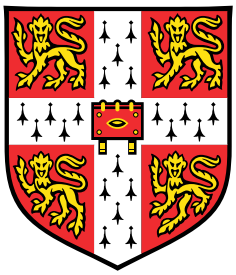
Joint Training Methods for Tandem and Hybrid Speech Recognition Systems using Deep Neural Networks



Chao Zhang

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Peterhouse July 2017

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and are my own work carried out at the Machine Intelligence Laboratory, Cambridge University Engineering Department. It includes nothing which is the outcome of any work done in collaboration except explicitly stated. Reference to the work of others is specifically indicated in the text where appropriate. Some of the materials have been presented at international conferences and workshops (Knill et al., 2013; Liu et al., 2015; Woodland et al., 2015; Yang et al., 2016; Zhang and Woodland, 2014, 2015a,b, 2016, 2017), and published in a software manual (Young et al., 2015). To my best knowledge, the length of this thesis including footnotes and appendices is approximately 57200 words.

Chao Zhang
July 2017

Acknowledgements

First, I would like to thank my supervisor Prof. Phil Woodland, for his guidance, support, and patience throughout my PhD study. I feel privileged to have the opportunity to work with Phil, and have learned a great deal from his experience and insight. I am grateful for the time Phil set aside for our meetings and replying my emails, even on weekends and holidays. In particular, Phil read and revised this thesis multiple times. I would also like to thank him for kindly securing funding support for me, and giving me the opportunity to get involved in the IARPA Babel, DARPA BOLT, and EPSRC NST research projects. On that note, I also thank the Cambridge Commonwealth, European & International Trust for their support during my first three years with a Cambridge International Scholarship.

I also want to thank Prof. Mark Gales, from whom I learned a lot through various projects, especially during the 2013 Babel project evluation period. Mark created the Babel scripts that I used almost everyday throughout the PhD study. He is also very generous in sharing wonderful ideas and providing valuable suggestions, for which I am truly grateful. Outstanding computing facilities are provided by the group and I must also thank Patrick Gosling and Anna Langley for maintaining the computer systems. There are many other people in the laboratory who have helped in various ways, and my thanks go to, in alphabetical order, Dr. Andrew Liu, Dr. Anton Ragni, Prof. Bill Byrne, Dr. Federico Flego, Dr. Haipeng Wang, Dr. Jingzhou Yang, Dr. Kate Knill, Dr. Matt Seigel, Dr. Matt Shannon, Dr. Penny Karanasou, Dr. Pierre Lanchantin, Dr. Rogier van Dalen, Dr. Shixiong Zhang, Dr. Tong Xiao, Dr. Xie Chen, Dr. Yanmin Qian, Dr. Yongqiang Wang, and Dr. Yu Wang. Meanwhile, my thanks also go to Prof. Steve Renals and Prof. Thomas Hain for co-leading the NST project. For three years I had a great time and I was fortunate to meet and collaborate with everyone in the NST project. Moreover, since I initially learned speech recognition using the HTK toolkit and all of my PhD work was based on HTK, I want to thank Prof. Steve Young for his work in initially creating such a great toolkit.

Finally, the biggest thanks go to my family to whom I always owe everything! My parents have always been very supportive and helpful during my school career, and I

know they will always be standing behind me. My dear wife, Linlin, has accompanied me and has been very patient with me. She undertook all the housework to save me time while writing this thesis, and helped me proof-read it. And to my baby daughter Zitong, you are the biggest motivation for me to complete my thesis!

Summary

Hidden Markov models (HMMs) have been the mainstream acoustic modelling approach for state-of-the-art automatic speech recognition (ASR) systems over the past few decades. Recently, due to the rapid development of deep learning technologies, deep neural networks (DNNs) have become an essential part of nearly all kinds of ASR approaches. Among HMM-based ASR approaches, DNNs are most commonly used to extract features (tandem system configuration) or to directly produce HMM output probabilities (hybrid system configuration).

Although DNN tandem and hybrid systems have been shown to have superior performance to traditional ASR systems without any DNN models, there are still issues with such systems. First, some of the DNN settings, such as the choice of the context-dependent (CD) output targets set and hidden activation functions, are usually determined independently from the DNN training process. Second, different ASR modules are separately optimised based on different criteria following a greedy build strategy. For instance, for tandem systems, the features are often extracted by a DNN trained to classify individual speech frames while acoustic models are built upon such features according to a sequence level criterion. These issues mean that the best performance is not theoretically guaranteed.

This thesis focuses on alleviating both issues using joint training methods. In DNN acoustic model joint training, the decision tree HMM state tying approach is extended to cluster DNN-HMM states. Based on this method, an alternative CD-DNN training procedure without relying on any additional system is proposed, which can produce DNN acoustic models comparable in word error rate (WER) with those trained by the conventional procedure. Meanwhile, the most common hidden activation functions, the sigmoid and rectified linear unit (ReLU), are parameterised to enable automatic learning of function forms. Experiments using conversational telephone speech (CTS) Mandarin data result in an average of 3.4% and 2.2% relative character error rate (CER) reduction with sigmoid and ReLU parameterisations. Such parameterised functions can also be applied to speaker adaptation tasks.

At the ASR system level, DNN acoustic model and corresponding speaker dependent (SD) input feature transforms are jointly learned through minimum phone error (MPE) training as an example of hybrid system joint training, which outperforms the conventional hybrid system speaker adaptive training (SAT) method. MPE based

speaker independent (SI) tandem system joint training is also studied. Experiments on multi-genre broadcast (MGB) English data show that this method gives a reduction in tandem system WER of 11.8% (relative), and the resulting tandem systems are comparable to MPE hybrid systems in both WER and the number of parameters. In addition, all approaches in this thesis have been implemented using the hidden Markov model toolkit (HTK) and the related source code has been or will be made publicly available with either recent or future HTK releases, to increase the reproducibility of the work presented in this thesis.

Contents

List of Figures

List of Tables

Nomenclature

Roman Symbols

$\mathbf{A}$	The weight matrix of a linear transform
a	The input activation value to an artificial neuron
a_{ij}	An HMM transition probability from HMM state i to j
$\mathbf{b}$	A bias vector
c_e	The context shift set associated with a feature element e
c_{im}	The weight value associated with a Gaussian mixture component m in HMM state i
C_k	The kth class among a set of classes
D	The dimension number of a vector or some constant value
$\mathbb{D}$	The Kullback-Leibler divergence function.
$\mathbb{E}$	The mathematic expectation function.
$\mathcal{F}$	An objective function
I_l	The number of input artificial neurons in layer l
J_l	The number of output artificial neurons in layer l
M	The number of Gaussian mixture components
$\mathcal{N}$	A Gaussian distribution
N_{min}	The minimum number of epochs to train by NewBob and NewBob$^+$ schedulers
$\mathbf{O}$	A speech observation sequence

$\mathbf{o}(t)$ A speech observation vector at time t

$\mathbf{Q}$ An HMM state sequence

S The number of HMM states

T The total number of frames in the observation sequence

t A time step

V A vocabulary

$\mathbf{W}$ A text string, usually words

$\mathbf{w}_k$ The inbound weight vector associated with artificial neuron k

$\mathbf{W}^{(l)}$ The weight matrix in layer l

$\mathbf{x}^{(l)}(t)$ The input vector of layer l at time t

$x_i^{(l)}(t)$ The input value to the ith input artificial neuron of layer l at time t

$\mathbf{y}^{(l)}(t)$ The output vector of layer l at time t

$y_j^{(l)}(t)$ The output value of the jth output artificial neuron of layer l at time t

$\mathbf{z}(t)$ A tandem feature vector at time t

$\alpha_i^{(l)}$ The first activation function parameter of artificial neuron i in layer l

$\alpha_i(t)$ The forward variable of being in HMM state i at time t

$\beta_i^{(l)}$ The second activation function parameter or the first batch normalisation parameter associated with artificial neuron i in layer l

$\beta_i(t)$ The backward variable given HMM state i at time t

δ The update value of a parameter in SGD.

$\delta_{kk'}$ The Kronecker delta function that returns 1 if $k = k^{'}$, and 0 otherwise.

η A learning rate

$\Delta\mathcal{F}_{\text{ramp}}$ The ramp state threshold of NewBob and NewBob$^+$ schedulers

$\Delta\mathcal{F}_{\text{stop}}$ The stopping threshold of NewBob and NewBob$^+$ schedulers

$\gamma_i^{(l)}$ The third activation function parameter or the second batch normalisation parameter associated with artificial neuron i in layer l

$\gamma_i(t)$ The posterior probability of being in an HMM state i at time t

$\gamma_{im}(t)$ The posterior probability of being in a Gaussian mixture component m of an HMM state i at time t

κ The inverse language model grammar scaling factor

Λ A set of HMMs

λ An HMM

μ A mean value

π The ratio of a circle's circumference to its diameter

$\pi_{im}(t)$ The posterior probability of a vector being generated by a Gaussian component m given in HMM state i at time t

ρ The momentum coefficient

Σ A covariance matrix of a multi-variate Gaussian distribution

σ A standard deviation value

Θ A set of model parameters

θ A model parameter

υ The gradient/update value clipping threshold

ε The coefficient of $L2$ regularisation or weight decay

Acronyms / Abbreviations

0-MN Zero mean normalisation

1-VN Unit variance normalisation

AI Artificial intelligence

ANN Artificial neural network

APS Arcs per second

ASR Automatic speech recognition

BLAS Basic linear algebra subprograms

BOLT Broad operational language technology

BPTT Backpropagation through time

BRNN Bidirectional recurrent neural network

BW Baum-Welch

CD Context-dependent

CDF Cumulative distribution function

CE Cross entropy

CI Context-independent

CLDNN Convolutional long-short term memory deep neural network

CMLLR Constrained maximum likelihood linear regression

CNN Convolutional neural network

CTC Connectionist temporal classification

CTS Conversational telephony speech

CUED Cambridge University Engineering Department

DAG Directed acyclic graph

DCG Directed cyclic graph

DCT Discrete cosine transform

DNN Deep neural network

DP Dynamic programming

EBP Error backpropagation

EBW Extended Baum-Welch

EM Expectation-maximisation

FBANK Log-Mel filterbank amplitudes

FB Forward-backward algorithm

FNN Feedforward neural network

FT Fine-tuning

GD Gradient descent

GEMM General matrix multiplication

GMM Gaussian mixture model

GPU Graphics processing unit

GRU Gated recurrent unit

HLDA Heteroscedastic linear discriminant analysis

HMM Hidden Markov model

HTK The hidden Markov model toolkit

KLD Kullback–Leibler divergence

KLT Karhunen-Loève transform

LHUC Learning hidden unit contribution

LM Language model

LSTM Long-short term memory

LVCSR Large vocabulary continuous speech recognition

MAP Maximum a posteriori

MBR Minimum Bayesian risk

MDNN Gaussian mixture density neural network

MFCC Mel-scale frequency cepstral coefficients

MGB Multi-genre broadcast

MLLR Maximum likelihood linear regression

ML Maximum likelihood

MLP Multi-layer perceptron

MMI Maximum mutual information

MMSE Minimum mean squared error

MPE Minimum phone error

OOV Out of vocabulary

PLP Perceptual linear prediction

p-ReLU Parameterised rectified linear unit

p-sigmoid Parameterised sigmoid unit

p-SoftPlus Parameterised soft rectified linear unit

PT Pretraining

RBM Restricted Boltzmann machine

ReLU Rectified linear unit

RNN Recurrent neural network

SAT Speaker adaptive training

SD Speaker dependent

SGD Stochastic gradient descent

SI Speaker independent

STC Semi-tied covariance matrices

TDNN Time delay neural network

WER Word error rate

WFST Weighted finite state transducer

WSJ Wall Street Journal

Chapter 1

Introduction

Talking to machines has been a dream of human beings for a long time. ASR technology allows computers to acquire the text embedded in human speech, and thus is essential in interacting with machines by speech. Nowadays, ASR integration is considered to be important for many software applications, especially when deployed on devices without a proper keyboard. Personal intelligent assistants, such as Apple Siri, rely on speech input and have changed many people's way of accessing information and acquiring services compared to a decade ago.

Nevertheless, ASR suffers from many known limitations and is by no means perfect. The first issue lies in the statistical modelling approach, which serves as the cornerstone of state-of-the-art ASR systems. The capability of accurate modelling using the seen (available for training) speech and the generalisation ability to the unseen data are crucial in ASR performance, but there is no perfect model for the purpose. Secondly, there are usually *acoustic and phonetic mismatches* between the seen and unseen data, caused by different speaker characteristics, pronunciations, accents, background noises and channels *etc.* Lastly and more fundamentally, language models (LMs) often suffer from *out of task* and *out of grammar* issues, and it is even impossible for ASR to recognise out of vocabulary (OOV) words. These issues often occur in real-world application, and are referred to as the *linguistic mismatches*.

Recently, there has been a breakthrough in ASR performance, which can notably improve the user experience, especially for applications like voice search and even CTS transcription, where ASR performance is quickly approaching human levels (Saon et al., 2016; Xiong et al., 2016). This leap forward is mainly due to the development of deep learning that can significantly increase the power of statistical models for classification and regression (Goodfellow et al., 2016; Hinton et al., 2012; Hinton and Salakhutdinov, 2006). Besides better accuracy and generalisation abilities, deep learning is also helpful

in solving the acoustic and phonetic mismatch problems by improving model accuracies. Alternatively, this issue can also be solved by collecting a diverse set of training data to cover more acoustic and phonetic scenarios. It is worth noting that the model power is also important in learning more statistical patterns from diversified data.

It is known that improving deep learning models can further improve ASR performance (Morgan, 2012), for example, using deeper and larger models to transform specific speech fragments into desired distributions, or using longer span temporal information to distinguish between confusable phonetic units (Bi et al., 2016; Graves and Schmidhuber, 2005; Sainath et al., 2015a; Sercu et al., 2016). An alternative way to improve statistical ASR modelling is to have the deep learning models configured and trained in a more suitable way, for instance, to use a more suitable set of phonetic units, or to learn the model parameters based on a more appropriate criterion (Kingsbury, 2009; Kingsbury et al., 2012; McDermott et al., 2014; Povey et al., 2016; Su et al., 2013; Veselý et al., 2013).

In this thesis, in order to help improve state-of-the-art system performance, the focus is to refine the integration of deep learning models into the *noisy source-channel model* based ASR framework (Jelinek, 1998). This ASR framework is also termed as statistical ASR or HMM based ASR approaches. The major advantage of this methodology is to allow the reuse of many existing methods developed in the past in the new deep learning based framework (Gales, 1998, 1999; Gauvain and Lee, 1994; Juang and Katagiri, 1992; Leggetter and Woodland, 1995; Povey and Woodland, 2002; Woodland et al., 1995; Woodland and Povey, 2002; Young et al., 1994). Moreover, it should be rather straightforward to apply the proposed approaches with alternative deep learning models, which allows them to have broader application.

1.1 Thesis Outline

In this thesis, the object of study to improve the statistical ASR approach is a DNN, and the proposed methods can be applied to alternative artificial neural network (ANN) models as well (Chung et al., 2014; Hochreiter and Schmidhuber, 1997; LeCun et al., 1998a; Rumelhart et al., 1986; Sainath et al., 2015a; Waibel et al., 1989). In this ASR approach, a DNN is often used for either feature extraction or acoustic modelling purposes, leading to the tandem or hybrid configurations for DNN integration. Instead of merely replacing the corresponding shallow models with cross entropy (CE) trained DNNs, the choice of the DNN targets, training criterion, adaptation scheme, along

with DNN feature extraction and acoustic model joint training are investigated to make DNNs more appropriately configured and trained for modelling human speech.

