldsymbol{x}_1 \cdots \boldsymbol{x}_S) \\ &= \underset{\mathcal{H}}{\operatorname{argmax}}\, P(\mathcal{H}|\mathcal{O}) \sum_{s=1}^{S} P(\boldsymbol{x}_s|\mathcal{O}, \mathcal{H}_s, \mathcal{H}) P(\mathcal{H}_s|\mathcal{O}, \mathcal{H}) \end{aligned} \tag{3.17}$$

The conditional dependence on the S models, $\{\mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}\}$, is dropped, for clarity. Hypotheses belong to two classes: C_c if they are correct, and C_e if not. Then it is assumed that

$$P(\boldsymbol{x}_s|\mathcal{O}, \mathcal{H}_s, \mathcal{H}) = \begin{cases} P(\boldsymbol{x}_s|C_c) & \text{if } \mathcal{H}_s \text{ correct} \\ P(\boldsymbol{x}_s|C_e) & \text{otherwise} \end{cases} \tag{3.18}$$

The distributions of the feature vectors for both correct and incorrect words $P(\boldsymbol{x}_s|C_c)$ and $P(\boldsymbol{x}_s|C_e)$ are are modelled by a histogram, and are estimated on a held-out set. Finally, it is assumed that the prior probability of being correct for each model is independent of the hypothesis and that the probability of being incorrect is equally distributed

$$P(\mathcal{H}_s|\mathcal{O}, \mathcal{H}) = \begin{cases} P(C_c|\mathcal{M}^{(s)}) & \text{if } \mathcal{H}_s \text{ correct} \\ P(C_e|\mathcal{M}^{(s)})/(N-1) & \text{otherwise} \end{cases} \tag{3.19}$$

and these scores can also be calculated from the held-out set. This is a form of the posterior voting scheme defined in equation 3.3, where the system weights are given by

$$\lambda_s = P(\boldsymbol{x}_s|\mathcal{O}, \mathcal{H}_s, \mathcal{H}) P(\mathcal{H}_s|\mathcal{O}, \mathcal{H}) \tag{3.20}$$

BAYCOM describes a scheme for finding optimal weights, which makes no assumptions about the nature of the recogniser scores $\boldsymbol{x}$, and does not require classifiers to have similar performances. Although it is defined here at the hypothesis level, it could easily be applied at the word level. However, BAYCOM does not address the underlying issue of poor confidence scores from the individual recognisers.

3.3.1.5 Frame-level Posterior Combination

The schemes described above combine posterior probabilities at the word or utterance level using a minimum Bayes' risk scheme. In general, combining posteriors at other levels is not normally carried out for ASR within the HMM framework of section 2.1. An alternative framework, the hybrid HMM/neural-network framework [112] allows posteriors to be easily combined at the frame level. In this framework, a number of neural networks are first trained to output frame level posteriors for each of the phones in the system. These phone posterior probabilities are then used as input to an HMM decoder. Hence, there is the possibility to combine the outputs from multiple neural networks before they are used as input to the HMM stage. One example of this form of posterior combination at the frame level is described

in [109]. In this system, multiple neural networks are used, each corresponding to different feature streams. A weighted combination of the posterior probabilities from the individual neural networks, as in equation 3.3, are then used as input to the HMM decoder. The weights are calculated using an inverse entropy measure; this measure is motivated because neural networks with high entropy have less discriminative power, and therefore should have a lower weighting. Other forms of posterior combination, such as discriminative model combination below, could also be used.

3.3.1.6 Discriminative Model Combination

Discriminative Model combination is a framework for weighted log-linear combination of models in a speech recognition system [9]. This framework generalises the posterior probability to a log-linear distribution. For combination of acoustic model log posteriors at the hypothesis level, the combined posterior becomes

$$P(\mathcal{H}|\mathcal{O},\mathcal{M}^{(1)}\cdots\mathcal{M}^{(S)}) = \frac{\exp\left\{-\sum_{s=1}^{S}-\lambda_s \log P(\mathcal{H}|\mathcal{O},\mathcal{M}^{(s)})\right\}}{\sum_{\tilde{\mathcal{H}}}\exp\left\{-\sum_{s=1}^{S}-\lambda_s \log P(\tilde{\mathcal{H}}|\mathcal{O},\mathcal{M}^{(s)})\right\}} \tag{3.21}$$

For a combination of language model or acoustic likelihood scores, the acoustic model posterior $P(\mathcal{H}|\mathcal{O},\mathcal{M})$ is replaced by the language model score $P(\mathcal{H})$ or the acoustic likelihood $P(\mathcal{O}|\mathcal{H},\mathcal{M})$, and the combination can be done at any level, e.g. sentence, word, phone. The weights, λ_s are estimated on a held-out set using a discriminative criterion to minimise word error rate. This form of combination could be used instead of the weighted posterior combination in equation 3.3, for example in word-level combination with CNC. However, difficulties arise when the probabilities to be combined are zero, as the log score becomes negative infinity. To avoid this problem, probabilities can be floored.

Discriminative model combination has previously been used for combining language models [10, 87] and acoustic models based on different feature sets [110, 169] and model topologies [10].

3.3.2 Distributional Combination Schemes

The previous section detailed various approaches to combining systems at the hypothesis level. An alternative approach is to consider combining distributions, either by combining likelihoods or by combining distribution parameters. The products and mixtures of experts discussed in section 3.2.3 are general examples of this form of combination. This section discusses synchronous and asynchronous combination, which are the general forms of model for combining distributions within an HMM-based ASR framework. Section 3.3.3 discusses forms of distributional combination where the approach is motivated by the nature of the likelihood combination.

The first form of combination process is a synchronous stream system, where at time t the multiple HMMs are constrained to be in the same state. The dynamic Bayesian network for a two stream synchronous system is shown in figure 3.4(a), and the likelihood of the observation is determined from the parameters of the individual state distributions.

$$p(\boldsymbol{o}_t|\boldsymbol{\theta}_j,\mathcal{M}^{(1)}\cdots\mathcal{M}^{(S)}) = p(\boldsymbol{o}_t|\theta_j^{(1)}\cdots\theta_j^{(S)},\mathcal{M}^{(1)}\cdots\mathcal{M}^{(S)}) \tag{3.22}$$

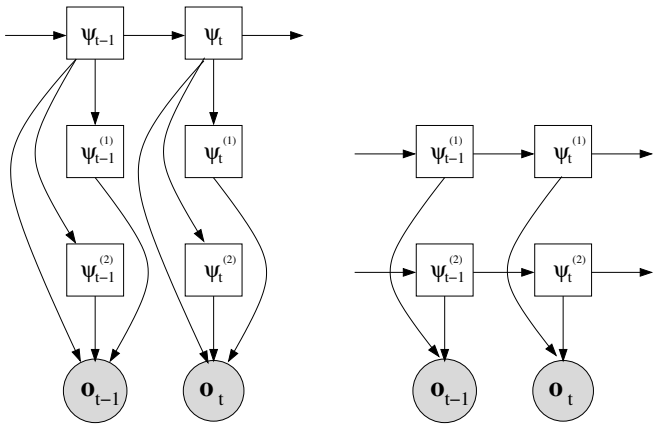
so the meta-state of the effective HMM, $\boldsymbol{\theta}_j$, is defined by the corresponding states of the individual systems $\boldsymbol{\theta}_j = \{\theta_j^{(1)} \cdots \theta_j^{(S)}\}$.

For the asynchronous system, shown in figure 3.4(b), there are two independent switching state processes. That is, the two HMMs are not restricted to be in the same state at time t. The meta-state $\boldsymbol{\psi}_t$ is defined by the hidden states of the individual systems, $\{\psi_t^{(1)} \cdots \psi_t^{(S)}\}$. The transition probability from one meta-state to another is determined by

$$P(\boldsymbol{\psi}_t|\boldsymbol{\psi}_{t-1}) = \prod_{s=1}^{S} P(\psi_t^{(s)}|\psi_{t-1}^{(s)}, \mathcal{M}^{(s)}) \tag{3.23}$$

as the multiple state switching processes are assumed to be independent. Also, the likelihood is now a function of the parameters of $\psi_t^{(1)} \cdots \psi_t^{(S)}$, where each independent state sequence is hidden

$$p(\boldsymbol{o}_t|\boldsymbol{\psi}_t, \mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}) = p(\boldsymbol{o}_t|\psi_t^{(1)} \cdots \psi_t^{(S)}, \mathcal{M}) \tag{3.24}$$



(a) Synchronous System (b) Asynchronous System

Figure 3.4: *Dynamic Bayesian Network Representation of an HMM*

Any combination of model parameters or likelihoods can be used, provided the resulting distribution is normalised to be a valid probability distribution. For a synchronous combination process, where the combination is done at the state level, the resulting output distribution must be valid

$$\int_{\mathcal{R}^d} p(\boldsymbol{o}_t|\theta_j, \mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}) d\boldsymbol{o}_t = 1 \tag{3.25}$$

and for an asynchronous combination, the condition to be satisfied is

$$\int_{\mathcal{R}^{Td}} p(\mathcal{O}|\mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}) d\mathcal{O} = 1 \tag{3.26}$$

where $\mathcal{O} = \{\boldsymbol{o}_1 \cdots \boldsymbol{o}_T\}$.

The factorial HMM [52] is an asynchronous combination of models where the mean of a meta-state is the sum of means of the individual model states. When each state output distribution is a single Gaussian, the likelihood becomes

$$p(\boldsymbol{o}_t|\boldsymbol{\psi}_t, \mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}) = \mathcal{N}(\boldsymbol{o}_t; \sum_{s=1}^{S} \boldsymbol{\mu}_{\psi_t^{(s)}}, \boldsymbol{\Sigma}) \tag{3.27}$$

where the covariance matrix is constrained to be the same for all components. The factorial HMM has the same problem as the multiple asynchronous stream systems of section 3.3.3.3, where the size of the meta-state space increases exponentially with the number of experts, thus complicating the training and decoding.

Other forms of synchronous and asynchronous combination may use interpolations between the state parameters. For example, using interpolation weights determined by the state of the first stream [40]. The following section details synchronous and asynchronous likelihood combination schemes in more detail.

3.3.3 Likelihood Combination Schemes

The distribution combination framework of the previous section is a general approach to combining distributions in an HMM-based system. However, many of the approaches which fall into this category were motivated by the form of likelihood combination. Two standard approaches are to use a mixture, or union, of the likelihoods [106, 144], and a product, or intersection, of the likelihoods [44, 70]. Finally, multiple stream systems are described in this likelihood combination framework. While these schemes are motivated from a likelihood combination perspective, they can also be described as parameter combination schemes, as the meta-component means and variances are simply functions of the individual expert means and variances.

3.3.3.1 Mixture Models for ASR

As previously discussed in section 3.2.3, a simple form of model combination is to use a mixture of distributions. This is extensively used for ASR in the form of a GMM for the HMM state output distributions, but can be used for the combination of arbitrary experts during decoding. For example, the weighted mixture of two experts at the state level is

$$p(\boldsymbol{o}_t|\theta_j, \mathcal{M}^{(1)}, \mathcal{M}^{(2)}) \quad = \quad (1-\lambda)p(\boldsymbol{o}_t|\theta_j, \mathcal{M}^{(1)}) + \lambda p(\boldsymbol{o}_t|\theta_j, \mathcal{M}^{(2)}) \tag{3.28}$$

where $\mathcal{M}^{(1)}$ and $\mathcal{M}^{(2)}$ are the individual models to be combined. This is referred to as a *union* of the two distributions. Any form of valid probability density function may be used, and the form of the two distributions need not be the same as long as they are valid probability distributions. Equation 3.25 will be satisfied if the individual mixture weights sum to one.

The extension of this form of model to the combination at the the HMM level is trivial and, provided the HMMs are valid probability models, the combined model will itself be valid. However, when using this union model for recognition, there are significant difficulties when combining at the HMM level. For the state level union, it is possible to directly use the Viterbi algorithm associated with the standard HMM recognition and combine likelihoods

from corresponding states in the multiple HMMs. When combining models at the HMM level the combination is complicated as there are multiple sets of hidden state sequences through the HMM. This could be addressed by expanding the recognition network so that it considers all possible state combinations between the models, and then performs recognition in that expanded space. This is similar to the generalised Viterbi decoding in [151].

3.3.3.2 Products of Experts

An alternative to the mixture model is to use the product of experts framework [45, 69]. Here the likelihoods from the models are producted together, effectively forming an intersection of the distributions. Again, considering the simplest case of only two experts and taking the product at the state observation level, the combination may be written as

$$p(\boldsymbol{o}_t|\theta_j, \mathcal{M}^{(1)}, \mathcal{M}^{(2)}) = \frac{1}{Z_{\theta_j}} \left(p(\boldsymbol{o}_t|\theta_j, \mathcal{M}^{(1)}) p(\boldsymbol{o}_t|\theta_j, \mathcal{M}^{(2)}) \right) \tag{3.29}$$

where the normalisation term, Z_{θ_j}, ensures that the resulting distribution is valid and equation 3.25 is satisfied. In general it may not be possible to find an analytical expression for Z_{θ_j} as the integral over the product of arbitrary experts is likely to be intractable. However, as the product of two Gaussians is itself a Gaussian, and consequently the product of GMMs is also a GMM. It is then possible to find Z_{θ_j} analytically for products of Gaussians and products of GMMs [44]. The product of GMM experts is itself a GMM, but now it is necessary to sum over all *meta-components*. These meta-components are created by taking all possible component pairings between the base GMM components, and leading to are $M^{(1)}M^{(2)}$ meta-components. The advantage of a product framework is that a more complex distribution can be modelled with fewer parameters. For the GMM, $M^{(1)}M^{(2)}$ components can be modelled with $O(M^{(1)} + M^{(2)})$ parameters. For S GMM experts:

$$p(\boldsymbol{o}_t|\boldsymbol{\theta}_j, \mathcal{M}) = \frac{1}{Z_{\theta_j}} \prod_{s=1}^{S} p(\boldsymbol{o}_t|\theta_j, \mathcal{M}^{(s)}) \tag{3.30}$$

$$= \frac{1}{Z_{\theta_j}} \sum_{\mathbf{m}} c_{\mathbf{m}} K_{\mathbf{m}} \mathcal{N}(\boldsymbol{o}_t; \boldsymbol{\mu}_{\mathbf{m}}, \boldsymbol{\Sigma}_{\mathbf{m}}) \tag{3.31}$$

where the summation is over all possible meta-components, $\mathbf{m}$ specifies the combination of components from each expert, and

$$K_{\mathbf{m}} = \frac{(2\pi)^{\frac{D}{2}} |\boldsymbol{\Sigma}_{\mathbf{m}}|^{\frac{1}{2}}}{\prod_{s=1}^{S} (2\pi)^{\frac{D}{2}} |\boldsymbol{\Sigma}_{jm}^{(s)}|^{\frac{1}{2}}} \exp\left(\frac{1}{2} \left(\boldsymbol{\mu}_{\mathbf{m}}^{\mathsf{T}} \boldsymbol{\Sigma}_{\mathbf{m}}^{-1} \boldsymbol{\mu}_{\mathbf{m}} - \sum_{s=1}^{S} (\boldsymbol{\mu}_{jm}^{(s)\mathsf{T}} \boldsymbol{\Sigma}_{jm}^{(s)} \boldsymbol{\mu}_{jm}^{(s)}) \right) \right) \tag{3.32}$$

$$\boldsymbol{\mu}_{\mathbf{m}} = \boldsymbol{\Sigma}_{\mathbf{m}} \left(\sum_{s=1}^{S} \boldsymbol{\Sigma}_{jm}^{(s)-1} \boldsymbol{\mu}_{jm}^{(s)} \right), \quad \boldsymbol{\Sigma}_{\mathbf{m}} = \left(\sum_{s=1}^{S} \boldsymbol{\Sigma}_{jm}^{(s)-1} \right)^{-1}, \quad c_{\mathbf{m}} = \prod_{s=1}^{S} c_{jm}^{(s)} \tag{3.33}$$

The normalisation term, Z_{θ_j}, is then:

$$Z_{\theta_j} = \sum_{\mathbf{m}} c_{\mathbf{m}} K_{\mathbf{m}} \tag{3.34}$$

Extending the framework from a product at the state distribution level to a product at the HMM level is complicated [18]. For the product at the model level it is necessary to write

$$p(\mathcal{O}|\mathcal{M}^{(1)}\cdots\mathcal{M}^{(S)}) = \frac{1}{Z^{(T)}}\prod_{s=1}^{S} p(\mathcal{O}|\mathcal{M}^{(s)}) \tag{3.35}$$

where now the normalisation term is dependent on the sequence length, T, and thus is complicated to calculate.

3.3.3.3 Multiple Streams and the Mixed Memory Model

Closely related to the product of HMM framework is the multiple stream framework [163]. Here at a state level, each of the *streams* of the observation vector are modelled separately. The separate streams can consist of different sources of information, including static and dynamic parameters [163], frequency bands [12, 146] or multiple sources of information such as speech and visual information [145]. The original feature vector may be rewritten as a concatenation of the feature vectors from each of the S streams:

$$\boldsymbol{o}_t = \begin{bmatrix} \boldsymbol{o}_t^{(1)} \\ \vdots \\ \boldsymbol{o}_t^{(S)} \end{bmatrix} \tag{3.36}$$

where $\boldsymbol{o}_t^{(s)}$ is the feature vector associated with stream s. If each of the streams are assumed to be conditionally independent given the state, then the state likelihood may be expressed as

$$p(\boldsymbol{o}_t|\boldsymbol{\theta}_j, \mathcal{M}^{(1)}\cdots\mathcal{M}^{(S)}) = \prod_{s=1}^{S} p(\boldsymbol{o}_t^{(s)}|\theta_j^{(s)}, \mathcal{M}^{(s)}) \tag{3.37}$$

For these multiple stream systems it is not necessary to include a normalisation term provided that each of the stream distributions is a valid distribution, as each stream is assumed to be independent. Multiple streams can also be used at the model level, yielding *asynchronous multiple stream* systems [116]. In this case there is again the issue of a more complex decoding scheme, but the generalised Viterbi decoding scheme in [151] may be used as there is no normalisation term. The use of this multiple stream framework has not yielded gains for speech recognition systems [44] due to the independence assumption.

A generalisation of the multiple streams for combining HMMs is the mixed memory model [117, 133]. A number of *meta-states* are composed of the individual states of the HMMs. Considering a particular *meta-state* at time t, $\boldsymbol{\psi}_t = \left\{\psi_t^{(1)}, \ldots, \psi_t^{(S)}\right\}$ where $\psi_t^{(s)}$ is the state of stream s at time time, the mixed memory model likelihood is given by

$$p(\boldsymbol{o}_t|\boldsymbol{\psi}_t, \mathcal{M}^{(1)}\cdots\mathcal{M}^{(S)}) = \prod_{s=1}^{S}\left(\sum_{u=1}^{S}\lambda_u^{(s)} p(\boldsymbol{o}_t^{(s)}|\psi_t^{(u)}, \mathcal{M}^{(u)})\right) \tag{3.38}$$

where the stream weights, $\lambda_u^{(s)}$, satisfy

$$\sum_{u=1}^{S} \lambda_u^{(s)} = 1, \quad \lambda_u^{(s)} \geq 0 \tag{3.39}$$

ensuring that the constraints in equation 3.26 are satisfied. The independent stream system is the specific case where $\lambda_u^{(u)} = 1$. This form of representation allows a coupling between the streams, but has not yielded gains for speech recognition [117].

3.3.4 Implicit Combination Schemes

One final class of combination scheme for ASR is an implicit combination, where multiple models are used in turn, with later models refining the output from previous models. This form of combination can easily be incorporated into the multi-pass framework of section 3.1. An example of this combination is acoustic codebreaking, discussed in section 4.2.4, where a second model is used to correct potential confusions made by the first. For example, [152] uses support vector machines to correct potential binary confusions. Other forms of implicit decoding, N-best and lattice rescoring, and cross-adaptation, are discussed in this section.

3.3.4.1 N-best and Lattice Rescoring

One form of combination which arises in the multi-pass decoding framework is N-best or lattice rescoring. For example, in figure 3.1, the second pass, P2, generates lattices which are rescored by multiple systems in passes P3a to P3x. Lattice rescoring can be done for either or both of the language and acoustic model scores, using different acoustic and language models. N-best [1, 76] and lattice [48, 62, 100, 126] rescoring are commonly used for LVCSR multi-pass decoding.

The advantage of N-best and lattice rescoring is that it is much faster than standard decoding, as there is no need to search over all possible hypotheses. Thus, it is possible to use complex models for rescoring, provided the N-best list or lattice is a good representation of the most likely hypotheses.

However, there are issues with this form of combination. The initial lattices generated must be representative of the most likely hypotheses, otherwise the rescoring from the second system is poor. This can be problematic if the two systems are very different, so the output lattice from the first system is not representative of the hypothesis space of the second. Additionally, a larger lattice slows rescoring, and so the level of word errors introduced by a small lattice must be balanced against the speed of rescoring.

3.3.4.2 Cross Adaptation

Another, more indirect, approach to system combination is cross-adaptation [48, 62, 126, 142]. This is a scheme which naturally arises in the multi-pass adaptive framework from section 3.1, where the output transcriptions from one system are used in the subsequent pass as the input hypothesis to perform unsupervised speaker adaptation of a second system. For example, in figure 3.1, different models can be used to obtain an initial transcription in the P2 pass, and to do the final rescoring stage in the P3 pass.

In practice, this form of system combination has led to improvements in performance. Cross-adaptation can be used in addition to any of the system combination methods discussed above, and with any form of model.

3.4 IDEAL combination

The output from system combination is one measure of how complementary two systems are, but it relies on the effectiveness of the combination method as well as the diversity of the systems. It is useful to consider a measure of *potential* performance in combination, to better judge whether two systems are complementary, independently of the combination algorithm used. This section discusses such a measure, although its use of confusion networks means it is not entirely independent of the combination algorithm.

A typical measure for potential performance is the *oracle* performance, which makes use of the reference labels to calculate a lower bound on performance. For CNC, this would align the confusion networks with the reference transcription and return the best possible transcription, regardless of the word posteriors. However, this may make use of words with very low posteriors. Furthermore, the oracle performance will improve as pruning is decreased during the CN generation, and more words are retained in the confusion network. Hence, an oracle combination is not appropriate for assessing potential performance for CNC.

If it is possible to accurately predict when a classifier is correct, that is if an accurate and reliable confidence measure exists, then it is possible to only combine with a second system for confusion segments when the first system is incorrect. IDEAL combination uses the reference transcription to identify word errors, and hence calculate a measure of potential performance. This form of combination is necessarily biased towards the reference, but shows the potential gains that could be achieved from CNC if it were possible to detect all the word errors accurately.

Figure 3.5 shows this combination in more detail. The confusion networks from two systems S0 and S1 are aligned with the reference transcription, and the bold words indicate those with highest posterior probability. For the segments where the best word corresponds with the reference, the second system is ignored, as shown by the dashed arcs in the confusion network for system S1. It is only for the remaining words that the combination is performed. Thus, in figure 3.5, only two of the confusion segments from the second system are combined with the first.

This combination scheme is not ideal in the sense of selecting the best possible hypothesis from the output hypotheses of the two systems, S0 and S1. Instead, it selects the best possible hypothesis from the first system and the combination of the two systems, S0 and S0+S1. This better matches the training approach used in this thesis, where the second system is trained for combination with the first. Also, the IDEAL combination gives a lower bound on error rate that could be achieved from combining the two systems, if it is possible to accurately detect word errors in decoding, as is done in training.

IDEAL combination can easily be extended to more than two systems. The multiple confusion networks are first aligned against each other and the reference. For a particular confusion segment, if the first system is correct then any later systems are ignored. If the first system is incorrect, then combination with the second system is performed and the new hypothesis compared against the reference. If this is correct, then the subsequent systems are ignored, or else combination with the third system is performed. This process is continued for

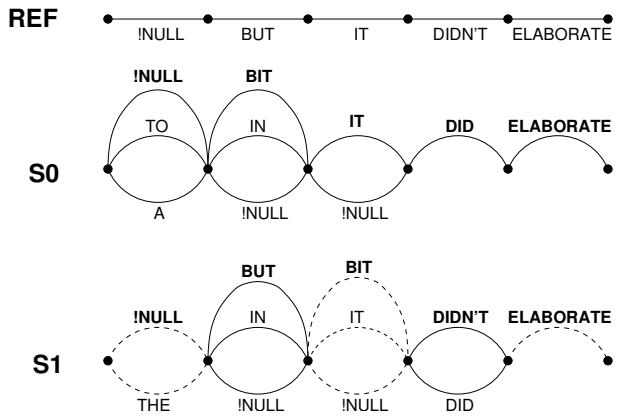


Figure 3.5: *IDEAL combination, only the solid CN arcs from system S1 are used in combination as these correspond to segments where the first system is incorrect*

any number of systems. However, as the number of systems increases, the IDEAL combination is likely to improve due to the additional diversity.

In practice, if a confidence measure is used to identify word errors rather than the reference transcription, then some segments will be falsely identified as errors. Of these, performing the combination may change the best word and introduce an error where there was none previously. Also, of those confusion segments correctly identified as errors, not all will be corrected by combining with the second system. Hence, for combination, there are only a subset of words or confusion segments where correctly identifying errors is important in terms of effect on the final word error rate. This is discussed in more detail in section 7.3.2 for the combination of complementary systems. This form of combination is related to acoustic codebreaking, discussed in section 4.2.4, as a second model is used to resolve only the confusions in the first.

3.5 Summary

This chapter has considered the task of combining multiple systems for ASR. First, a multi-pass combination framework for large vocabulary speech recognition was introduced, before general methods for combining arbitrary posteriors and likelihoods were presented. Next, approaches for combining recognition hypotheses, likelihoods, and model parameters for ASR were discussed. These approaches combine posteriors or distributions from decoding, or else an implicit combination can be performed where a second system refines the output from the first. To conclude, a discussion of an ideal combination scheme was presented, with respect to word-level confusion network combination.

In general, the distributional combination schemes require more complex decoding, and as seen in the following chapter, training algorithms. In contrast, the hypothesis combination schemes require an independent decoding pass for each system, before the outputs are combined. Thus, they need little change to the standard decoding procedure, and fit easily within the multi-pass combination framework described at the beginning of the chapter.

Chapter 4
Generating Complementary Systems

The previous chapter discussed several methods for combining systems, both in the general case and specifically for ASR. However, the combination of multiple systems is not useful unless they are complementary, i.e. make different errors. There are many approaches to building complementary systems, but, as for the combination of systems, many approaches are not directly applicable to ASR. This chapter reviews some of the existing approaches to generating complementary systems, for general tasks and also for ASR.

4.1 General Approaches

Ensembles of models for machine learning tasks has been a focus of research for several years [29], and approaches to building complementary systems fall broadly into two categories. First, diverse systems are somehow generated, and it is hoped they will have different errors. For ASR, examples of this approach are given in section 3.1, and include altering the training algorithm, frontend, and covariance modelling. Second, complementary systems can be explicitly trained. This can be done either by iteratively building multiple complementary systems, or by training the complementary systems in parallel. Approaches to generating complementary systems for a general task are discussed in this section.

4.1.1 Injecting Randomness

Aside from differences in the training approach, such as different forms of classifier, features or training algorithm, a straightforward way to build diverse systems is to introduce randomness into the training algorithm [29]. This can be done, for example, by adding noise onto the training data, initialising the parameters randomly, or by selecting random subsets of the training data. The latter method is known as bagging [14]. These are all general methods that can be used regardless of the system being built and the training algorithm.

If the specific classifier in question is a decision tree, then another approach to inject randomness is to grow the tree in a random manner, by randomly choosing a split from the top N, rather than grow the tree by choosing the best split each time. Repeated application of this algorithm builds a random forest [13]. The method works well as the splitting in a decision tree is a locally optimal split, and so very different decision trees can be built by small changes in the algorithm.

Injecting randomness has the advantage that any number of diverse systems can be built, although the process isn't deterministic and hence not repeatable. Some control over the degree of randomness can allow for building systems that are close to, or very different from, the starting system.

When diverse systems are built independently of each other, either by introducing randomness or by using alternative training approaches, any algorithm can be used for combining the systems. This differs from the training algorithms discussed below, where the combination scheme is specified by the training algorithm. Furthermore, existing classifiers and training algorithms do not need to be significantly altered, and the only additional cost is that of model combination. However, it is not guaranteed that the diverse systems will be complementary.

4.1.2 Boosting

Boosting, or leveraging, is a standard machine learning approach that allows complementary systems to be generated for a binary classification task, and has been extended to multiclass problems. It takes advantage of the fact that a combination of weak classifiers, i.e. those which perform slightly better than random, can perform as well as a single strong classifier [134]. In its strict definition, boosting defines a training procedure for building a set of weak classifiers, each of which perform slightly better than random, and a final classification for combining these classifiers using a weighted voting scheme, similar to those described in section 3.2.

AdaBoost [37, 38] is the most widely used boosting algorithm, and a multiclass version, known as AdaBoost.M2, is given in figure 4.1. This algorithm assumes a finite number of classes $\{C_1 \cdots C_V\}$, and a model which generates posterior probabilities of class membership $P(C_v|\boldsymbol{o}, \mathcal{M})$. If these posteriors are equal for all classes, then the model is uninformative. If, however, the posteriors are not equal, then the model $\mathcal{M}$ performs better than random and hence can be used as a weak learner in the boosting algorithm.

A distribution over all the training samples is maintained, $\boldsymbol{d} = \{d_1 \cdots d_N\}$. For those regions of space that are correctly classified the distribution weight is reduced, and for those that are incorrectly classified the weight is increased. As training progresses, this distribution evolves to make later classifiers focus on "difficult" training examples, i.e. data that the previous classifiers classified incorrectly. As part of this process the classifier importance, β_s for each classifier s is also computed.

Input:

A set of N labelled training sample pairs $\{\mathbf{o}_i, \mathcal{H}_i\}$ where $\mathcal{H}_i \in \{C_1 \cdots C_V\}$

Initialise:

Set the initial distribution weights $\boldsymbol{d}^{(1)}$ so $d_i^{(1)} = \frac{1}{N}$

For: s= 1:S

Train the classifier parameters, $\mathcal{M}^{(s)}$, with respect to the distribution $\boldsymbol{d}^{(s)}$
Obtain the posterior probabilities $P(C_v|\boldsymbol{o}_i, \mathcal{M}^{(s)})$
Calculate the pseudo-loss, ϵ_s, for the model $\mathcal{M}^{(s)}$

$$\epsilon_s = \frac{1}{2}\sum_{i=1}^{N} d_i^{(s)} \left(1 - P(C_i|\boldsymbol{o}_i, \mathcal{M}^{(s)}) + \sum_{v=1, v\neq i}^{V} P(C_v|\boldsymbol{o}_i, \mathcal{M}^{(s)})\right)$$

Set the classifier importance parameters, β_s,

$$\beta_s = \frac{\epsilon_s}{1-\epsilon_s}$$

Update the weight distribution function, $\boldsymbol{d}^{(s+1)}$

$$d_i^{(s+1)} = \frac{d_i^{(s)}}{Z_s}\beta_s^{\frac{1}{2}\left(1-P(C_i|\boldsymbol{o}_i, \mathcal{M}^{(s)})+P(C_v|\boldsymbol{o}_i, \mathcal{M}^{(s)})\right)}$$

where Z_s is a normalisation term so that the weights sum to one

$$Z_s = \sum_{i=1}^{N} d_i^{(s)}$$

Output: the final hypothesis

$$\hat{\mathcal{H}} = \text{argmax}_{\mathcal{H}} \left\{\sum_{s=1}^{S} \log\left(\frac{1}{\beta_s}\right) P(\mathcal{H}|\boldsymbol{o}, \mathcal{M}^{(s)})\right\}$$

Figure 4.1: *The multiclass AdaBoost.M2 algorithm*

The pseudo-loss, ϵ_s, is a discriminative measure of error which takes into account all classes $C_1 \cdots C_V$. An additional class weighting can be included in the third term of the pseudo-loss, to assign differing weights to different misclassifications, but is ignored here for clarity.

In the final step, AdaBoost combines the individual classifiers by summing their probabilistic predictions, using a weighting based on the classifier importance. This weighted voting scheme is the weighted posterior combination scheme described in section 3.2.2, where the classifier weights, λ_s, are given by

$$\lambda_s = \log\left(\frac{1}{\beta_s}\right) \tag{4.1}$$

Training the model parameters $\mathcal{M}^{(s)}$ with respect to the distribution can be be done in two ways. Either the weighting can be taken into account directly as part of the training algorithm, or a subset of training data can be sampled using the distribution and used for estimating the parameters. In the former, the modifications to the training algorithm depend on the form of classifier. The latter approach to data weighting differs from bagging, discussed above in section 4.1.1, as the subsets of training data are selected according to the distribution $\boldsymbol{d}^{(s)}$, not randomly.

Boosting is an algorithm where classifiers are trained iteratively, and hypotheses from existing classifiers are used to train later systems. Thus, the scheme naturally uses a hypothesis combination scheme in the final stage, rather than likelihood or parameter combination.

Where there is low classification noise on the training set, i.e. the training labels $\mathcal{H}_i$ are accurate, boosting outperforms randomisation as a method for generating complementary systems. In high classification noise however, the reverse is true as the later classifiers built by the boosting algorithm begin to focus on labelling errors rather than on data which is truly hard to classify. In contrast, bagging and randomisation perform much better in this situation because the randomness overcomes the classification noise [30].

4.1.3 Simultaneously Training Multiple Systems

In cases where there are few classifiers, the task is small or the classifiers are simple, it can be possible to simultaneously train an ensemble of complementary classifiers. For example, mixtures of experts, such as a GMM, can easily be trained using the EM algorithm or gradient descent to optimise the log-likelihood of the data.

Products of experts can also be trained in parallel. To maximise the log-likelihood using gradient descent, the gradient of the log-likelihood function is needed

$$\begin{aligned}\frac{\partial \mathcal{F}_{ML}(\mathcal{M})}{\partial \mathcal{M}} &= \frac{\partial}{\partial \mathcal{M}}\left\{\log \frac{1}{Z}\prod_{s=1}^{S}\lambda_s p(\boldsymbol{o}|\mathcal{H},\mathcal{M})\right\} \\ &= \frac{\partial}{\partial \mathcal{M}}\left\{\log \frac{1}{Z}\right\} + \sum_{s=1}^{S}\frac{\partial}{\partial \mathcal{M}}\log \lambda_s p(\boldsymbol{o}|\mathcal{H},\mathcal{M})\end{aligned} \tag{4.2}$$

The second term of this gradient can be found analytically, so the main complication is in estimating the first term, which involves the differential of Z with respect to the model parameters. Z is the integral to ensure that the product of experts is a valid distribution

$$Z = \int_{\mathcal{R}^D} \prod_{s=1}^{S} \lambda_s p(\boldsymbol{o}|\mathcal{H}, \mathcal{M}^{(s)}) d\boldsymbol{o} \tag{4.3}$$

For training products of GMMs, this normalisation term can be found analytically, as discussed in section 3.3.3.2. Their training is discussed below in section 4.2.3.1. In general, for products of arbitrary experts, it is not possible to analytically find this term and so an approximation must be used. One approach is to use numerical integration to estimate Z and its gradient, while a second approach is to estimate the gradient using sampling techniques such as Gibbs or Monte Carlo Markov Chain sampling [103]. However, these methods can be slow to converge and require many samples. Contrastive divergence training [20, 69] approximates the estimate of the gradient after many MCMC samples with the estimate after just one sample.

Contrastive divergence can be used for estimating the parameters of products of HMMs [18, 102]. However, as the normalisation term is needed in decoding for ASR, and is not calculated as part of the contrastive divergence optimisation, this method is not applicable to ASR.

A different approach to training models in parallel is to hypothesise an objective function which includes the parameters from multiple systems. For example, [115] proposes to train multiple classifiers by minimising the correlation between their errors. If the errors made by two classifiers are independent, they are more likely to be complementary. The training minimises the covariance between the expected errors of two classifiers over the training set.

Simultaneously training multiple systems typically involves a combination at the likelihood or parameter level during training, and hence this is the form of combination which naturally arises when these systems are used in practice.