In Chapter 2, the background of the statistical ASR approach is reviewed. Chapter 3 introduces and discusses the basics of deep learning. Baseline tandem and hybrid system configurations are also investigated in this chapter. In Chapter 4, a DNN acoustic model building procedure termed standalone training, that does not rely on any pre-existing system, is proposed. A context-independent (CI) initialisation method is also proposed as an effective regularisation method. Chapter 5 proposes the learning of sigmoid and ReLU hidden activation function parameters for SI DNN acoustic modelling. The effective parameters found in SI training are also used for speaker adaptation. Chapter 6 first studies hybrid system MPE training, which is then used later in the joint learning of DNN acoustic model and SD input transforms. SI tandem system joint training is also investigated in the chapter, which is based on training the DNN feature extraction and Gaussian mixture model (GMM) acoustic models together according to the MPE criterion. Finally, conclusions and suggestions to the future work are presented in Chapter 7.

1.2 Proposed Methods

Except for the baseline system configurations, the proposed methods in this thesis can be categorised into two classes: DNN acoustic model joint training and ASR system level joint training. DNN acoustic model joint training includes standalone training and parameterised sigmoid and ReLU function learning. CI initialisation and parameterised activation function adaptation methods are also developed based on these methods. ASR system level joint training is studied based on hybrid and tandem configurations. In the hybrid configuration, DNN-HMM discriminative sequence training is investigated and the SD feature transforms are jointly trained with the SI DNN acoustic model. The tandem system joint training is applied as the SI MPE training of the Gaussian mixture density neural network (MDNN) system.

1.2.1 Optimising baseline DNN systems

In Chapter 3, DNN based baseline tandem and hybrid system configurations are investigated. For tandem configurations, the DNN features are derived from a bottleneck (BN) hidden layer, which can generate features that are not only discriminative, but are also compact in size. This chapter starts with an investigation of the appropriate

DNN configuration for feature extraction, such as output targets, silence modelling, and the BN layer size and position, based on the traditional Cambridge University Engineering Department (CUED) tandem system setup (Park et al., 2011). Various silence modelling methods and input feature transforms are also compared with the hybrid system configuration. A joint decoding method is also investigated that is used as a tandem and hybrid system combination approach in this thesis.

1.2.2 Standalone DNN acoustic model training

DNN-HMM with CD phonetic units is a widely used type of acoustic model in ASR. However, the CE training of this method requires frame to label alignments produced by an existing system. The standalone training approach proposed in Chapter 4 trains a high performance CD-DNN-HMM system without relying on another system. This method first integrates realignments into the discriminative pretraining (PT) procedure to train a CI DNN model. Next, DNN based decision tree tying is proposed, which is a modification of the standard GMM-HMM based decision state tying approach (Young et al., 1994). The state output distributions used in maximum likelihood (ML) clustering are single Gaussians explicitly estimated as the approximations of equivalent terms to the CD-DNN output distribution. The resulting initial CD-DNN model is then optimised by the normal fine-tuning (FT) method. Experiments have shown that standalone training gives comparable WERs to the conventional procedure.

The idea of initialising the CD-DNN model with parameters trained for classifying CI states can be applied to the conventional CD-DNN training procedure as well. Compared to generative PT with restricted Boltzmann machines (RBMs) (Hinton et al., 2006), CI initialisation can improve ASR performance as it is discriminative, while keeping a unified training algorithm, error backpropagation (EBP), for DNN construction. Compared to the standard CD discriminative PT (Hinton et al., 2012), besides saving the time cost, CI initialisation also regularises CD training to prevent the lower layers from being over-specific to the CD state targets. Experiments have shown that the CI initialisation method outperforms CD discriminative PT on 15h and 66h data sets, and its regularisation effect is complementary to the *weight decay* method.

1.2.3 Parameterised sigmoid and ReLU functions

The choice of hidden activation functions is an important issue in deep learning. Common choices include sigmoid and ReLU functions (Glorot et al., 2011; Nair and

Hinton, 2010; Rumelhart et al., 1986). In Chapter 5, the use of *logistic functions* is proposed as the parameterised form of sigmoid functions, denoted as the p-Sigmoid. Its adaptive parameters can be learned together with standard DNN parameters in SI training. Since logistic functions can be used for the piecewise approximation to many other smoothed functions by varying their parameters, the p-Sigmoid allows each artificial neuron to learn the most appropriate activation function form. Similarly, the ReLU function is also parameterised to associate independent adaptive scaling factors to the positive and negative parts, which enables the discrimination ability of outputs of each artificial neuron to be controlled in learning.

In the first part of this chapter, experiments investigate different activation function parameters and training schemes. Moreover, as many of these parameters can be viewed as linear scales and biases, they can be integrated into the standard SI DNN weights and biases without resulting in any extra parameters.

In addition to SI model training, the most useful p-Sigmoid and p-ReLU configurations, using output value scaling factors, are also applied to speaker adaptation. Since these scaling factors can have a direct impact on the output of the layer, training these SD parameters is very efficient using limited speaker specific data for adaptation. Furthermore, a layer-wise adaptation scheme is also proposed that is found to stabilise and improve the adaptation process. Unsupervised test time adaptation experiments have been conducted with hybrid, tandem, and stacked hybrid systems. The results show that the proposed method not only outperforms but also is complementary to other commonly used DNN adaptation methods.

1.2.4 Hybrid system discriminative joint SAT

Traditionally, a DNN acoustic model is often trained to minimise the classification criterion like CE and minimum mean squared error (MMSE). Such criteria, however, only use the information within each input vector (usually formed by a small set of speech frames) and ignore the temporal structure embedded in the speech sequence. On the contrary, ASR is normally evaluated based upon a sequence of segments, which clearly generates a *criterion mismatch* and can cause the DNN parameters to converge to a local optimum that is not suitable for speech recognition. Chapter 6 first investigates DNN-HMM discriminative sequence training, such as the MPE and maximum mutual information (MMI) training methods.

Next, DNN-HMM MPE training is applied to optimise the SD input transforms with the DNN-HMM acoustic models on the training set, which is a discriminative joint SAT method. When estimating the test-time speaker SD transforms, MPE is

then smoothed by the frame level CE criterion and the SI DNN parameters are kept frozen. This can alleviate over-fitting to the supervision hypotheses in unsupervised adaptation.

1.2.5 SI tandem system joint training

In the latter part of Chapter 6, tandem system joint training studies training the BN DNN and GMMs together based on the MPE criterion. Traditional extended Baum-Welch (EBW) algorithm based GMM-HMM MPE training is first adapted to use the stochastic gradient descent (SGD) optimisation method, and to use the state-of-the-art general purpose graphics processing unit (GPU) devices. Unlike the EBW case, MPE training does not result in over-fitting with appropriate SGD configurations. However, techniques developed for EBW based MPE training, such as I-smoothing, dynamic MMI prior, and percentile based variance floor (Povey, 2003; Young et al., 2015), are still useful in improving SGD training performance. The MPE joint training is applied to the ML trained tandem system. The BN DNN and GMMs are seen as a single MDNN model, and the joint training becomes MDNN discriminative sequence training. A set of methods, such as linear to ReLU activation function conversion, relative update value clipping, amplified GMM learning, as well as various parameter update schemes are proposed and investigated to improve joint training stability and performance.

1.2.6 A generic ANN extension to HTK

The aforementioned joint training approaches are based on generic ANN models, whose related definitions, algorithms, and formulas are proposed in Chapter 3. Generic ANN support is implemented in the HTK toolkit, which reduces the difficulties in reproducing the experimental results and reusing the methods. Based on a flexible ANN structure and a vector/matrix based parameter sharing approach, commonly used models, such as DNNs, convolutional neural networks (CNNs), recurrent neural networks (RNNs), along with their variants and combinations are supported. HTK also supports both frame and sequence level ANN training methods, based on a powerful data cache that allows the data to be accessed randomly and efficiently in several different ways.

Another feature of the HTK ANN extension is the seamless integration with the pre-existing ASR approaches in HTK, such as decision tree state clustering, SI and SD transform estimation, and discriminative sequence training. The tandem system

implementation is an important instance, where the GMMs can directly use the ANN output values transformed in different ways. GMM training can be conducted using either EBW or SGD, and implicit data structure conversion happens automatically when using a different optimisation method. Details of the HTK ANN functions and their implementation can be found in Appendix B.

1.3 Summary of Contributions

The main contributions of this thesis are the following:

- Proposed a discriminative PT method with realignment, a DNN based decision tree tying method, and a CI initialisation method. These methods constituted a CD-DNN standalone training approach that was comparable in performance to traditional CD-DNN training.
- Proposed parameterised sigmoid and ReLU functions and used them in both SI and SD DNN acoustic modelling. For SI modelling, many such parameters could be integrated into the conventional DNN parameters. A layer-wise adaptation procedure analogous to discriminative PT and FT was also proposed.
- Studied DNN-HMM sequence training, and applied it to hybrid system discriminative joint SAT. The F-smoothing was found useful in alleviating the over-fitting issue in unsupervised discriminative adaptation.
- Proposed tandem system joint training based on MDNN-HMM discriminative sequence training. The I-smoothing, dynamic MMI prior, and percentile based variance floor techniques were adapted to the SGD framework. In order to improve training stability and performance, linear to ReLU activation function conversion, relative update value clipping, and amplified GMM learning were proposed.
- Implemented a generic ANN model extension in HTK. The new ANN functions were seamlessly integrated with the traditional HTK GMM-HMM functions.

Chapter 2

Automatic Speech Recognition

2.1 Automatic Speech Recognition System Structure

ASR is defined as an automatic process of translating a speech waveform into its corresponding transcription by computer (Rabiner and Juang, 1993). Usually the waveform is first converted into a sequence of speech *observation* vectors $\mathbf{O}$,

$$\mathbf{O} = \mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(T), \tag{2.1}$$

through a feature extraction process, where $\mathbf{o}(t)$ is the feature vector at time t derived from the corresponding speech frame (an audio segment), and T is the total number of frames in the utterance. The required transcription is presented as a text string $\mathbf{W}$ of a variable length L,

$$\mathbf{W} = w_1, w_2, \ldots, w_L. \tag{2.2}$$

w_l is the lth unit of the string given $w_l \in V$, and V is the *vocabulary* that contains all units that are possible to be recognised. Therefore, the function of ASR can be formally presented as

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{O}), \tag{2.3}$$

where $\mathbf{W}^*$ is *a posteriori* the most likely *hypothesis* of the text and is considered to be the 1-best ASR output. This is actually a *noisy source-channel model* (Jelinek, 1998; MacKay, 2003; Shannon and Weaver, 1949) where the speech production, propagation,

reception, quantification and waveform conversion processes are considered as a noisy channel, $\mathbf{O}$ is the observed channel output, and $\mathbf{W}$ is the intended source. Therefore, Eqn. (2.3) decodes the source text from the channel output and is referred to as the *maximum a posteriori* (MAP) *decoding rule*.

Applying *Bayes' rule*, we can get

$$P(\mathbf{W}|\mathbf{O}) \propto P(\mathbf{O}|\mathbf{W})P(\mathbf{W}), \tag{2.4}$$

since $P(\mathbf{O})$ is a constant given a certain utterance and is therefore independent from the hypotheses. $P(\mathbf{O}|\mathbf{W})$ is the likelihood of translating $\mathbf{W}$ to $\mathbf{O}$ through the channel; $P(\mathbf{W})$ is the probability distribution of the source. In ASR, $P(\mathbf{O}|\mathbf{W})$ and $P(\mathbf{W})$ are modelled by the acoustic model and the language model (LM), both of which are usually statistical models. The above-mentioned ASR approach is also referred to as the *statistical approach* (Jelinek, 1998) and is illustrated in Figure 2.1, with the waveform conversion presented as the feature extraction module.

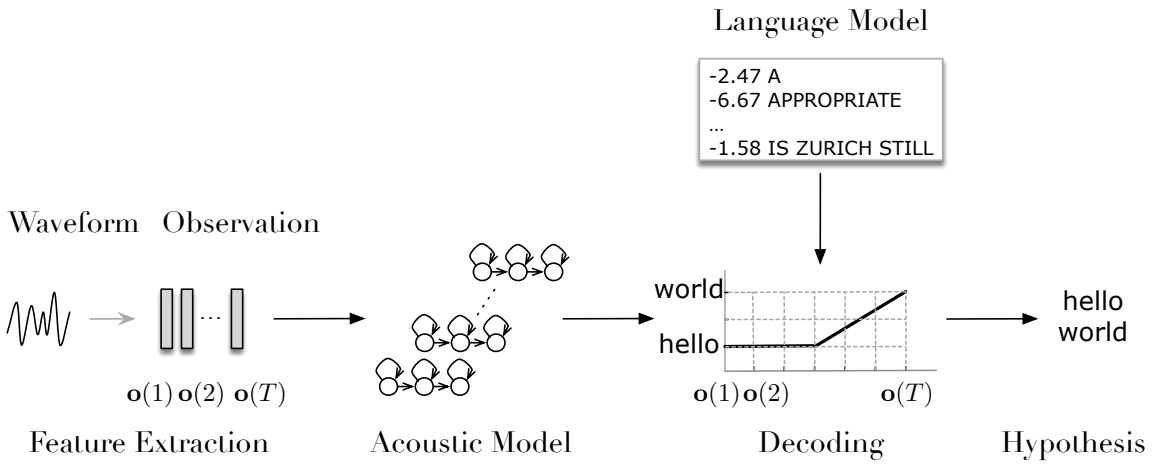


Figure 2.1 A statistical ASR system.

In the rest of this chapter, each ASR module will be introduced. However, since the exact ASR module setups differ significantly according to the recognition objective, it is useful to define the hypothesis spaces and their corresponding recognition tasks first. In $\mathbf{W}$, a unit w_l can be a phoneme, a syllable, a word, a phrase, or even a sentence, depending on the desired linguistic representation of the text. Assume that the duration of each frame is short enough that it cannot contain multiple linguistic units (which is generally true), then the upper bound of the hypothesis space is constrained by the

size of vocabulary, i.e.,

$$|V|^L \leqslant |V|^T. \tag{2.5}$$

If V is small, containing only phonemes, syllables, or some selected set of names and commands, the decoding cost for searching the hypothesis space is low. The model complexity is also reduced as not a full set of linguistic units needs to be covered by the acoustic model and LM. If V is large, for example, it contains tens or hundreds of thousands of words, both the decoding cost and model complexity would be much higher and that makes the problem much more difficult. In this thesis, all methods were evaluated by recognising a large vocabulary of words. In addition, rather than isolated words, the data used in this thesis contain either continuous read speech, or *spontaneous* speech with a natural speaking style from telephone conversations or broadcast shows. Such speech data contain more variations in pronunciation and are harder to recognise. The task defined above is called large vocabulary continuous speech recognition (LVCSR) and is one of the most generic and challenging problems in the ASR research field.

Besides the statistical approach, *expert systems* (Haton, 1985; Zue and Lamel, 1986), *template matching* (Berndt and Clifford, 1994; Gemmeke et al., 2011; Myers et al., 2003; Wachter et al., 2007), *detection based methods* (Lee, 2004), and *end-to-end methods* (Bahdanau et al., 2016; Graves and Jaitly, 2014) *etc.*, have been investigated as alternative approaches. In recent decades, mainly due to its superior performance and theoretical simplicity, statistical approaches have been dominant in practical systems. Therefore, this thesis is based on the statistical approach and refers to it as ASR without any further distinction.

2.2 Feature Extraction

The feature extraction module parameterises a raw waveform into a feature sequence $\mathbf{O}$, which is an adequate representation of the speech. Here a brief review of two types of speech features, Mel-scale frequency cepstral coefficients (MFCCs) (Davis and Mermelstein, 1980) and perceptual linear prediction (PLP) (Hermansky, 1990), is provided as an example of the process.