4.2 Methods in Automatic Speech Recognition

The methods described above for building complementary systems are not directly applicable to the task of ASR due to the dynamic nature of the data. Section 3.1 discussed an ad-hoc approach for generating complementary systems for ASR, by independently building a number of systems and selecting the ones which combine well together. This section discusses existing approaches to explicitly building complementary speech recognisers.

First, randomness and boosting-like algorithms for ASR are discussed. Next, methods for simultaneously building multiple systems for ASR are introduced. As these methods typically build multiple distributions, they naturally make use of the distributional combination schemes of the previous chapter. To conclude, a third approach is considered, acoustic code-breaking. This approach builds a second system for resolving specific confusions, and hence makes use of an implicit combination scheme.

4.2.1 Random Decision Trees

There are several options for introducing randomness into the training of an ASR system, such as those discussed above in section 4.1.1. However, HMM parameter estimation algorithms are reasonably robust to randomness, so an alternative is to introduce randomness into the decision tree algorithm by altering the question selection stage in the same way as when

building random forests [138, 162]. Instead of selecting the single best question, the tree is grown by randomly choosing a question from the best N questions [138] or by randomly selecting a subset of all available questions to build each random tree [162]. The difference between random trees for ASR and for general purpose classification is that the trees for ASR are used for clustering HMM states, and not for classification itself. Hence random decision trees for ASR are an indirect way to introduce randomness into the training.

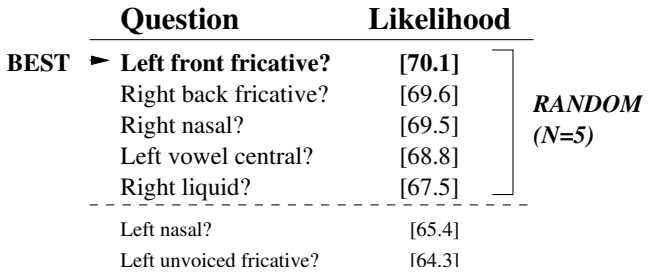


Figure 4.2: *Random tree question selection*

An example of the random tree question selection in [138] compared to the standard approach is given in figure 4.2. Rather than select the locally optimal question "*Left front fricative?*", a random choice from the top 5 is made. Thus the decision tree algorithm becomes:

1. **Statistics**
   - Obtain statistics for seen triphone contexts using the forward-backward algorithm
2. **Question Selection**
   - Recursively build the tree by *randomly selecting from the top N questions* which give the highest change in data likelihood
3. **Stopping criterion**
   - Stop building the tree when the data likelihood falls below a threshold

A randomised decision tree does not guarantee that the resulting systems are complementary, but using multiple systems built on random trees makes it more likely that confusable states will be separated in at least *some* of the trees. Hence, it is anticipated that the systems will make different errors and a combination of outputs from systems built on random trees will give improvements. The problem that systems should have comparable error rates is no longer an issue as altering the value of N provides some control over individual system performances. A larger value of N will tend to lead to trees being very different from the best tree, and it is likely that system performance will degrade. However, as there is no guarantee of obtaining complementary systems, random decision tree systems suffer from the problem that the optimal order of combination cannot be predicted and hence all system combinations must be performed in order to find the best. This is typically addressed by building many systems based on random trees and combining them all together [74, 126].

When training complementary systems, a form of system combination is often used in the training algorithm. Thus, it is expected that the same form of combination is used in

decoding for combining the systems, to obtain the best match between training and testing. When building random decision trees, no form of system combination is used, and hence any of the methods from chapter 3 are appropriate for combination. Systems built with random trees have previously been combined using hypothesis combination schemes within a multi-pass framework, such as ROVER or CNC [74, 126], or by combining acoustic scores during decoding [162].

4.2.2 Boosting for Automatic Speech Recognition

There are a number of issues with applying boosting, or boosting-like, algorithms to speech recognition. Speech has a large number of classes, the forms of classifier used are highly complex, the data is dynamic and, finally, the simple weighted voting scheme is not directly applicable. Much of the previous work on boosting for speech recognition has recast the problem as a phone classification task, and hence avoided some of the problems outlined above. For example, [170] builds GMMs for each phone and performs boosting at the frame level, [135] applies boosting to the Neural Network part of a hybrid HMM/NN system, and [31] applies boosting to whole-phone HMMs. For classification at the phone level, the models can be combined using a straightforward voting scheme. For the model combination in [31], the boosted HMMs are recombined by effectively treating them as different pronunciations of the same phone in decoding, with transition probabilities based on the classifier importance, β_s, of each model.

If the task is triphone classification, it may still be time consuming to calculate the pseudo-loss, but approximations can be used instead. For example, in [170], where boosting is applied to the task of frame level triphone classification, two approximations to the pseudo-loss are used. One restricts the number of phones over which the pseudo-loss is calculated to be a small set of most likely phones. The second partitions the phones into clusters, and only uses boosting within the clusters, thus limiting the number of alternatives in the pseudo-loss calculation.

Boosting-like algorithms for continuous speech recognition can be applied at different levels, for example at the speaker, utterance, frame, or word level. Previous work has implemented boosting at the utterance [108, 166], and at the frame level [167].

There are a number of issues which must be addressed before boosting-like schemes can be applied to continuous speech recognition. First, for these schemes, the pseudo-loss must be approximated. This can easily be done using utterance or frame level posterior probabilities, which are obtained from normalising the scores in an N-best list or a lattice. The restricted hypothesis space allows fast computation of the pseudo-loss by considering only the most likely alternative hypotheses. This will give an underestimate of the pseudo-loss, but should be a good approximation if the hypothesis space is representative.

Also, as boosting weights the training data to reflect previous errors, a weighted training scheme must be implemented such as that used for active training. In practice, provided the data can be weighted appropriately, it is trivial to modify the EM algorithm to take into account a weighting over the training data, whether it is at the utterance or the frame level. Data weighting is described in more detail for ASR in chapter 5.

Finally, the combination scheme must be considered as the simple voting scheme used by AdaBoost is not applicable to continuous speech recognition. [108] treats the models as pronunciation variants during decoding, with pronunciation probabilities derived from the model weights calculated during boosting. However, this approach increases the search space

for decoding, and can significantly slow the system down. Alternatively, a voting scheme for word-level combination like ROVER, as in [166], or CNC could be used, and this reflects the posterior voting scheme used for standard boosting in section 4.1.2.

The previous work on boosting for ASR has mainly used the AdaBoost algorithm in [37]. A generalised boosting algorithm, AnyBoost, was used with minimum classification error (MCE) in [168]. This algorithm allows an arbitrary loss function to be used in the training of ensembles.

4.2.3 Simultaneously Training Multiple Systems for ASR

Due to the complex nature of HMMs and speech data, it is not common for multiple systems to be trained in parallel, as for the general case in section 4.1.3. In particular, for the asynchronous decoding scheme discussed in section 3.3.2, the extended state space complicates the training. With a synchronous decoding scheme however, the state space does not increase exponentially and so it is possible to train distributions that are combined at the state or HMM level. This section discusses a state level product of experts, and the approximations needed for training an asynchronous system.

4.2.3.1 Products of GMMs

Mixtures of GMMs are commonly used as HMM state output distributions, and are trained using the EM algorithm. It is simple to extend this algorithm for training products of GMMs

$$p(\boldsymbol{o}|\mathcal{H}, \mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}) = \frac{1}{Z} \prod_{s=1}^{S} \left\{ \sum_{m=1}^{M} c_m^{(s)} \mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}_m^{(s)}, \boldsymbol{\Sigma}_m^{(s)}) \right\} \tag{4.4}$$

where $\boldsymbol{\mu}_m^{(s)}$, $\boldsymbol{\Sigma}_m^{(s)}$ and $c_m^{(s)}$ are the parameters of the mth component of the Sth GMM.

As mentioned in section 4.1.3, the normalisation term Z of an arbitrary product of experts is necessary for training and decoding in ASR. For the specific case of products of GMMs, it is possible to find an analytical expression for the normalisation term [44, 45], as discussed in section 3.3.3.2. Hence, it is possible to extend the EM algorithm described in section 2.4.1 to estimate the parameters of the product model. The E-step of the EM algorithm remains the same, while the M-step is altered to estimate the parameters of the product model.

4.2.3.2 Factorial HMMs

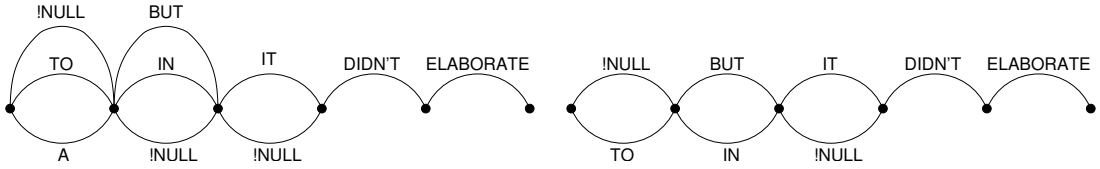
For the factorial HMM, and the other asynchronous likelihood combination schemes discussed in section 3.3.2, the naive EM algorithm requires translating the combined HMM into a single model where the effective number of states is $\prod_{s=1}^{S} M^{(s)}$. For large models, as this effective number of states increases, the E-step in the EM algorithm becomes intractable. To overcome this problem, approximate inference schemes based on sampling and variational approaches have been proposed [52, 103] to estimate the state posteriors, γ_j. Sampling methods can converge arbitrarily close to the exact values, although convergence can be slow. Variational schemes instead optimise a lower bound on the likelihood. Provided the state output distributions are of a form that can be optimised, such as Gaussians, GMMs or products of GMMs, then the exact M-step of the EM algorithm for asynchronous combination schemes is tractable.

Although it is possible to approximate the training for asynchronous models, the normalisation term is still required for decoding. Additionally, the extended state space slows decoding, and so these models are typically not used in practice.

4.2.4 Acoustic Code Breaking

With standard training and decoding, a single model is required to discriminate between all possible hypotheses. A typical speech recogniser is normally able to discriminate more easily between certain classes of hypothesis than others and hence it may be difficult to robustly train a system to correctly classify all data. However, a well-trained speech recogniser is normally able to handle the majority of the data. Rather than addressing this using boosting style approaches described above, a form of minimum Bayes decoding may be used. Here, a first pass with a general model is used to obtain an initial hypothesis, and a second pass is used only to resolve potential confusions in this hypothesis [152].

In this framework, it is now possible to use a distinct model to resolve each confusion. For example, one model might resolve confusions between the words '*has*' and '*had*', while another model might resolve confusions between '*had*' and '*have*'. Any number of models can be built to resolve common confusions.



(a) Confusion Network (b) Binary confusion network

Figure 4.3: *Pruning the Confusion Network for Acoustic Codebreaking*

Various implementations of acoustic codebreaking have been investigated. Typically, these use a confusion network to identify potential confusions, and the confusion networks are often pruned to leave only binary confusions, as shown in figure 4.3. In this figure, the words 'A' and '!NULL' are pruned from the first two confusion segments, so that the maximum number of words left in a segment is two. Different forms of classifier have been used to do the acoustic codebreaking. In [33] whole word HMMs were used as the second pass classifier, discriminatively trained for each of the confusion pairs. In [152] a support vector machine using a generative score-space [139] was used. Finally in [92], an augmented statistical model [92, 139] was used as the second pass classifier.

The next consideration is how to combine the second classifier with the first. For the application of these systems to LVCSR, the posteriors from the initial pass can be combined with those from the second pass to obtain the final score [92]. This form of combination is closely linked with the weighted posterior combination and confusion network combination. If the second classifier is a binary classifier, such as a support vector machine, in [152], then the second pass classifier can select the best word from the confusion set. Acoustic code-breaking is thus similar to the implicit combination schemes discussed in section 3.3.4, and can be incorporated into a multi-pass framework.

The posteriors and other statistics from the first system can be used directly in training the second pass model, as in [92, 152], or the confusion sets could be used to segment the training set and extract the relevant observations for training the second system [33].

A problem with this acoustic code-breaking framework is that, for LVCSR systems, the number of confusable pairs is very large, but corresponds to a small fraction of the training set. Hence, there is limited impact on the final word error rate. For example in [92] this form of acoustic code-breaking was applied to a Conversational Telephone Speech task. Using the fifteen most commonly confused pairs less than 3% of the hypothesised words were rescored. To increase this percentage, the number of classifier pairs needs to be dramatically increased, but then there is the problem of little training data for robustly estimating the second pass models. However, for smaller vocabulary tasks, such as digit string recognition, this is a powerful framework for building complementary classifiers.

4.3 Summary

This chapter has discussed general methods for building complementary systems, and their application to ASR. These methods include building diverse systems by injecting randomness, iteratively training complementary systems within a boosting-like framework, simultaneous training of multiple systems, for example products of GMMs, and training classifiers to be used in an implicit combination scheme. For many of these schemes, the combination approach used in the training algorithm specifies the appropriate combination algorithm for combining the final systems.

Of the algorithms discussed in this chapter, boosting and acoustic codebreaking require an approach for identifying poorly classified portions of data. This can be done with an appropriate data weighting, as is discussed in the following chapter.

Chapter 5
Data Weighting for ASR

The previous two chapters reviewed existing approaches for combining and generating complementary systems for ASR. This chapter discusses previous work on confidence measures and existing schemes for applying a data weighting during training for ASR. Confidence measures are used for combining multiple systems, and data weighting is important in schemes like boosting and active training, where the training takes into account the errors made by existing systems.

This chapter also presents a new form of data weighting which is used as the basis for building complementary systems in this thesis. This is a generic word-level weighting, which is obtained from the posterior probabilities in multiple confusion networks. The weighting is independent of the individual model topologies, and its application does not rely on an alignment of words to frames.

5.1 Confidence Measures

The output from a speech recogniser normally contains errors, and there are many applications, including system combination, where it is useful to identify incorrectly recognised portions of data. For this purpose, the use of a *confidence measure* has been proposed [77]. Confidence measures are typically real-valued numbers in the range 0 to 1, and measure the recogniser's certainty in its hypothesis. Often they can be interpreted as the posterior probability that a word is correct, $P(\mathcal{W}|\mathcal{M},\mathcal{O})$. Confidence measures can be calculated at any level, but word and sentence level confidences are most often used.

For example, in dialogue systems, the user may be asked to clarify responses where the recogniser was uncertain, thus improving performance without asking the user to repeat themselves unnecessarily. For an ASR task, specific models might be used to resolve potential

errors, such as acoustic codebreaking discussed in section 4.2.4. Confidence measures can also be used in training, for example active training in section 2.4.3, to identify poorly modelled portions of training data. Accurate confidence measures are important when weighting data, both for combination of multiple systems and also for building complementary systems.

This section summarises existing work on confidence measures. These largely fall into three categories - estimating posterior probability, alternative confidence scores, and combining multiple scores to predict confidence. Of these, the posterior probability is most widely used in practice.

5.1.1 Estimating Posterior Probability

A natural measure of word or hypothesis confidence is the posterior probability. For the best word, $\hat{\mathcal{W}}$, this is $P(\hat{\mathcal{W}}|\mathcal{M},\mathcal{O})$. Speech recognisers typically output acoustic and language model likelihoods which can be used for comparing competing word sequences, but cannot be used for assessment of whether a hypothesis is correct. Likelihoods are also not easily compared across different model sets. Hence, likelihoods should be normalised before use as a confidence measure. Again, it is not possible to consider every alternative hypothesis, so a subset of hypotheses are used to perform the normalisation, in the form of an N-best list or a lattice [157]. Alternatively, a filler model can be used to account for the competing hypotheses [164].

Likelihoods in a lattice can be converted to posteriors using the forward-backward algorithm [157]. This allocates each lattice arc a posterior, which takes into account both the acoustic and language model score. In a lattice, there are many arcs corresponding to the same word that overlap in time, and so using the word arc posterior probability will underestimate the word confidence. A better measure of posterior probability might take all overlapping word arcs into account. For example, confusion networks [105] cluster overlapping arcs, and the time frame word error [158] takes multiple overlapping lattice arcs into consideration.

The accuracy of the posterior probability depends whether the lattice or N-best list is a good representation of the competing hypotheses, and also on the particular bias of the model set. Posterior probabilities are often unreliable due to the incorrect models used to estimate them, and different model sets can estimate posteriors with different distributions, making comparison of posterior probabilities across different systems difficult.

5.1.2 Alternative Confidence Scores

The posterior probability has proved successful as a confidence measure, though other approaches have been proposed. One alternative is to use the likelihoods directly in a word or utterance verification framework [130]. This is a likelihood ratio test $LRT(\hat{\mathcal{W}})$, in which the likelihood of the best hypothesis is tested against the likelihood of all competing hypotheses

$$LRT(\hat{\mathcal{W}}) = \frac{p(\mathcal{O}|\hat{\mathcal{W}},\mathcal{M})P(\hat{\mathcal{W}})}{\sum_{\tilde{\mathcal{W}}\in\boldsymbol{\mathcal{W}}} p(\mathcal{O}|\tilde{\mathcal{W}},\mathcal{M})P(\hat{\mathcal{W}})} \tag{5.1}$$

where $\boldsymbol{\mathcal{W}}$ is the set of competing words. In [130], a filler model was used to model competing hypotheses.

Another measure which makes use of the posterior probabilities and takes into account all competing words is the entropy H

$$H = -\sum_{\tilde{\mathcal{W}} \in \boldsymbol{\mathcal{W}}} P(\tilde{\mathcal{W}}|\mathcal{M}) \log P(\tilde{\mathcal{W}}|\mathcal{M}) \tag{5.2}$$

where again $\boldsymbol{\mathcal{W}}$ is the set of competing words from a lattice or a confusion network. The entropy measures the 'disorder' in the probability distribution over words. While the likelihood ratio test score and the entropy cannot be interpreted as probabilities, they can be mapped to a confidence measure which lies between zero and one.

Other measures extracted from a lattice include the hypothesis density and acoustic stability [85]. The former assumes that high confidence regions in the lattice have fewer arcs, while the latter measures the stability of the hypothesis as the weighting between language and acoustic model is changed.

A different approach is to consider the correlation or agreement between multiple models as a measure of confidence. If two models agree on the output, then it is more likely to be correct than if two models disagree. In [147], precision and recall are calculated for the words on which two classifiers agree, and found to be a reliable indicator of confidence.

5.1.3 Combining Multiple Scores

The measures discussed above make use of a single feature extracted directly from the recogniser output to estimate confidence. A final class of algorithm for confidence estimation extracts a number of features from various knowledge sources, and combines them together with a classifier to predict a confidence. The final task may be the binary classification task of word error detection [2, 64], the regression task of confidence or posterior estimation [55, 154], or the problem of optimal system combination [68, 147, 148], though the approach used is the same. A number of different classifiers have been considered, including:

- Linear discrimination [64]
- Logistic regression [2, 55]
- Decision trees [85, 111, 161, 165]
- Support vector machines (SVMs) [111, 147, 148, 165]
- Neural networks [85, 154, 165]

A feature vector $\boldsymbol{x}$ is extracted for each word or utterance. The individual features which make up $\boldsymbol{x}$ are extracted as part of the decoding, or in a post-processing step, and can combine multiple knowledge sources. The only restriction is that the features should have different distributions for correct and incorrect words. Many features have previously been tried, including

- Posterior probability [2, 111, 161]
- Entropy [2]
- Number of alternative words in the CN segment [2]
- Duration of word [2, 55, 111]

- Acoustic and language model scores [55, 64, 111, 154, 161, 165]
- Language model backoff mode [154, 161, 165]
- Contextual information [2]
- Part-of-speech of word [147, 148]

These machine learning techniques to combine multiple scores have not proven more useful than the posterior probability alone. [165] suggests this is because the features are highly correlated, and the posterior probability is typically calculated by using many of the individual features. Thus, the information contained in these features is somehow already encoded in the posterior probability. Some success was seen in [165] with features from a parser, though these features may not work as well in spontaneous speech due to the less grammatical utterances.

Chapter 11 investigates these techniques for combining complementary systems using logistic regression as a binary classifier. Logistic regression was used rather than more complicated classifiers such as support vector machines [150], as initial experiments showed it to perform better.

5.1.3.1 Logistic Regression

Logistic regression is a commonly used classifier for the case when there are two classes. It is a particular form of a generalised linear model where the data is assumed to be binomially distributed. For word error detection, the two classes are those of a word being correct C_c or incorrect C_e. Logistic regression then can be used to estimate $P(C_c|\boldsymbol{x})$, the posterior probability that a word is correct given the extracted feature vector, $\boldsymbol{x}$.

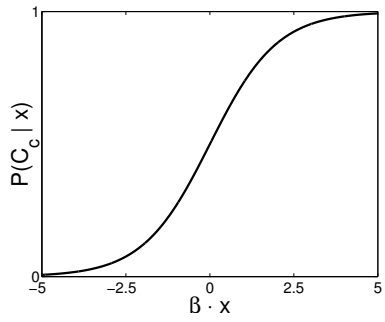


Figure 5.1: *Logistic Regression*

If the log odds, logit(P), is modelled as a linear function of the feature vector

$$\text{logit}\, P(C_c|\boldsymbol{x}) = \log\left(\frac{P(C_c|\boldsymbol{x})}{1 - P(C_c|\boldsymbol{x})}\right) = \boldsymbol{\beta}\cdot\boldsymbol{x} \tag{5.3}$$

then

$$P(C_c|\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{\beta}\cdot\boldsymbol{x})} \tag{5.4}$$

This function is a smoothly varying function of $\boldsymbol{\beta} \cdot \boldsymbol{x}$ between 0 and 1, as shown in figure 5.1. The function is differentiable, and allows the values of $\boldsymbol{\beta}$ to be trained via maximum likelihood. Once $\boldsymbol{\beta}$ has been trained, it is straightforward to predict the posterior probability of any new vector. For classification, a threshold P_{thresh} is chosen and $\boldsymbol{x}$ is classified according to

$$C(\boldsymbol{x}) = \begin{cases} C_c & \text{if } \boldsymbol{\beta} \cdot \boldsymbol{x} > P_{thresh} \\ C_e & \text{otherwise} \end{cases} \tag{5.5}$$

The values of $\boldsymbol{\beta}$ indicate weights for each of the corresponding individual features in $\boldsymbol{x}$.

5.2 Existing Approaches to Data Weighting for ASR

The confidence measures discussed above can be used to weight data in situations where a reference transcription is not available. For example, in unsupervised training and in decoding. This thesis has also discussed several applications where a data weighting is used as part of training, such as active training in section 2.4.3, discriminative training in section 2.4.2, and boosting in section 4.1.2. In these applications the training data is weighted to reflect errors made by existing systems, and so it is appropriate to make use of the correct transcriptions if available. Chapters 6 and 7 introduce two new approaches to generating complementary systems which rely on an appropriate weighting of the training data. This section discusses existing approaches to data weighting with both ML and discriminative training, and section 5.3 below presents a new approach to data weighting based on confusion network combination.

When applying a data weighting, it is first necessary to decide the level at which to weight the data. The most straightforward approach is to use an utterance-level selection scheme, as utterances are segmented manually or automatically and hence are clearly defined. For example, lightly supervised training (section 2.4.3) selects utterances which align with the closed caption subtitles; bagging (section 4.1.1) randomly selects utterances; and one implementation of boosting (section 4.1.2) is to select utterances according to a probability distribution [37, 166]. This form of data weighting effectively selects a subset of training data, and thus is appropriate for use with both ML and discriminative training. However, this may not be ideal in practice as a smaller training set could lead to issues with overtraining, but the selection method does allow control over the size of the subset to be used. For other forms of weighting discussed below, there is a dependence on the training scheme.

5.2.1 ML Training

Applying a data weighting with ML training is straightforward. The EM algorithm of section 2.4.1 can be altered to take into account a loss function $l_j(t)$ [4] and, for example, the mean update of equation 2.35 becomes

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} l_j(t)\gamma_j(t)\boldsymbol{o}_t}{\sum_{t=1}^{T} l_j(t)\gamma_j(t)} \tag{5.6}$$

The loss function can be calculated at any level, such as frame, state, phone, word or utterance, allowing for flexibility in the level at which the weighting is both calculated and applied.

Applying a weighting at the utterance level is straightforward as utterances are clearly defined. Examples of this include utterance level boosting [108, 166] and utterance-level active training [63]. With an utterance-level weighting scheme, the loss function $l_j(t)$ is constant for each utterance. The weight can also be applied at the frame level, where $l_j(t)$ is calculated for each frame. For example, boosting at the frame level is performed in [167].

Applying the weight at a different granularity, such as the word [80] or phone level, may require some further processing. One commonly used approach is to force-align the training data to obtain an alignment of words or phones to frames, and hence directly weight the observations, e.g. [80, 167]. Then, the weighting in equation 5.6 can be applied at the frame level, where each frame weight is obtained from the word or phone it is aligned with. However, this provides a hard assignment of frames to lexical units, and may lead to errors at the boundaries. This approach also has the disadvantage that the alignments may change during training, or if a very different model set is used, and so it may be necessary to recompute the alignment.

The second consideration is the form of the loss function or weighting to be used. One possibility is to use a confidence measure as a weighting, to identify correctly recognised portions of the training data. For example, the hypothesised word posterior probability [77] or likelihood-ratio [4] have previously been used. The drawback of this approach is that confidence measures are not always reliable, and the decoder output is not used to identify errors. Alternatively, the loss could be based on a high recognition error criterion [81] or word and utterance posterior probabilities [83]. For boosting [37], in figure 4.1, the data is weighted using the distribution over training samples $\boldsymbol{d}^{(s)}$, which is calculated from the pseudo-loss, ϵ_s. This is a discriminative measure which takes into account the classification of the training examples. Section 4.2.2 discussed approximations to this criterion suitable for ASR.

The purpose of the loss function is application specific, and high losses can be assigned to different portions of training data depending on the aim of the training. For example, boosting [37] applies a high loss to high error portions of the data in order to build complementary systems, while unsupervised training applies the weighting to focus the training on high confidence segments which are more likely to be correct. Thus, the form of the loss function is important in determining the overall effect of the training.

5.2.2 Discriminative Training

When using discriminative training, the application of a data weighting is complicated by the use of the reference and competing hypotheses as part of the training algorithm. Thus, a simple weighting is not applicable as for the ML training algorithm. Instead, an implicit data weighting is used as part of the discriminative training. For minimum Bayes' risk training, in section 2.4.2.2, the data weighting is defined by the loss function, $\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref})$, in the objective function

$$\mathcal{F}(\mathcal{M}) = \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M})\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}) \tag{5.7}$$

which is the equation previously given in 2.40.

$\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref})$ is a sentence level loss function, but is often a sum of losses calculated at a smaller level. For example, MMI training assigns each hypothesis a loss of 0 or 1 [6], depending on whether it is correct or not, and MPE training weights each phone arc in a lattice according to an estimate of its accuracy when compared against the reference [122]. MWE and MCE apply similar weightings at the word level. In order to calculate the loss for all competing hypotheses, an alignment of multiple hypotheses to the reference is required, as discussed in section 2.7.

5.3 A New Approach to Data Weighting for ASR

This thesis proposes two approaches for generating complementary systems, via the decision tree generation and the training algorithm. Both rely on an appropriate weighting of the training data, in order to focus on errors made by a number of previous systems. This section presents the form of word-level data weighting used with these algorithms for the experiments in chapters 9 and 10. A word-level weighting is used to reflect the word-level confusion network combination used for combining the multiple systems.

The two approaches are based on a boosting-like, or leveraging, approach to building complementary systems. An initial baseline system is first trained, and used as the starting point for complementary system training. Then, subsequent systems are trained using the data weighting described below. A second system is trained to focus on the errors made by the first, before a third system is trained to focus on the errors made by the combination of the first and second systems. This iterative approach allows any number of systems to be built by focusing on the errors made by the combination of all the previous systems.

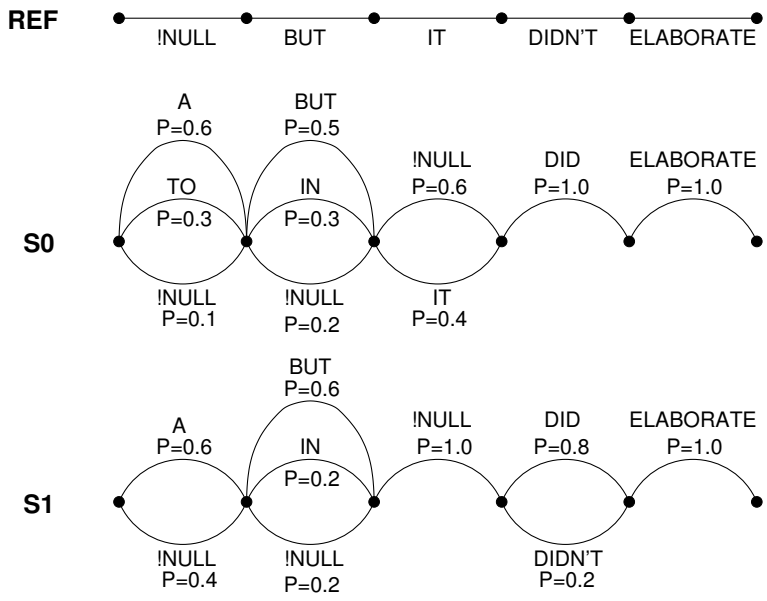


Figure 5.2: *Confusion Networks for systems S0 and S1 aligned with a reference transcription*

In this thesis, the data weighting is calculated using confusion networks for each training set utterance aligned with the reference transcription, as in figure 5.2. This makes it straightforward to see whether a word is correctly modelled or not. Figure 5.2 shows an

example alignment of two confusion networks with the reference. The confusion networks obtained from two systems, S0 and S1, are aligned against the reference transcription using the dynamic alignment algorithm discussed in section 2.7. This alignment makes it easy to see where the combination of previous models performs well, and where it is poor, based on the word posterior probabilities in the confusion networks. For example, in figure 5.2, the word 'ELABORATE' is modelled well by both S0 and S1, while the word 'IT' is not modelled well by either system. Any number of confusion networks can be aligned with the reference, allowing the weighting to make use of information from multiple systems using the average word posterior

$$P(\mathcal{W}_{ref}|\mathcal{O}, \mathcal{M}^{(0)}...\mathcal{M}^{(S-1)}) = \frac{1}{S}\sum_{s=0}^{S-1} P(\mathcal{W}_{ref}|\mathcal{O}, \mathcal{M}^{(s)}) \tag{5.8}$$

This is an unweighted version of the posterior combination in equation 3.3, and is the standard, unweighted, CNC posterior combination used in decoding.

The first step in the loss function calculation is to generate confusion networks for the training data. Due to the large amount of data, it is impractical to fully decode the data for every model. Alternatively, a set of lattices may be rescored by the multiple systems, as in section 3.3.4.1. The lattices must be representative of the hypothesis space to adequately represent all the confusions, and for this reason, the lattices for standard MPE training are used. These are generated using a pruned bigram language model, to increase the confusions that are modelled in the lattice. The lattices are then rescored, and converted to confusion networks using the algorithm in section 3.3.1.3. The use of the discriminative training lattices also has the advantage of a direct correspondence between the lattices and the confusion networks used in training, which simplifies the loss function calculation in the case of discriminative training.

The training data confusion networks are then aligned with the reference transcription using a dynamic alignment algorithm. Hence it is not necessary to use any time-stamp information in the alignment. This is advantageous as the confusion networks do not have reliable time stamps, and avoids the need to force-align the reference transcription.

For MPE training, a reference lattice is used to model multiple alignments of the reference transcription, and thus calculate the minimum phone loss, as in equation 2.45. However, when calculating a word-level loss function, it is not clear that using a lattice over a simple transcription would lead to improved results as multiple word-level pronunciations do not exist in the same way as there are multiple phone level pronunciations. Hence, for the loss function calculation in this work, the confusion networks are aligned with a single reference transcription.

The dynamic algorithm for aligning the reference transcription, discussed in section 2.7, naturally handles substitutions in the confusion network. For example, the substitution of reference word 'DIDN'T' for the word 'DID' in both confusion networks in figure 5.2. The alignment also handles insertions and deletions by inserting !NULL arcs into the appropriate transcription or confusion network. For example, the first CN segment in figure 5.2 is an insertion, while the third is a deletion. In the case of some deletions, such as with system S0, a CN segment exists that should be aligned with the reference word, though the reference word has low posterior in the segment. However, in other cases, such as with system S1, there is a true deletion and a CN segment must be inserted to perform the alignment.

In figure 5.2, !NULL links are inserted into both the reference and the confusion networks to allow the alignment of one reference word with one confusion segment, which simplifies the loss function calculation. Alternatively, CN segments could be aligned with the reference so that they span more than one reference word. Hence, insertions and deletions could be handled without the inclusion of additional !NULL links. However, it is not clear that this would provide an advantage in terms of a more accurate loss calculation, and would complicate the alignment algorithm.

When rescoring the lattices and converting to confusion networks, there is the problem that some lattice arcs are pruned and do not appear in the confusion networks. It is likely that these arcs have low posterior, and hence can be ignored when calculating the loss function. Additionally, the confusion segments do not include silence, and so do not allow a loss to be assigned to silence. Thus, the loss for silence models must be set appropriately. In this thesis, the loss of a silence model is set to zero, as this has the effect of not updating the model and it is assumed that a complementary silence model is not necessary.

As mentioned above, the form of the loss function is an important consideration. For building complementary models, the loss function should focus on the errors made by previous models. At the same time, it can ignore previously well modelled portions of data. For example, in figure 5.2 the words 'BUT' and 'ELABORATE' are well modelled by both S0 and S1. In contrast, the words 'DIDN'T' and 'IT' are not well modelled. Hence, complementary model training should focus on the latter two words at the expense of introducing errors where S0 and S1 are correct. The forms of loss function proposed below reflect this goal. This contrasts to previous applications of a data weighting, where the aim is to maintain a good model over all the training data.

For the directed decision tree and the ML word-level active training, it is necessary to weight reference words appropriately, to reflect the errors made by a number of previous models. For the discriminative MBRL training, it is necessary to weight arcs in the lattices used for training. These two tasks are related, and are both discussed in below.

5.3.1 Weighting Reference Words

For the directed decision tree algorithm in chapter 6, and the word-level active training algorithm discussed in section 7.1, it is necessary to apply a data weighting to reference words. Each word, $\mathcal{W}_{ref}^{(k)}$, in the reference hypothesis, $\mathcal{H}_{ref}$ is assigned a loss, $l(\mathcal{W}_{ref}^{(k)})$.

The alterations to the EM algorithm are as in section 5.2.1, and equation 5.6 is repeated here

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} l_j(t)\gamma_j(t)\boldsymbol{o}_t}{\sum_{t=1}^{T} l_j(t)\gamma_j(t)} \tag{5.9}$$

The weighting is applied in the forward-backward pass by multiplying each state occupancy count, $\gamma_j(t)$, by the weight of the word it belongs to. When $\boldsymbol{o}_t$ is the observation at time t, the accumulated statistics change from

$$\sum_t \gamma_j(t)\boldsymbol{o}_t \tag{5.10}$$

to

$$\sum_t l_j(t)\gamma_j(t)\boldsymbol{o}_t \tag{5.11}$$

Section 5.2 above discussed existing approaches to weighting training data, where the transcription is force-aligned to obtain a mapping from words to frames. An alternative application of the loss, used in this thesis, is to obtain a state level weighting from the corresponding word. Thus the loss $l_j(t)$ comes from the word level loss $l(\mathcal{W}_{ref})$, where state j forms part of word $\mathcal{W}_{ref}$. This is in contrast to the previous approaches, where the loss $l_j(t)$ is associated with a frame $\boldsymbol{o}_t$, via a forced-alignment.

This state-based approach to applying the loss function has the advantage that it links more closely with CNC as words are weighted rather than frames. It also allows for more flexibility as the effective weight for each frame may change as training progresses and the alignment of states to frames is altered. Furthermore, weighting at the state level easily allows the application of a weight obtained with one system to multiple different systems without the need to recompute the state alignment, and the application is independent of the particular model topologies.

From the training data CN aligned with the reference, as in figure 5.2, it is straightforward to implement a loss function for each reference word based on the posterior probability of the reference word in the aligned confusion segment. If the reference word does not appear in the confusion segment, then it is assumed the word has a posterior probability of 0.