It is known that the frequencies across the audio spectrum are resolved non-linearly by the human ear, which can be used to derive better features, such as MFCCs, to improve speech recognition performance. Initially, the waveform is segmented into short

time frames (25ms in length with a 15ms overlap throughout this thesis), over which the waveforms are assumed to be stationary. Based on the stationary assumption, the *short-term Fourier transform* (Oppenheim and Schafer, 1975) is applied within each segment and the resulting magnitude coefficients are weighted and summed over a set of triangular filters. MFCCs use filters equally spaced along the Mel-scale to acquire the desired non-linear resolution (Davis and Mermelstein, 1980),

$$\text{Mel}(f) = 2595 \log_{10}(1 + \frac{f}{700}). \tag{2.6}$$

The log amplitudes of the Mel-scale filterbanks are sometimes directly used as features denoted in HTK as FBANK. Otherwise, the log spectrum can be further linearly decorrelated by the discrete cosine transform (DCT) (Oppenheim and Schafer, 1975) to produce the *cepstral* vectors. For simplicity, many details, such as signal pre-emphasis, Hamming window framing, lower/upper frequency cut-off, and cepstral coefficient liftering *etc.*, are omitted from the above procedure (Young et al., 2015).

PLP features are often used as an alternative to MFCCs, which mainly approximate three main perceptual aspects: the non-linear frequency resolution curves, the equal-loudness curve, and the intensity-loudness power-law relation (Hermansky, 1990). In this thesis, the Mel-scale is also taken as the non-linear spectral scale for PLP as well (Woodland, 2002; Woodland et al., 1997). The Mel filterbank amplitudes are weighted by the equal-loudness emphasis to simulate the sensitivity of hearing, and then compressed by taking the cube root to convert to the auditory perceived intensity of strength. The resulting auditory spectrum is used for *linear prediction analysis* (Atal and Hanauer, 1971), and the cepstral coefficients are used as the PLP features.

Besides choosing the appropriate feature type, adding time derivatives to the basic static feature vector can also enhance the ASR performance (Furui, 1986). Such time derivatives are computed as the linear regression coefficients within a window (Young et al., 2015),

$$\mathbf{d}^{(1)}(t) = \frac{\sum_{\theta=1}^{2} (\mathbf{o}(t+\theta) - \mathbf{o}(t-\theta)) \cdot \theta}{2 \sum_{\theta=1}^{2} \theta^2}. \tag{2.7}$$

$\mathbf{d}^{(1)}(t)$ are called the first order regression coefficients of $\mathbf{o}(t)$. The second and third order coefficients $\mathbf{d}^{(2)}(t)$ and $\mathbf{d}^{(3)}(t)$ can be computed by replacing $\mathbf{o}(t)$ with $\mathbf{d}^{(1)}(t)$ and $\mathbf{d}^{(2)}(t)$ in Eqn. (2.7). Following the HTK convention (Young et al., 2015), the first, second, and third order regression coefficients are indicated by appending `_D`, `_A`, and `_T` qualifiers to the feature type.

In addition, a benefit of cepstral features like MFCCs and PLP, is the simplicity of compensation of different recording channels. Assume it is a *linear time-invariant system*, since the effect of adding a transmission channel is equivalent to that of multiplying the audio spectrum by the channel transfer function (Oppenheim and Schafer, 1975), then in the log cepstral domain, such multiplication becomes a simple addition. Therefore, if the averaged offset is zero and independent of the channel, normalising the cepstral mean values to zero by subtraction approximately compensates for the channel difference. More generally speaking, zero mean normalisation (0-MN), $\mathbf{o} = \mathbf{o} - \mu$, and unit variance normalisation (1-VN), $\mathbf{o} = \mathbf{o}/\sigma$, are common feature pre-processing techniques to standardise the range of the data and reduce variabilities, where μ and σ are the mean and standard deviation vectors of $\mathbf{o}$. Note that μ and σ estimates, together with the relevant 0-MN and 1-VN, are often based on a portion of data, such as an utterance, a conversation side, a broadcast show episode, which determines the normalisation level.

2.3 Acoustic Models

2.3.1 Acoustic modelling units

The acoustic model is often designed to model some units connected with the words in the vocabulary, since it handles the generation of likelihood $P(\mathbf{O}|\mathbf{W})$. For LVCSR tasks studied in this thesis, some fine-grained units that constitute parts of the spoken or written forms of words are often used as the acoustic modelling unit, which are often called *subword units*. There are some benefits of using subwords instead of words to construct acoustic models:

- Acoustic models are estimated from a training corpus of utterances and the associated transcriptions. Collecting a corpus with sufficient occurrences of a large vocabulary of words for reliable model parameter estimation is sometimes an unrealistic task. It is especially severe when taking the pronunciation variations into consideration, as every alternative pronunciation of each word needs sufficient samples. However, as there are far fewer phones than words, collecting sufficient data for phone model estimation is much more feasible.

- When deploying an LVCSR product, it is not uncommon to extend the vocabulary to recognise new words that did not appear in training. For word based acoustic models, they have to be rebuilt to model the new words. For phone based acoustic

models, on the other hand, as long as the phone set is sufficient to cover the pronunciations of the new words, no acoustic model modification is required, which makes the application more flexible.

In this thesis, all English ASR systems followed the phone based acoustic modelling strategy. Besides conventional phones, other phonetic units can be used as well, for instance, the phones can be associated with their in-word positions and the relevant tones (for tonal languages), which is adopted by all non-English ASRs in this thesis (Liu et al., 2011). For multi-lingual ASR systems, a global set of phone from the *international phonetic alphabet* (Selkirk, 1986) is sometimes used instead of language dependent phones (Schultz and Waibel, 2001), which can be decomposed into different categories of *articulatory features* representing their articulation status. The articulartory features can also be used to construct acoustic models (Kirchhoff, 1999). The syllable is another option for acoustic modelling units, but it is not always clearly defined in all languages (King et al., 1998). Subword units can be derived automatically and they were sometimes reported to outperform phones (Bacchiani and Ostendorf, 1998; Byrne et al., 2000). Moreover, rather than the spoken form, the written form of words, such as the characters, can be used for acoustic modelling as well (Gales et al., 2015; Kanthak and Ney, 2002).

2.3.2 Hidden Markov models and the forward-backward procedure

In the statistical ASR approach, each phone is normally modelled by an HMM, which assumes that the properties of its associated signals can be characterised by a *discrete time Markov process* or *Markov chain* (Gardiner, 1985). That is, the speech signal can be described at any time as being in one of the N distinct states, and at each time it may undergo a change of state according to a set of probabilities associated with the state, whose probabilistic description requires the specification of the current and the preceding states. An HMM is a first order Markov chain as its probabilistic description is truncated to just the current and the last state. More specifically, if $\mathbf{Q}$ is the state sequence,

$$\mathbf{Q} = q_1, q_2, \ldots, q_T, \tag{2.8}$$

where $q_t \in \{1, 2, \ldots, N\}$. The first order Markov property of an HMM λ is (Gardiner, 1985)

$$P(q_t = j | q_{t-1} = i, q_{t-2} = k, \ldots, \lambda) = P(q_t = j | q_{t-1} = i, \lambda), \tag{2.9}$$

$1 \leqslant i, j, k \leqslant N$. $P(q_t = j | q_{t-1} = i)$ is called a state transition probability and denoted as a_{ij}. In ASR, a_{ij} is often a fixed value, and therefore the duration staying in a particular HMM state follows the *Geometric distribution* (Gardiner, 1985). Based on Eqn. (2.9), we have

$$P(\mathbf{Q}|\lambda) = \prod_{t=1}^{T} P(q_t | q_{t-1}, \lambda). \tag{2.10}$$

Note that a virtual entry state $q_0 = 0$ is introduced in Eqn. (2.10) and $P(q_1 | q_0, \lambda) = P(q_1 | \lambda)$ follows the initial state distribution.

The other keyword of an HMM, *hidden*, means that the state sequence $\mathbf{Q}$ is not directly observable. It can only be observed through another state dependent stochastic process of the feature sequence $\mathbf{O}$, which is extracted from the speech waveform. Hence $\mathbf{O}$ is also called the observation sequence. The probability of observing $\mathbf{O}$ given $\mathbf{Q}$ is

$$P(\mathbf{O}|\mathbf{Q}, \lambda) = \prod_{t=1}^{T} P(\mathbf{o}(t) | q_t, \lambda), \tag{2.11}$$

where each feature vector $\mathbf{o}(t)$ is assumed to be independently generated by the state assigned to q_t. $P(\mathbf{o}(t) | q_t, \lambda)$ is called the output probability and is often denoted as $b_{q_t}(\mathbf{o}(t))$; λ is comprised of $\{a_{ij}\}$ and $\{b_j(\cdot)\}$. A sketch map of an HMM is illustrated in Figure 2.2, and note that the virtual entry state is not shown in it. Therefore, based on Eqns. (2.10) and (2.11), the likelihood $P(\mathbf{O}|\lambda)$ can be calculated by

$$P(\mathbf{O}|\lambda) = \sum_{\mathbf{Q}} P(\mathbf{Q}|\lambda) P(\mathbf{O}|\mathbf{Q}, \lambda) \tag{2.12}$$

$$= \sum_{q_1, q_2, \ldots, q_T} \prod_{t=1}^{T} a_{q_{t-1} q_t} b_{q_t}(\mathbf{o}(t)). \tag{2.13}$$

When evaluating $\mathbf{O}$ with an N state HMM using Eqn. (2.12), about $2T \cdot N^T$ calculations are required with all a_{ij} and $b_j(\mathbf{o}(t))$ available. The computational complexity grows exponentially when T increases, and can be intractable even for a short speech segment (e.g., 1 second with $T = 100$). A more efficient procedure is the

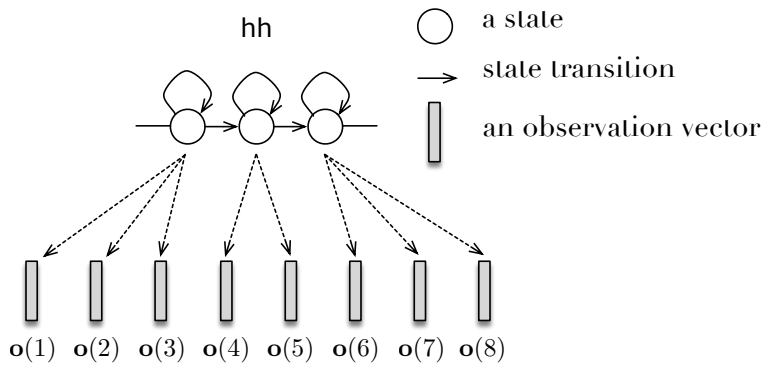


Figure 2.2 A sketch map of an HMM for phone hh.

forward-backward (FB) algorithm, which saves redundant calculations using a recursive structure. Define a *forward variable* $\alpha_i(t)$ as

$$\alpha_i(t) = P(\mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(t), q_t = i|\lambda), \tag{2.14}$$

which is the partial observation sequence $\mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(t)$ and state i at t. $\alpha_i(t)$ is computed through an induction step:

$$\alpha_j(t) = \left[\sum_{i=1}^{N} \alpha_i(t-1)a_{ij}\right] b_j(\mathbf{o}(t)), \tag{2.15}$$

for $1 < t \leqslant T$. This recursion depends on the fact that the probability of the joint event of being in state j at t and seeing $\mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(t)$ can be deduced by summing the transition probabilities weighted forward variables over all possible preceding states. The initial condition for $t = 1$ is

$$\alpha_j(1) = a_{0j} b_j(\mathbf{o}(t)). \tag{2.16}$$

When terminating the forward procedure, the likelihood is acquired by

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_i(T). \tag{2.17}$$

In this way, computing $P(\mathbf{O}|\lambda)$ requires only order $T \cdot N^2$ calculations.

In a similar way, the *backward variable* $\beta_i(t)$ is defined as

$$\beta_i(t) = P(\mathbf{o}(t+1), \mathbf{o}(t+2), \ldots, \mathbf{o}(T)|q_t = i, \lambda), \tag{2.18}$$

which is the conditional probability of observing the partial observation sequence from $t+1$ to the end, given that the model in state i at t. $\beta_i(t)$ can be calculated in a reverse time order by

$$\beta_i(t) = \sum_{j=1}^{N} a_{ij} b_j(\mathbf{o}(t+1)) \beta_j(t+1). \tag{2.19}$$

By multiplying the forward and backward variables, it is easy to see

$$\alpha_i(t)\beta_i(t) = P(\mathbf{O}, q_t = i|\lambda), \tag{2.20}$$

and the likelihood can be calculated by

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_i(t)\beta_i(t). \tag{2.21}$$

Therefore, we complete the evaluation of an HMM. Note this can also be achieved using only $\alpha_i(T)$ by

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_i(T), \tag{2.22}$$

but Eqn. (2.21) is more useful in HMM training, as shown later in Eqns. (2.27) – (2.30).

There are many different kinds of HMMs, and the most commonly seen one for acoustic modelling is the *continuous density HMM*, which employs a Gaussian mixture model (GMM) to model the output distribution (Juang, 1985). A GMM is defined as a weighted sum of a set of Gaussian components, i.e.,

$$\begin{aligned} p(\mathbf{o}(t)|q_t = i, \lambda) &= \sum_{m=1}^{M} c_{im} \mathcal{N}(\mathbf{o}(t)|\mu_{im}, \Sigma_{im}) \\ &= \sum_{m=1}^{M} c_{im} \frac{1}{(2\pi)^{D/2}|\Sigma_{im}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{o}(t)-\mu_{im})^{\mathrm{T}} \Sigma_{im}^{-1} (\mathbf{o}(t)-\mu_{im})\right\}, \end{aligned} \tag{2.23}$$

where D is the input feature vector size, M is the number of components in the GMM of state i; $\mathcal{N}(\mathbf{o}_t|\mu_{im}, \Sigma_{im})$ is the mth Gaussian component with μ_{im} and Σ_{im} representing its mean vector and covariance matrix; and c_{im} is the corresponding weight. When all dimensions of the feature $\mathbf{o}$, o_d, can be treated as independent variables, Σ_{im} becomes a diagonal matrix and its diagonal elements can be presented as a variance vector σ_{im}^2.

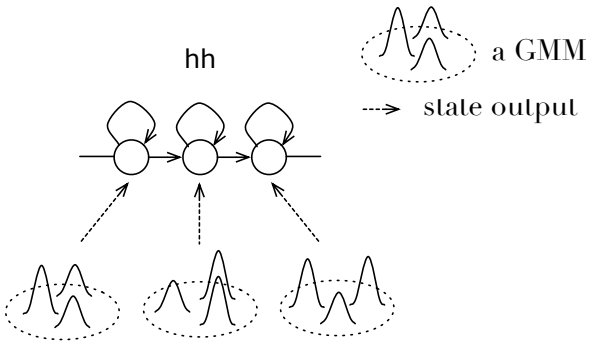


Figure 2.3 A sketch map of a GMM-HMM for phone hh.

Thus, Eqn. (2.23) becomes

$$p(\mathbf{o}(t)|q_t = i, \lambda) = \sum_{m=1}^{M} c_{im} \mathcal{N}(\mathbf{o}(t)|\mu_{im}, \sigma_{im}^2) \tag{2.24}$$
$$= \sum_{m=1}^{M} c_{im} \frac{1}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_{imd}} \exp\left\{ -\frac{1}{2} \sum_{d=1}^{D} \frac{(o_d(t) - \mu_{imd})^2}{\sigma_{imd}^2} \right\},$$

where μ_{imd} and σ_{imd} are the dth dimension of vector μ_{im} and σ_{im}. Note for multivariate Gaussian distributions, since a covariance matrix contains significantly more parameters than a variance vector, it often requires much more data to estimate Σ_{im} robustly. Thus, GMM with a variance vector is a more common choice, and $\mathbf{o}$ are linearly decorrelated as in Section 2.2. As the exact form of the distributions of the observation generation process in each state is unknown, a GMM is employed due to its ability to form smooth approximations to arbitrarily shaped densities (Juang, 1985). Continuous density HMMs are also denoted as GMM-HMMs in this thesis, and an example is shown in Figure 2.3.