There are many variants of the loss function which are suitable for performing the data weighting. Options for the loss function, $l(\mathcal{W}_{ref})$, considered in this thesis are a sum

$$l(\mathcal{W}_{ref}) = 1 - \frac{1}{S}\sum_{s=0}^{S-1} P(\mathcal{W}_{ref}|\mathcal{O},\mathcal{M}^{(s)})^{\alpha} \tag{5.12}$$

and a threshold

$$\begin{aligned} l(\mathcal{W}_{ref}) &= 1 \text{ if } \frac{1}{S}\sum_{s=0}^{S-1} P(\mathcal{W}_{ref}|\mathcal{O},\mathcal{M}^{(s)}) < \beta \\ &= 0 \text{ otherwise} \end{aligned} \tag{5.13}$$

where $P(\mathcal{W}_{ref}|\mathcal{O},\mathcal{M}^{(s)})$ is the posterior of the reference word $\mathcal{W}_{ref}$ according to system s. Hence, the summation in these two functions calculates the average posterior of the reference word with respect to the previous systems $\mathcal{M}^{(0)}\cdots\mathcal{M}^{(S-1)}$, in order to apply the weighting and train system S. The posterior probabilities are obtained directly from confusion networks.

In the sum function, the effect of increasing α is to reduce the proportional influence of well modelled words and increase the influence of poorly modelled words. This in turn leads the algorithm to focus more and more on the very poorly modelled training data. If a particular model leads to a large number of reference words with posterior close to 1, it may be useful to increase α and so decrease the influence of these words. In the threshold function, decreasing β also decreases the proportion of words which are assigned a loss of 1, leading the training to focus more on the errors.

Reference word	Threshold loss $\beta = 0.5$	Sum loss $\alpha = 1.0$
BUT	0.0	0.45
IT	1.0	0.80
DIDN'T	1.0	0.90
ELABORATE	0.0	0.00

Table 5.1: *Values of ML loss function for the alignment in figure 5.2*

With these two loss functions, the values of loss for figure 5.2 are given in table 5.1. The loss function focuses the weight on the words 'IT' and 'DIDN'T', which are not well modelled by S0 and S1. The threshold function effectively gives well modelled words a weight of 0, and poorly modelled words a weight of 1. The sum function assigns a continuous loss between 0 and 1.

When weighting reference words, it is only possible to assign a loss to a word which appears in the reference transcription. Hence, in figure 5.2, it is not possible to assign any weight to the insertion of the word 'A' in the first confusion segment. Thus, the training can only implicitly focus on insertions, by increasing the likelihood of the surrounding, correct, words. Deletions and substitutions can be assigned a weight, and hence their likelihood can directly be optimised.

5.3.2 Weighting Lattice Arcs

For discriminative training using a minimum Bayes' risk criterion, the hypothesis space is represented by a word lattice. Thus, the arcs in these training data lattices must be appropriately weighted. This form of weighting is used in section 7.2 to discriminatively train complementary systems.

In order to weight the lattice arcs, it is necessary to keep track of which lattice arcs are clustered to obtain each of the confusion network arcs. For example, in figure 5.3, the solid lattice arcs for the word 'IT' are clustered to give the solid CN arc 'IT'. Some lattice arcs will be pruned out as part of the confusion network generation, but these have low posterior and should so make little difference to the loss calculation overall.

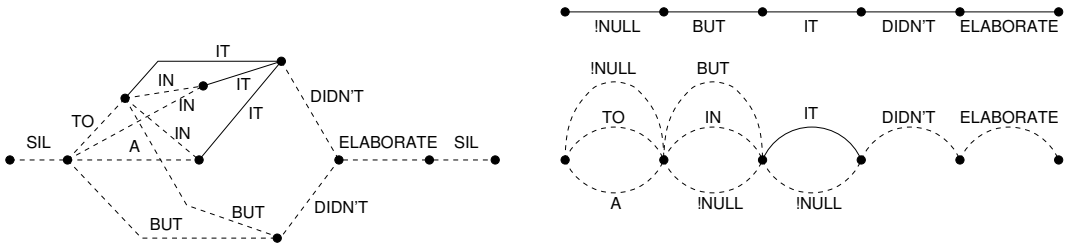


Figure 5.3: *Tracking lattice arcs in CN generation*

A loss must now be assigned to each word in the confusion network, and hence to each arc in the lattice. This is in contrast to the previous section where a loss was assigned only

to each reference word. The loss function can take any form but, as for weighting reference words above, in this thesis the average posterior probability of the reference word is used

$$P(\mathcal{W}_{ref}|\mathcal{O},\mathcal{M}^{(0)}...\mathcal{M}^{(S-1)}) = \frac{1}{S}\sum_{s=0}^{S-1} P(\mathcal{W}_{ref}|\mathcal{O},\mathcal{M}^{(s)}) \tag{5.14}$$

This is an unweighted version of the posterior combination in equation 3.3. If the reference word does not appear in the aligned confusion network segment, it is assumed to have a posterior of zero. The loss functions considered in this thesis are then a sum

$$l(\mathcal{W}^{(k)},\mathcal{W}^{(k)}_{ref}) = \begin{cases} 0 & \text{if } \mathcal{W}^{(k)} = \mathcal{W}^{(k)}_{ref} \\ 1 - P(\mathcal{W}^{(k)}_{ref}|\mathcal{O},\mathcal{M}^{(0)}...\mathcal{M}^{(S-1)}) & \text{otherwise} \end{cases} \tag{5.15}$$

and a threshold function

$$l(\mathcal{W}^{(k)},\mathcal{W}^{(k)}_{ref}) = \begin{cases} 0 & \text{if } \mathcal{W}^{(k)} = \mathcal{W}^{(k)}_{ref} \\ 0 & \text{if } P(\mathcal{W}^{(k)}_{ref}|\mathcal{O},\mathcal{M}^{(0)}...\mathcal{M}^{(S-1)}) \geq \beta \\ 1 & \text{if } P(\mathcal{W}^{(k)}_{ref}|\mathcal{O},\mathcal{M}^{(0)}...\mathcal{M}^{(S-1)}) < \beta \end{cases} \tag{5.16}$$

both based on the average posterior of the reference word in the segment here the word $\mathcal{W}^{(k)}$ is aligned with the reference word $\mathcal{W}^{(k)}_{ref}$ in the confusion network. Thus the overall loss function for the hypothesis is given by

$$\mathcal{L}(\mathcal{H},\mathcal{H}_{ref}) = \sum_{k=1}^{K} l(\mathcal{W}^{(k)},\mathcal{W}^{(k)}_{ref}) \tag{5.17}$$

These loss functions differ from existing discriminative loss functions, such as MPE, as their aim is to assign a low loss to correct words, a low loss to incorrect words where the corresponding reference word has a high posterior, and a high loss to incorrect words where the corresponding reference word has a low posterior. In contrast, MPE assigns a high loss to all errorful portions of data.

The corresponding values of word loss for these two functions are given in table 5.2 for the CN alignment in figure 5.2. The words 'A' and 'TO' in the first segment are assigned a loss as they are incorrect, and the word 'DID' is assigned a loss as the corresponding reference word for this segment, 'DIDN'T', has a low posterior. The remainder of the words are considered correct, and so have a low value of loss. This is because they are either correct themselves, or the corresponding reference word in the segment has a high posterior. Hence the loss focuses on the words which are poorly modelled, and ignores those which are incorrect yet have low posterior. For the purpose of loss function calculation, the !NULL arc is considered as a single word, though this may lead to overestimation of the loss.

Unlike the approach to weighting reference words discussed previously, it is now possible to directly weight words which correspond to insertions by weighting the corresponding lattice arcs. Hence, the training does not rely on improving the modelling of surrounding words to correct insertions, and can more directly focus on these errors.

Reference word	CN word	Threshold loss $\beta = 0.5$	Sum loss $\alpha = 1.0$
!NULL	A	1.0	0.75
	TO	1.0	0.75
BUT	BUT	0.0	0.00
	IN	0.0	0.00
IT	IT	0.0	0.00
DIDN'T	DID	1.0	0.90
	DIDN'T	0.0	0.00
ELABORATE	ELABORATE	0.0	0.00

Table 5.2: *Values of loss function for the alignment in figure 5.2*

Again, the loss calculation relies on an alignment of CN segments with the reference transcription. This allows a one-to-one correspondence between reference and hypothesis words. Other alignments of the confusion network, or the original lattice, with the reference may give an improved loss function, but it is not clear that this would impact significantly on the training.

5.3.3 Alternative to Confusion Networks

The approaches described above have relied on confusion network combination of training data confusion networks with the reference transcription, to obtain a loss function for training. However, confusion networks may not be the optimal method with which to encode and identify the confusions made by a recogniser.

An alternative approach would be to force-align the reference transcription and use a measure of performance based directly on the overlap between lattice arcs and reference words. This is the approach used for estimating the loss in MPE training in [122]. However, this approach doesn't make full use of the posterior information available, as the posterior probability of a word consists of contributions from multiple paths. Confusion networks address this by clustering overlapping words and hence obtaining a better estimate of word posterior probabilities.

Other possible approaches include the pinched lattices in [60], or the frame-based posterior estimation in [71]. The former elegantly handles insertions and deletions by allowing sub-lattices to align with reference words, while the latter calculates a frame-based word posterior.

Alternative representations like these could easily be aligned with a reference transcription in training, and hence used with the MBRL algorithm to identify confusions and assign a loss to lattice arcs or reference words. The loss function calculation would be altered to reflect the form of combination, then these methods could easily be used within the MBRL framework, and for combining the final systems in decoding.

For this work, however, confusion networks are used as a representation of recogniser confusions. They are straightforward to compute, and allow a simple interpretation of loss which can easily be applied in both the word-level active training and the discriminative MBRL training, regardless of the form of systems being trained. Confusion networks also allow for easy system combination, and thus allow the training to be embedded within an iterative leveraging framework for building multiple complementary systems.

5.4 Summary

This chapter first discussed confidence measures, which are used in this thesis for combination during testing and training. In particular, combination of complementary systems is examined in detail in chapter 11.

Data weighting is an important aspect of building and combining complementary systems, as it is necessary to incorporate information about errors made by existing systems. The algorithms presented in the following chapters make extensive use of a data weighting algorithm. This chapter first discussed existing approaches to data weighting for ASR, before presenting a new approach based on confusion network combination to identify errors made by previous systems. This approach allows the loss function to be calculated independently of the form of model, and is used with the directed decision tree and MBRL algorithms of the next chapters.

Chapter 6
Directed Decision Trees

Chapter 4 discussed existing approaches to generating complementary systems for ASR, including a randomised decision tree for parameter tying. This section proposes a new algorithm for generating complementary systems by altering the decision tree generation, and a divergence measure for comparing decision trees. This approach makes use of the data weighting proposed in the previous chapter.

If it were possible to obtain a well-trained triphone system without parameter tying, each individual state would have a distinct output distribution. However, due to insufficient training data and the memory requirements for building and using such a system, this is not possible. Thus, there is no adequate measure for assessing the impact of the decision tree state tying on recognition performance. In practice, the standard decision tree algorithm discussed in section 2.8 takes no account of system performance and confusability of states when building the tree, and so it is possible to cluster states which lead to confusions. This can impact on recognition performance, as clustered states rely on language model and context information to differentiate them in decoding. Thus, it is desirable to build decision trees in such a way that confusable states are explicitly separated.

The directed decision tree algorithm presented in this chapter concentrates states in contexts which were previously confusable. In this way, previous errors may be resolved, though new errors may be introduced by clustering states which were previously separate. If the two systems make different errors they will be complementary, and hence lead to improved performance when combined.

6.1 Directed Decision Tree Algorithm

Directed decision trees [16, 17] aim to separate confusable states when performing the decision tree clustering. This is done by using a second set of statistics when selecting the best question in decision tree generation. This second set of statistics is weighted so as to reflect confusions in the training set, so states which often lead to confusions are allocated a higher weight than those which don't. These weighted statistics are used in the question selection stage of the decision tree generation, so that states with high weights are not clustered together. The original statistics are used for the stopping criterion, to ensure that the trees are of a similar size and to avoid having to tune a new stopping criterion threshold. The directed decision tree algorithm from section 2.8 becomes

1. **Statistics**
   - Obtain original statistics for all seen triphone contexts
   - Obtain *weighted* statistics for all seen triphone contexts statistics
2. **Question Selection**
   - Recursively build the tree by selecting the question which gives the highest change in likelihood with respect to the *weighted* statistics
3. **Stopping criterion**
   - Stop building the tree when the data likelihood falls below a threshold, with respect to the *original* statistics

An example of this modified question selection is shown in figure 6.1. The question *'Right liquid?'* is chosen as this gives the largest data likelihood with respect to the weighted statistics, rather than the question *'Left front fricative?'* which is optimal with respect to the original statistics.

Question	Original	Weighted	
Right liquid?	**[67.5]**	**[37.4]**	◄ **DIRECTED**
Right nasal?	[69.5]	[37.2]	
Left nasal?	[65.4]	[36.7]	
Left vowel central?	[68.8]	[36.3]	
Left front fricative?	***[70.1]***	***[35.8]***	◄ ***ORIGINAL***
Left unvoiced fricative?	[64.3]	[35.3]	
Right back fricative?	[69.6]	[34.2]	

Figure 6.1: *Directed decision tree question selection*

The forward-backward algorithm in section 2.3.2 is used to estimate the state occupation counts γ_j which, along with the means and variances of the Gaussian state distributions, are the sufficient statistics needed for building the decision tree. The original set of statistics are calculated as in section 2.8, and the cluster occupancy count, $\gamma_{\boldsymbol{\Theta}}^{(1)}$, is found using equation 2.77, repeated here

$$\gamma_{\mathbf{\Theta}}^{(1)} = \sum_{t=1}^{T} \sum_{j=1}^{P} \gamma_j(t) \tag{6.1}$$

where P is the number of clustered states. For the second set of statistics, $\gamma_{\mathbf{\Theta}}^{(2)}$, equation 2.77 becomes

$$\gamma_{\mathbf{\Theta}}^{(2)} = \sum_{t=1}^{T} \sum_{j=1}^{P} l_j(t)\gamma_j(t) \tag{6.2}$$

The directed decision tree algorithm allows for any form of data weighting to be applied via the loss function $l_j(t)$, which can be applied at the state or frame level. The experiments in chapter 9 use the data weighting proposed in section 5.3.1 with the sum loss function of equation 5.12 to weight the training data and obtain the second set of statistics.

6.2 Multiple Complementary Systems

The form of loss function calculation in section 5.3.1 allows the incorporation of information from multiple previous systems. Hence, it is interesting to consider generating multiple complementary systems in a boosting-like framework, as shown in figure 6.2 for building three complementary systems. A baseline system, S0, is first trained in the usual way using unweighted statistics for the decision tree generation. Then, S0 is used to obtain confusion networks for the training data, these are aligned with the reference and used to generate the weighted statistics for building a second decision tree. Next, system D1 is built from this decision tree, and so will be complementary to S0. Using confusion networks from both S0 and D1 allows a second set of weighted statistics to be obtained and a third decision tree to be generated. This decision tree is then used to build D2, which is complementary to both S0 and D1. The combination of multiple previous confusion networks is done using CNC, as previously shown in figure 5.2.

This iterative method for building complementary systems has the advantage that the final order of combination is simply determined by the order in which systems were trained. The method of training and the system design for S0, D1 and D2 does not need to be the same, and further diversity can be obtained by varying, for example, the training approach, frontend or topology for each of the three systems. Experimental results in chapter 9 use a different frontend and training algorithm to introduce additional diversity.

6.3 Decision Tree Cluster Divergence Measure

An important aspect of using multiple directed decision trees is generating systems that are different. In this work, decision trees cluster states of triphones, and so they are built as a first step in training a triphone system. It is computationally expensive to evaluate multiple decision trees by training the multiple triphone systems and evaluating their performance. Hence, it is useful to have a method for comparing the clustering in decision trees without

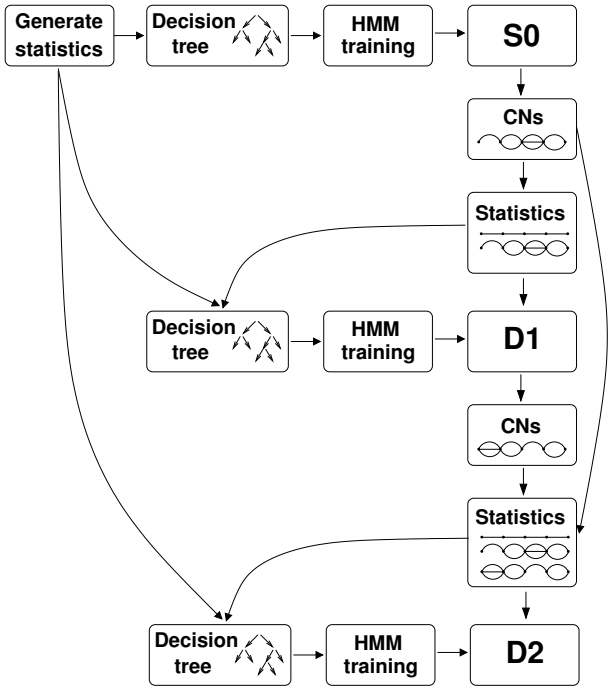


Figure 6.2: *Framework for building multiple directed decision trees*

having to fully train the corresponding system. This section describes a divergence measure for this purpose, which evaluates the similarity between the clustering in two decision trees.

It is possible to compare the clusterings in decision trees directly using, for example, a cluster similarity measure similar to that in [127]. However, these use many pairwise comparisons between clustered elements and prove expensive in practice when there are many thousands of states, so it is useful to make use of the properties of decision trees for ASR. As states are clustered at the tree nodes, each node has an associated Gaussian distribution. Hence each state has a Gaussian distribution in each of the two trees, which differs with the clustering.

Rather than directly measure cluster similarity, the divergence measure proposed here makes use of the fact that if two trees have similar clusterings then, for each state, the corresponding state output distributions from the two trees will also be similar. Similarly, if the clustering is very different, then it is expected that the state output distributions from the two trees will differ too. The tree divergence $\mathcal{D}$ is calculated as an average over all states of divergences between state output distributions from the two trees, weighted by the state occupation count γ_j

$$\mathcal{D} = \sum_j \gamma_j \mathrm{KL}_{sy}(\mathcal{N}(\mu_j^{(1)}, \Sigma_j^{(1)}), \mathcal{N}(\mu_j^{(2)}, \Sigma_j^{(2)})) \tag{6.3}$$

where

$$\mathrm{KL}_{sy}(\mathcal{N}^{(1)}, \mathcal{N}^{(2)}) = \frac{1}{2}\left(\mathrm{KL}(\mathcal{N}^{(1)}, \mathcal{N}^{(2)}) + \mathrm{KL}(\mathcal{N}^{(2)}, \mathcal{N}^{(1)})\right) \tag{6.4}$$

and $\mathcal{N}(\mu_j^{(s)}, \Sigma_j^{(s)})$ is the distribution of state θ_j in the s^{th} tree. The Kullback-Leibler divergence $\mathrm{KL}(\mathcal{N}, \mathcal{N})$ [88] is used as a measure of divergence between Gaussians from the two trees, but any suitable distance metric could be employed. The KL divergence between two Gaussians is

$$\mathrm{KL}(\mathcal{N}^{(1)}, \mathcal{N}^{(2)}) = \frac{1}{2}\left\{\log\left(\frac{|\boldsymbol{\Sigma}^{(2)}|}{|\boldsymbol{\Sigma}^{(1)}|} + \mathrm{trace}\left(\boldsymbol{\Sigma}^{(2)\text{-}1}\boldsymbol{\Sigma}^{(1)}\right)\right) + (\boldsymbol{\mu}^{(2)} - \boldsymbol{\mu}^{(1)})^{\mathsf{T}}\boldsymbol{\Sigma}^{(2)}(\boldsymbol{\mu}^{(2)} - \boldsymbol{\mu}^{(1)}) - D\right\} \tag{6.5}$$

Though this divergence measure does not indicate performance, it is useful in this work to determine whether two trees are close together. If two trees are very similar then it is unlikely that the resulting systems will differ enough for gains to be seen when combining them, and so it is not worthwhile to build the system. In particular, the effect of α in equation 5.12 on the decision tree generation is interesting, so to avoid the computational cost of building and decoding with many systems, the effect of α on decision tree divergence measure can be examined rather than its effect on final word error rate.

6.4 Summary

This chapter has presented an algorithm for altering the decision tree algorithm, to bias systems towards being complementary by separating confusable states in the decision tree. This algorithm uses a second set of statistics, weighted to reflect errors in the training data, to bias the decision tree generation against clustering confusable states.

Together with the loss function calculation discussed in section 5.3, this algorithm can be embedded within an iterative boosting-like framework for building multiple complementary systems. The statistics generation stage of the decision tree algorithm is altered to take this loss into account, and the weighted statistics are used during question selection stage of the decision tree generation.

To conclude, a divergence measure for comparing two decision trees was proposed. This measure uses a symmetric Kullback-Leibler divergence, averaged over states in the trees, to evaluate the diversity of the trees without having to build and evaluate the corresponding systems.

Chapter 7
Minimum Bayes' Risk Leveraging

The previous chapter presented an approach for generating complementary systems by altering the decision tree generation. This chapter presents two approaches for explicitly generating complementary systems through the training algorithm, based on both the maximum likelihood and minimum Bayes' risk criteria.

Standard training schemes, such as maximum likelihood and discriminative training, aim to build a single system with optimal performance. An alternative aim in training is to build an ensemble of systems which do not perform optimally individually, but have optimal performance in combination. One approach to achieve this is to alter the training algorithm to explicitly build complementary systems.

This chapter describes how confusion network combination and standard training algorithms can be used together in a boosting-like, or *leveraging*, scheme to allow the generation of complementary systems. The approaches described here differ from existing training criteria in their goal of building multiple complementary systems.

Modifications to the ML and MBR criteria of sections 2.4.1 and 2.4.2.2 are presented below, to explicitly generate complementary systems. Complementary systems make different errors, and so explicitly training one system to be complementary to another needs to take into account the errors made by the first. The algorithms presented in this chapter make use of confusion network combination to incorporate information about the performance of multiple previous systems into the training algorithm. First, the alterations to the training algorithms are presented, followed by a discussion of the potential issues with decoding using the complementary systems.

7.1 Word-level Active Training

Active training, described in section 2.4.3, alters the ML training algorithm to focus on subsets of training data. One application of active training is to focus on the errors made by a system in order to improve system performance. In the meantime, new errors may be introduced on previously well modelled data and it is hoped that the two systems will be complementary, particularly if the active training focuses extremely on the errors.

The ML objective function in equation 2.28 is altered to incorporate a loss function for each utterance $\mathcal{L}(\mathcal{H}_{ref})$, where the loss models how well the utterance is modelled by the previous systems $\mathcal{M}^{(0)} \cdots \mathcal{M}^{(s-1)}$. Hence, the objective function becomes

$$\mathcal{F}(\mathcal{M}^{(s)}) = \sum_{r=1}^{R} \mathcal{L}(\mathcal{H}_{ref}^{(r)} | \mathcal{O}, \mathcal{M}^{(0)} \cdots \mathcal{M}^{(s-1)}) \log p(\mathcal{O}^{(r)} | \mathcal{H}_{ref}^{(r)}, \mathcal{M}^{(s)}) \tag{7.1}$$

As discussed in section 5.3, the loss function in this work reflects the word errors made by a number of previous systems, to focus the training on the corresponding portions of data.

By incorporating multiple previous systems into the loss function, it is possible to build a series of complementary systems in an iterative fashion, similar to the boosting framework in section 4.1.2 and the directed decision tree approach in section 6.2. The individual systems can differ, for example in the frontend or model topology, provided the loss function reflects the errors made. Introducing extra diversity in this way may give additional gains. The multiple systems are combined in the training using confusion network combination.

Equation 7.1 specifies a loss at the utterance level. However, it is possible to calculate a loss at any level. In this work, the loss is calculated at the word level as this corresponds to the word level combination in CNC. During training, the loss is applied at the state level. The state occupation probability $\gamma_j(t)$ is multiplied by a state level loss $l_j(t)$, which is part of the corresponding reference word $l(\mathcal{W}_{ref})$. That is, $\sum_t \gamma_j(t)\boldsymbol{o}_t$ becomes $\sum_t l_j(t)\gamma_j(t)\boldsymbol{o}_t$ where state θ_j is derived from the word $\mathcal{W}_{ref}$. This is the same as the data weighting in section 5.3.1, and the sum and threshold loss functions in equations 5.12 and 5.13 are used in chapter 10 for building complementary systems, as they focus the training on previous errors.

Given this application of the loss in training, the changes to the standard EM parameter update equations are trivial. For example, the mean estimation becomes

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} l_j(t)\gamma_j(t)\boldsymbol{o}_t}{\sum_{t=1}^{T} l_j(t)\gamma_j(t)} \tag{7.2}$$

In decoding, the systems are used to independently decode the data, and then the outputs are combined. The order of combination, using CNC, is given by the order in which the systems were built. This allows the flexibility to change the loss function, the level at which the loss is applied, and the forms of the model being built, without having to alter the decoding process. Much of the discussion in the following section concerning discriminative training is also applicable to this word-level ML active training approach to generating complementary systems.

7.2 Minimum Bayes' Risk Leveraging Algorithm

The previous section described how the ML criterion can be modified to generate complementary systems. This section considers the changes needed to discriminative training, and makes use of the minimum Bayes' risk criterion, which optimises

$$\hat{\mathcal{M}} = \underset{\mathcal{M}}{\text{argmax}} \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M}) \mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}) \tag{7.3}$$

This objective function depends only on the existing model, $\mathcal{M}$. To build complementary systems it is necessary to take into account the performance of all previous recognisers. This can be done by altering the objective function in one of two ways. First, the other models can be taken into account via the posterior probability

$$\mathcal{F}(\mathcal{M}^{(s)}) = \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M}^{(0)} \cdots \mathcal{M}^{(s-1)}) \mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}) \tag{7.4}$$

This form of objective function is complicated to model due to extra dependencies, and proves difficult to optimise in practice. Hence, a second approach is proposed whereby the dependency on previous systems is introduced via the loss function

$$\mathcal{F}(\mathcal{M}^{(s)}) = \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M}) \mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}|\mathcal{O}, \mathcal{M}^{(0)} \cdots \mathcal{M}^{(s-1)}) \tag{7.5}$$

This objective function can be used to train a system $\mathcal{M}^{(s)}$ that is complementary to a number of previous systems, and hence can be embedded inside an iterative training scheme to train a number of systems, each complementary to the previous. This is called *Minimum Bayes' Risk Leveraging* (MBRL). In this thesis, the loss function for MBRL training is calculated by aligning confusion networks with the reference transcription, and calculating a loss for each word based on its posterior probability in the confusion network. This is described in more detail in section 5.3.2 above. Experimental results in chapter 10 use the threshold and sum loss functions of equations 5.16 and 5.15.

An outline for the scheme is given in figure 7.1, and is again similar to the boosting, or leveraging, schemes discussed in sections 4.1.2 and 4.2.2. It has many of the attributes of boosting, but without the theoretical aspects of setting the weights, calculating the pseudo-loss and maintaining a distribution over the training samples. The loss function $\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref})$ takes the place of the probability distribution, $\boldsymbol{d}$, over the training data, and allows a more general form of weighting to be applied than is used in boosting. Additionally, there is no need to approximate or estimate the discriminative pseudo-loss metric which is used in boosting. The overall algorithm, with the appropriate objective function and loss function, is also applicable to the ML active training described in the previous section.

For the discriminative MBR training, the loss function is a sentence level function. For MBRL, this function measures how well all previous classifiers perform. As word-level CNC is used for combination in the final decoding stage, a loss function is calculated at the word level to match the combination schemes in test and training. In general, the MBRL loss function is a sum of loss functions at a word level. When each competing hypothesis is aligned with

Input:
An initial model, $\mathcal{M}^{(0)}$

Initialise:
Using $\mathcal{M}^{(0)}$, generate initial set of training data CNs

For: s= 1:S
Combine training data CNs for systems $\mathcal{M}^{(0)} \cdots \mathcal{M}^{(s-1)}$
Align the combined CN with the reference, and calculate the word loss
Train $\mathcal{M}^{(s)}$ to minimise a cost function based on

$$\mathcal{F}(\mathcal{M}^{(s)}) = \sum_{\mathcal{H}} P(\mathcal{H}|\mathcal{O}, \mathcal{M}^{(s)}) \mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}|\mathcal{O}, \mathcal{M}^{(0)} \cdots \mathcal{M}^{(s-1)}))$$

Decode the training data with model $\mathcal{M}^{(s)}$ to obtain CNs

Output:
Test data independently decoded with each $\mathcal{M}^{(s)}$ to give hypothesis $\mathcal{H}^{(s)}$
The final hypothesis is based on CNC of hypotheses $\mathcal{H}^{(0)} \ldots \mathcal{H}^{(S)}$

Figure 7.1: *Minimum Bayes Risk Leveraging Algorithm for estimating and decoding with a baseline system* $\mathcal{M}^{(0)}$ *and* $S-1$ *complementary systems,* $\mathcal{M}^{(1)} \cdots \mathcal{M}^{(S)}$

the reference to give a set of word pairs, $\{\mathcal{W}^{(k)}, \mathcal{W}^{(k)}_{ref}\}$, as in section 2.7, the loss function becomes:

$$\mathcal{L}(\mathcal{H}, \mathcal{H}_{ref}) = \sum_{k=1}^{L} l(\mathcal{W}^{(k)}, \mathcal{W}^{(k)}_{ref}) \tag{7.6}$$

This sentence level loss function as a sum of losses is analagous to MPE training in equation 2.45, where the sentence loss is a sum of phone errors. MBRL differs from MPE training as it takes the performance of multiple classifiers into account. In contrast to the ML scheme described in the previous section, the loss function now assigns a loss to all words, whether they appear in the reference hypothesis or in a competing, incorrect, hypothesis. As for the ML algorithm, the loss should reflect which words are well modelled, and which are not. Each iteration of the MBRL training will then have a memory of *all* previous classifiers through the loss function. It is hoped that, while training in this way will not lead to a single *best* classifier, the resulting set of classifiers will give improved results when they are combined. MBRL differs from [168], a boosting algorithm for ASR with a general loss function, as it moves away from the strict boosting framework, and allows for more flexibility in the types of system that can be built and combined.

Parameter estimation with MBRL is straightforward and can be implemented in the same way as for MPE or other discriminative training schemes, simply by changing the loss function calculation.

Standard discriminative training schemes such as MPE and MWE effectively weight the data to reflect errors made by the current system. During training, they aim to reduce the

overall average word error rate, but in doing this, they may introduce errors in portions of the training data where the recogniser previously had good performance. Hence, the systems trained using standard discriminative training may be complementary. However, to improve overall performance, the discriminative training must keep a good model over data which is currently well recognised, thus restricting its ability to focus on the errors.

In contrast, MBRL and the ML active training in the previous section allow this requirement to be relaxed, by training systems that perform optimally in combination. The current system need not perform well on data which the previous systems recognise well, as the combination with previous systems should address any newly introduced errors. That is, by training explicitly for optimal performance in combination, the MBRL training can focus specifically on the harder portions of the training data, without losing the advantage of recognising easier portions, as these are already well recognised by existing systems.

The straightforward application of the MBRL scheme is to treat it as a standard discriminative training scheme, such as MPE training in section 2.4.2.2, where the model $\mathcal{M}^{(s-1)}$ is used as the starting point for training $\mathcal{M}^{(s)}$. This form of the algorithm may be susceptible to overtraining as many iterations of discriminative training are performed and successive models will drift further from the initial model. However, the weighting to focus the training on different areas of the training set may alleviate this problem, and smoothing, as discussed in section 2.4.2.5 can be used to prevent the models drifting too far.

A more general application of MBRL is to relax this requirement, and allow $\mathcal{M}^{(s)}$ to be trained from a system that differs from $\mathcal{M}^{(s-1)}$. This allows extra diversity to be incorporated into the training, which may give further gains. For example, $\mathcal{M}^{(s)}$ and $\mathcal{M}^{(s-1)}$ may differ in their frontend, decision tree, phone set, covariance modelling or topology. However, in contrast to the directed decision tree approach, the schemes in this chapter directly affect the training algorithm and so it is not possible to incorporate additional diversity by using complementary training algorithms. The use of CNC as the combination method in testing and training makes it straightforward to include very different models in the training algorithm, as the combination method can be used independently of the form of model.

7.3 Issues with Training Complementary Systems

The word-level active training and MBRL algorithms above offer a number of approaches that allow multiple complementary systems to be generated, and it is expected that they will perform well if the systems are not overtrained and if the word posteriors for use in combination are well estimated. However, there are some potential issues which arise from the mismatch of the training and the decoding algorithms.

When performing the word-level active training, some portions of the training data are weighted highly, while other portions have low weight. Initial experiments showed a tendency for the ML training to increase the state occupation posteriors on portions of the training data with low weight, and decrease the state occupation posteriors on portions with high weight. To avoid this, for the experiments in chapter 10, the state alignments were fixed in training using a second model set. Issues with the discriminative MBRL algorithm, including poor alignments in decoding adversely affecting the performance and a mismatch between the combination method used in testing and in training, are discussed in more detail below.

The directed decision tree approach of chapter 6 is not expected to suffer from the same issues as, although the data is weighted for the decision tree statistics generation, the training

still builds a model which performs well over all data. Hence it does not directly focus the training on specific portions of the training data in the same way as the approaches presented in this chapter.

7.3.1 Alignment

The MBRL training algorithm focuses on specific portions of the training set, and so the resulting systems are not expected to perform well on those portions that are not the focus of training. This is expected to be reflected in decoding, with the MBRL trained systems performing poorly on data where the original system may perform well. On these portions of the test data where the MBRL system performs badly, poor performance, and hence poor word alignments, may impact on the decoding in other segments of the utterance where the MBRL trained system performs well. This is due to the language model causing issues beyond the regions of data which are well modelled.

In order to restrict the impact of this potential misalignment, a restricted form of decoding is proposed. In training, the reference transcription is used to identify the errors. However, in decoding, the reference transcription is not available, and so it is interesting to consider a form of decoding which addresses the issue of poor alignments by only using the complementary system to decode segments where it is believed that the baseline system performs badly.

To perform decoding in this manner, the confusion networks obtained from decoding with the first system are used. Where it is believed that the first system is correct, the competing words are pruned out of the confusion network. Where it is believed that the first system is incorrect, the competing words remain in the confusion network. The potentially correct and incorrect words can be identified by a confidence measure, such as those discussed in section 5.1. This pruned confusion network is converted to a word lattice, and !NULL arcs in the confusion networks are treated as meaning the competing words in that segment are optional. The lattice is then expanded using a language model and rescored by the complementary system. Thus, only words where the first system is unsure are rescored by the second. The rescored lattices are then converted to confusion networks, for confusion network decoding. This approach is similar to the lattice rescoring and acoustic codebreaking approaches in sections 3.3.4.1 and 4.2.4.

This procedure allows paths in the lattice which were not present in the original decoding lattices, though the simple lattices allow for fast rescoring. Additionally, this form of decoding is more closely related to the threshold loss function in training and is similar to the acoustic codebreaking discussed in section 4.2.4 by its use of a second system to resolve confusions made by the first.

The posterior probability can be used as a confidence measure to identify those segments where the first system performs badly. An example of the confusion network pruning with a posterior threshold of 0.5 is shown in figure 7.2, and the corresponding lattice is shown in figure 7.3. For example, in the first segment of the confusion network, the best word ('OH') has a posterior of 0.8 and so the competing words ('TO' and 'A') are pruned out. In the second segment, the best word ('BUT') has a posterior of 0.4, and so the competing words are retained.

The use of a confidence measure to do this pruning means that some confusion segments will be incorrectly identified as errors, while some incorrect words will be falsely labelled as correct. In the first case, this could lead to previously correct words being made worse by the restricted decoding and degrading the word error rate. In the latter case however, the

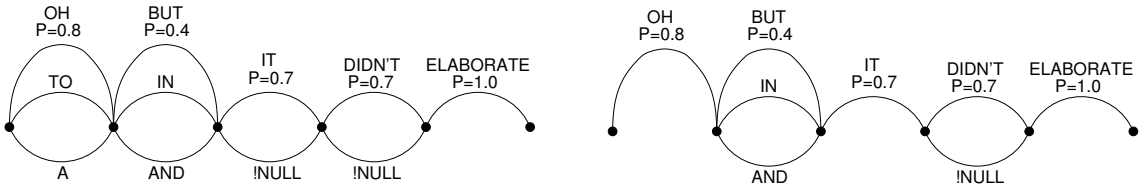


Figure 7.2: *Pruning the Confusion Network with a posterior threshold of 0.5*

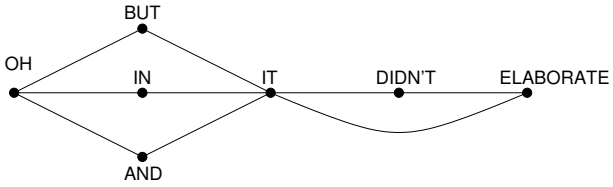


Figure 7.3: *Word graph resulting from the CN pruning in figure 7.2*

incorrect words are not able to be corrected, but this will have no impact on the word error rate.

The *reduced* lattices that are obtained from the pruned confusion networks are no longer a good representation of the hypothesis space. Words are pruned out based on the posterior of the corresponding *best word*, not the posterior of the word itself. Also, many low probability paths and words are pruned out. Hence, the posteriors which are calculated from this reduced lattice are no longer reliable. This means that confusion network combination using the final confusion networks is not appropriate. Thus, this scheme is an implicit combination scheme, where the output from the first system is rescored by the second.

7.3.2 Combination as a Binary Classification Task

A second issue with the complementary training algorithms presented in sections 7.1 and 7.2 is the mismatch between the combination algorithm in training and that used in decoding, when the threshold function in equation 5.13 is used in training. CNC in decoding uses the posterior combination in equation 3.3, while the threshold loss function makes a hard decision about the correctness of a word.

Hence, it may be necessary to alter the combination algorithm in decoding to more closely match that used in training. The threshold loss function effectively gives words a weight of 1 or 0, depending on whether they are correct or not. For this reason, it is interesting to consider the combination of two systems as a binary classification task, as in section 5.1.3. Confusion networks from two systems are aligned against each other. Then, for each segment, the binary choice is whether to use the output from the first system, or to combine the outputs from both the first and second systems. Thus, each segment in the second system is effectively assigned a weight of 1 or 0 in the combination, reflecting the threshold in training.

When considering the alignment of two confusion networks with a reference transcription, as shown in figure 7.4, it is possible that either or both of S0 and S1 are correct. However, for the task of combination to match the threshold loss function in training, it is not necessary to consider the errors made by S1, but to consider the errors made by S0 and the combination S0+S1. Thus, there are four outcomes from the word level combination that must be

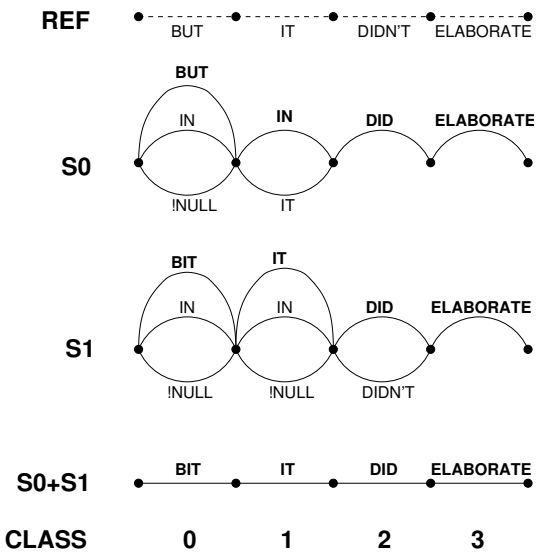


Figure 7.4: *Classes for Combination as a Binary Classification Task*

considered. These are illustrated in figure 7.4, where two systems S0 and S1 are combined. The reference transcription is shown alongside the confusion networks, and the four classes in this figure are

- **Class 0:** S0 is correct but the output from combination S0+S1 is not
- **Class 1:** S0 is incorrect but the combination of S0+S1 is correct
- **Class 2:** both S0 and the combination are incorrect
- **Class 3:** both S0 and the combination are correct

In the first confusion segment of figure 7.4, the best word from system S0, 'BUT', is altered to 'BIT' by combination of S0+S1, and performance is degraded. The second confusion segment shows the opposite case, where the word 'IN' is corrected to 'IT' by the combination. The third and fourth segments show the cases where both the first and second systems have the same hypothesis word, and so their combination has no effect on the final word error rate. Thus, it is only the first two cases which are important for the task of combination. Class 2 also includes the possibility of S0 and S1 both having different best hypothesis words, but both incorrect, as this outcome does not alter the combination.

Along with the confusion networks for each system, it is trivial to obtain a force-aligned transcription and language model information for the best hypothesis. These three sources of information may be used to obtain a feature vector for each confusion segment, as in figure 7.5, and hence train a binary classifier to detect the errors, as in section 5.1.3. Examples of previously used features, such as duration, acoustic model scores and word posteriors, are also given in section 5.1.3. If the best word in a confusion network is !NULL, then some of the features, such as word duration, may not be available.

First, a general error detection algorithm can be used. Such an algorithm aims to identify errors made by the first system, and hence combine segments only when the first system is incorrect. In this case, classes 1 and 2 are merged to give the class containing errors, and

classes 0 and 3 are merged to contain those words which are correct. However, an algorithm which detects errors reasonably well may not have a noticeable effect on the final word error rate after combination if it only performs well on those examples from classes 2 and 3.

Ideally, the error detection algorithm could be applied to just those examples in classes 0 and 1, in figure 7.5, although it is expected that these classes have far fewer examples than classes 2 and 3, possibly leading to data insufficiency if examples from classes 2 and 3 are discarded.

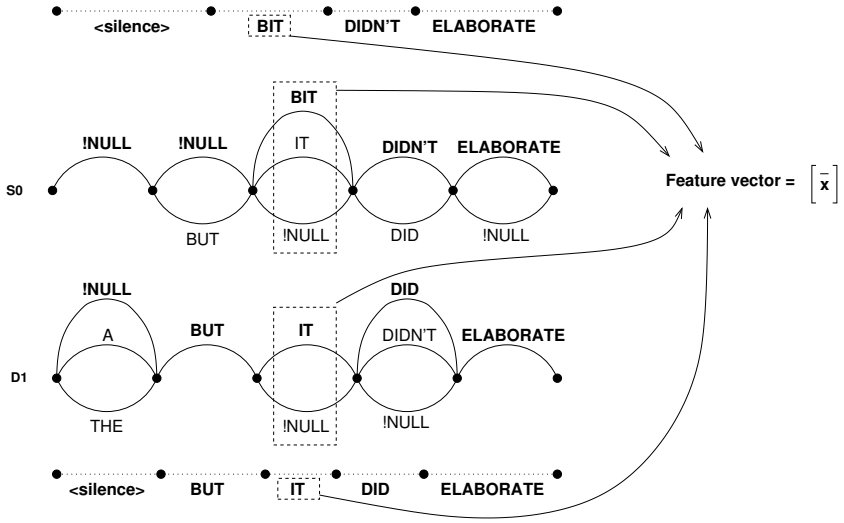


Figure 7.5: *Combination as a Binary Classification Task*

The IDEAL combination discussed in section 3.4 is a form of this scheme where the reference transcription is used to accurately identify the errors. That is, it uses an ideal confidence score to identify word errors. If the first system is correct, then its output is used. If the first system is incorrect, then the combination of the two systems is used.

Chapter 11 evaluates these approaches using logistic regression as the binary classifier. Word error detection using a classifier has had limited success in previous work [165]. However, the subset of words which affect the combination is much smaller than the entire test set, and training a classifier to perform well on this smaller task may prove effective. Additionally, as it is expected the decoding with the second system will perform differently depending on the errors made by the first system, these techniques may prove useful for combining complementary systems where the training algorithm is directly related to the errors.

7.4 Summary

This chapter has presented two algorithms for training complementary systems, based on the Maximum Likelihood and minimum Bayes' risk criteria. These use confusion network combination in both training and testing, to identify errors made by the previous systems and weight the data appropriately to focus on these portions of data. At the same time, the training may degrade performance on portions of the training set where the previous systems made no errors. In this way, the training aims to build systems which are optimal when combined, not individually.

The training is embedded within an iterative leveraging framework for building multiple complementary systems. The framework allows extra diversity to be incorporated by varying the multiple systems, for example in their frontend, decision tree, or dictionary.

To conclude, potential issues with the training algorithm are discussed, including poor alignments affecting the performance of the decoding algorithm and combination as a binary classification task to better match the training.

Chapter 8
Experimental Setup

The directed decision tree, word-level active training, and MBRL algorithms presented in the two previous chapters were evaluated on LVCSR broadcast news tasks in English, Mandarin and Arabic. Historically, much research for ASR has been carried out on English tasks. In recent years however, other languages have become more popular. Mandarin and Arabic have very different characteristics to English, which can affect the training algorithm, and so it is interesting to perform experiments on these three languages. The following three chapters present and discuss the results, but this chapter first details the experimental setup for the three tasks.

8.1 Broadcast News Training and Decoding

The training and decoding approaches for the three languages have some similarities. The common attributes of building a cross-word triphone system and two forms of decoding are discussed in this section, while language specific details are discussed below.

8.1.1 Training Approach

The overall training approach for building systems was the same for all three languages. The first step in building the systems is to perform the decision tree clustering, using an unclustered triphone model to generate the statistics. This decision tree was used as the basis for building an acoustic model with 16 components per state using a 39 dimensional feature vector with 12 PLP coefficients and energy, plus first and second derivatives. Cepstral mean normalisation was also used. This system, denoted by 'PLP' in the experimental results, was used for initial experiments with the directed and random decision trees in sections 9.2

and 9.3 as the quick development time made it easy to generate multiple systems based on multiple decision trees.

Next, the feature vector was increased to 52 dimensions by the addition of the 3rd derivatives, and an HLDA transform estimated to project back to 39 dimensions. For the Mandarin task only, the pitch and its first and second derivatives were then appended to the feature vector to give 42 dimensions. Then, the number of Gaussian components per state was reordered to be proportional to the state occupancy count, maintaining an average of 16 components per state. This more complex system, denoted by 'HLDA', was used as the ML baseline for experiments with the word-level active training in section 10.1.

MPE training was then performed. Phone-marked lattices were first generated using a pruned bigram language model to increase confusions, and these were used as the hypothesis space for MPE training. Eight iterations of training were performed and the smoothing, as described in section 2.4.2.5, was done using a dynamic MMI prior. This system, denoted 'MPE', was used as the baseline for experiments with the directed decision trees in sections 9.4 and 9.5, and the discriminative MBRL training in section 10.2.

Finally, gender dependent models were built using three iterations of MPE training to update only the means and variances, with the gender independent model as a static prior. These models were used for decoding in the multi-pass framework described below for experiments in section 9.6.

8.1.2 Single-pass Decoding

Results are first obtained using gender independent models in a single-pass decoding framework, without speaker adaptation. This decoding framework is the same for all three languages. First, lattices were obtained using a bigram language model. The bigram lattices were then expanded with a trigram language model, before being converted to confusion networks for confusion network decoding. All individual system results given are for confusion network decoding, as this allows the gains from combination to easily be seen. Combination was performed using confusion network combination. For all statistical significance tests, the matched-pairs test was used [54].

8.1.3 Multi-Pass Decoding Framework

Along with the singlepass unadapted framework, decoding was also performed in a multi-pass framework similar to that discussed in section 3.1. This allows for the use of complex acoustic and language models, and also additional techniques such as cross-adaptation and unsupervised speaker adaptation.

Decoding was performed in two multi-pass frameworks both shown in figure 8.1. In these frameworks, an initial transcription was generated using gender independent models and a trigram language model so that normalisation and adaptation could be performed, and lattices generated using gender dependent models and a fourgram language model. These lattices were then rescored, using multiple models, and the outputs combined using confusion network combination. Both MLLR mean and variance adaptation were used in the final pass, along with lattice-based adaptation, as in [48]. The first framework in figure 8.1(a) uses a common adaptation and lattice generation stage, using the S0 model, while the second in 8.1 (b) uses separate passes for each of the systems. Thus the framework in figure 8.1(a) includes cross-adaptation, while that in 8.1(b) does not.

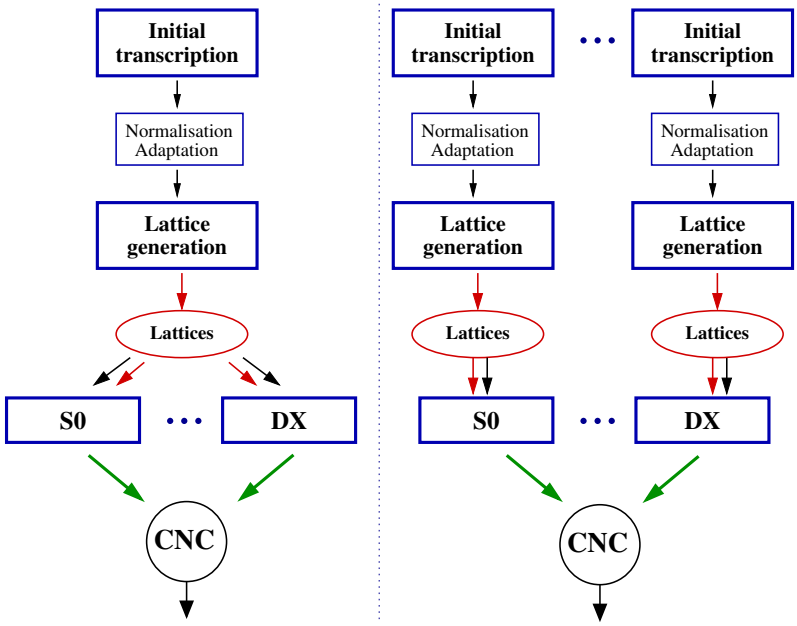


Figure 8.1: *Multi-Pass framework with (a) common lattice generation, (b) separate lattice generation passes*

As for the singlepass framework, all individual system results are for CN decoding, and the combination is done using CNC. The matched-pairs test was used for statistical significance testing.

8.2 Broadcast News English

The acoustic models for the English task were trained using approximately 144 hours of data recorded by the LDC in 1997 and 1998. The standard decision tree for state clustering had 6976 unique states. Additionally, MPE trained 4 and 8 component systems were built in the same way as the 16 component system. The three systems were trained using separate HLDA transforms and lattices for discriminative training, though the decision tree remained the same.

A trigram language model with a 59k wordlist was used for singlepass decoding. The language models were trained on approximately 1000M words, 4M from the acoustic training data, and the remainder from newswire articles.

Results are given on the `dev03` and `eval98` test sets, which are 2.7 and 2.9 hours respectively. The dev03 set was collected in 2001 and the eval98 set in 1998. Additionally, a 10 hour subset of training data was used for examining the effect of the algorithms on the training set. This subset of training data was not held-out, and so performance on this set is expected to be better than for the independent test sets. The baseline results for the English systems are given in table 8.1. On the dev03 set, the simpler PLP system gave an error of 19.5%, the introduction of HLDA and the reordering of Gaussian components improved performance to 18.1%, and the MPE training further improved performance to 14.7%. This system is similar to that used in [86].

	dev03	eval98	train subset
PLP	19.5	18.4	20.9
HLDA	18.1	16.6	19.3
MPE	14.7	13.2	12.0

Table 8.1: *Singlepass BN English baseline WER (%) results on the dev03 and eval98 test sets and the 10 hour training data subset*

8.3 Broadcast News Mandarin

Mandarin is spoken across much of mainland China, and is the official language for the Chinese media. It is a tonal language, with four different tones. Thus, along with the phonetic sound, the tone of a word distinguishes it. Hence, automatic speech recognition for Mandarin must take the tone of the speech into consideration along with the phonetic transcription. In practice, this requires the pitch of speech to be included in an ASR system, in the frontend and in the models, and tonal questions can be used in the decision tree for parameter tying. Mandarin is also a character based language, with words consisting of a number of characters, each character representing a syllable. Characters can be grouped in many ways to make words, which means in practice, an extra step is required in processing to automatically segment characters into words. Additionally, the metric used for evaluation is the character error rate, rather than the word error rate.

The baseline acoustic models for the Mandarin task were trained using 148 hours of data; 28 hours of Hub-4 data released by the Linguistic Data Consortium (LDC) with accurate transcriptions, and 120 hours of TDT4 data with only closed-caption references provided. Light supervision techniques were used on the latter portion. The baseline systems used a standard decision tree for state clustering, where there are 6070 unique states. Tonal questions were used in the decision tree clustering. The incorporation of tonal information into the models slows both training and decoding due to the increased number of tonal phones. The HLDA system was also used to generate statistics for Gaussianisation, discussed in section 2.2.2.2. The Gaussian frontend was used in addition to the complementary training algorithms of the previous chapters.

A trigram language model with a 50k wordlist was used for singlepass decoding. The language models were trained on approximately 366M words released by the LDC. These are the correct acoustic transcripts for the Hub-4 data, China radio, Mandarin TDT4 data, the Gigaword corpus, and People's daily. For the multipass decoding, the trigram and fourgram language models were built using a 58k word list, on a total of 20 Chinese text sources with 1.1G words, including acoustic transcriptions, LDC text releases and web data.

Results are given on the `bnm-dev06` test set, which is a combination of `dev04f` (0.5 hours of CCTV data from shows broadcast in November 2003), `eval04` (1 hour of data from CCTV, RFA and NTDTV broadcast in April 2004), `eval03m` (0.6 hours of mainland shows from February 2001) and `y1q1` (3 hours of data from October 2005). Baseline performance for decoding in the singlepass framework is given in table 8.2. As for the English task, performance improves as the system becomes more complex, with the MPE system yielding a character error rate of 18.4%.

For decoding in a multi-pass framework, a fourgram language model was used, and baseline performance is given in table 8.3 for both the P1+P2 and P3 passes. Performance is

	bnmdev06
PLP	23.4
HLDA	23.0
MPE	18.4

Table 8.2: *Singlepass BN Mandarin baseline CER (%) results on the bnmdev06 test set*

improved by using the multipass framework with gains achieved from the speaker adaptation and complex language models, and gender dependent acoustic models, yielding a best performance of 14.9%.

	Pass	bnmdev06
MPE	P1+P2	15.7
	P3	14.9

Table 8.3: *BN Mandarin baseline CER (%) results on the bnmdev06 test set in the multipass framework*

8.4 Broadcast News Arabic

Arabic has many dialects, but Modern Standard Arabic is spoken across much of north Africa and the Middle East, and is the standard dialect for use in the media. Arabic has the property that short vowels are missed from the words in transcription, and thus each grapheme has more pronunciations than, say, a word in English. Word pronunciations for Arabic can be generated, often by using a set of rules [19]. This leads to a large number of pronunciations per word - on average 4.3 compared to 1.1 for English. This issue needs to be addressed when building an Arabic system, and suggestions have included graphemic systems [1], general purpose 'short vowel' models [91], or variations on training to take the multiple pronunciations into account [49].

The MPE training criterion makes use of the phonetic transcription in training [49], and can be adjusted to take the multiple pronunciations into account. Section 2.4.2.2 discusses single and multiple pronunciation variants of MPE training which perform similarly but give improvements upon combination [49]. Thus they can be used in addition to the directed tree approach to incorporate extra diversity. However, as the MPE training criterion is directly altered, they cannot be used in addition to the explicit training schemes of chapter 7.

The baseline system for the Arabic task used in this thesis is a phonetic system, trained using 102 hours of data. 43 hours of this data is the FBIS data for which detailed transcriptions are available, and 59 hours is TDT4 data which was used in a lightly supervised fashion. MPE training was done using the single pronunciation criterion, but a second system was trained using the multiple pronunciation criterion. Decision tree state clustering was performed, with 3976 unique states in the standard tree.

The language models for decoding in both the single and multi pass frameworks were trained on approximately 435M words. Of this, approximately 1M was from the acoustic

training data, 77M from web data, and the remainder from the Gigaword corpus released by the LDC.

	bnad06	bcat06	bcad06	bnat06	average
PLP	38.3	48.3	46.6	41.1	43.5
HLDA	36.2	47.4	45.5	39.5	42.1
MPE	33.0	44.9	42.9	36.6	39.3

Table 8.4: *Singlepass BN Arabic baseline WER (%) results on the bnat06, bnad06, bcat06 and bcad06 test sets*

Results are given on four test sets, each around 3 hours long: `bnat06` and `bnad06` are broadcast news data while `bcat06` and `bcad06` consist of broadcast conversation shows, and hence are less closely matched to the training data. `bnat06` consists of data from November 2005 and January 2006, `bnad06` from November and December 2005, and January 2006, and `bcat06` and `bcad06` were collected in January 2006. Table 8.4 gives performance of the single pronunciation MPE trained baseline system. As for the English and Mandarin tasks, performance improves as training progresses.

Decoding was performed in the multi-pass framework described above. A fourgram language model was used. In this framework, the large number of Arabic pronunciations are handled in decoding by using pronunciation probabilities. The results obtained for decoding in the multipass framework are given in table 8.5. Again, gains are seen from the more complex decoding.

	Pass	bnad06	bcat06	bcad06	bnat06	average
MPE	P1+P2	31.8	43.9	41.3	35.0	38.0
	P3	30.8	43.6	40.6	33.9	37.2

Table 8.5: *BN Arabic baseline WER (%) results on the bnat06, bnad06, bcat06 and bcad06 test sets in the multipass framework*

Chapter 9
Experimental Results with Directed Decision Trees

This chapter presents experimental results for the directed decision tree discussed in chapter 6. First, the decision tree divergence measure is evaluated, and the effect of the data weighting on the resulting decision trees is seen. Next, initial experiments on an ML trained Mandarin system compare the random tree and the directed tree algorithms.

The effect of MPE training and the directed decision tree is then investigated for the Mandarin, Arabic and English tasks, followed by the combination of Gaussianisation and the directed tree algorithm for the Mandarin task.

The second half of this chapter uses the multi-pass combination framework introduced in the previous chapter for decoding. Results are obtained on both the Mandarin and Arabic tasks. Finally, the combination of directed and random decision trees with single and multiple pronunciation training for extra diversity is presented for Arabic.

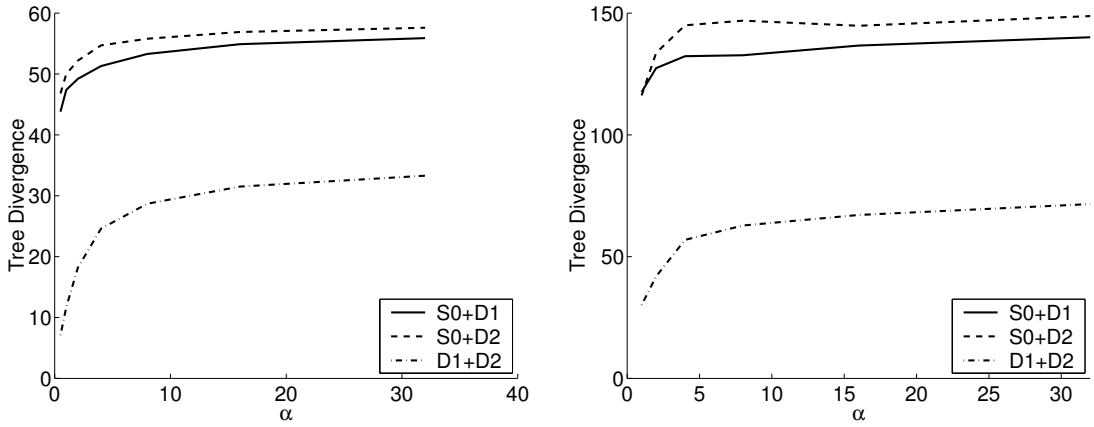
9.1 Decision Tree Divergence

To begin, the divergence measure of section 6.3 was evaluated on the Mandarin and Arabic tasks described in chapter 8. A baseline system, S0, was first built. Next, system D1 was built to be complementary to S0 using a directed decision tree, and D2 was then built to be complementary to both D1 and S0. D1 and D2 were built in exactly the same way as S0, the

only difference being in the decision tree generation. The weighting for reference words in the statistics generation was the sum function, previously given in equation 5.12, repeated here

$$l(\mathcal{W}_{ref}) = \left(1 - \frac{1}{S}\sum_{s=0}^{S-1} P(\mathcal{W}_{ref}|\mathcal{O}, \mathcal{M}^{(s)})\right)^{\alpha} \tag{9.1}$$

The effect of α on the decision tree generation is interesting, so to avoid the computational cost of building and decoding with many systems, the effect of α on decision tree divergence measure is examined rather than its effect on final word error rate.



(a) Mandarin (b) Arabic

Figure 9.1: *Decision Tree Divergence with α when comparing S0 and D1 (dotted line), S0 and D2 (solid), D1 and D2 (dashed)*

The tree divergence was measured for both Mandarin and Arabic broadcast news tasks, and figure 9.1 shows how the tree divergence varies with α in the loss function calculation. D1 was compared to just the baseline, S0, while D2 was compared both to S0 and to D1. The divergence tends to increase with α, as more emphasis is placed on harder to recognise training data. As α becomes larger, further increases in divergence are small. Both D1 and D2 are a similar distance from S0, but D1 and D2 are much closer together. For the Mandarin system in figure 9.1(a) the absolute values of the divergence are smaller. This could be because the Mandarin system includes tonal questions, hence there are many more similar questions for the Mandarin task than there are for Arabic. Thus, when selecting sub-optimal splits, the questions chosen are more likely to be similar to the optimal question.

For the Mandarin task, the baseline decision tree has 6070 unique states. Figure 9.2 shows the number of states in the directed decision tree, D1, as α in the loss function is changed. As α is increased from 0.5 to 32, the number of unique states in the resulting decision tree decreases from 5848 to 5643, and thus the directed tree is more compact than the baseline tree.

As a contrast to the directed tree approach, ten random decision trees were also built for the Mandarin task. Five of these were built with $N = 10$ and five with $N = 5$. N is the number of questions between which the random tree algorithm decides at each step in growing the tree. The divergences between S0 and the random trees are given in table 9.1 for both

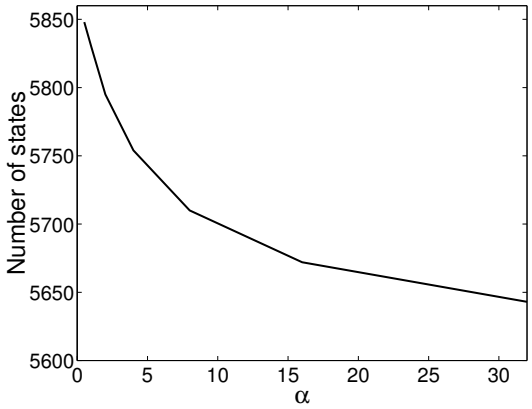


Figure 9.2: *Number of unique states with α for the Mandarin directed decision tree D1*

N=5 and N=10. When N=5, the divergence varies between 96.8 and 109.5, and for N=10 the divergence with the baseline varies between 167.7 and 188.7. Thus, the divergence increases with N as the random tree algorithm moves away from the baseline. These are greater than the divergences between the baseline and the directed tree in figure 9.1(a), which is to be expected as the random tree algorithm is far less restrained.

Table 9.1 also shows the number of unique states for each of the random trees. It can be seen that as N increases, the number of states decreases. This is as expected, as the increase in data likelihood will be slower than for the standard algorithm as the random tree is grown, and so the likelihood will fall below the stopping threshold earlier. However, there appears to be no correlation between the number of states and the divergence. The performance of these random systems is examined in the following section.

Decision Tree		Divergence with S0	# states
STANDARD	S0	-	6070
RANDOM: N=5	R1	102.2	5568
	R2	96.8	5618
	R3	105.5	5606
	R4	103.3	5585
	R5	109.5	5600
RANDOM: N=10	R1	182.1	5226
	R2	170.5	5188
	R3	188.7	5142
	R4	168.0	5222
	R5	167.7	5184

Table 9.1: *Random Trees for ML trained BN Mandarin systems - number of states and divergence with baseline S0 system*

It is suspected that if the divergence between two trees is small, then their combination will not yield gains. For this reason, in the following sections, the first directed tree D1 was built with $\alpha = 1$ and the second D2 was built with $\alpha = 16$.

9.2 Random Decision Trees

First, the performance of random decision trees was examined on the Mandarin task. A baseline decision tree was built using the standard decision tree algorithm discussed in section 2.8. From the ten random decision trees of the previous section, ten random systems were also built. Each system was built with a 39 dimensional PLP feature vector and the number of components per state was 16. ML training was performed, and no HLDA transform was included in the frontend. This was to allow quick development of the ten random systems.

Decoding was run in a singlepass unadapted framework to evaluate the system performances. The individual system results, and their performance in combination, on the `bnmdev06` test set are given in table 9.2. These results show that the random trees individually perform worse than the baseline, and as N is increased the individual system performance gets worse. The average result for N=10 is 23.8% CER, compared to 23.6% for when N=5, and 23.4% for the baseline. This is expected, as the random decision trees are never optimal and drift further from the optimal decision tree as N is increased. That the random tree systems are sub-optimal can be seen by a comparison of the ML criterion on the training data, when averaged over all frames. The ML criterion for the baseline system, S0, is -68.31, while the best ML criterion for the random systems is slightly worse at -68.40, and occurs with system R4 when N=5.

In contrast to the individual performances, when these random systems were combined with the baseline the performance improved, dropping from an error rate of 23.4% baseline to an average of 22.9% for combination with the baseline when N=5, and 23.0% when N=10. Thus, both values of N yield gains in combination, with N=5 performing slightly better than N=10. All the gains achieved by CNC are statistically significant when compared with the baseline S0 system. It is also worth noting that the best individual system does not necessarily give the best results in combination, and there is some variation between best and worst performance. For example, when N=5, the best performing system has an error rate of 23.5% and the worst an error rate of 23.7%. When combined with the baseline, the best performance is 22.8%, and the worst performance is 23.1%. When comparing the performances of these ten random systems with the number of states and divergences in table 9.1, it appears that there is no correlation between word error rate and number of states or divergence with the baseline. For the Mandarin results below, just the average of the random tree results is given for N=5 as this allows for easy comparison.

The performance of the random tree systems suggests that small perturbations in the decision tree generation can lead to gains in practice. This gain might also be expected if, for example, the decision tree was generated with an alternative decision tree algorithm. However, these trees were not explicitly trained with the goal of being complementary, as is the case for the directed trees in the following experiments.

9.3 Directed Decision Trees

To evaluate the directed decision tree algorithm, and for direct comparison with the random tree systems of the previous section, initial experiments built complementary decision trees using ML training and a PLP frontend, with no HLDA transform and 16 Gaussian components per state.

Decision Tree		bnmdev06 (CER %)	
		Individual	CNC with S0
PLP	S0	23.4	-
RANDOM: N=5	R1	23.5	22.9
	R2	23.5	22.9
	R3	23.6	22.8
	R4	23.6	22.9
	R5	23.7	23.1
	AVG	*23.6*	*22.9*
RANDOM: N=10	R1	23.6	22.9
	R2	23.9	22.9
	R3	23.9	23.1
	R4	23.9	23.0
	R5	23.8	22.8
	AVG	*23.8*	*23.0*

Table 9.2: *Random Tree Mandarin performance for ML trained systems, bnmdev06 (CER %)*

The ML trained baseline, S0, was used to weight the training data and build a directed decision tree. This tree was then used to train a directed tree system, D1, which was trained in the same way as the baseline and the random systems above, with the exception of the decision tree generation. A second directed tree system, D2, was also built in the same way, but complementary to S0+D1. That is, the loss function for the data weighting was calculated using the combination of S0+D1. When building the trees, D1 used a value of $\alpha = 1$ in the loss function calculation, while D2 used $\alpha = 16$. Decoding was run in a singlepass unadapted framework to evaluate the system performance.

Table 9.3 shows a comparison of the average random (R) and the directed tree performance. For combining random systems and the baseline, the average over all combinations is given. For example, S0+R denotes the average performance when combining S0 and the five random systems. S0+R+R denotes the average performance when all the combinations of two random systems are used.

It can be seen that the individual directed decision tree systems perform slightly better than the baseline while the random trees perform worse. The performance of D2 improves the baseline performance from 23.4% to 23.2% while the average random tree has an error rate of 23.6%. This is most likely due to an implicit discriminative training effect obtained from the data weighting.

In combination with the baseline system, the directed trees perform as well as the average random tree. For example, the combination of S0+D1 gives an error rate of 22.9%, which is just 0.1% worse than the best combination, S0+R3, from table 9.2. Additionally, S0+D1+D2 gave an error rate of 22.7%, which a 0.7% absolute improvement over the baseline performance, while the average combination of S0 and two random trees gave a comparable error rate of 22.8%. The gains achieved from CNC are statistically significant when compared to the baseline performance. The majority of this gain is seen when combining just two systems, with a much smaller gain achieved by the third. For the directed tree systems, this pattern is similar to that seen in the divergence measure in the previous section. For further gains from

the second system, it may be possible to increase the diversity of the second directed tree, D2. However, with the loss function equation used for these experiments, figure 9.1 showed that increasing the value of α further did not lead to increased diversity.

	Complementary to	System	bnmdev06
PLP	-	S0	23.4
DIRECTED	S0	D1	23.3
	S0+D1	D2	23.2
RANDOM	-	R	23.6
CNC		S0+D1	22.9
		S0+D1+D2	22.7
		S0+R	22.9
		S0+R+R	22.8

Table 9.3: *Comparison of Directed and Random Tree Mandarin results for ML trained systems, bnmdev06 set (CER %)*

The directed decision tree algorithm makes use of a data weighting, and thus has some implicit discriminative effect in training. This accounts for the improvements seen in the individual directed tree results over the baseline ML trained system. This implicit discriminative effect is additional to the gains achieved due to the systems being complementary, making it difficult to see exactly where gains are achieved. Thus, ML trained systems are not examined for the English and Arabic tasks, and the following results use an MPE trained baseline.

It is expected that if many random trees are built, the best of these would outperform both the baseline and the directed tree systems. However, it is impractical to build many complex systems, and these results show that, for a small number of random trees, the directed tree performance is close to that of the best random tree, without the uncertainty associated with randomness.

9.4 MPE Training and Directed Decision Trees

Next, the effect of directed decision trees on an MPE trained system was considered. It is computationally expensive to generate lattices for discriminative training, so MPE training was only performed for the baseline and for one directed tree system on the Mandarin task. For the directed tree system, the HLDA transform and the MPE training lattices were regenerated.

Table 9.4 shows the results obtained in a singlepass decoding framework. First, the individual results for the baseline and the directed tree systems are shown. In contrast to the ML system in the previous section, the individual directed decision tree system no longer outperforms the baseline. This suggests that any implicit discriminative effect from the data weighting is subsumed by the explicit discriminative training. When the two systems are combined, statistically significant improvements in error rate are still seen. The improvement from the baseline to the combined performance was 0.4% absolute, a drop in error from 18.4% to 18.0%. The IDEAL performance, introduced in section 3.4, is 17.2%. This combination scheme uses the reference to identify word errors made by the baseline, and only combines with the second system when the first makes an error. Thus, the IDEAL performance shows that some additional gain could be achieved with an optimal method of combination.

	System	Complementary to	bnmdev06
MPE	S0	-	18.4
DIRECTED	D1	S0	18.5
CNC	S0+D1		18.0
IDEAL	S0+D1		17.2

Table 9.4: *Directed tree performance for Mandarin with MPE trained systems, bnmdev06 testset (CER %)*

Due to the inclusion of tonal phones and tonal questions in the decision tree for Mandarin, lattice generation for MPE training is slowed. Hence, to examine the effect of a second MPE trained complementary system, the Arabic and English tasks were used. For these tasks, S0 is again an MPE trained system with an HLDA frontend and an average of 16 components per state. The complementary systems D1 and D2 were trained in the same manner as S0, with separate HLDA transforms and training lattices.