Besides using the standard GMMs defined in Eqns. (2.23) and (2.24), continuous density HMMs can be reformed in different ways by having the GMM parameters shared at different levels. Instead of using diagonal covariance matrices to reduce the number of parameters, a full covariance matrix can be shared among all GMMs, which is equivalent to a *log-linear mixture model* (Heigold, 2010). If the GMM in each HMM state is adapted from a globally shared GMM using state dependent vectors, it is called a *subspace GMM-HMM* (Povey et al., 2010). If the GMM parameters are shared at the Gaussian level, or every GMM is formed by Gaussians selected from the same codebook, it is a *semi-continuous density HMM* (Huang, 1989). In addition, it is also possible to replace GMMs with other statistical models, such as *Dirichlet mixture models* (Neal,

2000), *support vector machines* (Vapnik, 1998a,b), and in particular artificial neural network (ANN) models (Bishop, 1995; Bourlard and Morgan, 1993) that are widely used nowadays and will be introduced in detail later in Section 3.1. When the GMM is replaced by discrete probabilities, it is a *discrete density HMM* (Rabiner, 1989). Meanwhile, the use of an autoregression process for observation generation leads to the autoregressive HMM class (Rabiner, 1989). In additional to the output probabilities, the transition probabilities can also follow distributions other than the Geometric distribution, such as the Gamma, Gaussian, and Poisson distributions forming *hidden semi-Markov models* (Levinson, 1986; Russell and Moore, 1985; Yoshimura et al., 1998).

2.3.3 The ML HMM training problem

So far, the acoustic modelling units and model types have been discussed. However, these are still not sufficient to distinguish a set of acoustic models as the model parameters have not yet been determined. Since HMMs are evaluated based on Eqn. (2.12), it is natural to use parameters that can maximise the likelihood. Given an HMM λ, the maximum log-likelihood objective function is defined as

$$\mathcal{F}^{\mathrm{ML}}(\mathbf{O}) = \ln p(\mathbf{O}|\lambda), \tag{2.25}$$

and the maximum likelihood (ML) parameter estimation procedure can be specifically presented as

$$\lambda^* = \arg\max_{\lambda} \sum\nolimits_{\mathbf{O}} \mathcal{F}^{\mathrm{ML}}(\mathbf{O}). \tag{2.26}$$

Prior to describing the actual HMM training procedure, there are several useful variables to define based on the FB algorithm. The first one is $\xi_{ij}(t)$ defined as the probability of being in state i at t and state j at $t+1$, given the model and the entire observation sequence, i.e.,

$$\xi_{ij}(t) = P(q_t = i, q_{t+1} = j|\mathbf{O}, \lambda) \tag{2.27}$$

According to Bayes' rule and Eqn. (2.21), $\xi_{ij}(t)$ can be calculated by

$$\begin{aligned} P(q_t = i, q_{t+1} = j|\mathbf{O}, \lambda) &= \frac{p(\mathbf{O}, q_t = i, q_{t+1} = j|\lambda)}{p(\mathbf{O}|\lambda)} \\ &= \frac{\alpha_i(t) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_j(t+1)}{\sum_{i'=1}^{N} \alpha_{i'}(t) \beta_{i'}(t)}. \end{aligned} \tag{2.28}$$

The next variable $\gamma_i(t)$ is the posterior probability of being in state i at t,

$$\gamma_i(t) = P(q_t = i|\mathbf{O}, \lambda). \tag{2.29}$$

Similarly, $\gamma_i(t)$ can also be computed using the FB algorithm as

$$\begin{aligned} P(q_t = i|\mathbf{O}, \lambda) &= \frac{p(\mathbf{O}, q_t = i|\lambda)}{p(\mathbf{O}|\lambda)} \\ &= \frac{\alpha_i(t)\beta_i(t)}{\sum_{i'=1}^{N} \alpha_{i'}(t)\beta_{i'}(t)}. \end{aligned} \tag{2.30}$$

$\gamma_i(t)$ is often termed as the state occupancy as it reflects *how probable* that a frame $\mathbf{o}(t)$ is aligned with a certain state i. The last two variables are associated with GMMs and require an assumption that each sample is generated by only one Gaussian component. The posterior probability of being generated by the mth Gaussian component given state i and the observation sequence is represented by a variable $\pi_{im}(t)$. That is,

$$\begin{aligned} \pi_{im}(t) =& P(g_t = m|q_t = i, \mathbf{O}, \lambda), \\ =& \frac{c_{im}\mathcal{N}(\mathbf{o}(t)|\mu_{im}, \Sigma_{im})}{\sum_{m'=1}^{M} c_{im'}\mathcal{N}(\mathbf{o}(t)|\mu_{im'}, \Sigma_{im'})}, \end{aligned} \tag{2.31}$$

where g_t is the auxiliary variable revealing the membership of $\mathbf{o}(t)$, and $1 \leqslant m \leqslant M$. Finally, we define $\gamma_{im}(t)$ as

$$\gamma_{im}(t) = P(q_t = i, g_t = m|\mathbf{O}, \lambda), \tag{2.32}$$

and then we have

$$\begin{aligned} P(q_t = i, g_t = m|\mathbf{O}, \lambda) =& P(q_t = i|\mathbf{O}, \lambda)P(g_t = m|q_t = i, \mathbf{O}, \lambda) \\ =& \gamma_i(t)\pi_{im}(t), \end{aligned} \tag{2.33}$$

based on Eqn. (2.29) and Eqn. (2.31).

2.3.4 The Baum-Welch algorithm

Next, we are going to solve Eqn. (2.26). Since there is no known way to analytically find the solution λ^*, HMM parameters are often estimated through iterative optimisation procedures (Baum and Egon, 1972; Dempster et al., 1977; Gill et al., 1981), among which the most common one is the Baum-Welch (BW) algorithm (Baum and Egon,

1972). Instead of directly maximising $\mathcal{F}^{\mathrm{ML}}$, the BW algorithm is derived by maximising *Baum's auxiliary function* (Baum and Egon, 1972), which is also named the $\mathcal{Q}$ function,

$$\mathcal{Q}(\lambda,\hat{\lambda}) = \sum_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{O},\lambda)\ln p(\mathbf{O},\mathbf{Q}|\hat{\lambda}), \tag{2.34}$$

where $\hat{\lambda} = \{\{\hat{a}_{ij}\},\{\hat{b}_j(\cdot)\}\}$ is the re-estimated parameter set. The solution found by maximising the $\mathcal{Q}$ function also maximises $\mathcal{F}^{\mathrm{ML}}$, as shown later in Section 2.3.5. By taking Eqns. (2.10) and (2.11) into the $\mathcal{Q}$ function, it can be re-written as

$$\mathcal{Q}(\lambda,\hat{\lambda}) = \sum_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{O},\lambda)\sum_{t=1}^{T}\ln\hat{a}_{q_{t-1}q_t} + \sum_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{O},\lambda)\sum_{t=1}^{T}\ln\hat{b}_{q_t}(\mathbf{o}_t). \tag{2.35}$$

Since the two terms of Eqn. (2.35) are associated with different types of HMM parameters, they can be treated separately.

By re-organising and applying *Lagrange multipliers* (Gill et al., 1981) to the first term in Eqn. (2.35) with constraints $\sum_{j=1}^{N}\hat{a}_{ij}=1$ and $\sum_{m=1}^{M}\hat{c}_{im}=1$, we can get

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} P(q_{t-1}=i,q_t=j|\mathbf{O},\lambda)}{\sum_{t=1}^{T} P(q_{t-1}=i|\mathbf{O},\lambda)} = \frac{\sum_{t=1}^{T}\xi_{ij}(t-1)}{\sum_{t=1}^{T}\gamma_i(t-1)}, \tag{2.36}$$

$$\hat{c}_{im} = \frac{\sum_{t=1}^{T} P(q_t=i,g_t=m|\mathbf{O},\lambda)}{\sum_{t=1}^{T}\sum_{m'=1}^{M} P(q_t=i,g_t=m'|\mathbf{O},\lambda)} = \frac{\sum_{t=1}^{T}\gamma_{im}(t)}{\sum_{t=1}^{T}\sum_{m'=1}^{M}\gamma_{im'}(t)}. \tag{2.37}$$

Meanwhile, by taking the derivatives of the second item with respect to $\hat{\mu}_{im}$ and $\hat{\Sigma}_{im}$ and setting them to zero, we have

$$\begin{aligned}\hat{\mu}_{im} &= \frac{\sum_{t=1}^{T} P(q_t=i,g_t=m|\mathbf{O},\lambda)\,\mathbf{o}(t)}{\sum_{t=1}^{T} P(q_t=i,g_t=m|\mathbf{O},\lambda)} \\ &= \frac{\sum_{t=1}^{T}\gamma_{im}(t)\,\mathbf{o}(t)}{\sum_{t=1}^{T}\gamma_{im}(t)},\end{aligned} \tag{2.38}$$

$$\begin{aligned}\hat{\Sigma}_{im} &= \frac{\sum_{t=1}^{T} P(q_t=i,g_t=m|\mathbf{O},\lambda)(\mathbf{o}(t)-\hat{\mu}_{im})(\mathbf{o}(t)-\hat{\mu}_{im})^{\mathrm{T}}}{\sum_{t=1}^{T} P(q_t=i,g_t=m|\mathbf{O},\lambda)} \\ &= \frac{\sum_{t=1}^{T}\gamma_{im}(t)\mathbf{o}(t)\mathbf{o}(t)^{\mathrm{T}}}{\sum_{t=1}^{T}\gamma_{im}(t)} - \hat{\mu}_{im}\hat{\mu}_{im}^{\mathrm{T}}.\end{aligned} \tag{2.39}$$

When building a set of GMM-HMM acoustic models, an HMM is constructed for each phone unit (including silence, as described in Section 2.8 in detail). For

each utterance in the training corpus, it is segmented into speech segments according to phone to frame alignments, and each such segment is regarded as an individual observation sequence in this section. The GMM-HMM parameter update formulas in Eqns. (2.37) – (2.39) are executed once their numerators and denominators have been accumulated from every relevant segment of all utterances in the training corpus. Note that phone to frame alignments can be acquired by converting the word level *reference* transcriptions into phone sequences and finding the time boundaries of each phone manually or automatically with the Viterbi algorithm (see Section 2.4.2). As an alternative to hard segmentations, the FB algorithm can be applied at the utterance level using a *composite HMM* to produce "soft" segmentations, which is introduced in detail in Section 2.8.

2.3.5 Properties of the BW algorithm

Next, we will show that the BW algorithm can guarantee the monotonic increase of the training data likelihood. First, $\mathcal{F}^{\mathrm{ML}}$ is re-arranged as

$$\begin{aligned}\ln p(\mathbf{O}|\hat{\lambda}) &= \ln \sum\nolimits_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{O},\lambda)\frac{p(\mathbf{O},\mathbf{Q}|\hat{\lambda})}{P(\mathbf{Q}|\mathbf{O},\lambda)} \\ &\geqslant \sum\nolimits_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{O},\lambda)\ln\frac{p(\mathbf{O},\mathbf{Q}|\hat{\lambda})}{P(\mathbf{Q}|\mathbf{O},\lambda)}. \end{aligned} \tag{2.40}$$

The inequality in Eqn. (2.40) makes use of *Jensen's inequality* (Bishop, 2006). Since the BW algorithm increases the $\mathcal{Q}$ function value such that

$$\sum\nolimits_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{O},\lambda)\ln p(\mathbf{O},\mathbf{Q}|\hat{\lambda}) \geqslant \sum\nolimits_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{O},\lambda)\ln p(\mathbf{O},\mathbf{Q}|\lambda), \tag{2.41}$$

from Eqn. (2.40) we have

$$\ln p(\mathbf{O}|\hat{\lambda}) \geqslant \ln \sum\nolimits_{\mathbf{Q}} P(\mathbf{Q}|\mathbf{O},\lambda)\ln\frac{p(\mathbf{O},\mathbf{Q}|\lambda)}{P(\mathbf{Q}|\mathbf{O},\lambda)} \geqslant \ln p(\mathbf{O}|\lambda), \tag{2.42}$$

which shows that the BW algorithm guarantees that the likelihood will not decrease.

Further examing the $\mathcal{Q}$ function, Eqn. (2.34) becomes

$$\mathcal{Q}(\lambda,\hat{\lambda}) = \mathbb{E}_{\mathbf{Q}|\mathbf{O},\lambda}\left[\ln p(\mathbf{O},\mathbf{Q}|\hat{\lambda})\right], \tag{2.43}$$

where $\mathbb{E}_{\mathbf{Q}|\mathbf{O},\lambda}[\cdot]$ is the mathematical expectation with respect to $P(\mathbf{Q}|\mathbf{O},\lambda)$. Therefore, the BW algorithm can be interpreted as an implementation of the more general expec-

tation maximisation (EM) algorithm (Dempster et al., 1977). From this perspective, the BW algorithm comprises the E (expectation) step, which calculates the $\mathcal{Q}$ function and the related variables using the FB algorithm, and the M (maximisation) step that updates the parameters to maximise the expected log-likelihood based on the E step. It should be noted that λ^* found by the BW algorithm is only a local rather than global optimum (Bishop, 2006), which is usually one of the multiple local optima on a complex optimisation surface.

2.4 Language Models and The Decoding Process

2.4.1 Language modelling

In ASR, besides a dictionary with a vocabulary of words and their pronunciations, linguistic and semantic knowledge are also used in the LM, which provides the prior probability of each word sequence, $P(\mathbf{W})$, in the MAP decoding rule in Eqn. (2.3).

$$\begin{aligned} P(\mathbf{W}) &= P(w_1, w_2, \ldots, w_L) \\ &= P(w_1)P(w_2|w_1)\cdots P(w_L|w_1, w_2, \ldots, w_{L-1}) \end{aligned} \tag{2.44}$$

If $\mathbf{W}$ is assumed to have an nth order Markov property, that is, each word depends on at most $n-1$ preceding words, then the above equation can be re-written as

$$P(\mathbf{W}) = \prod_{l=1}^{L} P(w_l|w_{l-n+1}, w_{l-n+2}, \ldots, w_{l-1}). \tag{2.45}$$

Eqn. (2.45) defines an n-gram model that has been the most successful and commonly used LM (Jelinek, 1991; Manning and Schütze, 1999). $P(w_l|w_{l-n+1}, w_{l-n+2}, \ldots, w_{l-1})$ is estimated based on ML as

$$P(w_l|w_{l-n+1}, w_{l-n+2}, \ldots, w_{l-1}) = \frac{\text{count}(w_{l-n+1}, w_{l-n+2}, \ldots, w_l)}{\text{count}(w_{l-n+1}, w_{l-n+2}, \ldots, w_{l-1})}, \tag{2.46}$$

where $\text{count}(w_{l-n+1}, w_{l-n+2}, \ldots, w_l)$ is the number of observed $w_{l-n+1}, w_{l-n+2}, \ldots, w_l$ units in the training corpus.

Taking a deeper look at the n-gram model, for a vocabulary sized $|V|$, it contains $|V|^n$ discrete probabilities. However, when n is large, most n-gram units do not have sufficient occurrences in the training corpus for reliable estimation giving rise to a severe data sparseness issue. Unseen n-gram units with zero frequencies can even prohibit

ASR from outputting such units in the hypotheses, regardless of how unambiguous the acoustic signal is. This can certainly be alleviated by training the LM on more data to cover the n-gram units better. Meanwhile, the estimation of infrequent n-gram units can be smoothed by "backing-off" to models with smaller histories (Manning and Schütze, 1999). Different smoothing techniques have been developed based on such backoff strategies (Church and Gale, 1991; Katz, 1987; Kneser and Ney, 1995).