Table 9.5 shows the performance obtained on the Arabic BN task. The directed tree systems individually do not normally improve over the baseline S0 system. For example, on bnad06, the baseline performance is 33.0%, compared to 33.1% for D1 and 33.2% for D2. When combined with the baseline, large gains are achieved from the combination S0+D1, while further small gains are obtained on two of the test sets for the combination S0+D1+D2. On the bnad06 set, the performance of S0+D1 gives an error rate of 32.5%, which is a 0.5% improvement over the baseline system. The combination of S0+D1+D2 gives a further 0.1% improvement to 32.4%.

	System	Comp. to	bnad06	bcat06	bcad06	bnat06	average
MPE	S0	-	33.0	44.9	42.9	36.6	39.3
DIRECTED	D1	S0	33.1	45.1	42.6	36.6	39.3
	D2	S0+D1	33.2	44.9	42.5	36.8	39.3
CNC	S0+D1		32.5	44.5	42.2	36.0	38.8
	S0+D1+D2		32.4	44.5	42.1	36.0	38.7

Table 9.5: *Directed tree performance for Arabic using a singlepass decoding framework with single pronunciation MPE trained systems, bnat06, bcat06, bnad06 and bcat06 testsets (WER %)*

The same behaviour can be seen in table 9.6 for the two English test sets. The directed tree systems individually do not improve performance over the baseline, but statistically significant gains can be seen in combination. The largest gain is achieved from the addition of one complementary system, and additional small gains are seen when incorporating the second. On the dev03 test set, the baseline and the directed trees all have an error rate of 14.7%, which is improved by the combination of multiple systems: S0+D1 gives 14.4% and S0+D1+D2 gives 14.3%. The IDEAL combination shows that there is some potential for improvement over confusion network combination, with larger potential gains possible from the combination of three systems.

	System	Complementary to	dev03	eval98
MPE	S0	-	14.7	13.2
DIRECTED	D1	S0	14.7	13.1
	D2	S0+D1	14.7	13.5
CNC	S0 + D1		14.4	12.9
	S0 + D1 + D2		14.3	12.8
IDEAL	S0 + D1		13.6	12.2
	S0 + D1 + D2		13.1	11.8

Table 9.6: *Directed tree performance for English with MPE trained systems, dev03 and eval98 testsets (WER %)*

9.5 Combination of Complementary Approaches

The experiments presented above in this chapter use baseline and complementary systems which are trained in an identical manner, with the exception of the decision tree. Thus, the differences in performance are solely due to the decision tree generation.

The standard approach to generating complementary systems is simply to vary the training of individual systems and choose those which combine well together. Despite the ad-hoc nature of this approach, some techniques have consistently been found to be complementary. Hence, it is interesting to look at using the directed decision tree approach in addition to these other methods, to determine whether the gains from directed trees add to those from other sources. This can be done by altering the training of D1, so both the training method and the decision tree differ from S0, compared to the previous results where the only difference was in the decision tree.

Gaussianisation, discussed in section 2.2.2.2, is an alternative frontend which has been shown to be complementary to the HLDA frontend used in the previous results [98]. Gaussianisation was applied to the Mandarin MPE trained baseline system, S0, while using the same decision tree, resulting in a baseline Gaussianised system G0. Also, Gaussianisation was applied to the directed tree D1 to build DG1. DG1 is thus complementary to S0, but also has extra diversity due to the addition of a Gaussian frontend. The statistics for performing the Gaussianisation were generated in the HLDA domain, due to the assumption of independence between the feature vector dimensions, and thus the baseline decision tree was not altered. Separate Gaussian transforms were estimated for both the baseline and the complementary systems. Thus, S0 and G0 make use of the same decision tree, as do D1 and DG1.

Table 9.7 shows how the combination of the Gaussian frontend with the directed decision tree improve over the techniques individually. As expected, Gaussianised frontend improves over the PLP frontend. The individual system performance dropped from 18.4% to 17.8%, and also the combination of S0 and G0 resulted in a further drop in error to 17.4%. Similarly, changing the frontend of the directed tree improved its performance from 18.5% to 17.7%. The combination of the baseline S0 system with the Gaussianised directed tree system, DG1, outperformed both of these with an error of 17.2%. The gain is small, suggesting that the performance of the PLP and Gaussian frontends is too different, or the two were highly complementary to begin with. The same behaviour is seen in the IDEAL combination.

Decision Tree		Complementary to	System	bnmdev06
STANDARD	MPE	-	S0	18.4
	MPE+GAUSS	-	G0	17.8
DIRECTED	MPE	S0	D1	18.5
	MPE+GAUSS	S0	DG1	17.7
	CNC		S0+G0	17.4
			S0+DG1	17.2
	IDEAL		S0+G0	16.8
			S0+DG1	16.4

Table 9.7: *Mandarin Directed Tree performance in addition to Gaussianisation, bnmdev06 testset (CER %)*

9.6 Multi-pass Performance

The previous results have all used a singlepass unadapted framework for decoding. It is interesting to see how the performance is affected by using a more complex multi-pass combination framework, with speaker adaptation and cross-adaptation.

First, the MPE trained Mandarin baseline, S0, and the directed tree system, D1, were decoded in the two multi-pass frameworks described in section 8.1.3. The systems S0 and D1 are gender dependent versions of those previously evaluated in the singlepass framework in table 9.4. Results for the P3 passes and the final confusion network combination are given in tables 9.8 and 9.9.

The results show that, for the Mandarin task, the gains from the directed tree system in the singlepass unadapted framework are lost when using the multi-pass framework. Using separate P1 and P2 adaptation and lattice generation passes, in table 9.8, the combination of the baseline and the complementary system gives an improvement of just 0.1% over the baseline, from 14.9% to 14.8%. No gain is seen in the case of a common adaptation and lattice generation pass, in table 9.9, where the baseline system S0 was used to perform the common P1+P2 pass and obtain the lattices for rescoring. Additionally, no cross-adaptation effect is seen in this setup, as the performance of the directed tree system alone, 15.0%, is 0.1% worse than its performance in table 9.8 where no cross-adaptation is performed.

	System	bnmdev06
MPE	S0	14.9
DIRECTED	D1	14.9
CNC	S0+D1	14.8

Table 9.8: *Directed tree performance for Mandarin with MPE trained systems in a multi-pass adaptive framework with separate adaptation and lattice generation passes, bnmdev06 testset (CER %)*

The multi-pass combination framework was also used to evaluate the random and directed tree results on the Arabic task. This task has the additional option of single and multiple pronunciation MPE training, which can be used in addition to the directed decision tree for

	System	bnmdev06
MPE	S0	14.9
DIRECTED	D1	15.0
CNC	S0+D1	15.0

Table 9.9: *Directed tree performance for Mandarin with MPE trained systems in a multi-pass adaptive framework with a common adaptation and lattice generation pass, bnmdev06 testset (CER %)*

	System	Comp. to	bnad06	bcat06	bcad06	bnat06	average
MPE	S0	-	30.4	42.7	39.9	33.5	36.6
RANDOM	R	-	30.6	42.9	39.9	33.5	36.7
DIRECTED	D1	S0	30.5	42.8	39.8	33.4	36.6
	D2	S0+D1	30.4	43.0	39.8	33.4	36.6
CNC	S0+D1		30.1	42.5	39.5	33.1	36.3
	S0+R		30.1	42.4	39.6	33.1	36.3
	S0+D1+D2		29.9	42.5	39.6	33.1	36.2

Table 9.10: *Arabic results using a common adaptation and lattice generation pass with single pronunciation MPE trained systems, bnat06, bcat06, bnad06 and bcat06 testsets (WER %)*

additional diversity. The initial baseline system, S0, and the directed systems, D1 and D2, were trained using the single pronunciation MPE criterion described in section 2.4.2.2, and are gender dependent versions of the models used above in table 9.5. Three random tree systems were built with $N = 5$, also using the single pronunciation criterion, and the the average results (R) are given for clarity. The results obtained using a multi-pass decoding framework with a common lattice generation pass are shown in table 9.10. These results show a similar pattern to the previous unadapted singlepass results on the Mandarin and English tasks. First, the individual directed and random tree results are slightly worse than the baseline. Second, the combination of the directed tree systems and the baseline leads to improvements, and the combination of the baseline with the random tree systems performs slightly worse than the directed trees. However, the addition of a second directed tree system does not improve performance. For example, on the `bnad06` set, the baseline performance is 30.4% word error rate. The average random tree performance is 30.6%, while D1 and D2 give individual performances of 30.5% and 30.4% respectively. The combination of S0+D1, and the average combination with the random trees, S0+R, both give a small absolute gain of 0.3% over the baseline, with a performance of 30.1%. These gains are consistent across the four test sets.

It has previously been seen that the single and multiple pronunciation criteria are complementary for the broadcast news Arabic task [49]. This section also considers the combination of the directed decision tree algorithm with the complementary training criteria, in a multi-pass decoding framework.

M0 is a second baseline system trained using the multiple pronunciation MPE criterion discussed in section 2.4.2.2. The only difference between M0 and S0 is the loss function used in the MPE training stage. DM1 and DM2 are both built using directed decision trees in

the same way as M0, yet complementary to S0. That is, they are the same as D1 and D2 from the previous section, except for the loss function used in the MPE training. Hence they should retain the gains seen previously, but also include extra diversity from the difference in training. The three random tree systems were also trained using the multiple pronunciation MPE training and hence should also show additional gains. The average performance of these systems (RM) is given for clarity, and again S0+RM and S0+RM+RM denote the average of the possible combinations.

Tables 9.11 and 9.12 show results obtained using these systems in the two multi-pass frameworks previously described in section 8.1.3. The individual system performances for M0 and S0 are very similar, but those for M0 are slightly improved over S0 in table 9.11 due to the cross-adaptation. S0 and M0 also give gains when combined. For example, in table 9.11, the baseline performances are 30.4% and 30.2% for S0 and M0 respectively on `bnad06`, and combining them gives a performance of 29.7%. This pattern is consistent for all testsets in both decoding frameworks.

Considering the first directed tree system, DM1, the individual system performances are similar to the baseline systems S0 over the four sets of results, again with some cross-adaptation gains. However, the performance of S0+DM1 improves over that of S0+M0. For example, in table 9.11, on `bnad06`, S0+DM1 gives an error rate of 29.5%, compared to 29.7% obtained from combining S0 and M0.

Further gains can be obtained from introducing a second directed tree system, DM2. This system was built to be complementary to S0+DM1. Comparing table 9.11 to table 9.10 where both D1 and D2 were single pronunciation systems, the results are further improved, implying that the directed tree gains are additional to those obtained from the multi-pronunciation training. Again, the directed tree systems perform the same or slightly better in combination than the random tree systems, over all four testsets.

	System	Comp. to	bnad06	bcat06	bcad06	bnat06	average
MPE	S0	-	30.4	42.7	39.9	33.5	36.6
	M0	-	30.2	42.1	39.2	32.9	36.1
RANDOM	RM	-	30.3	42.3	39.3	33.3	36.3
DIRECTED	DM1	S0	29.9	42.0	39.3	33.1	36.0
	DM2	S0+DM1	30.2	41.9	39.3	33.1	36.1
CNC	S0+M0		29.7	41.8	39.0	32.5	35.7
	S0+DM1		29.5	41.8	38.9	32.6	35.6
	S0+RM		29.8	41.8	38.9	32.7	35.8
	S0+DM1+DM2		29.5	41.4	38.8	32.5	35.5
	S0+RM+RM		29.8	41.8	38.8	32.6	35.7

Table 9.11: *Arabic results using a common adaptation and lattice generation pass and both single/multiple pronunciation MPE training, bnat06, bcat06, bnad06 and bcat06 testsets (WER %)*

The different results in tables 9.11 and 9.12 show clearly the cross adaptation effect that can be achieved when using one system to perform adaptation and generate lattices, and another to rescore them. The effect is demonstrated by the improved individual system performances for the directed tree systems when using a shared adaptation and lattice generation

	System	Comp. to	bnad06	bcat06	bcad06	bnat06	avg
MPE	S0	-	30.4	42.7	39.9	33.5	36.6
	M0	-	30.3	42.3	39.6	33.1	36.3
RANDOM	RM	-	30.6	42.7	39.6	33.8	36.6
DIRECTED	DM1	S0	30.1	42.4	39.6	33.6	36.4
	DM2	S0+DM1	30.6	42.1	39.4	33.5	36.4
CNC	S0+M0		29.8	42.0	39.1	32.7	35.9
	S0+DM1		29.4	41.6	38.7	32.4	35.5
	S0+RM		29.5	41.7	38.7	32.5	35.6
	S0+DM1+DM2		29.2	41.2	38.7	32.5	35.4
	S0+RM+RM		29.2	41.2	38.9	32.8	35.5

Table 9.12: *Arabic results using separate adaptation and lattice generation passes and both single/multiple pronunciation MPE training, bnat06, bcat06, bnad06 and bcat06 testsets (WER %)*

pass (table 9.11) over using independent passes (table 9.12). It is interesting then that these gains don't follow through when system combination is performed as might be expected. It is the framework with no cross-adaptation effect that gives the best results in combination for the directed tree systems. This shows that individual system error rate is not necessarily a good indicator of whether two systems are complementary or not, and, as such, simply aiming to optimise individual system performances may not lead to optimal performance in combination. The first framework is more efficient for decoding due to the shared P1+P2 pass, and it may be the case that relaxing the pruning on the first passes to give more diverse lattices for rescoring will combine the efficiency benefits of the first framework with the performance gains of the second. However, as discussed in section 3.3.4, there is a trade-off between efficiency and search errors introduced by lattice rescoring.

A second observation concerns the relative gains obtained when combining the baseline, S0, with either one or two complementary systems, DM1 and DM2, for example in table 9.11. The largest gains over the baseline are obtained by combining two complementary systems and the baseline, though the contribution from adding the second complementary system is small compared to that of just the first. Although the directed decision tree algorithm is not a boosting approach, it does try and focus the statistics generation on harder parts of the training set. Also, the broadcast news task makes use of lightly supervised training so there are likely to be some transcription errors in the training data. However, unlike discussed in section 4.1.2, the observation that random trees perform better in classification noise is unlikely to apply here. Firstly, too few decision trees are built for the effect to be noticed, and secondly the weighting is only done at a state level so the influence of poor transcriptions is expected to be minimal. It is likely that the two directed decision trees are not diverse enough, as seen with the tree divergence measure in figure 9.1(b). The fact that D1 and D2 are close together in both divergence and error rate suggest a need to somehow increase diversity between these two trees in order to see gains from a second complementary system. This cannot be done by simply increasing α, as section 9.1 shows that the tree divergence does not increase further as α becomes larger.

While the gains seen from the directed decision trees aren't as large as those from combining S0 and M0, they consistently add to the the gains from combining the single and multiple pronunciation MPE systems, while performing slightly better than the average random tree systems.

9.7 Summary

Experimental results were presented on three Broadcast News tasks - Mandarin, English and Arabic. The divergence measure was first evaluated on these Arabic and Mandarin tasks. There appears not to be a strong correlation between divergence and word error rate, as seen by the high divergences between the random trees and the baseline on the Mandarin task, which in fact have similar performances to the directed trees. The divergence was instead used to evaluate the similarity of the trees and hence determine suitable parameters for building the directed decision trees.

Preliminary results on the Mandarin task using ML trained models show that systems built using directed trees perform as well, or slightly better, than the average of those built using random trees, without the individual fluctuations in performance seen when using multiple random trees. Compared to the random tree systems, the directed tree is a deterministic algorithm allowing more control over the tree generation process, and also has the advantage that the order of combination in decoding is the same as that in training.

Next, results presented on the three languages show that the combination of complementary directed tree systems and the baseline give consistent performance gains in a range of conditions - using both ML and MPE trained models, and performing unadapted singlepass decoding or using a more complex multi-pass decoding scheme. Typically, the largest gains are obtained from the combination of one directed tree system with the baseline, but further small gains are seen from the addition of a second directed tree system.

Directed decision trees were then used in addition to two other techniques that have been empirically shown to be complementary. These were a Gaussianised frontend for the Mandarin task, and single/multiple pronunciation MPE training for the Arabic. In both cases, the combination of directed trees with these techniques has shown gains additional to those seen from just using the different techniques to independently train systems. Again, large gains are seen from the addition of one complementary system, with further small gains from a second, and the directed trees typically perform as well as or better than the average random tree.

Chapter 10
Experimental Results with Data Weighting

This chapter presents experimental results achieved when the approaches discussed in chapter 7 are used to explicitly train complementary systems. Results are presented on broadcast news English and Mandarin tasks, though the Arabic task is not used. This is because the approaches examined in this chapter alter the training algorithm, and thus cannot be used in addition to the single and multiple pronunciation training which was used in the previous chapter to incorporate additional diversity. This chapter concludes by addressing the issue of poor alignments in decoding, as discussed in section 7.3. In contrast to the experiments of chapter 9, this chapter focuses on altering the training algorithm via a data weighting, and the standard decision tree is used for all systems.

In contrast to the directed decision tree experiments in the previous chapter, a distance metric for evaluating the diversity of HMMs isn't considered in this chapter. While such a measure could prove informative, it is not expected to be as useful for evaluation. First, the explicit training schemes in this chapter are more closely linked with word error rate when compared to the implicit directed decision tree approach. Second, the time saving which could be achieved by evaluating a distance measure for HMMs is much smaller than that achieved by a distance metric for decision trees.

10.1 Word-level Active Training Results

This section considers the active ML training approach introduced in section 7.1. First, experiments on the English task show the effect of the active training as the loss function,

number of training iterations, and updated parameters are changed. Initially, a well-trained 16 component system is used as the baseline, though a simpler 4 component system is also considered to evaluate generalisation. Performance is examined on both the test and training data. To conclude, the active training is also evaluated on the Mandarin task, and is used in addition to Gaussianisation to incorporate further diversity.

As discussed in section 7.3, initial experiments showed a tendency for the ML training to increase the state occupation posteriors on portions of the training data with low weight, and decrease the state occupation posteriors on portions with high weight. Thus, the state alignments were fixed in training using a second model set. For all the experiments below, the baseline model was used as the second model for fixing the state alignments. Silence models are always given a weight of 0, and hence not updated, as these are assumed to be well trained and a complementary silence model is not expected to be useful in practice.

10.1.1 Test Data Performance

The first experiments were carried out on the English broadcast news task with an ML trained baseline model, S0, which has an average of 16 components per state. The frontend consists of a 39 dimensional feature vector including 12 PLP coefficients plus energy, first, second and third derivatives, and an HLDA transform. Training data confusion networks were obtained for S0, and used to train a complementary system, D1.

The word-level active training was evaluated as the number of training iterations increased. The threshold loss function used was

$$\begin{aligned} l(\mathcal{W}_{ref}) &= 1 \text{ if } \frac{1}{S}\sum_{s=0}^{S-1} P(\mathcal{W}_{ref}|\mathcal{O},\mathcal{M}^{(s)}) < \beta \\ &= 0 \text{ otherwise} \end{aligned} \tag{10.1}$$

This is the loss function of equation 5.13, and a value of $\beta = 0.25$ was used. That is, any reference word with a posterior less than 0.25 in the confusion network was considered incorrect, and used for training with a weight of 1. All other words were assigned a loss of 0. The threshold of 0.25 corresponds to 36.2% of the reference words being used for training. During training, the means, variances and component weights were updated, but the transition probabilities were not as they are not expected to influence the results. The results in table 10.4 below confirm this.

Table 10.1 shows the impact of the number of training iterations on the word error rate. As the number of iterations increases, the individual systems typically perform slightly worse than the baseline. For example, on the dev03 set, the baseline S0 has a word error rate of 18.1% and, after 8 iterations of training, the word error rate has increased to 18.2%. In contrast, the combination of the baseline and the complementary system improves the error rate. Again, for the dev03 set, after 8 iterations of training the combination of the two systems gives an error rate of 17.7%, which is a 0.4% absolute improvement over the baseline. On the dev03 set, the combination of S0+D1 is statistically significant for all iterations when compared with the baseline using the matched pairs test. On the eval98 set however, only the combination of S0 and D1 for the fourth iteration is significant. On the dev03 set, the performance of the active training after two iterations is 18.0%, which is a 0.1% improvement

System	Iteration		dev03	eval98
S0	HLDA		18.1	16.6
D1	THRESHOLD	2	18.0	16.7
		4	18.1	16.8
		6	18.2	16.8
		8	18.2	16.8
S0+D1	CNC	2	17.8	16.5
		4	17.7	16.4
		6	17.7	16.4
		8	17.7	16.4
S0+D1	IDEAL	2	17.0	15.7
		4	16.9	15.6
		6	16.8	15.6
		8	16.8	15.5

Table 10.1: *BN English word-level active ML training WER (%) results as the number of iterations increases. Threshold loss function with* $\beta = 0.25$ *on the dev03 and eval98 test sets*

over the baseline. This could be due to the discriminative effect of the active training giving an improvement in performance, such as is seen with MPE training.

It is interesting to see that the IDEAL combination of the two systems gives a further improvement over the CNC performance. For example, the IDEAL combination of the baseline and the complementary system after eight iterations of training is 16.8% on the dev03 set, compared to the CNC performance of 17.7%. The IDEAL combination only uses the second system when the first is incorrect, and so its improved performance suggests that combination with the second system does correct many of the errors which the first system makes. However, as CNC performs worse than the IDEAL combination, this suggests that the combination of the second system not only corrects errors made by the first, but also introduces new errors. This implies that the training algorithm does indeed correct errors made by the baseline system, but also that the degredation on the previously well modelled data is large and degrades the overall performance.

The IDEAL performance improves as the number of iterations increases, implying that the ML active training is driving the system to be more complementary as less training data is used, but the combination scheme is not taking full advantage.

Next, to determine how dependent the active training is on the form of loss function, a different loss function was used. The sum loss function of equation 5.12 with $\alpha = 1$ was used for training, repeated here:

$$l(\mathcal{W}_{ref}) \quad = \quad 1 - \frac{1}{S} \sum_{s=0}^{S-1} P(\mathcal{W}_{ref}|\mathcal{O}, \mathcal{M}^{(s)})^{\alpha} \tag{10.2}$$

This function does not make a hard decision about whether a reference word is correctly modelled or not, but instead uses a weighting based on its posterior probability in the confusion network. Thus, a far greater percentage of the data is used for training, and reference words have a continuous weighting between 0 and 1.

System	Iteration		dev03	eval98
S0	HLDA		18.1	16.6
D1	SUM	2	17.8	16.4
		4	17.8	16.5
		6	17.8	16.6
		8	17.7	16.6
S0+D1	CNC	2	17.8	16.4
		4	17.8	16.3
		6	17.7	16.3
		8	17.7	16.3
S0+D1	IDEAL	2	17.1	15.8
		4	17.1	15.7
		6	17.0	15.7
		8	16.9	15.6

Table 10.2: *BN English word-level active ML training WER (%) results as the number of iterations increases. Sum loss function on the dev03 and eval98 testsets*

The results for training with the sum loss function in table 10.2 show the same trends as for the threshold loss function results in table 10.1, implying that the form of loss function does not greatly alter the training. However, the behaviour of the two functions is slightly different. First, the individual systems perform better when training with the sum loss function, and this is probably due to the larger proportion of training data used. There is little difference in the CNC performance between the two loss functions, though all combinations are statistically significant when compared to the baseline, for both test sets. The IDEAL performance with the sum loss function is worse than for the threshold. This is probably due to the larger amount of training data used, which restricts the degree to which the training can focus on the errors, and hence the degree to which the two systems are complementary.

The difference between the performance with sum and the threshold functions suggests the effective amount of training data influences the active training. To examine this, the performance was evaluated as the threshold in the loss function changed. As the threshold increases, the proportion of reference words with weight 1 increases, and hence the effective amount of training data is increased. As words are assigned a loss of 0 or 1, a simple measure of how much data is used is the proportion of reference words with a posterior below the threshold and hence assigned a loss of 1.

The results in table 10.3 show the performance as the loss threshold changes, for two iterations of training where the means, variances and component priors are updated during training. The threshold of $\beta \leq 1.0$ corresponds to two further iterations of standard ML training with 100% of the training set being used, while the decreasing threshold corresponds to an effectively smaller training set. Table 10.3 also shows the percentage of the training data reference words assigned a loss of 1.

For a high threshold, the behaviour is like that of active training discussed in section 2.4.3, where the individual system performance improves over the baseline. For example, with a threshold of 0.75 on the dev03 set, the performance improves from 18.1% WER to 17.8%. As the threshold decreases further, the training focuses more on the errors and the individual

system performance degrades. For example, when using just 24.2% of the reference words for training, or a threshold of 0.0025, the performance on the dev03 set degrades by 1.0% absolute, to 19.1% WER. This is again similar to previous work with active training, where it has been suggested that it is not sensible to focus too much on the errors, as they are often outliers.

System		Threshold (β)	% words	dev03	eval98
S0	HLDA			18.1	16.6
D1	THRESHOLD	$\leq$1.00	100	18.0	16.6
		0.75	46.7	17.8	16.5
		0.50	41.2	17.9	16.5
		0.25	36.2	18.0	16.7
		0.0025	24.2	19.1	17.6
S0+D1	CNC	$\leq$1.00	100	18.0	16.5
		0.75	46.7	17.8	16.4
		0.50	41.2	17.8	16.4
		0.25	36.2	17.8	16.5
		0.0025	24.2	18.0	16.6
S0+D1	IDEAL	$\leq$1.00	100	17.8	16.4
		0.75	46.7	17.2	15.8
		0.50	41.2	17.1	15.8
		0.25	36.2	17.0	15.7
		0.0025	24.2	16.9	15.6

Table 10.3: *BN English word-level active ML training WER (%) results as the threshold in the loss function changes. 2 iterations of training, % words = percentage of reference words below threshold, dev03 and eval98 testsets*

It is interesting to note that the pattern of the IDEAL combination results in table 10.3 is reversed from that of the individual system results. That is, as the training focuses more on the errors, the systems become more diverse and introduce different errors, hence the IDEAL performance improves. However, the CNC performance does not reflect this increased diversity, and the combination results are stable over a range of different thresholds. The combination of the baseline and the complementary system yields some gain, with a word error rate of 17.8% for thresholds of 0.25-0.75 on the dev03 set. This is a 0.3% absolute improvement over the baseline performance of 18.1%. The IDEAL combination performance improves from 17.8% to 16.9% as the threshold decreases from 1.0 to 0.0025, and the training focuses more on the errors. Of the results obtained from confusion network combination of S0 and D1 on the dev03 set, all results are statistically significant with the exception of a threshold of 1.0, i.e. standard ML training.

The experiments above have updated the means, variances and GMM component priors. Table 10.4 shows the effect of updating the different HMM parameters using the word-level active training. Two iterations of training were performed, using the threshold loss function with $\beta = 0.25$. The means (m), variances (v), transition probabilities (t) and component priors (w) were updated. Table 10.4 shows that most of the effect of the active training is

seen when updating the means and variances, while the component weights and the transition probabilities have little impact on the overall results.

System		Updated	dev03	eval98
S0	HLDA		18.1	16.6
D1	THRESHOLD	m	17.9	16.5
		v	18.1	16.7
		mv	18.0	16.7
		mvw	18.0	16.7
		tmvw	18.0	16.7
S0+D1	CNC	m	17.8	16.4
		v	18.0	16.6
		mv	17.8	16.5
		mvw	17.8	16.5
		tmvw	17.8	16.5
S0+D1	IDEAL	m	17.2	15.9
		v	17.5	16.1
		mv	17.0	15.7
		mvw	17.0	15.7
		tmvw	17.0	15.7

Table 10.4: *BN English word-level active ML training WER (%) results as the updated parameters change. 2 iterations of training, loss threshold function with $\beta = 0.25$. m=means, v=variances, t=transition probabilities and w=component priors, dev03 and eval98 testsets*

The results above used a baseline system, S0, with an average of 16 Gaussian components per state. This system is well trained, and so it is interesting to examine the effect of the active training on a system which is not well trained, and thus evaluate generalisation. For this purpose, a baseline system with an average of 4 Gaussian components per state was built in the same way as the 16 component system, including the same decision tree. The results are shown in table 10.5. The 16 component baseline performs better than the 4 component baseline, as expected. For example, on the dev03 set, the baseline error rate for the 4 component system is 20.6% compared to 18.1% for the 16 component system. Next, two iterations of the word-level active training were performed, with a threshold of $\beta = 0.25$ in the loss function, to build a system, D1, complementary to the 4 component baseline. Performing the active training improves the performance to 20.2%, and combination of the two systems yields a final error rate of 20.0%. This is a similar pattern to the results obtained with the 16 component system, and suggests that the active training does not quickly lead to overtraining.

The results presented in this section show that it is possible to achieve gains over a baseline system by performing word-level active training. For gains to be seen for the individual systems, the complementary system must not be too different from the baseline system. That is, gains can be seen if the number of iterations is not too high and the loss function is chosen so the active training uses a large proportion of the training set. This effect has been seen with previous work on utterance-level active training as discussed in section 2.4.3.

System		dev03	eval98
S0	HLDA (4 COMPONENT)	20.6	19.5
D1	THRESHOLD	20.2	19.1
S0+D1	CNC	20.0	19.1
	IDEAL	19.4	18.4

Table 10.5: *BN English word-level active ML training WER (%) with a 4 component system. 2 iterations of training, loss threshold function with $\beta = 0.25$, dev03 and eval98 testsets*

Gains can be seen from combining the individual systems with the baseline, even when the active training has degraded the performance of the individual system. However, if the active training has driven the individual system to be too different from the baseline, then CNC does not take full advantage of the diversity. CNC appears relatively stable when combining systems which were trained using a range of parameters. In contrast, the IDEAL combination performance improves as the complementary system is driven to be more diverse than the baseline, for example by performing more iterations of training, or by focusing more on the errors through the loss function.

10.1.2 Effect on Training Data

The previous section considered the performance of the word-level active training on two independent test sets. It is also informative to consider the effect of the training on the training data. This allows issues such as overtraining and lack of generalisation to more easily be seen. Additionally, it allows a better assessment of whether the training is achieving its goal of focusing the complementary models on the errors made by the baseline system. In order to examine the performance on the training set, both recognition performance and the effect on the word posteriors are examined.

Recognition performance was evaluated on a 10 hour subset of the English training data, and decoding was carried out in the same way as for the independent test set results above. The performance on this training data subset as the number of training iterations increases is shown in table 10.6. The 16 component ML trained baseline was used, and eight iterations of word-level active training were performed using a threshold of $\beta = 0.25$ in the loss function. Thus these results are directly comparable to those in table 10.1 on the dev03 and eval98 test sets.

The results show a large improvement in performance from doing active training, which is not seen on the independent test set results of table 10.1. This performance increase is is seen to a greater extent in section 10.2.1 below when considering the effect of MPE training on the training data, and suggests that the large improvement in performance is due to the discriminative effect of the active training, rather than overtraining. After eight iterations of active training, the word error rate on this subset has dropped from 19.3% to 17.4%. The CNC performance however is slightly worse than the individual system results, though there is a potentially large gain to be achieved from the IDEAL combination.

The word-level active training algorithm directly uses the posterior probabilities of the training data reference words, in order to appropriately weight the training data. Hence, it is interesting to see the effect of the training on these word posteriors.

System	Iteration		train subset
S0	HLDA		19.3
D1	THRESHOLD	2	17.8
		4	17.6
		6	17.6
		8	17.4
S0+D1	CNC	2	17.9
		4	17.6
		6	17.6
		8	17.5
S0+D1	IDEAL	2	16.2
		4	15.9
		6	15.8
		8	15.7

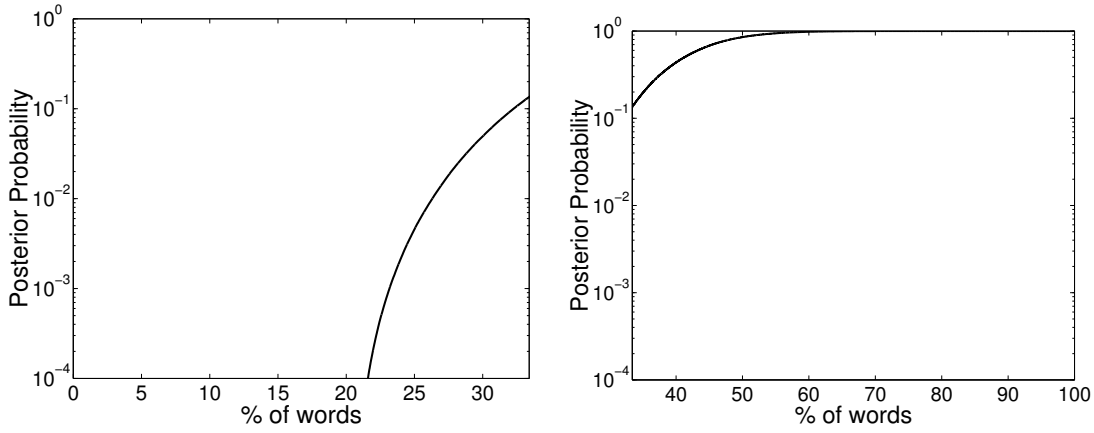
Table 10.6: *BN English active ML training WER (%) results on a 10 hour training data subset, as the number of iterations increases, threshold loss* $\beta = 0.25$

Figures 10.1(a) and (b) show the cumulative distributions of the training data reference word posteriors, obtained from the training data confusion networks from the 16 component baseline system, for (a) training data words with a posterior less than 0.25, and (b) words with a posterior greater than 0.25. This corresponds to the threshold used for training, and so the 36% of words in figure 10.1(a) had a weight of 1 and were used in the active training, while the remaining 64% in figure 10.1(b) had a weight of 0 in training. Figures 10.1(c) and (d) show the cumulative distribution of the same word posteriors after the active training, where (c) shows the same selection of training data words as in (a), and (d) shows the same words as in (b). Two iterations of active training were performed, with the threshold loss function and $\beta = 0.25$.

Thus, the effect of the training can clearly be seen on the portion of data with weight 1 by comparing figures 10.1(a) and (c). The posteriors of these reference words have increased due to the training. In figure 10.1(a), all 36% of words have a posterior of less than 0.25, while in figure 10.1(c), 31% of the words have a posterior of less than 0.25. Thus, the active training has increased the posterior of 5% of the initially badly modelled words to be above the threshold of 0.25. Additionally, in figure 10.1(a), approximately 22% of the words had a low posterior and were pruned out of the confusion networks. After active training, this figure decreased to approximately 19%.

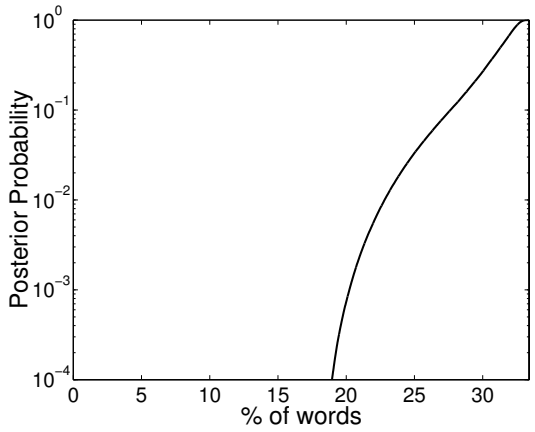
A comparison of figures 10.1(b) and (d) shows that the posteriors of previously well modelled words were also decreased by the training. In 10.1(b), the top 64% of words have a posterior greater than 0.25. In figure 10.1(d) however, the posterior of 0.25 corresponds to a figure of 40% on the x-axis, compared to 36% in figure 10.1(b), showing that the training has degraded the posterior of 4% of the words to below the threshold of 0.25. Figure 10.1(d) also shows that, of the initially well modelled words, the posterior of approximately 1% degraded sufficiently that they were pruned out of the confusion networks.

The effect of training on the reference word posteriors suggest that the training algorithm is achieving its goal of better modelling previously badly modelled data. At the same time,

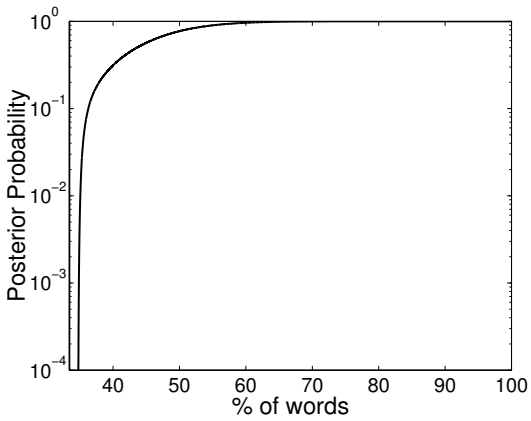


(a) S0 (bottom 36% of data)

(b) S0 (top 64% of data)



(c) D1 (the same 36% of data as in (a))



(d) D1 (the same 64% of data as in (b))

Figure 10.1: *CDFs of training data reference word posteriors, (a) and (b) before and (c) and (d) after active training with* $\beta = 0.25$ *for the BN English task*

performance deteriorates on the portion of training data which was well modelled by the baseline. While the active training improves the posteriors for the poorly modelled reference words, as in figure 10.1(c), the effect on the previously well modelled words in figure 10.1(d) is substantial. This explains the large gains achieved by the IDEAL combination. In contrast, for CNC where the posterior probabilities are used directly, the improvement seen on one portion of data may not be large enough to counteract the degradation on the other. This explains the lack of improvement from CNC on the training data, in table 10.6 above.

10.1.3 Two Complementary Systems

The word-level active training can be performed within an iterative framework for building multiple complementary systems, as was done for directed decision trees. Table 10.7 shows the effect of building a second complementary system on the English task. D1 and D2 are both trained using two iterations of active training, with a threshold loss function and $\beta = 0.25$. D1 is complementary to S0, while D2 is complementary to S0+D1.

System	Complementary to		dev03	eval98
S0	HLDA		18.1	16.6
D1	THRESHOLD	S0	18.0	16.7
D2		S0+D1	18.1	16.9
S0+D1	CNC		17.8	16.5
S0+D1+D2			17.8	16.5
S0+D1	IDEAL		17.0	15.7
S0+D1+D2			16.6	15.5

Table 10.7: *BN English active ML training WER (%) results for two complementary systems, threshold loss $\beta = 0.25$, 2 iterations of training, dev03 and eval98 testsets*

Individually, D2 performs worse than S0 and D1. This may be because the training of D1 degrades the posteriors of the well modelled data, as seen in figure 10.1 in the previous section, and D2 focuses training too much on these errors introduced by D1. When combined with the baseline and the first complementary system, D2 does not alter the CNC performance over the combination of S0 and D1, although potential improvements from the IDEAL performance can be seen.

These results suggest that modifications to the training algorithm are needed to restrict the degradation on previously well modelled data, before the algorithm is useful for building multiple complementary systems.

10.1.4 Mandarin Results

To check that the behaviour of the previous sections is not specific to the BN English task, word level active training was also evaluated on the Mandarin broadcast news task. The baseline system, S0, for this task is an ML trained model, with an HLDA frontend and a 42 dimensional feature vector, including 12 PLP coefficients, 1st, 2nd and 3rd derivatives, energy and pitch, with an average of 16 components per state. Gaussianisation was then incorporated into the frontend, to give a second baseline system, G0. Next, D1 was trained, using S0 as a starting point, to be complementary to S0 using the word-level active training. Two iterations of training were performed using the threshold loss function and $\beta = 0.25$. Additionally, the word-level active training was performed using system G0 as the starting point, but to build a model complementary to S0, yielding system DG1. DG1 is thus complementary to S0, but incorporates additional diversity due to the Gaussianised frontend.

The results in table 10.8 show the performance of these four systems individually and when combined with the S0 baseline. The addition of a Gaussianised frontend improves the baseline performance from 23.0% to 20.1% CER, and their combination further improves the performance to 21.7% CER, as expected. The word-level active training also improves the individual system performance from 23.0% to 21.1% CER, though the combination of S0 and D1 yields no gain with a performance of 24.4% CER. However, the addition of the Gaussianisation and the word-level active training gives the best performance of 21.6% when S0 and DG1 are combined, with a corresponding IDEAL performance of 19.0%. The word-level active training does give gains additional to those seen from Gaussianisation, though the

System		Complementary to	bnmdev06
S0	HLDA	-	23.0
G0	HLDA+GAUSS	-	20.1
D1	THRESHOLD	S0	21.1
DG1	THRESHOLD+GAUSS	S0	20.4
S0+G0	CNC		21.7
S0+D1			22.4
S0+DG1			21.6
S0+G0	IDEAL		19.2
S0+D1			20.2
S0+DG1			19.0

Table 10.8: *16-component BN Mandarin active training CER (%) results, 2 iterations of training, threshold loss function $\beta = 0.25$, bnmdev06 testset*

gains are small as the performance gap between the HLDA and GAUSS frontends is large to begin with.

10.2 Discriminatively Training Complementary Systems

This section discusses experiments performed with the discriminative Minimum Bayes' Risk Leveraging (MBRL) training presented in section 7.2. Results are initially presented on the broadcast news English task. First, MPE training is considered as a method for generating complementary systems. Next, MBRL training is used, and the effect of the loss function and the smoothing examined. Results are presented on both the test and training data. Additionally, the effect of altering the size of the system, via the number of Gaussian components, is examined. Finally, experiments are presented on the broadcast news Mandarin task, in addition to a Gaussianised frontend for additional diversity.

Unlike for the ML training, the loss function for MBRL training does not directly alter the effective size of the training set. Hence, there is no need for a second model to fix the state alignments.

10.2.1 MPE Training for Complementary Systems

MPE, and other discriminative training schemes, focus on errors as part of the training. Thus, it is interesting to consider standard discriminative training as a method for generating complementary systems. Normally, eight iterations of MPE training are performed to yield a well-trained system. In this section, sixteen iterations of MPE training are performed to yield an overtrained system. This overtrained system is interesting to consider as a complementary system as its training focuses more on errors and hence might be expected to be complementary to a well-trained MPE system.

An ML trained HLDA system was used as the starting point for performing MPE training. This system is the baseline S0 system of the previous section. Sixteen iterations of MPE

	Iteration(s)	dev03	eval98	train subset
MPE	0	18.1	16.6	19.3
	6	14.9	13.3	12.7
	8	14.7	13.2	12.0
	12	14.7	13.1	11.1
	16	15.2	13.7	10.8
CNC	8+0	15.3	13.8	13.4
	8+6	14.8	13.2	12.4
	8+12	14.5	13.0	11.5
	8+16	14.7	13.1	11.0
IDEAL	8+0	13.6	12.3	11.5
	8+6	14.4	12.7	11.9
	8+12	14.1	12.7	10.9
	8+16	13.7	12.4	10.3

Table 10.9: *BN English MPE training for generating complementary systems, WER (%), on the dev03 and eval98 testsets, and a 10 hour subset of training data*

training were performed, with an MMI prior for smoothing. Table 10.9 shows the results obtained on the dev03 and eval98 test sets as training progresses. As expected, the first iterations of MPE training improve the performance, reaching a minimum after about 12 iterations, before further training leads to overtrained models and a decreased performance. After 8 iterations of training, on the dev03 set, the word error rate has decreased from 18.1% to 14.7%. Normally, 8 iterations of MPE training result in a well-trained system, and this is the MPE trained system used below in this chapter as a baseline. Many iterations of MPE training degrade the performance, and after 16 iterations, performance on the dev03 set has dropped to 15.2%.

Next, this 8th iteration well-trained MPE system was combined with models obtained from different iterations. Table 10.9 shows the CNC and IDEAL performance results. Small, statistically significant, gains can be achieved by combining the 8th and the 12th iteration. For example, the error rate of the combination is 14.5% on the dev03 set, while the 8th and and 12th iterations both give error rates of 14.7%. Other combinations do not lead to gains. The IDEAL combination results show mostly small potential gains, except for the combination of the 8th iteration with either the 0th or 16th iterations. These systems are further away from the 8th iteration in terms of number of training iterations, and the gains achieved from an IDEAL combination are larger. This is similar to the results seen in the previous section, where diverse systems tend not to give good CNC performance but good IDEAL performance, and systems that are close give a much smaller IDEAL combination gain.

Table 10.9 also shows the performance obtained on a subset of training data as MPE training progresses. As was seen for the active training in section 10.1.2, there is a large jump in performance from carrying out the discriminative training, and after eight iterations of MPE training, the word error rate has dropped from 19.3% to 12.0%.

10.2.2 MBRL Test Data Performance

MBRL training is expected to have many of the same attributes as active learning, and some initial experiments showed this to be the case. Further iterations of training are expected to drive the complementary system further from the baseline, degrading the individual system performance yet leading to improved IDEAL gains. Also, it is expected, as in table 10.4, that updating the component priors and transition probabilities will have little effect on the overall performance. Hence, this section considers just the loss function and the smoothing when performing the MBRL training. The baseline system, S0, used in this section is the system obtained from eight iterations of MPE training, and is used as a starting point for MBRL training, where just the means and variances were updated.

10.2.2.1 Effect of Loss Function

Section 5.3.2 gives two loss functions for the discriminative MBRL training - a sum function and a threshold, in equations 5.15 and 5.16 respectively. The former, the threshold function, was first used for training. With this function, a lattice arc is given a weight of 1 if the corresponding reference word has a posterior less than the threshold.

The effect of the threshold for the MBRL training differs from its effect in the word-level ML active training. Previously, the threshold, β, in the loss function directly alters the amount of training data used by assigning portions of the data a weight of zero. In contrast, for the MBRL training, the threshold does not alter the effective amount of training data, but the weighting of arcs in the training lattices.

The training was performed with S0 as a static prior, as in section 2.4.2.5, and a smoothing value of $\tau = 70$. This is the same training as is performed for building gender dependent MPE trained models for the broadcast news tasks. Two iterations of training were performed to avoid overtraining the complementary systems. D1 was trained to be complementary to S0. The effect of altering the loss threshold can be seen in the results of table 10.10, and the reduced impact of the threshold on the training set size can clearly be seen as the operation of the MBRL training is stable over a range of thresholds. The table also shows, for each threshold, the percentage of reference words which are below the threshold.

For a threshold value of $\beta = 0.25$, on the dev03 test set, the performance of the individual system degrades from 14.7% to 14.9%, but the combination of S0 and D1 gives a small gain, with an error rate of 14.6%, although this gain is not statistically significant. The IDEAL combination of these two systems is 14.1% word error rate, and the IDEAL gains are also stable over a range of thresholds.

Next, to evaluate a different form of loss function, the sum function in equation 5.15 was used for training. Again, two iterations of training were carried out. Table 10.11 shows the results obtained using this different loss function. The results are very similar to those obtained with the threshold loss function. The complementary system performs worse individually than the baseline system, and gives no statistically significant gain when combined with the baseline. There are small potential gains to be obtained from performing an IDEAL combination. Thus, as for the previous active training results, the form of the loss function does not greatly alter the effect of the training.

		β	% words	dev03	eval98
S0	MPE			14.7	13.2
D1	THRESHOLD	0.99	59.1	15.0	13.5
		0.75	44.4	15.0	13.5
		0.50	39.9	15.1	13.5
		0.25	35.8	14.9	13.5
		0.01	27.2	15.1	13.6
S0+D1	CNC	0.99	59.1	14.6	13.1
		0.75	44.4	14.7	13.2
		0.50	39.9	14.7	13.2
		0.25	35.8	14.6	13.1
		0.01	27.2	14.7	13.2
S0+D1	IDEAL	0.99	59.1	14.2	12.7
		0.75	44.4	14.1	12.7
		0.50	39.9	14.2	12.7
		0.25	35.8	14.1	12.7
		0.01	27.2	14.1	12.7

Table 10.10: *English BN MBRL training results, with change in threshold, WER (%), % words = percentage of reference words below threshold, dev03 and eval98 testsets*

System		dev03	eval98
S0	MPE	14.7	13.2
D1	SUM	15.0	13.5
S0+D1	CNC	14.7	13.1
	IDEAL	14.2	12.7

Table 10.11: *English BN MBRL training results, with the sum loss function, WER (%), two iterations of training, dev03 and eval98 testsets*

10.2.2.2 Effect of Smoothing

The previous section showed that the MBRL training is not affected by the form of the loss function. This section examines the effect of smoothing on the complementary system training. As discussed in section 2.4.2.5, the smoothing interpolates the model parameter estimates between the baseline, S0, and the estimated model parameters. If the smoothing is too relaxed, the training allows the complementary system model parameters to drift too far from the baseline, and performance may be poor. Conversely, if the smoothing is too restrictive, then the complementary system may be too similar to the baseline and the systems will not be complementary. The impact of the smoothing is thus similar to the threshold, β in the loss function, and also the value of α used in the decision tree algorithm, in section 9.1. Additionally, decreasing the smoothing is expected to have a similar effect to increasing the number of training iterations.

System		τ	dev03	eval98
S0	MPE	-	14.7	13.2
D1	THRESHOLD	∞	14.7	13.2
		2500	14.8	13.3
		300	14.8	13.4
		70	15.1	13.5
		10	16.4	14.7
		0	18.8	16.5
S0+D1	CNC	∞	14.7	13.2
		2500	14.7	13.2
		300	14.6	13.2
		70	14.7	13.2
		10	14.8	13.2
		0	15.3	13.6
S0+D1	IDEAL	∞	14.7	13.2
		2500	14.5	13.1
		300	14.3	13.0
		70	14.2	12.7
		10	13.6	12.3
		0	13.5	12.1

Table 10.12: *English BN MBRL training results, with change in smoothing, WER (%), threshold loss function with $\beta = 0.5$, 2 iterations of training, dev03 and eval98 testsets*

The baseline system, S0, is the same MPE system as above, and is again used as a static prior during training. A smoothing value of $\tau = \infty$ effectively keeps the model parameters the same as the baseline, while a smoothing of $\tau = 0$ implies no smoothing. Two iterations of MBRL training were performed, using the threshold loss function with $\beta = 0.5$, and only the means and variances were updated.

The results are given in table 10.12 for a range of values of τ. As expected, as the smoothing decreases, the complementary model parameters move further from the baseline and the individual model performance degrades. With a smoothing of $\tau = 70$, the performance degrades from 14.7% to 15.1% on the dev03 set, and with no smoothing the performance

further degrades to 18.8%. However, with the exception of $\tau = 0$, the CNC performance is fairly insensitive to the smoothing, giving error rates that are very similar to the baseline system alone. In contrast, as the smoothing decreases and the complementary system becomes more diverse, the IDEAL performance improves. The only statistically significant results obtained from combination of the baseline and the complementary system are when $\tau = 0$. Previous work has found that combining systems with very different error rates leads to poor performance. It is interesting to note for these results, that the performance doesn't greatly degrade when combining systems with differing error rates. For example, S0 and the complementary system with $\tau = 10$ have error rates of 14.7% and 16.4% respectively. Their combination gives an error rate of 14.8%, which is just 0.1% worse than the baseline. For two independently trained systems with similarly differing performance, their combination would be expected to perform considerably worse.

10.2.3 Effect on the Training Data

Experiments discussed in the previous section show that the MBRL training tends to degrade the test performance of the individual systems, but that small gains can sometimes be achieved by combining the MBRL trained systems with the baseline. However, in general, the combination of the MBRL trained system and the baseline does not improve over the baseline alone. To see why this might be, the performance and effect of the algorithm on the training data is examined.

Recognition results were obtained using the same ten hour subset of training data used in section 10.1.2, as the number of iterations of MBRL training was increased. The baseline system, S0, is the same well-trained MPE system as in the previous section, and the threshold loss function was used with $\beta = 0.5$. Smoothing was performed with a static prior using $\tau = 70$. The results are shown in table 10.13.

System		Iteration	train subset
S0	MPE		12.0
D1	THRESHOLD	2	12.1
		4	12.6
		8	13.7
S0+D1	CNC	2	11.9
		4	11.9
		8	11.9
S0+D1	IDEAL	2	11.6
		4	11.4
		8	11.3

Table 10.13: *BN English MBRL training data subset recognition performance as number of iterations increases, threshold loss* $\beta = 0.5$, $\tau = 70$

The baseline performance on this subset of training data gives a word error rate of 12.0%, with the performance of the complementary system degrading as the number of iterations increases. After eight iterations of training, the word error rate is 13.7%. However, the combination of the baseline and the complementary system consistently gives an error rate

of 11.9%. Again, the IDEAL gains improve as the number of iterations increases and the complementary system becomes more diverse.

This behaviour differs from the performance of the ML word-level active training on the training set, as seen in section 10.1.2, and differs from the effect of MPE training shown previously in table 10.9. For both active and MPE training, the training improves results on the training data subset, while MBRL degrades performance.

It is also interesting to see the effect of training on the training data reference word posteriors. Figure 10.2 shows the effect of training on the posteriors, and is similar to figure 10.1 for the active training. However, the graphs in figures 10.2 and 10.1 are not directly comparable due to the different baselines and initial distributions over word posteriors, and also the different choice of threshold.

Figures 10.2(a) and (b) show the cumulative densities of the reference word posteriors obtained using S0, for the words (a) with a posterior below 0.5, and (b) with a posterior above 0.5. This threshold corresponds to 40% of words with a posterior below 0.5, and 60% with a posterior above, with approximately 23% of reference words not appearing in the confusion networks. Figures 10.2(c) and (d) show the cumulative density functions of the same portions of training data after two iterations of the MBRL training have been performed. In figure 10.2(c), approximately 36% of the reference words have a posterior greater than 0.5, compared to 40% in figure 10.2(a), showing that the MBRL training has improved the posterior of 4% of words to above the threshold of 0.5. Figure 10.2(d) shows that 2% of reference words had a degredation in their posterior from above to below 0.5.

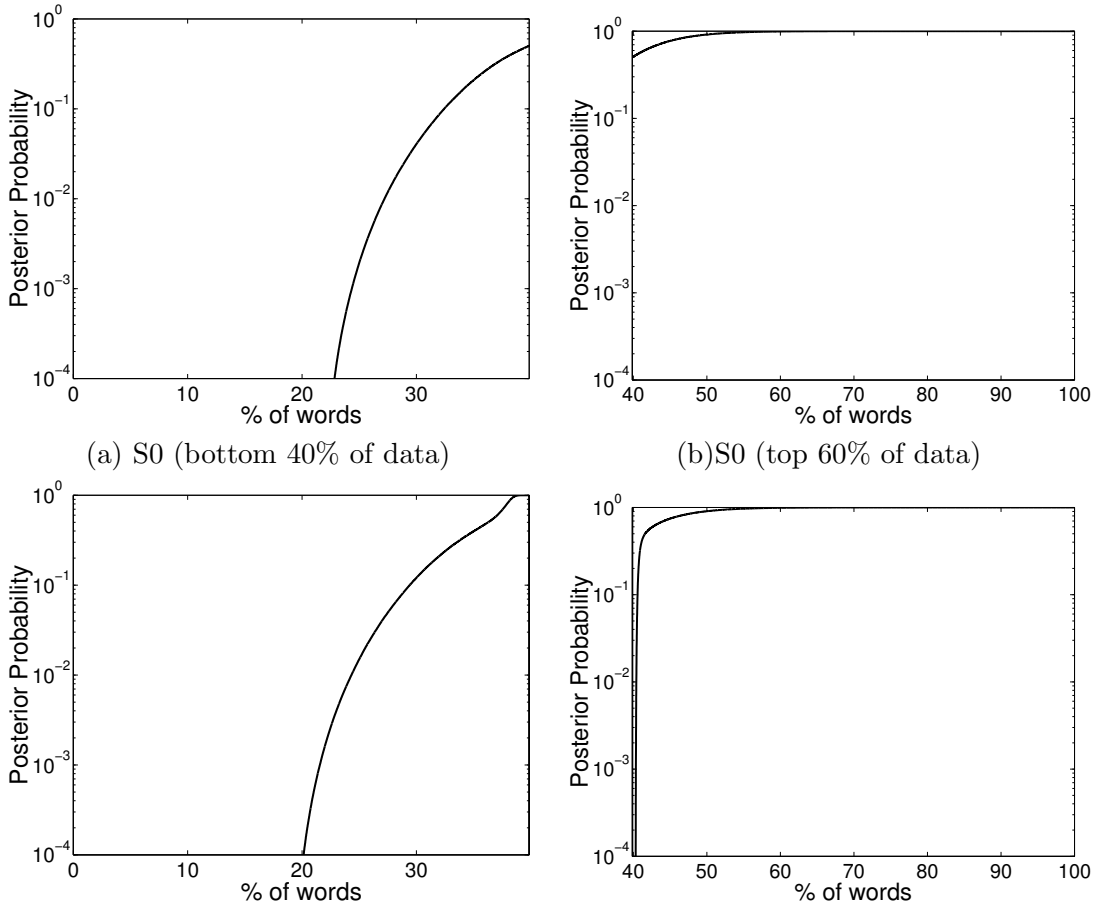
It can be seen that the effect of the MBRL training is to increase the posteriors of words which were previously badly modelled, and meanwhile decreases the posteriors of word which were previously well modelled. Thus the baseline and the MBRL trained system make different errors. As for the active training case, the MBRL training decreases the percentage of reference words which do not appear in the confusion networks. However, the improvement on the poorly modelled portion of data may not be enough to overcome the degradation on the previously well modelled data, and could lead to poor performance using confusion network combination which uses the posterior probabilities directly.

10.2.4 Overtraining and Generalisation

The results discussed above in this chapter have used a 16 component MPE trained system as the starting point for performing MBRL training. This system is well-trained, and further MPE training begins to degrade performance. Hence, to further investigate the issue of generalisation and overtraining, it is interesting to consider the MBRL training starting with a system which is not well trained. For this purpose, 8 and 4 component MPE trained systems were built, and used as the starting point for MBRL training. The 16, 8 and 4 component MPE trained systems were built in an identical manner, but with separate HLDA transforms and MPE training lattices.

The results for the 16, 8 and 4 component systems are given in table 10.14. Individually, it can be seen that the performance degrades as the number of components decreases. For example, on the dev03 set, the performance on the 16 component baseline system was 14.7%, on the 8 component system it was 15.3%, and on the 4 component system performance had dropped to 16.2%. This suggests that the 8 and 4 component systems are not well trained.

Two iterations of MBRL training were performed for each system, with a threshold loss function and $\beta = 0.5$. Smoothing with the baseline as a static prior was used, with $\tau = 70$,



(a) S0 (bottom 40% of data)

(b)S0 (top 60% of data)

(c) D1 (the same 40% of data as in (a))

(d) D1(the same 60% of data as in (b))

Figure 10.2: *CDFs of training data reference word posteriors, (a) and (b) before and (c) and (d) after MBRL training with $\beta = 0.5$, for the BN English task*

System		Components	dev03	eval98
S0	MPE	4	16.2	14.8
		8	15.3	13.9
		16	14.7	13.2
D1	THRESHOLD	4	18.4	16.6
		8	16.1	14.4
		16	15.1	13.5
S0+D1	CNC	4	16.3	14.7
		8	15.4	13.9
		16	14.7	13.2
S0+D1	IDEAL	4	15.0	13.7
		8	14.5	13.2
		16	14.2	12.7

Table 10.14: *English BN Results MBRL 4, 8 and 16 components, dev03 and eval98 testsets*

and just the means and variances were updated. It can be seen that, although absolute performance improves as the number of components increases, the pattern of results on all three systems is the same.

First, performing the MBRL training degrades the individual system results. For example, with the 4 component system, the MBRL training degrades performance from 16.2% to 18.4%, while on the 16 component system the performance degrades from 14.7% to 15.1%. Next, combining the baseline and the MBRL trained system performs about as well as the baseline alone. The combination of the 4 component baseline and the MBRL trained system gives an error rate of 16.3%, which is 0.1% absolute worse than the baseline. For the 16 component system, the combination of S0 and D1 gives an error rate of 14.7%, which is identical to the baseline performance alone. As has previously been seen, large gains could be achieved from the IDEAL combination. These results further suggest that overtraining is not an issue with the MBRL training, and starting from a simpler system does not affect the performance of the algorithm.

10.2.5 Building Multiple Complementary Systems

As with the directed decision trees and the word-level active training, MBRL training may be embedded within an iterative boosting-like framework to build multiple systems. Table 10.15 shows the results obtained when building two systems on the English task. D1 is built to be complementary to S0, as before, while D2 is trained to be complementary to S0+D1.

Individually, D2 improves over D1, suggesting that it attempts to fix some of the errors that D1 introduced. However, in combination, the IDEAL performance improves from the addition of a second system, but the CNC results do not. The performance of the CNC for the combinations S0+D1 and S0+D1+D2 is the same for both test sets.

System		Complementary to	dev03	eval98
S0	MPE	-	14.7	13.2
D1	THRESHOLD	S0	15.1	13.5
D2		S0+D1	14.9	13.2
S0+D1	CNC		14.7	13.2
S0+D1+D2			14.7	13.2
S0+D1	IDEAL		14.2	12.7
S0+D1+D2			13.9	12.6

Table 10.15: *BN English results for two complementary systems, threshold loss $\beta = 0.5$, dev03 and eval98 testsets*

10.2.6 MBRL on Broadcast News Mandarin

The discriminative MBRL results in this section have been presented on the English broadcast news task. To conclude, the training was evaluated on the Mandarin task, and the effect of training in addition to Gaussianisation was considered.

A 16 component MPE trained baseline system, S0, was used, and a 16 component system with a Gaussian frontend, G0, was also built. Then, two iterations of MBRL training were

performed, with a threshold loss function and $\beta = 0.5$. Both a standard MPE and a Gaussianised system were trained to be complementary to S0. Results are shown in table 10.16. The Gaussianised system, G0, improves over the MPE system, with the error rate dropping from 18.4% to 17.8%. The combination of S0 and G0 improved further, with an error rate of 17.4%. This effect has been seen in previous work using Gaussianisation.

The MBRL training degrades the individual system performance, with D1 yielding an error of 19.5%, compared to the baseline of 18.4%. The combination of the two performs 0.2% absolute worse than the baseline, with 18.6% CER. A fourth system, DG1, was trained to be complementary to S0, but with a Gaussianised frontend. The MBRL training degrades the performance of the Gaussianised system from 17.8% to 19.0%. S0 and DG1 have individual performances of 18.4% and 19.0% respectively, while their combination yields a performance of 17.8%. However, this performance is 0.4% absolute worse than the combination of the standard baseline and Gaussianised system.

The pattern of results for the Mandarin task are similar to the results seen on the English task, suggesting that the effect of the MBRL training is consistent across the different languages. It also does not appear beneficial to improve MBRL training by combining it with the additional diversity obtained from Gaussianisation. This is in contrast to the directed decision tree results of table 9.7, which uses the same baseline systems, and where Gaussianisation gave small additional gains to the complementary system training.

System		Complementary to	bnmdev06
S0	MPE	-	18.4
G0	MPE+GAUSS	-	17.8
D1	THRESHOLD	S0	19.5
DG1	THRESHOLD+GAUSS	S0	19.0
S0+G0			17.4
S0+D1	CNC		18.6
S0+DG1			17.8
S0+G0			16.8
S0+D1	IDEAL		17.7
S0+DG1			16.9

Table 10.16: *Mandarn BN MBRL results (CER %) in addition to Gaussianisation, $\beta = 0.5$, bnmdev06 testset*

10.3 Addressing Alignment Issues

As discussed in section 7.3, there are potential issues with the decoding of complementary systems due to poor performance on segments of the data influencing the performance on other segments. To address this, the form of decoding proposed in section 7.3.1 was examined on the English task. In this form of decoding, the confusion networks obtained from the baseline system are pruned using a confidence measure, converted to lattices, and rescored by the complementary system. Thus, this is an implicit form of system combination. This section uses the posterior probability as a confidence measure and so decoding with the complementary system is only performed on portions of the test set where the baseline best

hypothesis word has a low posterior. This is closely matched to the threshold loss function used in training though, unlike with the IDEAL combination, no use is made of the reference transcription in decoding.

The MPE trained baseline was used as the starting point for eight iterations of MBRL training, using a threshold loss function with $\beta = 0.5$ and a smoothing value of $\tau = 70$. C2 is the complementary system obtained after two iterations of MBRL training, while C8 is the system obtained after 8 iterations of MBRL training. Table 10.17 gives the baseline performance obtained when performing standard CN decoding and combination. The initial confusion networks for the restricted decoding were obtained from decoding with the baseline system, S0.

System		dev03	eval98	train subset
S0	MPE	14.7	13.2	12.0
C2	THRESHOLD-2	15.1	13.5	12.1
C8	THRESHOLD-8	16.5	14.9	13.7
S0+C2	CNC	14.7	13.2	11.9
S0+C8		14.8	13.3	11.9
S0+C2	IDEAL	14.2	12.7	11.6
S0+C8		13.8	12.4	11.3

Table 10.17: *English BN Results MBRL 16 component results (WER %)*

The restricted decoding was performed for a variety of posterior thresholds for pruning, and figures 10.3 and 10.4 show the results on the two test sets, while figure 10.5 shows the results obtained on the 10 hour subset of training data. These plots show the word error rate after this restricted decoding as the percentage of baseline CN segments being rescored is increased. For the test sets, performance obtained from rescoring the lattices with C2 is relatively consistent over a range of thresholds. However, the performance decreases as the proportion of rescored data increases when using C8 to rescore the lattices. For example, on the dev03 set, the performance degrades from 15.0% to 15.9% as the threshold increases from 0.4 to 0.95 and the percentage of data rescored increases to 35%. The performance however improves over the standard decoding result of 16.5% in table 10.17, which is expected as this form of decoding is much more restricted.

In contrast, for the training data subset, the performance first improves as the threshold is increased, and then decreases. The baseline performance is 12.0% word error rate. Rescoring the lattices with the C2 system improves the error rate to 9.0% when the threshold is 0.99 and 46% of the baseline confusion network segments are rescored. This is a 3.0% absolute improvement in word error rate, and equates to a 25% relative improvement in word error rate. This suggests that there is some issue with poor alignments affecting the decoding, and accounts for the improvement in the training subset performance as the complementary models are biased towards correcting the particular errors. Additionally, the restricted form of decoding improves markedly over the IDEAL performance, which for the combination of S0 and C8 is 11.3%. This suggests that restricting the hypothesis space for decoding means the complementary system is better able to select the correct word than when unrestricted decoding is performed.

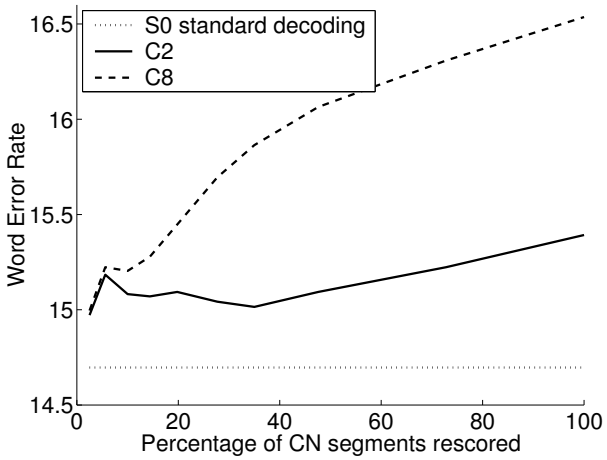


Figure 10.3: *English BN Results MBRL 16 component training subset results (WER %) for the restricted decoding with percentage of CN segments rescored, dev03*

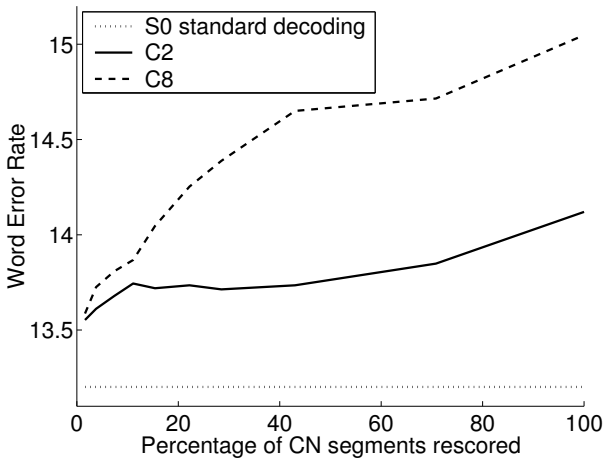


Figure 10.4: *English BN Results MBRL 16 component training subset results (WER %) for the restricted decoding with percentage of CN segments rescored, eval98*

	BASELINE	
	# incorrect	# correct
S0	2995	1943
	AFTER RESCORING	
	# corrected	# deteriorated
C2	265	523
C8	341	640

Table 10.18: *Effect of the restricted decoding on the rescored CN segments, i.e. those where the best word posterior* ≤ 0.7*, on the dev03 set using systems C2 and C8*

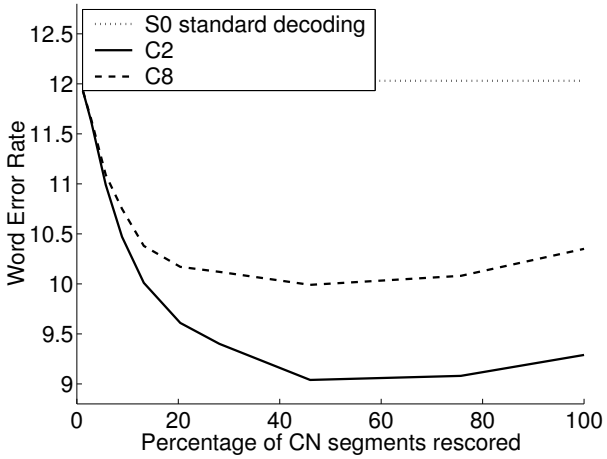


Figure 10.5: *English BN Results MBRL 16 component training subset results (WER %) for the restricted decoding with percentage of CN segments rescored, training data subset*

Table 10.18 shows a more detailed breakdown of the effect of the restricted decoding on the dev03 set, with a posterior threshold of 0.7 for pruning the confusion networks obtained from decoding with the S0 system. 4938 confusion network segments have a best word with a posterior less than 0.7, which corresponds to 14% of the total confusion network segments. Of these, 1943 are correct, and 2995 are incorrect. Thus, 60.7% of the rescored segments are incorrect. Compared to a baseline word error rate of 14.7%, this suggests that the posterior is a relatively good indicator of word errors. When rescoring these segments using C2, 265 of the previously incorrect segments are corrected, although 523 of the segments which were previously correct using S0 are made worse by rescoring with C2. Similarly, for rescoring with C8, 341 segments are corrected and 640 are made worse. Thus, the restricted decoding with the complementary system does correct a large proportion of the errors - around 10% of the segments which could be corrected are improved by rescoring - although at the cost of introducing more new errors than are corrected. Hence, with the use of an improved confidence measure or error detection algorithm, this form of decoding should yield improvements for complementary systems.

10.4 Summary

This chapter has presented experimental results on two Broadcast News tasks - English and Mandarin - using the word-level active training and the MBRL training schemes presented in chapter 7. These schemes explicitly focus on training data errors in order to build systems which make different errors, and hence are complementary. An ML trained baseline was used for the word-level active training results, and an MPE trained baseline for the MBRL training.

For both algorithms, results were first presented on the English task, varying parameters such as number of iterations, smoothing and loss function, in order to examine the effect of the training on an independent test set. Results were then presented on the training data to determine the effect of the training, and on simpler 4 and 8 component systems to examine

the issue of overtraining. To conclude, both algorithms were evaluated on the Mandarin task in addition to a Gaussianised frontend for further diversity.

Results for the ML word-level active training showed that individual system performance can be improved by the active training, and further gains can be achieved by combination with the baseline. If the training focuses too much on the errors, or drives the complementary system too far from the baseline, then the individual system performance degrades, though potential gains could be achieved using an IDEAL combination. This suggests that an alternative combination scheme might prove useful when combining the multiple systems.

In contrast, for the discriminative MBRL training, little gain is seen from training the complementary systems. Again, as the training drives the system to be more diverse, the IDEAL combination improves, but the results from CNC are reasonably stable over a range of conditions.

For both training algorithms, analysis of the training data reference word posteriors shows that the training is successful in focusing more on the errors made by the baseline, while also degrading performance on previously well modelled data. Additionally, the results obtained from decoding with the training data subset suggest that overtraining is not an issue as the performance is similar to that seen from MPE training. Results presented on simpler systems show the same patterns as on a complex system, suggesting that the training has a similar effect whether the baseline system is well-trained or not. Finally, results presented on the Mandarin task show similar patterns to the English task.