2.4.2 Decoding process

The decoding process relies on creating a finite state graph structure that integrates the acoustic model, dictionary and language model. This graph can be constructed statically (Baker, 1975; Mohri et al., 2002; Young et al., 1989), dynamically (Odell, 1995; Odell et al., 1994), or in a hybrid manner that dynamically adds language model information to a static graph (Demuynck et al., 2000; Odell, 1999). The graph includes often context dependent HMM models appropriately connected according to the dictionary and language model. The speech recognition task then corresponds to finding the most likely path through the graph that will have generated the utterance to be decoded according to the MAP decoding rule in Eqn. (2.3). Common breadth-first, depth-first, and heuristic pathfinding algorithms can be used for this task (Huang et al., 2001; Young, 1996). The most widely used search method is *Viterbi decoding* (Viterbi, 1967), which is a dynamic programming (DP) algorithm (Cormen et al., 1990) for finding the most likely sequence of hidden states. Let $\phi_j(t)$ represent the maximum likelihood of being in state j at time t and observing $\mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(t)$. $\phi_j(t)$ can be computed recursively by

$$\phi_j(t) = \max_i \{\phi_i(t-1) + \ln a_{ij}\} + \ln b_j(\mathbf{o}(t)). \tag{2.47}$$

When computing each $\phi_j(t)$, the maximum operation removes the paths that can not form the overall optimal path. At the last time T, the maximum likelihood value of generating $\mathbf{O}$ with HMM λ is obtained by

$$\hat{\phi}(T) = \max_e \left\{ \max_i \{\phi_i(T-1) a_{ie}\} + \ln b_e(\mathbf{o}(T)) \right\}, \tag{2.48}$$

where state e is any of the exit states of HMM λ.

Equivalently, the *token passing model*, an alternative form of the Viterbi algorithm, is often used in practical decoders (Woodland et al., 1994; Young et al., 1989). In this conceptual model, at time t, an HMM state j holds a moveable token that contains

$\phi_j(t)$ and the history to achieve it. At each time t, a copy of every token in every state i is passed to all of their possible succeeding states j, and each token has its value increased by $\ln a_{ij} + \ln b_j(\mathbf{o}(t))$. Afterwards, in each state, discard all tokens except for the one with the highest value. When multiple HMMs are considered, a token in an exit state of an HMM can be passed to the connected states in other HMMs. If the two HMMs belong to different words, say w_l and w_{l+1}, the token passing causes a word transition and the token value needs to be increased further by $\ln P(w_{l+1}|w_l)$. By assuming every utterance starts with a silence word, w_0, initially $\ln P(w_1)$ is added to the token value by setting $\ln P(w_1|w_0) = \ln P(w_1)$[1]. These enable the token passing algorithm to search for the maximum posterior probability rather than the maximum likelihood path by taking the LM into account. The LM log-probability is usually linearly scaled by a *grammar scaling factor* (Bahl et al., 1980) when combined with the acoustic log-likelihood. This is necessary since HMM acoustic models often produce a wider dynamic range of likelihood values due to the underestimation of likelihood arising from invalid assumptions (Woodland and Povey, 2002). At the end of the search, again every utterance is assumed to end with silence, then the best path with the highest probability is acquired only from the tokens staying in the exit states of the silence HMM at T. In addition, the token passing model is also useful for finding N-best paths by allowing each model state to hold multiple tokens to increase the number of different token histories that can be maintained. Tokens from different preceding words are regarded as distinct. Empirically, this has been shown to well approximate the optimal N-best paths acquired without discarding any token (Young et al., 2015).

For LVCSR, as discussed in Section 2.1, efficiency becomes more critical due to the increase of decoding complexity. A common strategy is to reduce the search space by sharing possible phone model evaluations in different pronunciations. However, this is still not adequate in practice. Since Viterbi decoding is a time-synchronous breadth-first search, it is possible to prune the searching space by reducing the "breadth". That is, at each time, any model whose token value falls more than some threshold below the maximum among all models is deactivated. It works as to only search within paths defined by a beam of a certain width and is therefore named as *beam search* (Lowerre, 1976). A similar method can also be applied among the tokens being propagated into new words, which is called *word end pruning*. Note that the beam search can prune the optimal path and cause word error rate (WER) increase, but normally a good result accuracy/computation complexity trade off can often be found with appropriate

[1] These LM scores can also be pre-compiled into the graph.

thresholds. The HTK LVCSR decoder, `HDecode`, is used in all experiments in this thesis (Young et al., 2015). In `HDecode`, the LM is decoupled from the dictionary, in order to handle their information separately using multiple tokens. The dictionary and HMMs are used to construct a compact static graph, while the LM information is incorporated into the search dynamically at the run time. This achieves a good trade off between the computation and storage complexity, and allows a flexible choice of the LM. In addition, decoders can also be equivalently designed based on weighted finite state transducers (WFST) by representing each of the acoustic model, LM, and lexicon as a *weighted finite state automata* to simplify their integration in decoding (Mohri, 1997; Mohri et al., 2002).

2.4.3 Representing hypotheses using lattices

Section 2.4.2 briefly describes decoders based on Viterbi decoding. In general, more sophisticated linguistic and phonetic knowledge, such as a higher order LM, is often used in decoding, which can even result in an unmanageably large search space or infeasible decoder implementation. One solution is to apply the "divide and conquer" strategy to decompose the searching objective into multiple progressive stages, which leads to the *multi-pass decoding* (Huang et al., 2001; Young, 1996). For example, to decode using a 4-gram LM, a bigram LM can be applied in the first decoding pass, while in the second pass the 4-gram LM is used to rescore the resulting word level hypotheses by keeping the acoustic model scores produced in the first pass. For this purpose, a data structure is required to represent and cache the word level hypotheses. A straightforward solution is the *word lattice* (Woodland et al., 1995), or equivalently the *word graph* (Ney and Aubert, 1994), which is a weighted DAG specified for each utterance with time information. A word lattice consists of a number of nodes placed along the time axis and many arcs connecting the nodes. Except for the entry node, every node serves as the common end of its inbound words and is associated with its ending time t. An arc between two nodes with t_1 and t_2 is a directed link corresponding to a hypothesised word aligned with the observation sequence $\mathbf{o}(t_1), \mathbf{o}(t_1+1), \ldots, \mathbf{o}(t_2)$. Therefore, every path covering time from 1 to T is a valid hypothesis, and the common words among some hypotheses are efficiently represented by the shared arcs of corresponding paths.

One benefit of using a word lattice, is to efficiently rescore the embedded hypotheses using a lattice based modification of the FB algorithm. Like in Section 2.3.2, we can define the lattice based forward and backward variables α_q and β_q analogous to the

$\alpha_i(t)$ and $\beta_i(t)$. Let Λ be the HMM set used to produce the lattice,

$$\alpha_q = p(\mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(t_q^e - 1), q|\Lambda) \tag{2.49}$$

$$\beta_q = p(\mathbf{o}(t_q^e), \mathbf{o}(t_q^e + 1), \ldots, \mathbf{o}(T)|q, \Lambda), \tag{2.50}$$

where q is an arc whose starting and ending times are t_q^s and t_q^e. Their properties are found similar to Eqns. (2.15) and (2.19). Provided that arcs h and q are the immediate preceding arcs of q and r, denoting $p(q) = p(\mathbf{o}(t_q^s), \mathbf{o}(t_q^s + 1), \ldots, \mathbf{o}(t_q^e - 1)|q, \Lambda)$, the recursive calculation formulas of α_q and β_q are

$$\alpha_q = \sum_h \alpha_h P(q|h) p(q) \tag{2.51}$$

$$\beta_q = \sum_r \beta_r P(r|q) p(r). \tag{2.52}$$

Back to the example of rescoring a lattice with a 4-gram LM, $p(q)$ is the acoustic score saved from lattice generation, and $P(q|p)$ is the LM score. However, since the 4-gram LM requires more preceding words than h, the lattice needs to be expanded before rescoring to make the nodes represent multiple preceding words (Liu et al., 2013; Woodland et al., 1995).

It is empirically found that though the whole hypothesis space is very large, it can be well approximated by a small subset of hypotheses. Therefore, the word lattices from the first pass are normally pruned (Woodland et al., 1995), which can significantly reduce the search space of the following decoding stages. The pruned lattices can be measured by *lattice density*, which is often defined as the averaged number of arcs per second (APS) indicating the amount of hypotheses reserved in the lattice. The lowest WER for any word sequence in the lattice is often used to evaluate the quality of the lattice, since pruning can remove such sequences from the lattice (Woodland et al., 1995). Note that there exist multiple lattice definitions. For example, time information can be excluded from the word lattice or word graph (Ljolje et al., 1999). Equivalently, WFST based technology can be applied to lattices as well (Povey et al., 2012).

2.4.4 WER evaluation

The choice of the evaluation standard of an ASR system depends on the specific recognition task. For English LVCSR tasks studied in this thesis, WER is often used, which measures the ratio of words differing between the reference and hypothesis

transcriptions[1]. However, since it is not necessary to have the same number of words in a pair of reference and hypothesis, each hypothesis word needs to be aligned to a reference word for comparison purpose. Three kinds of errors arise from such comparisons:

- *Substitution error*, which occurs when a hypothesis word is aligned with a reference word and they are different. This means a word is misrecognised as another.
- *Insertion error*, which occurs when a hypothesis word cannot be aligned with any reference word. This indicates an additional word has been produced by ASR.
- *Deletion error*, which occurs when a reference word is not aligned to any word in the hypothesis indicating a word has been omitted by ASR.

Therefore, word sequence alignment is critical in hypothesis evaluation, which is often carried out so as to minimise the *Levenshtein edit distance* between the two sequences using a DP procedure (Levenshtein, 1966). The Levenshtein edit distance is defined as

$$\begin{cases} 1 & \text{if correct} \\ 0 & \text{if a substitution error} \\ -1 & \text{if an insertion error} \end{cases}. \tag{2.53}$$

Based on Eqn. (2.53), the word accuracy is

$$\%\text{WordAcc} = \frac{H - I}{N} \times 100\%, \tag{2.54}$$

where H, I, and N are the numbers of correctly recognised words, insertion errors, and words in the reference sequence. WER is therefore calculated by

$$\%\text{WER} = 100\% - \%\text{WordAcc}. \tag{2.55}$$

In addition to the Levenshtein edit distance, there are other edit distances serving as the DP cost functions for hypothesis evaluation purpose. If the insertion error cost is changed to 0 in Eqn. (2.53), i.e., word insertions are not penalised, the evaluation standard becomes the word correctness (Young et al., 2015). However, it is not as useful an evaluation standard as WER for ASR outputs with variable lengths. To show

[1]For the Mandarin, Cantonese, and Tamil speech recognition systems in this thesis, character error rate (CER) is used instead of WER.

this, a dummy hypothesis can be made as an example,

$$w_1^{(1)}, w_1^{(2)}, \ldots, w_1^{(|V|)}, w_2^{(1)}, w_2^{(2)}, \ldots, w_2^{(|V|)}, \ldots, w_T^{(1)}, w_T^{(2)}, \ldots, w_T^{(|V|)}, \tag{2.56}$$

where $w_t^{(1)}, w_t^{(2)}, \ldots, w_t^{(|V|)}$ are all words in the vocabulary enumerated for frame t. This dummy hypothesis always has 100% word correctness (assuming there are no OOVs), but is useless in practice. The edit distance can also be expanded to include unit dependent costs to weigh the errors of different units differently. This is typically used for specific ASR error pattern extraction (Fung et al., 2000).

Besides WER, ASR performance is sometimes evaluated at a different level. For phone recognition, *phone error rate* is often used and calculated similar to WER, with the word errors replaced by corresponding phone errors. Another useful metric is the sentence error rate (SER) that counts utterances fully correctly recognised using a *0-1 loss function*. The 0-1 loss function is defined as (Bishop, 2006)

$$\delta_{\mathbf{W}\mathbf{W}'} = \begin{cases} 1 & \text{if } \mathbf{W} = \mathbf{W}' \\ 0 & \text{otherwise} \end{cases}, \tag{2.57}$$

where $\mathbf{W}$ and $\mathbf{W}'$ are the reference and hypothesis sequences, and $\delta_{\mathbf{W}\mathbf{W}'}$ is also termed as *Kronecker delta function*.

2.5 Context-Dependent Acoustic Modelling

In continuous speech processing, an important factor to take into account is the *coarticulation* effect. Coarticulation refers to a situation that an isolated speech sound is influenced by its neighbouring speech sounds, which is generally produced through the continuous movement of speech organs when articulating (Selkirk, 1986). The resulted pronunciation changes are usually handled by extending the acoustic modelling units to depend on their neighbouring units for better categorisation of sounds. For example, a phone a can be specified to b−a+c, by taking its preceding phone b and succeeding phone c into consideration. Such an extended phone unit is called a *triphone* (Schwartz et al., 1985), which is a particular type of context-dependent (CD) units, while standard phones are often referred to as a *monophone* that belongs to the context-independent (CI) unit class. An alternative solution is to directly model the dependencies between the current and context phones using a more flexible model rather than HMM, for example, *dynamic Bayesian networks* (Bishop, 2006; Frankel et al., 2007).

When constructing an HMM for each triphone, an issue encountered is explosion in the number of parameters. For instance, if there are 40 monophones in the phone set, a maximum number of $40^3 = 64000$ triphones could be generated. Though in practice the actual triphones (referred to as *logical triphones*) are normally fewer than the maximum number due to the linguistic rules applied through the dictionary, HMMs still suffer from the data sparsity issue since many triphones may have insufficient occurrences in the training corpus. A solution is to tie some parameters of some HMMs together, which can share samples of the involved triphones for better estimation. This parameter tying can happen at different levels in an HMM. The most commonly used strategy is to tie the GMMs in HMM states as the tied state systems. Another important choice in parameter tying is the clustering approach. Two types of clustering approaches are often used: bottom-up and top-down. The bottom-up approaches often do data-driven clustering of the individual untied states according to some metric (Chesta et al., 1997; Young et al., 1994; Young and Woodland, 1993), while in contrast, the top-down approaches split a collection of untied states into classes through the growth of a decision tree (Bahl et al., 1991). For triphone state clustering, the top-down approaches have some key advantages that unseen triphone states can be clustered and the linguistic knowledge is easier to be utilised, while comparable in performance with the bottom-up approaches (Young et al., 1994). Next, a detailed description of the ML criterion based decision tree tying approach is presented (Odell, 1995; Young et al., 1994).