Finally, this chapter evaluated a different form of decoding, where the complementary system is used to rescore only low confidence portions of data obtained from decoding with the baseline system. This form of decoding yields improvements on the subset of training data, but not on the independent test sets where more errors were introduced than corrected. This suggests that the word posterior probability alone is not accurate enough as a measure for identifying errors, but an improved error detection algorithm could yield performance gains on the test sets with this form of decoding.

Chapter 11
Combination of Complementary Systems

The previous chapter evaluated the performance of the MBRL algorithm for explicitly training complementary systems. The results show that the training drives the systems to be diverse, and diversity can be increased by, for example, performing more iterations of training or updating more parameters. However, confusion network combination does not take full advantage of the diversity, and the combination of the baseline and the complementary system is consistent over a range of training conditions. Hence, this chapter considers approaches to combination based on the techniques discussed in sections 5.1.3 and 7.3.2. These attempt to approach the IDEAL performance by training a classifier to accurately predict word errors. These techniques are evaluated for word error detection on all data, and also on the subset of word errors which will affect the outcome of combination.

Another application of the word error detection is pruning the confusion networks for the restricted decoding evaluated previously in section 10.3, where it was seen that the posterior probability alone was not a suitable indicator of word error. With an accurate word error detection algorithm, this restricted decoding could outperform the standard combination. However, the techniques in this chapter are evaluated in the context of confusion network combination. This is due to the ease of evaluating a word-error detection task, and the IDEAL combination giving a lower bound on final word error rate if a perfect word-error detection algorithm was available.

There is some potential gain from the IDEAL combination with the directed and random decision tree systems investigated in chapter 9. However, due to the implicit complementary system generation, there is not the property that the IDEAL performance markedly improves

as the systems become more diverse through training. Thus, while the methods in this chapter may give gains with the directed decision tree systems, it is more interesting to consider the application to MBRL trained systems. Also, these techniques are not evaluated for the word-level active training, as the results are expected to be similar to those obtained with the MBRL systems.

The results in this chapter are evaluated on the English task, using the same systems as for the restricted decoding in section 10.3. Table 11.1 shows the baseline performance of the S0 system, and the performance of two complementary systems, C2 and C8. These are obtained from two and eight iterations of MBRL training respectively. It can be seen that the IDEAL combination gives gains of approximately 0.5% and 1.0% absolute over the baseline and CNC performances.

This chapter first investigates two approaches to combination using a global parameter - a system weighting and a threshold. Next, logistic regression is used as a classifier for word error detection, and its application to system combination is discussed.

System		dev03	eval98
S0	MPE	14.7	13.2
C2	THRESHOLD-2	15.1	13.5
C8	THRESHOLD-8	16.5	14.9
S0+C2	CNC	14.7	13.2
S0+C8		14.8	13.3
S0+C2	IDEAL	14.2	12.7
S0+C8		13.8	12.4

Table 11.1: *English BN Results MBRL 16 component*

11.1 Global Approaches to Combination

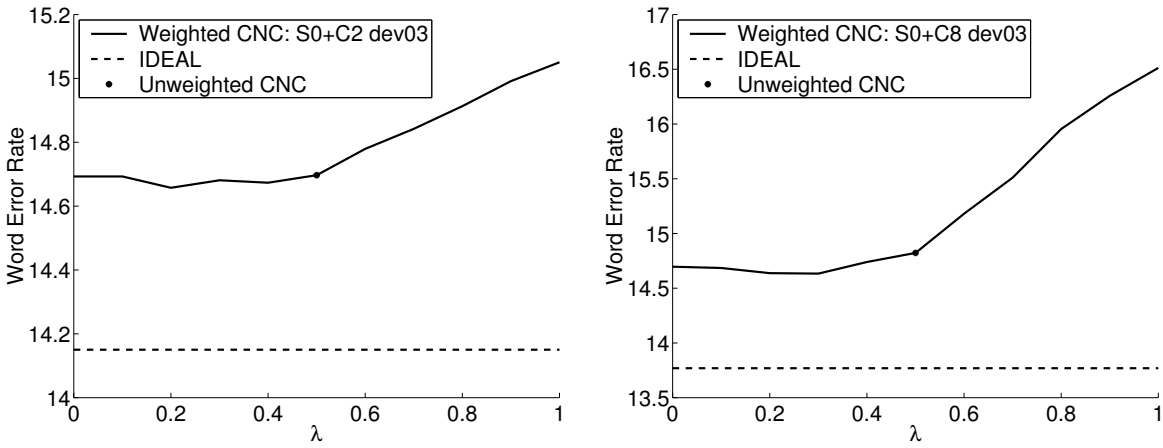
Confusion network combination used in the previous two chapters treats all systems equally during combination. The word posterior is calculated as a simple unweighted average of word posteriors from each system. However, the systems being combined typically don't have equal performance, and it might be beneficial to take this into account during combination. Alternatives are to introduce a global weighting for each system, or to use a global posterior threshold for performing the combination. These two methods are discussed in this section.

11.1.1 Global Weighting

A weighting for each system during confusion network combination, as discussed in section 3.3.1.4, calculates a weighted average of word posteriors. For the combination of two systems, the posterior of a word $\mathcal{W}$ given systems S0 and C2 becomes

$$P(\mathcal{W}|\text{S0},\text{C2},\mathcal{O}) = (1-\lambda)P(\mathcal{W}|\text{S0},\mathcal{O}) + \lambda P(\mathcal{W}|\text{C2},\mathcal{O}) \tag{11.1}$$

where λ is the weight assigned to system S0. Weights can be calculated in different ways, though better systems typically have a higher weighting. In this form of combination, $\lambda = 0$



(a) S0+C2 (b) S0+C8

Figure 11.1: *Global weighting for confusion network combination, dev03 set*

corresponds to just using S0, while $\lambda = 1$ corresponds to just using the complementary system C2. $\lambda = 0.5$ corresponds to the unweighted CNC. This weighted combination of complementary systems is similar to that used in boosting, although the MBRL algorithm does not calculate weights as part of the training.

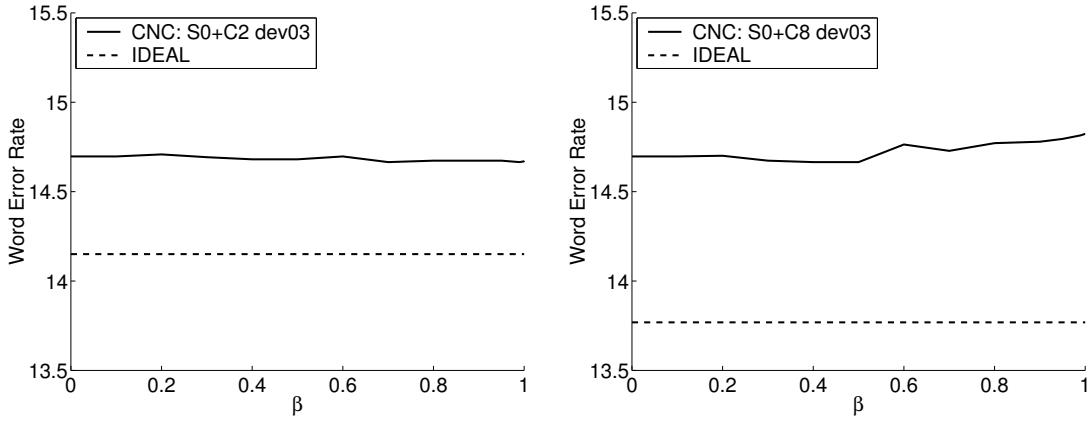
The graphs in figure 11.1 show the effect of altering λ on the dev03 set performance. Figure 11.1(a) shows the combination of S0 and C2, while figure 11.1(b) shows the combination of S0 and C8. The IDEAL combination of the two systems is also shown. As λ increases from 0 to 0.5, a small gain over the baseline and the unweighted combination can be seen. For S0+C8, the combination of the two systems yields a performance of 14.6% for $\lambda = 0.2$, while the baseline performance is 14.7%. However, the weighted combination does not reach the IDEAL combination of 13.8%.

11.1.2 Global Posterior Threshold

The performance of the global weighting for combination suggests a need to alter the combination at a more local level. That is, to decide at a finer granularity whether to perform the combination of the baseline with the complementary system. Selecting whether to combine with a second system at the confusion segment level is close to the IDEAL combination, which uses the reference transcription to make the decision. A straightforward approach in decoding is to use a threshold on the best word posterior in a confusion segment to decide whether to combine with the corresponding confusion segment in the second system.

Thus, if the posterior of the best word $\hat{\mathcal{W}}$ is above a threshold, β, then it is considered correct and is selected as the hypothesis word. Otherwise, the combination with the second system is carried out for that segment. This mirrors the IDEAL combination, and also the threshold loss function in training. The combination S0+C2 becomes

$$P(\mathcal{W}|\text{S0},\text{C2}) = \begin{cases} P(\mathcal{W}|\text{S0}) & \text{if } P(\hat{\mathcal{W}}|\text{S0}) > \beta \\ \frac{1}{2}P(\mathcal{W}|\text{S0}) + \frac{1}{2}P(\mathcal{W}|\text{C2}) & \text{otherwise} \end{cases}$$



(a) S0+C2 (b) S0+C8

Figure 11.2: *Global posterior threshold for confusion network combination, dev03 set*

In this form of combination, a threshold of $\beta = 0$ corresponds to only using the system S0. As the threshold increases, the number of CN segments combined with the second system increases, and a threshold of 1 corresponds to the standard unweighted CNC of S0 and the complementary system C2. A weighted combination was not used as the previous section showed little gain from weighting the systems.

The graphs in figure 11.2 show the effect of altering the posterior threshold for this form of CNC on the dev03 set, along with the IDEAL combination result. Over a range of thresholds, the effect of the posterior threshold has very little effect on the combination performance, for both combinations S0+C2 and S0+C8 in figures 11.2(a) and (b) respectively. As for the restricted decoding in section 10.3, this suggests that the posterior probability is not informative for use with system combination.

11.2 Word Error Detection and Combination using Single Features

The global weighting and threshold on the best word posterior in the previous section do not yield a gain which is close to the IDEAL performance. However, the posterior probability is known to be a reasonable confidence measure, and so its use might be expected to yield performance gains when used for combination. This section examines the performance of the posterior threshold in more detail, and also considers two other features for error detection and combination - the number of alternative words in the confusion segment, and the entropy of the segment, H, given by

$$H = -\sum_{\mathcal{W} \in \boldsymbol{\mathcal{W}}} P(\mathcal{W}) \log P(\mathcal{W}) \tag{11.2}$$

where $\boldsymbol{\mathcal{W}}$ is the set of all competing words in the confusion network segment. These three single features can be used in a threshold combination scheme, like that in section 11.1.2 above. A high posterior probability for the best hypothesised word, low entropy, and low

number of alternative words in the CN segment all indicate a high confidence. The use of single features for word error detection is a simplified version of the classification approach presented in section 7.3.2. The four classes previously discussed are used in this chapter. They are

- **Class 0:** S0 is correct but the output from combination S0+S1 is not
- **Class 1:** S0 is incorrect but the combination of S0+S1 is correct
- **Class 2:** both S0 and the combination are incorrect
- **Class 3:** both S0 and the combination are correct

Table 11.2 gives the number of examples in each class for the combination of S0 and C2. Both the dev03 and eval98 sets are included in the totals, and all confusion segments from the two sets are used, whether the hypothesised best word is a valid word or a deletion, indicated by a !NULL arc. Classes 2 and 3 are those where the combination of the two systems does not alter the best word hypothesis, while classes 0 and 1 are those CN segments where the combination does alter the final word error rate. As expected, classes 2 and 3 have many more examples than classes 0 and 1.

	# examples
class 0	396
class 1	394
class 2	16378
class 3	66002

Table 11.2: *Class sizes for error detection, combined dev03 and eval98 sets, for the combination S0+C2*

For the task of word error detection, classes 0 and 3 are combined to give the class of words which are correctly hypothesised by the baseline system S0, while classes 1 and 2 are combined to give the class of word errors. There are many more segments belonging to the class of correct words, as the word error rate is around 14%. Figure 11.3 shows the ROC plot for word error detection for the three features; posterior, entropy and number of alternative words. The ROC plot shows the proportion of correct words falsely identified as errors against the proportion of errors correctly identified, as the threshold is altered. Thus the IDEAL combination corresponds to the top left hand corner of the ROC plot where all errors are correctly identified, and no correct words are mistakenly identified as an error. The dotted line on the diagonal corresponds to a feature which is not informative for the task of word error detection. It can be seen that all three features are reasonably good indicators for word error. Posterior and entropy perform similarly, and the number of alternatives is a slightly worse indicator. Other evaluation metrics, such as the F-measure, are not relevant for evaluating performance as the precision is more important than the recall for application of these methods to combination.

As previously mentioned, a good feature for error detection does not necessarily aid combination if the correctly classified words mainly belong to classes 2 and 3, i.e. those which

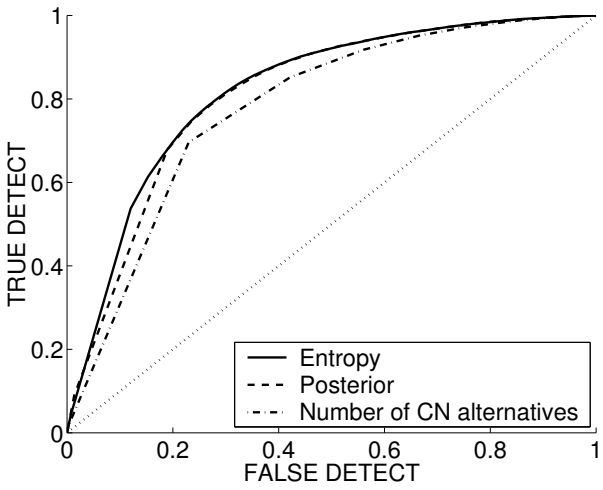


Figure 11.3: *ROC for word error detection using system S0 with the posterior probability, number of alternative words, and CN segment entropy*

do not alter the combination performance. Figure 11.4 shows the ROC curve for error detection as the threshold is changed for the three features, but only on the subset of examples from classes 0 and 1 which alter the performance after combining the S0 and C2 systems. Again, the IDEAL combination corresponds to the top left hand corner of the ROC plot. In contrast to the previous ROC plot in figure 11.3, it can be seen that the three features are not useful for identifying the subset of errors which alter the combination performance as they are much closer to the diagonal. This explains the poor performance of the posterior threshold in combination, as seen in figures 11.2 (a) and (b) of the previous section, as the posterior probability is not a good indicator for errors which affect the combination. In both plots 11.3 and 11.4, the posterior probability, entropy and number of alternative words in the confusion segment have similar performance for error detection, which is likely to be because the features are highly correlated. This is because the posterior probability and the number of alternatives are both used in the calculation of the entropy, in equation 11.2.

Section 11.1.2 showed the recognition results obtained when using a threshold on posterior probability to identify word errors. The recognition performance using thresholds on the entropy and number of alternative words are not evaluated, as they are expected to perform similarly to the posterior probability.

11.3 Word Error Detection and Combination using Multiple Features

A natural extension to using single features for word error detection is to train a classifier using multiple features for detecting errors. In this section, logistic regression, discussed in section 5.1.3.1, is used as the classifier as initial experiments showed it outperformed the more complex support vector machine. Classification techniques for word error detection have previously shown limited results [165]. However, while classification techniques have been applied to ROVER combination [68], they have not been applied to confusion network

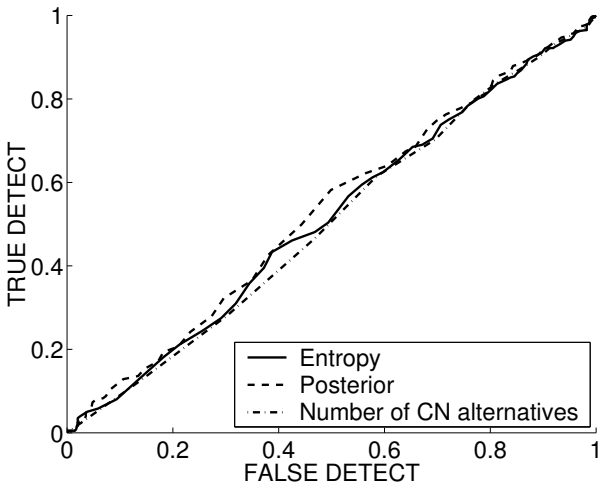


Figure 11.4: *ROC for word error detection from the S0 system, only on the subset of words which alter the combination S0+C2*

combination. Additionally, it is expected that, due to the MBRL training algorithm, the complementary systems combined in this chapter will perform well on some parts of the data and poorly on others. Hence, classification techniques may prove useful at identifying relevant errors for combining the complementary systems.

To begin, features are extracted from each segment of the confusion networks obtained from decoding with S0, and also from the force-aligned best transcription, as proposed in section 7.3.2. All features are normalised to have a mean of 0 and a variance of 1. Five sets of features were extracted, grouped according to their source:

Confusion Segment Features - features extracted directly from the confusion segment

- segment entropy
- number of alternative words in segment
- posterior probability of best word

Lattice Arc Features - features calculated from the lattice arcs which were clustered to give the best word in the CN segment

- number of clustered lattice arcs
- best language model score of clustered arcs
- best acoustic model score of clustered arcs
- best arc likelihood of clustered arcs
- variance in start times of clustered arcs
- variance in end times of clustered arcs

Segment Context - information about the preceding and following confusion segments

- best word to left is !NULL
- best word to right is !NULL
- segment at start of utterance

- segment at end of utterance

Duration - extracted from the force-aligned best transcription

- duration of word in characters
- duration of word in seconds

Language Model Information - extracted from the best transcription

- LM backoff mode
- LM probability

The CN segment features - posterior probability, entropy, and number of alternatives - are the single features used in the previous section. Of the available features, only the CN segment and context information are available if the best word in the segment is a deletion, or !NULL arc. This is because a !NULL word does not appear in the force-aligned best transcription, and there may not necessarily be an appropriate silence segment to align it with. Thus, the two cases are treated differently, and only the full set of features are used when the best word is not !NULL. An alternative would be to assign default values for !NULL words and train just one classifier, but this could distort the results. The number of examples for each class are given in table 11.3.

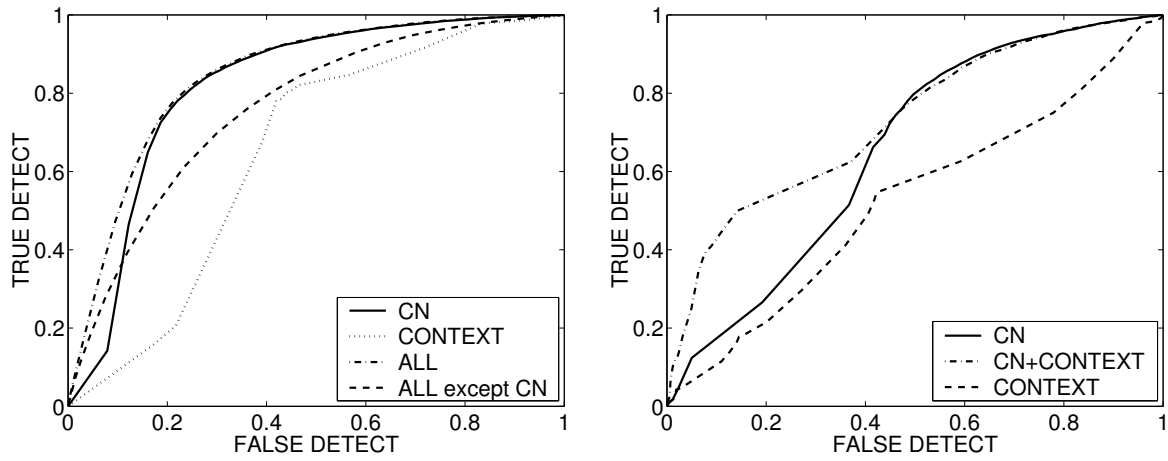
	# error	# correct	# total
!NULL	1999	11364	13363
non-!NULL	10255	48348	58603

Table 11.3: *Class sizes for error detection of words hypothesised by S0, split into !NULL and non-!NULL words*

To evaluate the performance of logistic regression for error detection, n-fold cross-validation was used with all of the data from the eval98 and dev03 test sets. The segmentation into $n = 27$ subsets was done to avoid words from one speaker being in multiple sets. Figure 11.5 shows the ROC plots obtained from the task of error detection, while figure 11.6 shows the performance of the logistic regression on the subset of examples from classes 0 and 1 which affect the outcome of the combination S0+C2. Figure 11.5 corresponds to figure 11.3 of the previous section, and figure 11.6 to figure 11.4, but the plots in this section are split according to whether the best hypothesised word is a valid word or a deletion, i.e. !NULL arc.

Figure 11.5(a) shows the performance of the logistic regression on the non-!NULL words using the CN segment features, the context information, all features, and all features with the exception of the CN segment information. Other combinations of the five sets of features perform similarly to all features and so aren't shown. The context information by itself is a reasonably good indicator for word errors, and the addition of the duration, arc, and language model information improves the detection further. However, the CN segment information alone outperforms all the other features. The use of all five feature sets performs similarly to just the CN information alone, suggesting that they do not contain extra information which is not already contained in the three CN segment features.

Figure 11.5(b) shows the error detection performance on the subset of !NULL words, using only the CN segment information, the context information, and the combination of the two. In this case, the context information is far less useful than for the non-!NULL words, and



(a) non-!NULL words (b) !NULL words only

Figure 11.5: *ROC for error detection with multiple features, all of the eval98 and dev03 sets*

the CN segment information again performs reasonably well. However, for both the !NULL and non-!NULL words, the use of multiple features and logistic regression does not appear to outperform the use of single features, as presented in the previous section. Additionally, the features are more informative for the case of non-!NULL words than for !NULL words.

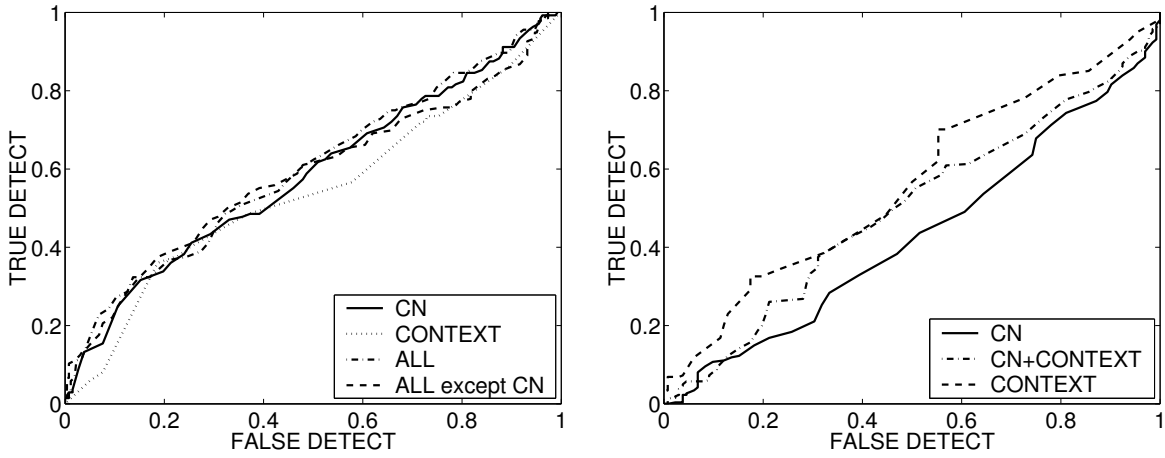
Figures 11.6 (a) and (b) show the same plots as in figure 11.5 but for the subset of features which make a difference to combination. As above, the classifier is trained to detect errors, using all words from the dev03 and eval98 sets, but the evaluation is performed on just those examples from classes 0 and 1 which alter the final word error rate. This is due to there not being enough training examples to train a classifier just on the examples from classes 0 and 1. As with the single features above, while the multiple features are a reasonable indicator for error detection in figure 11.5, they do not perform well on the subset of words which affect the combination for both the !NULL and non-!NULL words, in figure 11.6. Thus, it is expected that integrating multiple baseline features and logistic regression with CNC will not improve performance and so no recognition performance was obtained.

The classifiers above have only been trained on features extracted from the baseline system, S0. It is possible to extract the same features from the complementary system, and thus double the size of the feature vector. Additionally, three further features can be extracted from the combination:

Combination information

- best word posterior after combination
- posterior of best word from the baseline system in the complementary system CN
- posterior of best word from the complementary system in the baseline CN

However, the case of !NULL words must be appropriately dealt with. In contrast to the previous experiments, there are now four cases which must be handled as either the first, second, or both systems may have the best word in the CN segment being !NULL. To investigate the performance of these additional features, only the case of both systems having



(a) non-!NULL words (b) !NULL words only

Figure 11.6: *ROC for error detection with multiple features, on the subset of the eval98 and dev03 sets which alter the combination S0+C2*

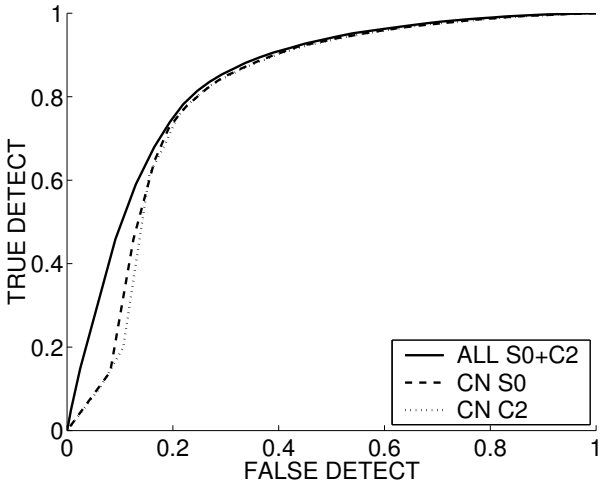


Figure 11.7: *ROC for word error detection using multiple features from systems S0 and C2, for all words*

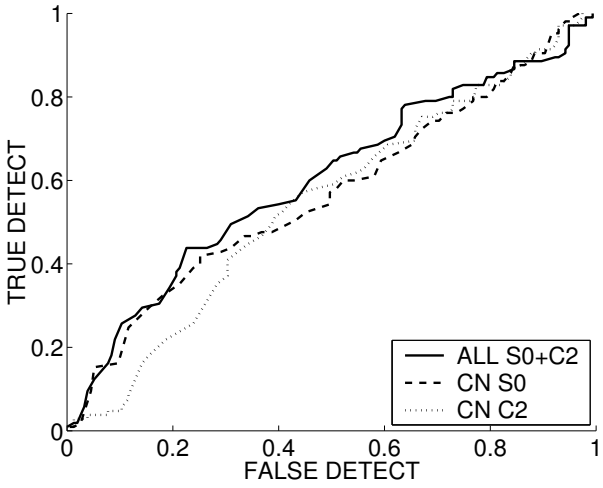


Figure 11.8: *ROC for word error detection using multiple features from systems S0 and C2, for the subset of words which affects the combination*

non-!NULL words is considered, as the plots above show poorer results when identifying errorful !NULL words.

Again, n-fold cross-validation was used when training the classifier, and ROC plots were obtained for the available features. Figure 11.7 shows the performance of all the available features from both systems, along with the CN segment information from each system alone. Figure 11.8 shows the performance of the same classifier when applied to the subset of examples which affect the combination. The effect of the logistic regression is the same as seen previously. All the features do perform slightly better when used together than the CN segment information alone, and they appear to be a relatively good indicator for error detection. However, when applied to the subset of examples which affect the combination, they are no longer a reliable indicator as the ROC curves lie close to the diagonal.

This section has evaluated the application of word error detection to combination, with features extracted from both the baseline and the complementary systems. It would be preferable to train the classifier only on those examples from classes 0 and 1 which alter the combination. However, there is far less data available from these classes, and so it is not possible to robustly train a classifier. This is a particular problem when the task is divided further to account for !NULL arcs in one or both of the systems as not all features are available for these arcs.

11.4 Summary

This chapter has looked at alternative approaches for combining complementary systems. These are motivated by the potential IDEAL combination gains seen in the previous experimental results, which are not seen with confusion network combination. If a suitable confidence measure could be computed, and word errors accurately detected, then the combination could be improved to achieve the IDEAL performance. Additionally, the restricted decoding scheme evaluated in the previous chapter may lead to improved results if used with an accurate word error detection algorithm.

First, a global weighting and posterior threshold were evaluated, and it was seen that these do not improve over the standard confusion network combination. Next, word error detection and its application to combination was investigated. Single features were first used, including posterior probability, entropy and number of alternatives in the CN segment. Then, logistic regression was used to train a classifier using multiple features extracted from the decoding. It was seen that the multiple features do not perform better than the single features for word error detection, and using the best word posterior or CN segment entropy gives the best prediction of word errors. However, when applied to combination, these techniques do not perform well enough on the relevant subset of data to expect gains from combination.

Chapter 12
Conclusions

In this thesis, the problem of generating and combining complementary systems for ASR is investigated. First, an approach to building complementary systems was proposed where the decision tree generation is altered to avoid clustering states which are associated with confusions, leading to systems making different errors. Next, two approaches to directly training complementary systems were proposed, based on maximum likelihood and minimum Bayes risk training. The three algorithms take into account the performance of a number of previous systems through a data weighting based on confusion network combination, and can be embedded within an iterative framework for building multiple complementary systems. Finally, alternative methods for combination of complementary systems were investigated, based on existing approaches for predicting word errors and confidence. In this chapter, a detailed summary of this thesis is presented, followed by possible directions for further work.

12.1 Review of Work

Chapters 6 and 7 of this thesis presented three algorithms for generating complementary systems for ASR, based on a data weighting proposed in section 5.3. Typically, complementary systems for ASR are found in an ad-hoc manner, by building multiple diverse systems and selecting those which yield improvements upon combination. These systems can be diverse in many ways, for example using different frontends, training algorithms or segmentations, as discussed in section 3.1. It has been found that independently trained systems with very different error rates do not normally give improvements when combined. Also, the performance of independent systems when combined cannot be predicted before performing the combination. Thus, previous work has looked at explicitly training complementary systems, and

existing approaches were discussed in chapter 4. These approaches include injecting randomness, adapting the boosting algorithm for ASR, training specialist models to fix errors, and training multiple model parameters in parallel. The latter method is feasible if the models are combined in a synchronous manner during decoding, so they are constrained to be in the same state at the same time. For asynchronous combination however, where there are two independent state switching processes, the training is infeasible and approximate techniques must be used. When the complementary models are trained in an iterative fashion, such as in boosting, training must take into account the performance of previous systems. Hence, the appropriate training algorithm for generating complementary systems depends on the method employed to combine the complementary systems.

The three algorithms presented in this thesis use confusion network combination (CNC) as a method for combining systems in both the testing and training algorithms. CNC and other existing combination methods are detailed in chapter 3. The three main classes of combination algorithm are hypothesis combination, which includes CNC, likelihood combination, and implicit combination. This thesis makes use of hypothesis combination schemes, based on confusion networks, as these allow the easy separation of the training algorithm and combination scheme. Confusion network combination is used to incorporate information from multiple models, and weight the training appropriately to focus on errorful portions of data, thus generating a system with complementary errors.

Section 6.1 presents the directed decision tree algorithm for generating complementary systems. This algorithm biases the decision tree to separate states which previously led to errors. Thus, the different clustering of states in the decision tree allows a model which was not possible with the original tying. Additionally, a divergence measure was proposed in section 6.3 to allow easy comparison of decision trees without having to train the corresponding systems. Rather than measure cluster similarity, this measure makes direct use of the structure of decision trees, and calculates an efficient metric based on the similarity between state distributions in the two trees.

The modified ML algorithm presented in section 7.1 uses active training to explicitly generate complementary systems. This algorithm differs from previous work with active training in its application to complementary system generation by taking multiple previous systems into account during the training data weighting. Additionally, the weighting is applied at the word level to better match the word level combination scheme. The minimum Bayes' risk leveraging (MBRL) algorithm presented in section 7.2 makes use of the discriminative minimum Bayes' risk criterion to build complementary systems. It is related to existing discriminative training criteria, and can be optimised in the same way, but differs in its aim of incorporating information from multiple previous systems into the training. Both MBRL and the word-level active training weight the training data so that the training focuses on errors made by previous systems, and not on data which was previously well modelled. In this way, the complementary systems have a greater flexibility to model errors, although they may degrade performance on the previously well modelled data.

The three algorithms use a common data weighting approach, discussed in section 5.3. This data weighting uses confusion networks aligned with the reference transcription, and weights words or lattice arcs based on their posterior in the confusion networks. These weights are then applied directly at the state level of the HMM. This differs from previous data weighting methods, discussed in section 5.2, as it does not rely on force-aligning the data. Hence, a data weighting obtained with one system can easily be applied to another, without needing to realign the data. In this thesis, a threshold and a sum function are used

to obtain the data weighting, although these could be replaced by any suitable function. Furthermore, the data weighting easily takes multiple previous systems into account, hence the three algorithms can be embedded within an iterative boosting-like framework to build multiple complementary systems.

MBRL and the word-level active training scheme explicitly focus on previous errors in training, and hence there are potential issues with their performance in decoding. Particularly, it is expected that they will perform well on specific portions of the data and badly on other portions. Section 7.3 discusses alternative approaches to combination, to address these potential issues. First, a modified form of decoding was presented in section 7.3.1 where the complementary system is only used to rescore portions of the data where it is believed that the first system makes an error. This is used in a similar way to the acoustic code-breaking discussed in section 4.2.4, as the second model is used only to resolve potential confusions made by the first. Next, section 7.3.2 discusses the application of classification techniques to improve the combination. This is related to the confidence prediction and word error detection methods detailed in section 5.1.3. Both approaches attempt to bring the final combination closer to that used in training by mirroring the threshold loss function, and thus improve the combination of the baseline and complementary systems.

Experimental results obtained using the directed decision tree algorithm on three LVCSR tasks are presented in chapter 9. First, two directed trees were built and the divergence measure evaluated. It was found that focusing more on the errors increased the tree divergence, though a second complementary decision tree is less diverse than a first. Next, the directed tree was compared to the performance of a random decision tree in sections 9.2 and 9.3. Results show that the directed tree performs better than the average random tree, but without the fluctuations seen from introducing randomness.

Results were then obtained from decoding in a single and multi-pass framework using the directed tree systems. Sections 9.4 and 9.6 respectively show that gains can be achieved from the addition of one complementary system, and further small gains achieved from a second. Finally, the directed tree approach was used in addition to an alternative frontend in section 9.5, and an altered training algorithm in section 9.6, to include extra diversity, and results show that further gains can be achieved. Thus, the directed tree algorithm is an effective approach for generating complementary systems for large vocabulary recognition.

Chapter 10 presents experimental results for the word-level active training and the MBRL algorithm on two LVCSR tasks. Section 10.1 first discusses results for the active training algorithm. Performance was evaluated as the number of training iterations increased, and it was seen that further training leads to more diverse systems. Next, the effect of the data weighting was investigated. When a large portion of the training set is used, the algorithm behaves just as standard active training and improves the individual system performance. However, as the weighting focused more on the errors, the individual system performance degrades, while gains were seen from combination. As the systems become more diverse through further training iterations or an increased focus on the errors, the IDEAL combination performance improves, suggesting that the systems do make complementary errors. On a simpler system, the same behaviour was seen, while the addition of a second complementary system showed no additional gains. To investigate the potential issue of overtraining, performance was examined on the training data. Analysis of the reference word posteriors before and after training show that the algorithm has the desired effect of improving the modelling of badly modelled data, while degrading performance on well modelled data. Finally, the addition

of the active training to an alternative frontend for additional diversity led to further small gains.

Section 10.2 presented results obtained using the discriminative MBRL algorithm for building complementary systems. First, MPE training was examined as a method of generating complementary systems, and small gains were seen. Next, the MBRL algorithm was evaluated as the loss function and smoothing were varied. The results were similar to the active training algorithm, showing that the training can drive the system to be more diverse, though these gains were not taken advantage of with the standard combination. The same behaviour was seen for two simpler systems. Analysis of the training data performance again showed that the MBRL training degrades performance on previously well modelled data, but improves performance on previously badly modelled data, as anticipated. Thus, both the active training and MBRL algorithms perform as expected on the training data, and the algorithms do drive the systems to be more diverse and make complementary errors. However, the current combination approach does not take advantage, probably because the degradation on previously well modelled data is large.

Section 10.3 examined a restricted form of decoding to better match the training algorithm, where the complementary system was only used to rescore segments of the data when the first system was not confident. On the training data, this decoding substantially improved the results. On the test data however, the performance degraded using this form of decoding. Although a number of errors were corrected by this approach, further errors were introduced, implying that the posterior probability as a confidence measure was not accurate enough for identifying poorly recognised regions of data.

Finally, chapter 11 examined approaches to word error detection for improving the combination of multiple systems. First, a global weighting and posterior threshold were used for combination, and results showed no improvement over standard confusion network combination. Next, the performance of both single and multiple features was examined for identifying word errors, the latter using logistic regression to train a binary classifier. These features were shown to be a reasonably good indicator for predicting word errors, although a combination of multiple features performed only slightly better than the posterior probability alone. When applied to the subset of errors which affect the combination however, the features were no longer a reliable indicator. Hence, an improved method of word error detection is needed to achieve the potential gains from combination.

12.2 Future Work

Very little work has previously looked at explicitly generating complementary systems for automatic speech recognition, and hence there is much scope for further work. In particular, several areas of this thesis may be expanded. These are summarised below.

- The experimental results with directed decision trees used a divergence measure for evaluating the distance between trees. A similar metric, such as in [22], for similarity between HMMs might prove useful when building complementary systems, to provide further insight into the effect of the algorithm.

- The three algorithms in this thesis used confusion networks for encoding the confusions made by previous systems. Other forms of combination, such as fWER combination or

pinched lattices might perform differently. This would require alterations to the training and data weighting to take these different forms of combination into account.

- In the same way as the MPE criterion has been applied to feature transform estimation to calculate discriminative features [125], the MBRL or related criterion might be applicable to estimating complementary features.

- The experimental results in this thesis have evaluated performance on three similarly sized broadcast news tasks with different languages. These have common properties, such as unsupervised or lightly supervised training, noise, and ungrammatical, spontaneous speech. Hence, it would be interesting to evaluate the algorithms on other tasks, whether they differ in their properties or size.

- Gaussianisation and multiple/single pronunciation MPE training were used in this thesis to introduce extra diversity into training. However, there are many other techniques which have been empirically shown to be complementary, and could be used in addition to the algorithms proposed in this thesis. For example, previous work has shown that different frontends have complementary features, or that multiple stream systems with streams for different frequency bands can provide complementary information. With additional forms of diversity, it might be possible to successfully build multiple complementary systems.

- The results in this thesis showed that large gains could be achieved with an IDEAL combination, but this is difficult to achieve in practice. Hence, other methods of combination might achieve a performance closer to the IDEAL. For example, additional features might be incorporated into the classification task of chapter 11, or with a large enough test set there may be enough examples to explicitly train a classifier for the task of combination rather than word error detection. This task differs from standard error detection as only a subset of errors are important for system combination. It might also prove useful to somehow decouple the confidence scoring from features calculated during recognition, as these are normally correlated with the posterior probability.

References

[1] M. Afify, L. Nguyen, B. Xiang, S. Abdou, and J. Makhoul. Recent Progress in Arabic Broadcast News Transcription at BBN. In *Proc. Interspeech*, 2005. 3.1, 3.3.4.1, 8.4

[2] A. Allauzen. Error Detection in Confusion Network. In *Proc. Interspeech*, 2007. 5.1.3

[3] T. Anastasakos, J. McDonough, R. Schwartz, and J. Makhoul. A compact model for speaker-adaptive training. In *Proc. ICASSP*, 1999. 2.9

[4] L.M. Arslan and J.H.L. Hansen. Selective Training for Hidden Markov Models with Applications to Speech Classification. *IEEE Trans. on Speech and Audio Processing*, 7 (1):46–54, 1999. 2.4.3, 5.2.1, 5.2.1

[5] S. Axelrod, R. Gopinath, and P. Olsen. Modeling with a Subspace Constraint on Inverse Covariance Matrices. In *Proc. ICSLP*, 2002. 2.3.1

[6] L.R. Bahl, P.F. Brown, P.V.d Souza, and R.L. Mercer. Maximum Mutual Information Estimation of Hidden Markov Model Parameters for Speech Recognition. In *Proc. ICASSP*, 1986. 2.4.2.1, 2.4.2.4, 5.2.2

[7] L.R. Bahl, P.V. de Souza, P.S. Gopalkrishnan, D. Nahamoo, and M.A. Picheny. Context Dependent Modelling of Phones in Continuous Speech using Decision Trees. In *Proc. DARPA Speech and Natural Language Processing Workshop*, pages 264–270, 1991. 2.8

[8] L.E. Baum and J.A. Eagon. An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology. *Bull. Amer. Math. Soc.*, 73:360–363, 1967. 1.1, 2.3, 2.4.1

[9] P. Beyerlein. Discriminative Model Combination. In *Proc. ASRU*, 1997. 3.3.1.6

[10] P. Beyerlein, X. Aubert, R. Haeb-Umach, M. Harris, D. Klakow, A. Wendemuth, S. Molau, M. Pitz, and A. Sixtus. The Philips/RWTH System for Transcription of Broadcast News. In *Proc. DARPA Broadcast News and transcription workshopa*, 1999. 3.3.1.6

[11] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. 2.2.2.3, 3.2.3

[12] H. Bourlard and S. Dupont. A new ASR approach based on independent processing and recombination of partial frequency bands. In *Proc. ICSLP*, 1996. 3.3.3.3

[13] L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001. 4.1.1

[14] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996. 4.1.1

[15] C. Breslin and M.J.F. Gales. Generating Complementary Systems for Speech Recognition. In *Proc. ICSLP*, 2006. (document)

[16] C. Breslin and M.J.F. Gales. Generating Complementary Systems using Directed Decision Trees. In *Proc. ICASSP*, 2007. (document), 6.1

[17] C. Breslin and M.J.F. Gales. Building Multiple Complementary Systems using Directed Decision Trees. In *Proc. Interspeech*, 2007. (document), 6.1

[18] A.D. Brown and G. E. Hinton. Products of Hidden Markov Models. Technical Report GCNU TR 2000-08, Gatsby Computational Neuroscience Unit, 2000. 3.3.3.2, 4.1.3

[19] T. Buckwalter. *Buckwalter Arabic morphological analyzer version 2.0*. In LDC2004L02, Linguistic Data Consortium, 2004. 8.4

[20] M.A. Carreira-Perpinan and G.E. Hinton. On contrastive divergence learning, 2005. URL `citeseer.ist.psu.edu/735735.html`. 4.1.3

[21] I.F. Chen and L.S. Lee. A New Framework for System Combination based on Integrated Hypothesis Space. In *Proc. ICSLP*, 2006. 3.3.1.1

[22] J.Y. Chen, P. Olsen, and J. Hershey. Word Confusability - Measuring Hidden Markov Model Similarity. In *Proc. Interspeech*, 2007. 12.2

[23] S.S. Chen and R.A. Gopinath. Gaussianization. In *Proc. Advances in NIPS*, 2000. 2.2.2.2, 2.2.2.2

[24] T. Cincarek, T. Toda, H. Saruwatari, and K. Shikano. Selective EM Training of Acoustic Models Based on Sufficient Statistics of Single Utterances. In *Proc. ASRU*, 2005. 2.4.3

[25] T. Cincarek, T. Toda, H. Saruwatari, and K. Shikano. Utterance Based Selective Training for the Automatic Creation of Task Dependent Acoustic Models. *IEICE Transactions on Info and Systems*, 89(3):962–969, 2006. 2.4.3

[26] D.A. Cohn, L. Atlas, and R.E. Ladner. Improving Generalization with Active Learning. *Machine Learning*, 15(2):201–221, 1992. 2.4.3

[27] S.B. Davis and P. Mermelstein. Comparison of Parametric Representations for Monosyllable Word Recognition in Continuously Spoken Sentences. *IEEE Trans. on Speech and Audio Processing*, 28(4):357–366, 1980. 2.2, 2.2.1

[28] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39:1–39, 1977. 2.4.1

[29] T.G. Dietterich. Ensemble Methods in Machine Learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000. 1.2, 4.1, 4.1.1

[30] T.G. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting and Randomization. *Machine Learning*, pages 1–22, 1999. 4.1.2

[31] C. Dimitrakakis and S. Bengio. Boosting HMMs with an Application to Speech Recognition. In *Proc. ICASSP*, 2004. 4.2.2

[32] V. Doumpiotis and W. Byrne. Pinched Lattice Minimum Bayes Risk Discriminative Training for Large Vocabulary Continuous Speech Recognition. In *Proc. ICSLP*, 2004. 2.4.2.2, 2.4.2.3, 2.4.2.3, 2.6.2, 2.7.3

[33] V. Doumpiotis, S. Tsakaliis, and W. Byrne. Discriminative Training for Segmental Minimum Bayes Risk Decoding. In *Proc. ICASSP*, 2003. 4.2.4

[34] G. Evermann and P.C. Woodland. Posterior Probability Decoding, Confidence Estimation and System Combination. In *Proceedings Speech Transcription Workshop*, 2000. 2.6.2.1, 2.7.3

[35] G. Evermann and P.C. Woodland. Design of Fast LVCSR Systems. In *Proc. ASRU*, 2003. 3.1

[36] J.G. Fiscus. A Post-Processing System to Yield Reduced Word Error Rates: Recogniser Output Voting Error Reduction (ROVER). In *Proc. ASRU*, 1997. 2.7.3, 3.3.1.2

[37] Y. Freund and R. Schapire. Experiments with a New Boosting Algorithm. In *Proceedings of the thirteenth International Conference on Machine Learning*, 1996. 3.2.2, 4.1.2, 4.2.2, 5.2, 5.2.1

[38] Y. Freund and R. Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, pages 55:119–139, 1997. 4.1.2

[39] S. Furui. Speaker independent isolated word recognition using dynamic features of speech spectrum. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 34:52–59, 1986. 2.2.1

[40] M.J.F. Gales. Transformation Streams and the HMM Error Model. *Computer Speech and Language*, 2002. 3.3.2

[41] M.J.F. Gales. The generation and the use of regression class trees for MLLR adaptation. Technical Report CUED/F-INFENG/TR263, Cambridge University, 1996. (via anonymous) `ftp://svr-www.eng.cam.ac.uk`. 2.9

[42] M.J.F. Gales. Maximum Likelihood Linear Transformations For HMM-Based Speech Recognition. *Computer Speech and Language*, 12, 1998. 2.9, 2.9

[43] M.J.F. Gales. Semi-tied Covariance Matrices for Hidden Markov Models. *IEEE Trans. on Speech and Audio Processing*, 7:272–281, 1999. 2.2.2.3, 2.3.1

[44] M.J.F. Gales and S.S. Airey. Product of Gaussians for Speech Recognition. Technical Report CUED/F-INFENG/TR458, University of Cambridge, 2003. Available via anonymous ftp from: svr-www.eng.cam.ac.uk. 3.3.3, 3.3.3.2, 3.3.3.3, 4.2.3.1

[45] M.J.F. Gales and S.S. Airey. Product of Gaussians for Speech Recognition. *Computer Speech and Language*, 2006. 3.2.3, 3.3.3.2, 4.2.3.1

[46] M.J.F. Gales and P.C. Woodland. Mean and Variance Adaptation Within the MLLR Framework. *Computer Speech and Language*, 10:249–264, 1996. 2.9

[47] M.J.F. Gales, B. Jia, K.C. Sim, P.C. Woodland, and K. Yu. Development of the CUHTK 2004 RT04F Mandarin Conversational Telephone Speech Transcription System. In *Proc. ICASSP*, 2005. 3.1

[48] M.J.F Gales, D.Y. Kim, P.C. Woodland, H.Y. Chan, D. Mrva, R. Sinha, and S.E. Tranter. Progress in the CU-HTK broadcast news transcription system. *IEEE Trans. on Speech and Audio Processing*, 14:1513–1525, September 2006. 1.1, 2.2.2.3, 3.1, 3.1, 3.3.4.1, 3.3.4.2, 8.1.3

[49] M.J.F. Gales, F. Diehl, C.K. Raut, M. Tomalin, P.C. Woodland, and K. Yu. Development of a Phonetic System for Large Vocabulary Arabic Speech Recognition. In *Proc. ASRU*, 2007. 2.4.2.2, 3.1, 3.1, 8.4, 9.6

[50] J. Gauvain and C.H. Lee. Maximum a Posteriori Estimation for Multivariate Gaussian Misture Observations of Markov Chains. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 2:291–299, 1994. 2.9

[51] J.L. Gauvain, L. Lamel, and G. Adda. The LIMSI Broadcast News Transcription System. *Speech Communication*, pages 37(1–2):89–108, 2002. 3.1

[52] Z. Ghahramani and M. Jordan. Factorial Hidden Markov Models. *Machine Learning*, pages 245–275, 1997. 3.3.2, 4.2.3.2

[53] M. Gibson and T. Hain. Hypothesis Spaces for Minimum Bayes Risk Training in Large Vocabulary Speech Recognition. In *Proc. Interspeech*, 2006. 2.4.2.2

[54] L. Gillick and S.J. Cox. Some statistical issues in the comparison of speech recognition algorithms. In *Proc. ICASSP*, 1989. 8.1.2

[55] L. Gillick, Y. Ito, and J. Young. A Probabilistic Approach to Confidence Estimation and Evaluation. In *Proc. ICASSP*, 1997. 5.1.3

[56] J. Glass, T.J. Hazen, S. Cyphers, I. Malioutov, H. Huynh, and R. Barzilay. Recent Progress in the MIT Spoken Lecture Processing Project. In *Proc. Interspeech*, 2007. 3.1

[57] V. Goel and W. Byrne. Minimum Bayes-risk automatic speech recognition. *Computer Speech and Language*, pages 115–135, 2000. 2.6.2, 2.6.2, 2.6.2

[58] V. Goel, W. Byrne, and S. Khudanpur. LVCSR rescoring with modified loss functions: a decision theoretic perspective. In *Proc. ICASSP*, 1998. 2.6.2

[59] V. Goel, S. Kumar, and W. Byrne. Segmental Minimum Bayes-Risk ASR Voting Strategies. In *Proc. ICSLP*, 2000. 3.3.1.2

[60] V. Goel, S. Kumar, and W. Byrne. Segmental Minimum Bayes-risk decoding for Automatic Speech Recognition. *IEEE Trans. on Speech and Audio Processing*, pages 12:234–249, 2004. 5.3.3

[61] R. Haeb-Umbach, X. Aubert, P. Beyerlein, D. Klakow, M. Ullrich, A. Wendemuth, and P. Wilcox. Acoustic modeling in the Philips Hub-4 continuous-speech recognition system. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, 1998. 2.3.1

[62] T. Hain, L. Burget, J. Dines, G. Garau, V. Wan, M. Karafiat, J. Vepa, and M. Lincoln. The AMI System for the Transcription of Speech in Meetings. In *Proc. ICASSP*, 2007. 2.2.2.3, 3.1, 3.1, 3.3.4.1, 3.3.4.2

[63] D. Hakkani-Tr, G. Riccardi, and A. Gorin. Active Learning for Automatic Speech Recognition. In *Proc. ICASSP*, 2002. 2.4.3, 5.2.1

[64] T.J. Hazen and I. Bazzi. A Comparison and Combination of Methods for OOV Word Detection and Word Confidence Scoring. In *Proc. ICASSP*, 2001. 5.1.3

[65] G. Heigold, W. Macherey, R. Schluter, and H. Ney. Minimum Exact Word Error Training. In *Proc. ASRU*, 2005. 2.4.2.2

[66] H. Hermansky. Perceptual Linear Predictive (PLP) Analysis of Speech. *Journal of the Acoustic Society of America*, 87(4):1738–1752, 1990. 2.2, 2.2.1

[67] D. Hillard and M. Ostendorf. Compensating for Word Posterior Estimation Bias in Confusion Networks. In *Proc. ICASSP*, 2006. 2.6.2.1

[68] D. Hillard, B. Hoffmeister, M. Ostendorf, R. Schluter, and H. Ney. iROVER: Improving system combination with classification. In *Proceedings HLT*, 2007. 3.3.1.2, 5.1.3, 11.3

[69] G. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14:1771–1800, 2002. 3.3.3.2, 4.1.3

[70] G. Hinton. Products of Experts. In *Proceeding of ICANN*, 1999. 3.2.3, 3.3.3

[71] B. Hoffmeister, T. Klein, R. Schluter, and H. Ney. Frame Based System Combination and a Comparison with Weighted ROVER and CNC. In *Proc. ICSLP*, 2006. 3.3.1.1, 3.3.1.3, 5.3.3

[72] B. Hoffmeister, D. Hillard, S. Hahn, R. Schluter, M. Ostendorf, and H. Ney. Cross-site and Intra-site ASR System Combination: Comparisons on Lattice and 1-best Methods. In *Proc. ICASSP*, 2007. 3.3.1.1

[73] B. Hoffmeister, C. Plahl, P. Fritz, G. Heigold, J. Loof, R. Schluter, and H. Ney. Development of the 2007 RWTH Mandarin LVCSR System. In *Proc. ASRU*, 2007. 3.1

[74] J. Huang, E. Marcheret, L. Visweswariah, V. Libal, and G. Potamianos. Detection, Diarization, and Transcription of Far-Field Lecture Speech. In *Proc. Interspeech*, 2007. 3.1, 3.1, 4.2.1

[75] X.D. Huang, A. Acero, and H.W. Hon. *Spoken Language Processing*. Prentice Hall, 2001. 2.6.1, 2.6.1

[76] M.Y. Hwang, W. Wang, X. Lei, J. Zheng, O. Cetin, and G. Peng. Advances in Mandarin Broadcast Speech Recognition. In *Proc. Interspeech*, 2007. 3.1, 3.1, 3.3.4.1

[77] H. Jiang. Confidence measures for speech recognition: A survey. *Speech Communication*, pages 45:455–470, 2005. 5.1, 5.2.1

[78] D. Jurafsky and J.H. Martin. *Speech and Language Processing*. Prentice Hall, 2000. 2.3.2, 2.5

[79] J. Kaiser, B. Horvat, and Z. Kacic. Overall Risk Criterion estimation of hidden Markov model parameters. *Speech Communication*, pages 38:383–398, 2002. 2.4.2.3

[80] T.M. Kamm. *Active Learning for Acoustic Speech Recognition Modeling*. PhD thesis, John Hopkins University, 2004. 1.2, 2.4.3, 5.2.1

[81] T.M. Kamm and G.G.L. Meyer. Automatic Selection of Transcribed Training Material. In *Proc. ASRU*, 2001. 2.4.3, 5.2.1

[82] T.M. Kamm and G.G.L. Meyer. Selective Sampling of Training Data for Speech Recognition. In *Proceedings HLT*, 2002. 2.4.3

[83] T.M. Kamm and G.G.L. Meyer. Word-selective Training for Speech Recognition. In *Proc. ASRU*, 2003. 2.4.3, 5.2.1

[84] A. Kannan, M. Ostendorf, and J.R. Rohlicek. Maximum likelihood clustering of Gaussians for speech recognition. *IEEE Trans. on Speech and Audio Processing*, 2:3:453–455, 1994. 2.8

[85] T. Kemp and T. Schaaf. Estimating Confidence Using Word Lattices. In *Proc. Eurospeech*, 1997. 5.1.2, 5.1.3

[86] D.Y. Kim, G. Evermann, T. Hain, D. Mrva, S.E. Tranter, L. Wang, and P.C. Woodland. Recent Advances in Broadcast News Transcription. In *Proc. ASRU*, 2003. 3.1, 8.2

[87] D. Klakow. Log-linear interpolation of language models. In *Proc. ICSLP*, 1998. 3.3.1.6

[88] S. Kullback and R.A. Leibler. On Information and Sufficiency. *Annals of Mathematical Statistics*, pages 22:79–86, 1951. 6.3

[89] N. Kumar. *Investigation of Silicon-Auditory Models and Generalization of Linear Discriminant Analysis for Improved Speech Recognition*. PhD thesis, John Hopkins University, 1997. 2.2.2.3

[90] L. Lamel. Lightly Supervised and Unsupervised Acoustic Model Training. *Computer, Speech and Language*, 2002. 2.4.3

[91] L. Lamel, A. Messaoudi, and J.L. Gauvain. Improved Acoustic Modeling for Transcribing Arabic Broadcast Data. In *Proc. Interspeech*, 2007. 8.4

[92] M.I. Layton and M.J.F. Gales. Maximum Margin Training of Generative Kernels. Technical Report CUED/F-INFENG/TR.484, Cambridge University Engineering Department, 2004. 4.2.4

[93] B. Lecouteux, G. Linares, Y. Esteve, and J. Mauclair. System Combination by Driven Decoding. In *Proc. ICASSP*, 2007. 3.3.1.1

[94] L. Lee and R.C. Rose. Speaker Normalisation using Efficient Frequency Warping Procedures. In *Proc. ICASSP*, 1996. 2.2.2

[95] C.J. Legetter and P.C. Woodland. Maximum Likelihood Linear Regression Speaker Adaptation of Contiuous Density HMMs. *Computer Speech and Language*, 1997. 2.9

[96] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707 – 710, 1966. 2.7.2

[97] R. Lippmann. Speech Recognition by Machines and Humans. *Speech Communication*, 22(1):1–16, 1997. 1.1

[98] X. Liu, M.J.F. Gales, K.C. Sim, and K. Yu. Investigation of Acoustic Modeling Techniques for LVCSR Systems. In *Proc. ICASSP*, 2005. 2.2.2.2, 3.1, 9.5

[99] A. Ljolje. The importance of cepstral parameter correlations in speech recognition. *Computer Speech and Language*, 8:223–232, 1994. 2.2.2

[100] J. Loof, C. Gollan, S. Hahn, G. Heigold, B. Hoffmeister, C. Plahl, D. Ryback, R. Schluter, and H. Ney. The RWTH 2007 TC-STAR Evaluation System for European English and Spanish. In *Proc. Interspeech*, 2007. 2.2.2.3, 3.1, 3.1, 3.3.4.1

[101] W. Macherey, L. HaferKamp, R. Schluter, and H. Ney. Investigations on Error Minimizing Training Criteria for Discriminative Training in Automatic Speech Recognition. In *Proc. Interspeech*, 2005. 2.4.2.2

[102] D.J.C. MacKay. Ensemble Learning for Hidden Markov Models. `http://www.inference.phy.cam.ac.uk/mackay/abstracts/ensemblePaper.html`, 1997. 4.1.3

[103] D.J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. Available from http://www.inference.phy.cam.ac.uk/mackay/itila/. 4.1.3, 4.2.3.2

[104] L. Mangu. *Finding Consensus in Speech Recognition*. PhD thesis, The John Hopkins University, 2000. 2.6.2.1

[105] L. Mangu, E. Brill, and A. Stolke. Finding Consensus Among Words: Lattice-Based Word Error Minimization. In *Proc. Eurospeech*, 1999. 2.4.2.3, 2.6.2, 2.6.2, 2.6.2.1, 5.1.1

[106] G.J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997. 3.3.3

[107] A. Messaoudi, J.L. Gauvain, and L. Lamel. Arabic Broadcast News Transcription using a One Million Word Vocalized Vocabulary. In *Proc. ICASSP*, 2006. 3.1

[108] C. Meyer and H. Schramm. Boosting HMM acoustic models in large vocabulary speech recognition. *Speech Communication*, pages 48:532–548, 2006. 4.2.2, 5.2.1

[109] H. Misra, H. Bourlard, and V. Tyagi. New Entropy Based Combination Rules in HMM/ANN Multi-Stream ASR. In *Proc. ICASSP*, 2003. 3.3.1.5

[110] P. Momayyez, J. Waterhouse, and R. Rose. Exploiting Complementary Aspects of Phonological Features in Automatic Speech Recognition. In *Proc. ASRU*, 2007. 3.1, 3.3.1.6

[111] P. Moreno, B. Logan, and B. Raj. A Boosting Approach for Confidence Scoring. In *Proc. Eurospeech*, 2001. 5.1.3

[112] N. Morgan and H. Bourlard. An Introduction to Hybrid HMM/Connectionist Approach. *IEEE Signal Processing Magazine*, 12:25–42, 1995. 3.3.1.5

[113] K. Na, B. Jeon, D. Chang, S. Chae, and S. Ann. Discriminative Training of Hidden Markov Models using Overall Risk Criterion and Reduced Gradient Method. In *Proc. Eurospeech*, 1995. 2.4.2.2

[114] L. Nguyen, S. Matsoukas, J. Davenport, F. Kibala, R. Schwartz, and J. Makhoul. Progress in transcription of Broadcast News using Byblos. *Computer, Speech and Language*, 2002. 3.1

[115] P. Niyogi, J.B. Pierrot, and O. Siohan. Multiple Classifiers by Constrained Minimisation. In *Proc. ICASSP*, 2000. 4.1.3

[116] H. Nock. *Techniques for modelling Phonological Processes in Automatic Speech Recognition*. PhD thesis, University of Cambridge, 2001. 3.3.3.3

[117] H.J. Nock and S.J. Young. Loosely coupled HMMs for ASR. In *Proc. ICSLP*, 2000. 3.3.3.3, 3.3.3.3

[118] Y. Normandin. *Hidden Markov Models, Maximum Mutual Information Estimation and the Speech Recognition Problem*. PhD thesis, McGill University, Montreal, 1991. 2.4.2, 2.4.2.1, 2.4.2.4

[119] J.J. Odell. *The Use of Context in Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge, 1995. 2.8, 2.8

[120] M.K. Omar and L. Mangu. An Evaluation of Lattice Scoring using a Smoothed Estimate of Word Accuracy. In *Proc. ICSLP*, 2007. 3.3.1.1

[121] S. Ortmanns, H. Ney, and A. Aubert. A Word Graph Algorithm for Large Vocabulary Continuous Speech Recognition. *Computer Speech and Language*, pages 43–72, 1997. 2.1, 2.7.1

[122] D. Povey. *Discriminative Training for Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge, 2003. 2.4.2.2, 2.4.2.3, 2.4.2.4, 2.4.2.4, 2.4.2.4, 5.2.2, 5.3.3

[123] D. Povey and B. Kingsbury. Evaluation of Proposed Modifications to MPE for Large Scale Discriminative Training. In *Proc. ICASSP*, 2005. 2.4.2.2

[124] D. Povey and P.C. Woodland. Minimum Phone Error and I-Smoothing for Improved Discriminative Training. In *Proc. ICASSP*, 2002. 2.4.2.5

[125] D. Povey, B. Kingsbury, L. Mangu, G. Saon, H. Soltau, and G. Zweig. fMPE: Discriminatively trained features for speech recognition. In *Proc. ICASSP*, 2005. 2.4.2.2, 12.2

[126] B. Ramabhadran, O. Siohan, L. Mangu, G. Zweig, M. Westphal, H. Schluz, and A. Soneiro. The IBM 2006 Speech Transcription System for European Parlimentary Speeches. In *TC-STAR Workshop on Speech-to-Speech Translation*, 2006. 3.1, 3.1, 3.3.4.1, 3.3.4.2, 4.2.1

[127] W.M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the Americal Statistical Association*, 66:846–850, 1971. 6.3

[128] G. Riccardi and D. Hakkini-Tr. Active and Unsupervised Learning for Automatic Speech Recognition. In *Proc. Eurospeech*, 2003. 2.4.3

[129] R. Riccardi and D. Hakkani-Tr. Active Learning: Theory and Applications to Automatic Speech Recognition. *IEEE Trans. on Speech and Audio Processing*, 13(4):504–511, 2005. 2.4.3

[130] R.C. Rose, B.H. Juang, and C.H. Lee. A Training Procedure for Verifying String Hypotheses in Continuous Speech Recognition. In *Proc. ICASSP*, 1995. 5.1.2, 5.1.2

[131] A. Sankar. Bayesian Model Combination (BAYCOM) for Improved recognition. In *Proc. ICASSP*, 2005. 3.3.1.4

[132] G. Saon, S. Dharanipragada, and D. Povey. Feature Space Gaussianization. In *Proc. ICASSP*, 2004. 2.2.2.2, 2.2.2.2

[133] L.K. Saul and M.I. Jordan. Mixed Memory Markov Models. *Machine Learning*, pages 75–87, 1999. 3.3.3.3

[134] R.E. Schapire. The Strength of Weak Learnability. *Machine Learning*, 5:197–227, 1990. 4.1.2

[135] H. Schwenk. Using boosting to improve a hybrid HMM/Neural-Network Speech Recogniser. In *Proc. ICASSP*, 1999. 4.2.2

[136] H. Schwenk and J.L. Gauvain. Combining Multiple Speech Recognisers using Voting and Language Model Information. In *Proc. ICSLP*, 2000. 3.3.1.2

[137] R. Sinha, M.J.F. Gales, D.Y. Kim, X.A. Liu, K.C. Sim, and P.C. Woodland. The CU-HTK Mandarin Broadcast News Transcription System. In *Proc. ICASSP*, 2006. 3.1, 3.1

[138] O. Siohan, B. Ramabhadran, and B. Kingsbury. Constructing Ensembles of ASR Systems using Randomized Decision Trees. In *Proc. ICASSP*, 2005. 4.2.1, 4.2.1

[139] N.D. Smith and M.J.F. Gales. Using SVMs and Discriminative Models for Speech Recognition. In *Proc. ICASSP*, 2002. 4.2.4

[140] H. Soltau, G. Saon, B. Kingsbury, J. Kuo, L. Mangu, D. Povey, and G. Zweig. The IBM 2006 GALE Arabic ASR System. In *Proc. ICASSP*, 2007. 2.2.2.3, 3.1

[141] A. Stolke, Y. Konig, and M. Weintraub. Explicit Word Error Minimization in N-Best List rescoring. In *Proc. Eurospeech*, 1997. 2.6.2, 2.6.2, 3.3.1.1

[142] S. Stuker, C. Fugen, S. Burger, and M. Wolfel. Cross-System Adaptation and Combination for Continuous Speech Recognition: The Influence of Phoneme Set and Acoustic Front-end. In *Proc. ICSLP*, 2006. 3.3.1.3, 3.3.4.2

[143] S. Stuker, C. Fugen, F. Kraft, and M. Wolfel. The ISL 2007 English Speech Transcription System for European Parlimentary Speeches. In *Proc. Interspeech*, 2007. 3.1

[144] T.P. Tan and L. Besacier. Acoustic Model Interpolation for Non-Native Speech Recognition. In *Proc. ICASSP*, 2007. 3.3.3

[145] M.J. Tomlinson, M.J. Russell, and N.M. Brooke. Integrating Audio and Visual Information to provide Highly Robust Speech Recognition. In *Proc. ICASSP*, 1996. 3.3.3.3

[146] M.J. Tomlinson, M.J. Russell, R.K. Moore, A.P. Buckland, and M.A. Fawley. Modelling Asynchrony in Speech using Elemental Single-signal Decomposition. In *Proc. ICASSP*, 1997. 3.3.3.3

[147] T. Utsuro, Y. Kodama, T. Watanave, H. Nishizaki, and S. Nakagawa. Confidence of Agreement among Multiple LVCSR Models and Model Combination by SVM. In *Proc. ICASSP*, 2003. 5.1.2, 5.1.3

[148] T. Utsuro, Y. Kodama, T. Watanabe, H. Nishizaki, and S. Nakagawa. An Empirical Study on Multiple LVCSR Model Combination by Machine Learning. In *Proceedings HLT*, 2004. 5.1.3

[149] V. Valtchev, J.J. Odell, P.C. Woodland, and S.J. Young. MMIE training of large vocabulary recognition systems. *Speech Communication*, 22:303–314, 1997. 2.4.2.1

[150] V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons Inc., 1998. 5.1.3

[151] A.P. Varga and R.K. Moore. Hidden Markov Model Decomposition of Speech and Noise. In *Proc. ICASSP*, 1990. 3.3.3.1, 3.3.3.3

[152] V. Venkataramani and W. Byrne. Lattice Segmentation and Support Vector machines for Large Vocabulary continuous Speech Recognition. In *Proc. ICASSP*, 2005. 3.3.4, 4.2.4, 4.2.4

[153] L. Wang, M.J.F. Gales, and P.C. Woodland. Unsupervised Training for Mandarin Broadcast News and Conversation Transcription. In *Proc. ICASSP*, 2007. 2.4.3

[154] M. Weintraub, F. Beaufays, Z. Rivlin, Y. Konig, and A. Stolcke. Neural-Network Based Measures of Confidence for Word Recognition. In *Proc. ICASSP*, 1997. 5.1.3

[155] F. Wessel and H. Ney. Unsupervised training of acoustic models for large vocabulary continuous speech recognition. *IEEE Trans. on Speech and Audio Processing*, 13(1): 23–31, 2005. 2.4.3

[156] F. Wessel, R. Schluter, and H. Ney. Using Posterior Word Probabilities for Improved Speech Recognition. In *Proc. ICASSP*, 2000. 2.6.2

[157] F. Wessel, R. Schluter, K. Macherey, and H. Ney. Confidence measures for Large Vocabulary Continuous Speech Recognition. *IEEE Trans. on Speech and Audio Processing*, pages 288–298, 2001. 2.7.1, 5.1.1

[158] F. Wessel, R. Schluter, and H. Ney. Explicit Word Error Minimization Using Word Hypothesis Posterior Probabilities. In *Proc. ICASSP*, 2001. 2.6.2, 5.1.1

[159] P.C. Woodland and D. Povey. Large Scale Discriminative Training of Hidden Markov Models for Speech Recognition. *Computer Speech and Language*, 16:25–47, 2002. 2.4.2.3, 2.4.2.4

[160] J. Xue and Y. Zhao. Improved Confusion Network Algorithm and Shortest Path Search from Word Lattice. In *Proc. ICASSP*, 2005. 2.6.2.1

[161] J. Xue and Y. Zhao. Random Forests-based Confidence Annotation using Novel Features from Confusion Network. In *Proc. ICASSP*, 2006. 5.1.3

[162] J. Xue and Y. Zhao. Random Forests of Phonetic Decision Trees for Acoustic Modeling in Conversational Speech Recognition. *IEEE Trans. on Audio, Speech, and Language Processing*, pages 16(3):519–528, 2008. 4.2.1, 4.2.1

[163] S.J. Young, G. Evermann, M.J.F. Gales, T. Hain, D. Kershaw, G. Moore, J.J. Odell, D. Ollason, D. Povey, V. Valtchev, and P.C. Woodland. *The HTK Book (for HTK Version 3.3)*. University of Cambridge, 2004. 2.2, 2.4.2.3, 2.6.1, 3.3.3.3

[164] S.R. Young. Detecting Misrecognitions and Out-of-Vocabulary Words. In *Proc. ICASSP*, 1994. 5.1.1

[165] R. Zhang and A.I. Rudnicky. Word Level Confidence Annotation using Combinations of Features. In *Proc. Eurospeech*, 2001. 5.1.3, 7.3.2, 11.3

[166] R. Zhang and A.I. Rudnicky. Comparative Study of Boosting and Non-boosting Training for Constructing Ensembles of Acoustic Models. In *Proc. Eurospeech*, 2003. 4.2.2, 5.2, 5.2.1

[167] R. Zhang and A.I. Rudnicky. A Frame Level Boosting Training Scheme for Acoustic Modelling. In *Proc. ICSLP*, 2004. 4.2.2, 5.2.1

[168] R. Zhang and A.I. Rudnicky. Investigations of Issues for Using Multiple Acoustic Models to Improve Continuous Speech Recognition. In *Proc. ICSLP*, 2006. 3.3.1.2, 4.2.2, 7.2

[169] A. Zolnay, D. Kocharov, R. Schluter, and H. Ney. Using Multiple Acoustic Feature Sets for Speech Recognition. *Speech Communication*, pages 514–525, 2007. 3.3.1.6

[170] G. Zweig and M. Padmanabhan. Boosting Gaussian Mixtures in an LVCSR System. In *Proc. ICASSP*, 2000. 4.2.2