In the decision tree state tying approach, a decision tree is often constructed for every state of every non-silence monophone, which restricts the tying to happen only among the same states of the triphone HMMs with the same central phone. With this configuration, it is implicitly assumed that the states assigned to different decision trees should not be tied as they belong to different linguistic spaces. Other configurations, such as a single decision tree for all states, are also possible (Knill et al., 2013; Paul, 1997). In each tree node, the state set S_p in the node is split into two classes $S_{r_1^*}$ and $S_{r_2^*}$ according to their answers to a binary question r^*, and each class is associated with a child node. Question r^* is a linguistic rule selected from a pre-designed question set $\mathcal{R}$ that results in the largest likelihood increase by splitting the current node,

$$r^* = \arg\max_{r \in \mathcal{R}} \Delta\mathcal{L}(S, r), \tag{2.58}$$

where

$$\Delta\mathcal{L}(S,r^*)=\mathcal{L}(S_{r_1^*})+\mathcal{L}(S_{r_2^*})-\mathcal{L}(S_p). \tag{2.59}$$

$\mathcal{L}(S)$, the log-likelihood of generating a set of observation $\mathbf{O}$ with a state set S, is

$$\begin{aligned}\mathcal{L}(S)&=\sum_{t=1}^{T}\ln p(\mathbf{o}(t)|S)\\&=\sum_{t=1}^{T}\ln\left(\sum_{s\in S}P(q_t=s|\mathbf{o}(t),S)\frac{p(\mathbf{o}(t),q_t=s|S)}{P(q_t=s|\mathbf{o}(t),S)}\right)\\&\geqslant\sum_{t=1}^{T}\sum_{s\in S}\ln\mathcal{N}(\mathbf{o}(t)|\mu_S,\Sigma_S)\gamma_s(t).\end{aligned} \tag{2.60}$$

Similar to the case in Section 2.3.5, the inequality is again due to Jensen's inequality. $P(q_t=s|\mathbf{o}(t),S)$ is denoted as $\gamma_s(t)$ since the HMM transition probabilities are ignored and each frame is generated individually. Furthermore, $p(\mathbf{o}(t)|S)$ is replaced by $\mathcal{N}(\mathbf{o}(t);\mu_S,\Sigma_S)$ by assuming that the samples are (single) Gaussian distributed. By taking equality

$$\sum_{t=1}^{T}\sum_{s\in S}(\mathbf{o}(t)-\mu_S)^{\mathrm{T}}\Sigma_S^{-1}(\mathbf{o}_t-\mu_S)\gamma_s(t)=D\sum_{t=1}^{T}\sum_{s\in S}\gamma_s(t), \tag{2.61}$$

into Eqn (2.60), we can get

$$\mathcal{L}(S)\approx-\frac{1}{2}\left(D\ln(2\pi)+\ln|\Sigma_S|+D\right)\sum_{t=1}^{T}\sum_{s\in S}\gamma_s(t), \tag{2.62}$$

where D is the dimension number. Since $\sum_{t=1}^{T}\sum_{s\in S}\gamma_s(t)$ can be calculated using the FB algorithm, computing $\mathcal{L}(S)$ requires Σ_S. Denote the single Gaussian output distribution of state s as $\mathcal{N}(\mathbf{o}(t)|\mu_s,\Sigma_s)$, and obtain it from pre-trained untied triphone HMMs. According to Eqns (2.38) and (2.39), Σ_S can be acquired as

$$\Sigma_S=\frac{\sum_{s\in S}\left(\Sigma_s+\mu_s^{\mathrm{T}}\mu_s\right)\sum_{t=1}^{T}\gamma_s(t)}{\sum_{t=1}^{T}\sum_{s\in S}\gamma_s(t)}-\mu_S^{\mathrm{T}}\mu_S, \tag{2.63}$$

where

$$\mu_S = \frac{\sum_{s \in S} \mu_s \sum_{t=1}^{T} \gamma_s(t)}{\sum_{t=1}^{T} \sum_{s \in S} \gamma_s(t)}. \tag{2.64}$$

At each step, split the node with the question that can bring the highest likelihood gain. This single tree node splitting procedure is applied to each node unless:

- There is no more questions available in the question set.
- S has only one state.
- $\sum_{t=1}^{T} \sum_{s \in S} \gamma_s(t)$ is smaller than a threshold.
- $\mathcal{L}(S, r^*) = \max \Delta\mathcal{L}(S, r)$ falls below a threshold.

The tree construction stops once no more tree node splits occur. Note that it is often assumed that the alignments throughout the tree construction procedure are fixed, which makes $\sum_t \gamma_s(t)$, μ_s, and Σ_s constant and can reduce the computation cost. In the final stage, the decrease in log-likelihood by merging leaf nodes with different parents is calculated using Eqn. (2.59). Any pairs of tree leaf nodes whose log-likelihood decrease falls below a threshold are merged. Once the trees are constructed, the triphone states unseen in the training corpus can be classified into the existing tree leaf nodes by answering the questions attached to the tree nodes. Moreover, this state level tying usually results in many logical triphone HMMs with the same tied states, which therefore, can be tied together. This leads to a more compact set of HMMs with identical state definitions termed as *physical HMMs* (Young et al., 2015).

Further improvements to the standard ML based decision tree can remove the single Gaussian distribution assumption by using GMMs (Reichl and Chou, 2000), or adding MAP (Gauvain and Lee, 1994) based hierarchical priors (Zen and Gales, 2011). Alternatively, *entropy* (Hwang et al., 1996) and *minimum description length* (MacKay, 2003; Shinoda and Watanabe, 2000) are also widely used as the criterion for decision tree construction. In addition, the requirement of question set can be removed as well (Beulen and Ney, 2000; Chou, 1991; Povey et al., 2011).

2.6 Maximum Likelihood Linear Transforms

Various linear transformations have been applied to HMM-based ASR in the past decades by assuming that the mismatch between the original models and a particular

data set is piece-wise linear (Gales, 1998). This section describes three linear transformations involved in this thesis, namely maximum likelihood linear regression (MLLR) (Leggetter and Woodland, 1995), semi-tied covariance matrices (STC) (Gales, 1999), and heteroscedastic linear discriminant analysis (HLDA) (Kumar, 1997; Liu et al., 2003), all of which are estimated to maximise the likelihood of the data generated by the HMMs.

2.6.1 Maximum likelihood linear regression

It is well known that there exist unique characteristics in speech from distinct speakers (Choukri and Chollet, 1986). Rather than speaker independent (SI) acoustic models, a general model set built upon many speakers' data using speaker specific acoustic models, is therefore considered to have a better chance to model such characteristics well. However, constructing such a model set usually requires a large amount of data from target speakers, which is often hard or infeasible to collect. An alternative solution is to adapt SI models using a small portion of speaker specific data to capture the characteristics of his/her voice, which is termed *speaker adaptation* (Choukri and Chollet, 1986; Cox and Bridle, 1989; Gales, 2000; Gauvain and Lee, 1994; Leggetter and Woodland, 1995; Woodland, 2001).

MLLR is a widely used speaker adaptation method that employs linear transforms to adapt the mean and covariance values of Gaussian components by

$$\begin{aligned}\hat{\mu} =& \mathbf{A}\mu + \mathbf{b} \\ =& \mathbf{W}\begin{bmatrix}\mu \\ 1\end{bmatrix}\end{aligned} \tag{2.65}$$

$$\hat{\Sigma} = \mathbf{H}\Sigma\mathbf{H}^{\mathrm{T}}, \tag{2.66}$$

where $\hat{\mu}$ and are $\hat{\Sigma}$ are the transformed mean vector and covariance matrix, and $\mathbf{W} = [\mathbf{A}\ \mathbf{b}]$ and $\mathbf{H}$ denote their associated linear transforms. As its name suggests, the linear transforms are estimated by maximising the likelihood of generating the speaker specific data from the models, which is a linear regression problem. From Section 2.3.4, this could be done using the BW algorithm. The $\mathcal{Q}$ function from Eqn. (2.35) applied on HMM set Λ equals to

$$\mathcal{Q}(\Lambda,\hat{\Lambda}) \propto -\frac{1}{2}\sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{t=1}^{T}\gamma_{im}\left[D_{im}+\ln|\hat{\Sigma}_{im}|+(\mathbf{o}(t)-\hat{\mu}_{im})^{\mathrm{T}}\hat{\Sigma}_{im}^{-1}(\mathbf{o}(t)-\hat{\mu}_{im})\right], \quad (2.67)$$

where D_{im} is the normalisation constant associated with the Gaussian component m in state i. The re-estimation formulae of $\mathbf{W}$ can be obtained by differentiating $\mathcal{Q}(\Lambda,\hat{\Lambda})$ with respect to $\mathbf{W}$ and equating to zero (Leggetter and Woodland, 1995). Note that for each speaker, the speaker dependent (SD) linear transforms can be tied across a number of distributions through a *regression class tree* to share the limited adaptation samples (Leggetter and Woodland, 1995). $\mathbf{W}^{(r)}$ is the regression matrix of the rth class. Once $\mathbf{W}^{(r)}$ is updated, $\mathbf{H}^{(r)}$ is then re-estimated using a formulae obtained in a similar way (Gales and Woodland, 1996). Note that since MLLR allows different models to have different linear transforms, the overall transform is actually piece-wise linear, and therefore the BW algorithm is often performed for multiple iterations.

Besides applying the linear transforms to model parameters, MLLR can also be applied to input features. This is achieved by constraining the mean and covariance of a Gaussian component to share the same linear transform (Digalakis et al., 1995; Gales, 1998), i.e, let $\mathbf{H}=\mathbf{A}$. Since

$$\mathcal{N}(\mathbf{o}(t)|\mathbf{A}\mu+\mathbf{b},\mathbf{A}\Sigma\mathbf{A}^{\mathrm{T}}) = |\mathbf{A}^{-1}|\mathcal{N}(\mathbf{A}^{-1}\mathbf{o}(t)-\mathbf{A}^{-1}\mathbf{b}|\mu,\Sigma), \quad (2.68)$$

the constrained Gaussian mean and covariance transform is equivalent to transform the feature vector by

$$\begin{aligned}\hat{\mathbf{o}}(t) =& \mathbf{A}^{-1}\mathbf{o}(t)-\mathbf{A}^{-1}\mathbf{b} \\ =& \mathbf{W}\begin{bmatrix}\mathbf{o}(t)\\ 1\end{bmatrix}.\end{aligned} \quad (2.69)$$

$\mathbf{W}$ can be acquired similarly using Eqn. (2.67) by replacing $\mathbf{o}(t)$, $\hat{\mu}_{im}$, and $\hat{\Sigma}_{im}$ with $\hat{\mathbf{o}}(t)$, μ_{im}, and Σ_{im}, respectively. A major benefit of applying this constrained MLLR (CMLLR) transform is to perform speaker adaptation without changing the model parameters (Gales, 1998). More details of CMLLR can be found in (Gales, 1998).

In addition to applying test-time speaker adaptation, it is also of a broad interest to apply adaptation in both training and testing (Anastasakos et al., 1996; Gales, 1998; Pye and Woodland, 1997). At test time, SD transforms are applied to a model set trained using the adaptation scheme instead of the SI model trained in the standard

way. Therefore, this scheme is referred to as speaker adaptive training (SAT). In this method, standard acoustic model parameters are estimated to model only the phonetically relevant variations since the speaker variations are assumed to have been modelled separately by the training set SD parameters. When applying SAT based on MLLR, given SD transforms, the ML estimation of μ_{im} and Σ_{im} is achieved by maximising Eqn. (2.67). This becomes very efficient for CMLLR since the re-estimation formulas are equivalent to the standard ML re-estimation formulas in Eqns. (2.38) and (2.39) with the transformed observation $\hat{\mathbf{o}}(t)$ (Gales, 1998).

2.6.2 Heteroscedastic linear discriminant analysis

Recall Section 2.3.2, in practice, it is often the case that GMMs with diagonal covariance matrices are used for acoustic modelling, which assumes that all dimensions of $\mathbf{o}_t$ are independent variables. A solution is to decorrelate the GMM input features with a linear Karhunen-Loève transform (KLT) (or its simplified approximation the DCT, as for MFCC), which is a fixed transform estimated by modelling all data samples with a single Gaussian distribution (Bishop, 2006). The KLT can find some *nuisance dimensions*, which are less important in data modelling and can be discarded for data compression purpose. However, for speech data, the nuisance dimensions found by knowing the distribution of each class (e.g., a phonetic unit) are often different from those found by KLT (Bishop, 2006). When each class is modelled by a Gaussian distribution with a shared covariance matrix, a linear decorrelation transform can be estimated by maximising the likelihood of generating the data. Equivalently, it can also be acquired by discriminating the classes by maximising the inter-class distances while minimising intra-class distances, and the nuisance dimensions are those useless for class discrimination. Therefore, this approach is named as linear discriminant analysis (LDA) (Bishop, 2006). A further improvement of LDA can be derived by allowing each Gaussian distribution to have a distinct covariance matrix, which results in the HLDA transform (Kumar, 1997).

Let the top d dimensions be the *valuable dimensions* supposed to contain the discriminative information and the other $D-d$ dimensions be the nuisance dimensions. Therefore, the transformed mean vector $\hat{\mu}^{(c)}$ and diagonal covariance matrix $\hat{\Sigma}^{(c)}$

associated with class c are

$$\hat{\mu}^{(c)} = \begin{bmatrix} \hat{\mu}^{(c)}_{[d]} \\ \hat{\mu}_{[D-d]} \end{bmatrix} \tag{2.70}$$

$$\begin{aligned} \hat{\Sigma}^{(c)} &= \begin{bmatrix} \hat{\Sigma}^{(c)}_{[d]} & \mathbf{0} \\ \mathbf{0} & \hat{\Sigma}^{(g)}_{[D-d]} \end{bmatrix} \\ &= \begin{bmatrix} \operatorname{diag}\left(\mathbf{A}_{[d]}\Sigma^{(c)}\mathbf{A}^{\mathrm{T}}_{[d]}\right) & \mathbf{0} \\ \mathbf{0} & \operatorname{diag}\left(\mathbf{A}_{[D-d]}\Sigma^{(g)}\mathbf{A}^{\mathrm{T}}_{[D-d]}\right) \end{bmatrix}, \end{aligned} \tag{2.71}$$

where $\mathcal{N}(\hat{\mathbf{o}}_{[d]}(t)|\hat{\mu}^{(c)}_{[d]}, \hat{\Sigma}^{(c)}_{[d]})$ and $\mathcal{N}(\hat{\mathbf{o}}_{[D-d]}(t)|\hat{\mu}^{(g)}_{[D-d]}, \hat{\Sigma}^{(g)}_{[D-d]})$ are the Gaussian distribution generating the samples of c in the valuable dimensions and a global Gaussian distribution generating the information in the nuisance dimensions. $\mathbf{A}$ is the HLDA transform and

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{[d]} \\ \mathbf{A}_{[D-d]} \end{bmatrix}. \tag{2.72}$$

The transformed observation vector $\hat{\mathbf{o}}(t)$ is

$$\hat{\mathbf{o}}(t) = \mathbf{A}\mathbf{o}(t), \tag{2.73}$$

and $\mathbf{A}$ can be estimated using the BW algorithm. By taking the equality Eqn. (2.61) into Eqn. (2.34), the $\mathcal{Q}$ is positively associated with

$$\frac{1}{2}\sum_t\sum_c \gamma_c(t) \ln\left(\frac{|\mathbf{A}|^2}{\left|\operatorname{diag}\left(\mathbf{A}_{[d]}\Sigma^{(c)}\mathbf{A}^{\mathrm{T}}_{[d]}\right)\right|\left|\operatorname{diag}\left(\mathbf{A}_{[D-d]}\Sigma^{(g)}\mathbf{A}^{\mathrm{T}}_{[D-d]}\right)\right|}\right) \tag{2.74}$$

and the full covariance matrices $\Sigma^{(c)}$ and $\Sigma^{(g)}$ are estimated with the samples associated with class c and all the samples using Eqn. (2.39). Further steps for calculating $\mathbf{A}$ are described in (Gales, 1998, 1999, 2002). In practice, the nuisance dimensions are often discarded by using $\mathbf{A}_{[d]}$ as the HLDA projection, which results in a *subspace* based on the valuable dimensions that can be more suitable for modelling.

2.6.3 Semi-tied covariance matrices

Since it is well known that the distributions of phonetic units are not Gaussian, HLDA still has unsatisfied assumptions. As alternative to using a fixed feature level linear

transform, features can also be decorrelated by transforming the GMM parameters as shown in Eqn. (2.68). This has an important advantage that one can use many transforms for decorrelation. STC is one of the model level approaches that associates an independent linear transform $\mathbf{A}^{(r)}$ to one of the R transform groups to decorrelate their covariance matrices (Gales, 1999). Transform groups can be acquired by partitioning the acoustic models in some way, for example, triphones with the same center phone, and STC can thus find a suitable subspace for each phonetic unit. Specifically,

$$\hat{\Sigma}^{(c,r)} = \text{diag}\left(\mathbf{A}^{(r)}\Sigma^{(c,r)}\mathbf{A}^{(r)^{\text{T}}}\right), \tag{2.75}$$

where $\Sigma^{(c,r)}$ is the full covariance matrix of the cth class in the rth group and $\hat{\Sigma}^{(c,r)}$ is the relevant diagonal matrix transformed by $\mathbf{A}^{(r)}$. Each $\mathbf{A}^{(r)}$ is separately estimated by maximising the likelihood of generating the samples of each class with their relevant models, which is similar to HLDA, and a class can be any Gaussian component from the GMMs in the same group. Therefore, by setting no nuisance dimension in Eqn. (2.74), the HLDA estimation technique can be applied to obtain each $\mathbf{A}^{(r)}$ independently. Complete STC estimation steps can be found in (Gales, 1999), which are designed to increase the likelihood. Moreover, if $R = 1$, the STC transform is a global one, and is equivalent to HLDA without nuisance dimensions. In this case, the inverse of the STC transform can be applied to the features by Eqn. (2.68).

2.7 Discriminative Sequence Training

From Section 2.3.3, training HMMs with the BW algorithm can find a local optimum for the acoustic model definition $p(\mathbf{O}|\mathbf{W})$, and indirectly maximise the MAP decision rule in Eqn. (2.3) given a fixed LM. As shown in (Brown, 1987), the ML method is actually optimal for ASR by assuming that:

- The true distribution family of $p(\mathbf{O}|\lambda)$ and the true LM are known;
- The training data set is sufficiently large;
- The system performance cannot get worse when its parameters get closer to the true ones.

Unfortunately, as the actual $p(\mathbf{O}|\lambda)$ distribution form is unknown, a mismatch is likely to happen with the chosen form (HMM in this thesis), not to mention the true LM. Meanwhile, a finite training data set is often known to be inadequate (Juang

et al., 1997). In practice, these broken assumptions cause ML trained ASR not necessarily to have optimal performance. Intuitively, as the ML method learns the data generation process with a supposed distribution form, the resulting parameters are not guaranteed to be the most effective to discriminate the subword units for recognition. An alternative solution is *discriminative training* that either models the posterior probabilities, or optimises a *discriminant function* that is directly associated with the decision rules (Bishop, 2006). By contrast, approaches that model the input or output data distributions are known as *generative models*, which include ML estimated HMMs (Bishop, 2006).

2.7.1 Maximum mutual information

Maximum mutual information (MMI) is a common discriminative training criterion that comes from *information theory* (MacKay, 2003; Shannon and Weaver, 1949). This was derived based on the concept of entropy. The entropy of a random event of communicating a word string is defined as

$$-\sum_{\mathbf{W}} P(\mathbf{W}) \ln P(\mathbf{W}), \tag{2.76}$$

which measures the amount of information that is missing before reception, or the *uncertainty* in that random event. For spoken word recognition, the averaged uncertainty in recovering a word string by perceiving an acoustic observation sequence is defined as *conditional entropy*,

$$-\sum_{\mathbf{W}} \int p(\mathbf{W}, \mathbf{O}) \ln P(\mathbf{W}|\mathbf{O})\, d\mathbf{O}. \tag{2.77}$$

In general, we would like to minimise the conditional uncertainty to construct a high performance ASR (Brown, 1987). Since the equation above can be re-organised as

$$-\sum_{\mathbf{W}} P(\mathbf{W}) \ln P(\mathbf{W}) - \sum_{\mathbf{W}} \int p(\mathbf{W}, \mathbf{O}) \ln \frac{p(\mathbf{W}, \mathbf{O})}{P(\mathbf{W}) p(\mathbf{O})}\, d\mathbf{O}, \tag{2.78}$$

and the first term is the entropy of the word string that is only related to the LM, optimising the conditional entropy by refining the acoustic model is to optimise the second term. In information theory, the negative of the second term is defined as the *mutual information*, which is a measure that quantifies the amount of information acquired about one random variable through another. Therefore, given a fixed LM,

the ASR with minimum conditional entropy is achieved when the mutual information is maximised. Mutual information can also be viewed as *Kullback-Leibler divergence* (KLD)(MacKay, 2003). That is,

$$\sum_{\mathbf{W}} \int p(\mathbf{W},\mathbf{O}) \ln \frac{p(\mathbf{W},\mathbf{O})}{P(\mathbf{W})p(\mathbf{O})} d\mathbf{O} = \mathbb{D}\left[p(\mathbf{W},\mathbf{O})||P(\mathbf{W})p(\mathbf{O})\right], \tag{2.79}$$

where $\mathbb{D}\left[p(\mathbf{W},\mathbf{O})||P(\mathbf{W})p(\mathbf{O})\right]$ is the KLD that measures the information lost when $P(\mathbf{W})p(\mathbf{O})$ is used to approximate $p(\mathbf{W},\mathbf{O})$, or in other words, the information loss caused by assuming $\mathbf{W}$ and $\mathbf{O}$ are independently generated.

Since $p(\mathbf{W},\mathbf{O})$ is unknown, calculating mutual information by definition is infeasible. To get rid of the joint probability, it is assumed that a particular pair of $\mathbf{W}$ and $\mathbf{O}$ is representative (Brown, 1987) and therefore, the above function can be simplified as

$$\ln \frac{p(\mathbf{W},\mathbf{O})}{P(\mathbf{W})p(\mathbf{O})} = \ln \frac{p(\mathbf{O}|\mathbf{W})}{\sum_{\mathbf{W}'} p(\mathbf{O}|\mathbf{W}')P(\mathbf{W}')}. \tag{2.80}$$

Again by assuming the LM is fixed, $P(\mathbf{W})$ is constant. Hence, by adding a constant term $\ln P(\mathbf{W})$, it is equivalent to maximising

$$\ln \frac{p(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{\sum_{\mathbf{W}'} p(\mathbf{O}|\mathbf{W}')P(\mathbf{W}')}. \tag{2.81}$$

Note by applying the Bayes' rule, this equals $\ln P(\mathbf{W}|\mathbf{O})$, which is also the objective function of the *conditional maximum likelihood* criterion that maximises the posterior probabilities (Nádas, 1983). Since $p(\mathbf{O}|\mathbf{W})$ is normally computed using embedded-unit training with a composite HMM $\lambda_{\mathbf{W}}$, we have $p(\mathbf{O}|\mathbf{W}) = p(\mathbf{O}|\lambda_{\mathbf{W}})$. Furthermore, as mentioned in Section 2.4.2, the LM log probability is normally scaled to match the wide range of log-likelihoods from the HMM acoustic model. Here, alternatively, the acoustic model log-likelihoods are scaled by an *acoustic scaling factor* κ, the inverse of the LM scaling factor, for the same purpose (Woodland and Povey, 2002). This is due to the fact that using acoustic scaling factors tends to increase the amount of confusable data in training and improve the generalisation ability by weighting several paths fairly similarly, while using LM scaling factors tends to make one path dominate the search process (Woodland and Povey, 2002). By integrating the above modifications, the most commonly seen form of the MMI objective function is (Woodland and Povey, 2002)

$$\mathcal{F}^{\text{MMI}}(\mathbf{O}) = \ln \frac{p(\mathbf{O}|\lambda_{\mathbf{W}})^{\kappa} P(\mathbf{W})}{\sum_{\mathbf{W}'} p(\mathbf{O}|\lambda_{\mathbf{W}'})^{\kappa} P(\mathbf{W}')}. \tag{2.82}$$

2.7.2 The extended Baum-Welch algorithm

For a discrete HMM parameter θ, the re-estimation formulae obtained by the BW algorithm is in the form of

$$\hat{\theta} = \frac{\partial \mathcal{F}^{\mathrm{ML}}(\mathbf{O})/\partial \ln \theta}{\sum_{\theta'} \partial \mathcal{F}^{\mathrm{ML}}(\mathbf{O})/\partial \ln \theta'}. \tag{2.83}$$

For example, when $\theta = a_{ij}$, the re-estimation formulae is defined in Eqn. (2.36). Analogous to the BW algorithm, it has been proven that a parameter of a discrete density HMM θ can be re-estimated using this formulae

$$\hat{\theta} = \frac{\partial \mathcal{F}^{\mathrm{MMI}}(\mathbf{O})/\partial \ln \theta + D\theta}{\sum_{\theta'} \left(\partial \mathcal{F}^{\mathrm{MMI}}(\mathbf{O})/\partial \ln \theta' + D\theta'\right)}, \tag{2.84}$$

where D is a sufficiently large constant to guarantee the training convergence (Gopalakrishnan et al., 1991). In order to handle continuous density HMMs, Normandin (1991) proposed to use discrete output probabilities to arbitrarily approximate Gaussian density functions, which results in the same re-estimation formulae as Eqn. (2.84).

To investigate the derivatives of $\mathcal{F}^{\mathrm{MMI}}(\mathbf{O})$ with respect to $\ln\theta$, it can be written as

$$\begin{aligned}
\frac{\partial \mathcal{F}^{\mathrm{MMI}}(\mathbf{O})}{\partial \ln \theta} =& \kappa \frac{\ln p(\mathbf{O}|\lambda_{\mathbf{W}})}{\partial \ln \theta} - \kappa \sum_{\mathbf{W}'} \frac{p(\mathbf{O}|\lambda_{\mathbf{W}'})^{\kappa} P(\mathbf{W}')}{\sum_{\mathbf{W}''} p(\mathbf{O}|M_{\mathbf{W}''})^{\kappa} P(\mathbf{W}'')} \frac{\partial \ln p(\mathbf{O}|\lambda_{\mathbf{W}'})}{\partial \ln \theta} \\
=& \kappa \frac{\sum_{\mathbf{W}' \neq \mathbf{W}} p(\mathbf{O}|\lambda_{\mathbf{W}'})^{\kappa} P(\mathbf{W}')}{\sum_{\mathbf{W}'} p(\mathbf{O}|\lambda_{\mathbf{W}'})^{\kappa} P(\mathbf{W}')} \frac{\ln p(\mathbf{O}|\lambda_{\mathbf{W}})}{\partial \ln \theta} - \\
& \kappa \sum_{\mathbf{W}' \neq \mathbf{W}} \frac{p(\mathbf{O}|\lambda_{\mathbf{W}'})^{\kappa} P(\mathbf{W}')}{\sum_{\mathbf{W}''} p(\mathbf{O}|\lambda_{\mathbf{W}''})^{\kappa} P(\mathbf{W}'')} \frac{\partial \ln p(\mathbf{O}|\lambda_{\mathbf{W}'})}{\partial \ln \theta}.
\end{aligned} \tag{2.85}$$

$\partial \ln p(\mathbf{O}|\lambda_{\mathbf{W}})/\partial \ln \theta$ is the derivative of $\mathcal{F}^{\mathrm{ML}}(\mathbf{O})$ with respect to $\ln\theta$, and the difference between directions of the MMI and ML derivatives actually lies in the second term of the equation above. This reveals a fundamental difference between MMI and ML that MMI training not only tries to increase the likelihood of the reference $\mathbf{W}$, but also tries to decrease the likelihood of every hypothesis $\mathbf{W}'$ (Brown, 1987). In particular,

for $\partial \mathcal{F}^{\text{MMI}}(\mathbf{O})/\partial \ln b_i(\mathbf{o}(t))$, there is

$$\begin{aligned}\frac{\partial \mathcal{F}^{\text{MMI}}(\mathbf{O})}{\partial \ln b_i(\mathbf{o}(t))} =& \kappa \gamma_{i|\lambda_{\mathbf{W}}}(t) - \kappa \sum_{\mathbf{W}'} \frac{p(\mathbf{O}|\lambda_{\mathbf{W}'})^{\kappa} P(\mathbf{W}')}{\sum_{\mathbf{W}''} p(\mathbf{O}|M_{\mathbf{W}''})^{\kappa} P(\mathbf{W}'')} \gamma_{i|\lambda_{\mathbf{W}'}}(t) \\ =& \kappa \gamma_i^{\text{num}}(t) - \kappa \gamma_i^{\text{den}}(t). \end{aligned} \tag{2.86}$$

It utilises the fact that

$$\frac{\mathcal{F}^{\text{ML}}(\mathbf{O})}{\partial \ln b_i(\mathbf{o}(t))} = \gamma_i(t), \tag{2.87}$$

which is because

$$\begin{aligned}\frac{\partial \ln p(\mathbf{O}|\lambda)}{\partial \ln p(\mathbf{o}(t)|q_t = i, \lambda)} &= \frac{1}{p(\mathbf{O}|\lambda)} \frac{\partial \sum_{j=1}^{N} p(\mathbf{O}, q_t = j|\lambda)}{\partial \ln p(\mathbf{o}(t)|q_t = i, \lambda)} \\ &= \frac{p(\mathbf{O}, q_t = i|\lambda)}{p(\mathbf{O}|\lambda)} \frac{\partial \ln p(\mathbf{O}, q_t = i|\lambda)}{\partial \ln p(\mathbf{o}(t)|q_t = i, \lambda)} \\ &= p(q_t = i|\mathbf{O}, \lambda). \end{aligned} \tag{2.88}$$

In Eqn. (2.86) we denote the first and second terms as $\gamma_i^{\text{num}}(t)$ and $\gamma_i^{\text{den}}(t)$, as they are derived separately from the numerator and denominator of the MMI objective function. The Gaussian specific posterior probabilities $\gamma_{im}^{\text{num}}(t)$ and $\gamma_{im}^{\text{den}}(t)$ are still obtained using Eqn. (2.32). For LVCSR, as in the decoding process, it is computationally expensive to calculate the exact $\gamma_i^{\text{den}}(t)$ due to the number of hypotheses. Therefore, a pruned lattice is normally used to represent the likely alternatives as an approximation to the hypothesis space. Such lattices are referred to as *denominator lattices*. To make Eqns. (2.86) and (2.87) more similar in form, an *MMI state occupancy* $\gamma_i^{\text{MMI}}(t)$ is defined as

$$\gamma_i^{\text{MMI}}(t) = \gamma_i^{\text{num}}(t) - \gamma_i^{\text{den}}(t). \tag{2.89}$$

Based on Eqns. (2.84) and (2.86), the estimations of the mean and covariance of an infinite number of MMI discrete output probabilities are

$$\hat{\mu}_{im} = \frac{\sum_{t=1}^{T} \left(\gamma_{im}^{\text{num}}(t) - \gamma_{im}^{\text{den}}(t)\right) \mathbf{o}(t) + D_{im}\mu_{im}}{\sum_{t=1}^{T} \left(\gamma_{im}^{\text{num}}(t) - \gamma_{im}^{\text{den}}(t)\right) + D_{im}} \tag{2.90}$$

$$\hat{\Sigma}_{im} = \frac{\sum_{t=1}^{T} \left(\gamma_{im}^{\text{num}}(t) - \gamma_{im}^{\text{den}}(t)\right) \mathbf{o}(t)\mathbf{o}(t)^{\text{T}} + D_{im}\mathbf{G}_{im}}{\sum_{t=1}^{T} \left(\gamma_{im}^{\text{num}}(t) - \gamma_{im}^{\text{den}}(t)\right) + D_{im}} - \hat{\mu}_{im}\hat{\mu}_{im}^{\text{T}}, \tag{2.91}$$

where

$$\mathbf{G}_{im} = \Sigma_{im} + \mu_{im}\mu_{im}^{\mathrm{T}}. \tag{2.92}$$

These equations are used as the re-estimation formulas in MMI training (Normandin, 1991). Comparing Eqns. (2.90) and (2.91) with Eqns. (2.38) and (2.39), we can see MMI re-estimation formulas differ from the ML re-estimation in two aspects: the use of $\gamma_{im}^{\mathrm{MMI}}(t)$ instead of $\gamma_{im}(t)$, and the existence of D_{im}. Actually if we only replace $\gamma_{im}(t)$ with $\gamma_{im}^{\mathrm{MMI}}(t)$ in Eqn. (2.39), the modified formulae can produce non-positive covariance values since $\sum_{t=1}^{T}\gamma_{im}^{\mathrm{MMI}}(t)$ contains $-\sum_{t=1}^{T}\gamma_{im}^{\mathrm{den}}(t)$. Therefore, the use of D_{im} should at least ensure positive covariances. That is,

$$D_{im} = \max\left\{E\sum_{t=1}^{T}\gamma_{im}^{\mathrm{den}}(t), 2D_{im}^{\min}\right\}, \tag{2.93}$$

where $D_{im}^{\min}$ is the minimum among all D_{im} that ensures the variances are positive, and E is a configurable global constant often set to 2 (Woodland and Povey, 2002). The reason of using Gaussian dependent constants D_{im} instead of a global constant D is to accelerate the learning process (Woodland and Povey, 2002), since D is found to control the learning rate that determines both convergence and learning efficiency (Normandin, 1991). When approximating a GMM more accurately with more discrete probabilities, D increases and makes training slow but stable. Actually when using infinite discrete probabilities to perfectly approximate a GMM, D becomes infinity, and therefore, it is impossible to use D in its convergence region (Normandin, 1991).

However, it has been found that Eqn. (2.84) will cause mixture weights to be extremely sensitive to small-valued parameters, and therefore, an alternative function is maximised to get the following re-estimation formulae (Povey and Woodland, 1999)

$$\hat{c}_{im} = \frac{\sum_{t=1}^{T}\gamma_{im}^{\mathrm{num}}(t) + k_{im}\,c_{im}}{\sum_{m'=1}\left(\sum_{t=1}^{T}\gamma_{im'}^{\mathrm{num}}(t) + k_{im'}\,c_{im'}\right)}, \tag{2.94}$$

where

$$k_{im} = \max_{m'}\left\{\frac{\sum_{t=1}^{T}\gamma_{im'}^{\mathrm{den}}(t)}{c_{im'}}\right\} - \frac{\sum_{t=1}^{T}\gamma_{im}^{\mathrm{den}}(t)}{c_{im}}. \tag{2.95}$$

As transition probabilities suffer from the same issue by using Eqn. (2.84), their re-estimation formulas are obtained similar to c_{im} by replacing $\gamma_{im}^{\text{num}}(t)$ and $\gamma_{im}^{\text{den}}(t)$ with $\xi_{im}^{\text{num}}(t)$ and $\xi_{im}^{\text{den}}(t)$ (Povey and Woodland, 1999).

2.7.3 Minimum phone error rate

Though MMI training maximises the posterior probability of the reference by increasing its discrimination ability from the hypotheses, it does not directly optimise the evaluation standard defined in Section 2.4.4. An alternative to MMI is minimum Bayes' risk (MBR) that allows to integrate a function associated with the evaluation standard explicitly into the objective function (Bishop, 2006; Doumpiotis and Byrne, 2004; Kaiser et al., 2002). Let $\text{Loss}(\mathbf{W}, \mathbf{W}')$ be the loss function that measures a hypothesis $\mathbf{W}'$ given the reference $\mathbf{W}$, and the Bayes' risk is defined as the expectation with respect to the joint distribution $p(\mathbf{W}, \mathbf{O})$ of the empirical loss of a particular utterance,

$$\begin{aligned} &\mathbb{E}_{\mathbf{W},\mathbf{O}}\left[\mathbb{E}_{\mathbf{W}'|\mathbf{O}}\left[\text{Loss}(\mathbf{W}, \mathbf{W}')\right]\right] \\ =&\int p(\mathbf{O}) \sum_{\mathbf{W}} P(\mathbf{W}|\mathbf{O}) \sum_{\mathbf{W}'} P(\mathbf{W}'|\mathbf{O}) \text{Loss}(\mathbf{W}, \mathbf{W}') \, \mathrm{d}\mathbf{O}, \end{aligned} \tag{2.96}$$

Since $p(\mathbf{W}, \mathbf{O})$ is unknown, similar to the MMI case, we assume a pair of $\mathbf{W}$ and $\mathbf{O}$ is representative to minimise the empirical loss $\mathbb{E}_{\mathbf{W}'|\mathbf{O}}\left[\text{Loss}(\mathbf{W}, \mathbf{W}')\right]$ instead. This results in the MBR criterion with the objective function defined as

$$\mathcal{F}^{\text{MBR}}(\mathbf{O}) = \frac{\sum_{\mathbf{W}'} p(\mathbf{O}|\mathbf{W}')^{\kappa} P(\mathbf{W}') \text{Loss}(\mathbf{W}, \mathbf{W}')}{\sum_{\mathbf{W}'} p(\mathbf{O}|\mathbf{W}')^{\kappa} P(\mathbf{W}')}. \tag{2.97}$$

In Eqn. (2.97), the choice of the loss function determines the effect of MBR training. As discussed before, we would like to use a function directly associated with the error rates. If the 0-1 loss function (defined in Eqn. (2.57)) is used, by assuming the reference $\mathbf{W}$ is encoded in the lattice, the $\text{Loss}(\mathbf{W}, \mathbf{W}')$ measures the sentence correctness, and $\mathcal{F}^{\text{MBR}}$ becomes $\mathcal{F}^{\text{MMI}}$. This reveals MMI may be more suitable for systems evaluated by SER. For LVCSR, to minimise WER, a word based loss function should be more desirable. However, in practice, phone and HMM state based loss functions are more commonly used due to data sparsity reasons. In this thesis, the minimum phone error

(MPE) criterion is used. The MPE loss is defined as a raw phone accuracy

$$\text{PhoneAcc}(\mathbf{W},\mathbf{W}^{'})=\sum_{q\in\mathbf{W}^{'}}\max_{h\in\mathbf{W}}\begin{cases}-1+2e(h,q) & \text{if } h \text{ and } q \text{ are the same}\\ -1+e(h,q) & \text{if } h \text{ and } q \text{ are different}\end{cases}, \tag{2.98}$$

where $e(h,q)$ returns the proportion of the length of a reference phone h that is overlapped with a hypothesis phone q (Povey, 2003; Povey and Woodland, 2002). This raw phone accuracy function is used since it removes the need for DP for accuracy calculations. Alternatively, better approximations to the Levenshtein edit distance can be used to replace the raw phone accuracy loss function (Povey, 2003).

Similar to MMI training, hypotheses are often presented as a denominator word lattice for efficiency reasons in MPE training. To calculate the raw phone accuracy, each word arc is replaced by a set of arcs associated with its constituent phones. This is often called a *phone marking* procedure that produces phone lattices (Woodland and Povey, 2002). Note that like word lattice generation, phone marking again preserves only the high likelihood paths found by the Viterbi search. To see how MPE works, $\partial\mathcal{F}^{\text{MPE}}(\mathbf{O})/\partial\ln\theta$ is investigated,

$$\begin{aligned}\frac{\mathcal{F}^{\text{MPE}}(\mathbf{O})}{\ln\theta}=&\kappa\sum_{\mathbf{W}'}\frac{p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})\text{PhoneAcc}(\mathbf{W},\mathbf{W}^{'})}{\sum_{\mathbf{W}''}p(\mathbf{O}|\mathbf{W}'')^{\kappa}P(\mathbf{W}'')}\frac{\partial\ln p(\mathbf{O}|\mathbf{W}^{'})}{\partial\ln\theta}-\\ &\kappa\frac{\sum_{\mathbf{W}'}p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})\text{PhoneAcc}(\mathbf{W},\mathbf{W}^{'})}{\sum_{\mathbf{W}'}p(\mathbf{O}|\mathbf{W}')^{\kappa}P(\mathbf{W}')}\sum_{\mathbf{W}'}\frac{p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})}{\sum_{\mathbf{W}''}p(\mathbf{O}|\mathbf{W}'')^{\kappa}P(\mathbf{W}'')}\frac{\partial\ln p(\mathbf{O}|\mathbf{W}^{'})}{\partial\ln\theta}.\end{aligned} \tag{2.99}$$

As in Section 2.4.3, let $p(q)$ be the likelihood of arc q, and $\partial\ln p(\mathbf{O}|\mathbf{W})/\partial\ln p(q)$ equals to one instead of zero only if $\mathbf{W}$ passes through q. Denote each hypothesis including q as $\mathbf{W}_q$, and therefore,

$$\begin{aligned}\frac{\mathcal{F}^{\text{MPE}}}{\ln p(q)}=&\kappa\frac{\sum_{\mathbf{W}_q'}p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})}{\sum_{\mathbf{W}'}p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})}\left(\frac{\sum_{\mathbf{W}_q'}p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})\text{PhoneAcc}(\mathbf{W},\mathbf{W}^{'})}{\sum_{\mathbf{W}_q'}p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})}\right.\\ &\left.-\frac{\sum_{\mathbf{W}'}p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})\text{PhoneAcc}(\mathbf{W},\mathbf{W}^{'})}{\sum_{\mathbf{W}'}p(\mathbf{O}|\mathbf{W}^{'})^{\kappa}P(\mathbf{W}^{'})}\right)\\ =&\kappa\gamma_q(c(q)-c_{\text{avg}}),\end{aligned} \tag{2.100}$$

where γ_q is the occupancy passing through arc q as defined in Section 2.4.3; c_{avg} and $c(q)$ are the weighted average correctness of all hypotheses and the hypotheses including arc q. c_{avg} and $c(q)$ can be calculated using a procedure similar to the lattice FB

algorithm (Povey, 2003; Povey and Woodland, 2002). Then $\partial \mathcal{F}^{\text{MPE}}(\mathbf{O})/\partial \ln b_i(\mathbf{o}(t))$ can be acquired by the *chain rule* as

$$\begin{aligned}\frac{\partial \mathcal{F}^{\text{MPE}}(\mathbf{O})}{\partial \ln b_i(\mathbf{o}(t))} =& \frac{\partial \mathcal{F}^{\text{MPE}}(\mathbf{O})}{\partial \ln p(q)} \frac{\partial \ln p(q)}{\partial \ln b_i(\mathbf{o}(t))} \\ =& \kappa \gamma_q (c(q) - c_{\text{avg}}) \gamma_i(t) \\ =& \kappa \gamma_i^{\text{MPE}}(t). \end{aligned} \tag{2.101}$$

Note that $\gamma_i^{\text{MPE}}(t)$ is not really a posterior probability. It is defined like this only to make $\partial \mathcal{F}^{\text{MPE}}(\mathbf{O})/\partial \ln b_i(\mathbf{o}(t))$ more consistent in form with the $\gamma^{\text{MMI}}(t)$ defined in Eqn. (2.85), and the extended Baum-Welch (EBW) algorithm can be applied to MPE training simply by replacing $\gamma^{\text{MMI}}(t)$ with $\gamma^{\text{MPE}}(t)$. A further modification is to set

$$\begin{cases} \gamma_i^{\text{num}}(t) = \gamma_i^{\text{MPE}}(t) & \text{and} \quad \gamma_i^{\text{den}}(t) = 0 & \text{if} \quad \gamma_i^{\text{MPE}}(t) > 0 \\ \gamma_i^{\text{num}}(t) = 0 & \text{and} \quad \gamma_i^{\text{den}}(t) = -\gamma_i^{\text{MPE}}(t) & \text{otherwise} \end{cases}. \tag{2.102}$$

This not only guarantees $\gamma_i^{\text{num}}(t) \geqslant 0$ and $\gamma_i^{\text{den}}(t) \geqslant 0$, like the MMI numerator and denominator occupancies, but also allows the EBW re-estimation formulae, Eqns. (2.90) – (2.94), to be applied to MPE training without modifications.

In addition to MMI and MBR, there are other common discriminative training approaches for HMMs. Minimum classification error (MCE) employs a differentiable approximation to the 0-1 loss function as the discriminant function to minimise SER (Juang et al., 1997). Large margin GMMs maximise the distance between the speech samples and the decision boundaries dividing the feature space into different classes (Sha and Saul, 2006), which minimises the *structural risk* instead of the empirical risk by MMI and MBR (Vapnik, 1998a). There are also other HMM large margin training methods including modifications to MCE, MMI, and MBR (Heigold et al., 2008; Li et al., 2006; Povey et al., 2008; Yu et al., 2006).

2.7.4 I-smoothing and percentile based variance floor

It has been found that MPE with the EBW algorithm for GMM-HMMs suffers from a severe *over-fitting* issue (Povey, 2003; Povey and Woodland, 2002), that is, MPE training can reduce the errors when recognising the training set but not those when recognising the unseen test set. This generalisation issue also exists in other MBR GMM-HMM training (Gibson and Hain, 2006; Povey and Kingsbury, 2007).

I-smoothing is a method for reducing GMM-HMM over-fitting issues, the functions by applying a data dependent interpolation between the discriminative criterion and the ML criterion (Povey and Woodland, 2002). It takes the data availability of each Gaussian component into account with a component dependent coefficient

$$\tau_{im}^{\mathrm{ML}}(t) = \frac{\tau^{\mathrm{ML}}}{\gamma_{im}(t)}. \tag{2.103}$$

The smoothed objective function can be written as

$$\mathcal{F}^{\mathrm{MPE}}(\mathbf{O}) + \tau_{im}^{\mathrm{ML}}(\mathbf{O})\,\mathcal{F}^{\mathrm{ML}}(\mathbf{O}). \tag{2.104}$$

Taking Eqn. (2.104) into Eqn. (2.84) and seeing $\tau_{im}^{\mathrm{ML}}(\mathbf{O})$ as a constant when differentiated, we get

$$\hat{\mu}_{im} = \frac{\sum_{t=1}^{T}\left(\gamma_{im}^{\mathrm{num}}(t) - \gamma_{im}^{\mathrm{den}}(t)\right)\mathbf{o}(t) + D_{im}\mu_{im} + \tau^{\mathrm{ML}}\,\mu_{im}^{\mathrm{prior}}}{\sum_{t=1}^{T}\left(\gamma_{im}^{\mathrm{num}}(t) - \gamma_{im}^{\mathrm{den}}(t)\right) + D_{im} + \tau^{\mathrm{ML}}} \tag{2.105}$$

$$\hat{\Sigma}_{im} = \frac{\sum_{t=1}^{T}\left(\gamma_{im}^{\mathrm{num}}(t) - \gamma_{im}^{\mathrm{den}}(t)\right)\mathbf{o}(t)\mathbf{o}(t)^{\mathrm{T}} + D_{im}\mathbf{G}_{im} + \tau^{\mathrm{ML}}\,\mathbf{G}_{im}^{\mathrm{prior}}}{\sum_{t=1}^{T}\left(\gamma_{im}^{\mathrm{num}}(t) - \gamma_{im}^{\mathrm{den}}(t)\right) + D_{im} + \tau^{\mathrm{ML}}} - \hat{\mu}_{im}\hat{\mu}_{im}^{\mathrm{T}}, \tag{2.106}$$

where

$$\mathbf{G}_{im}^{\mathrm{prior}} = \Sigma_{im}^{\mathrm{prior}} + \mu_{im}^{\mathrm{prior}}\,\mu_{im}^{\mathrm{prior}\,\mathrm{T}}, \tag{2.107}$$

and $\mu_{im}^{\mathrm{prior}}$ and $\Sigma_{im}^{\mathrm{prior}}$ are the prior distribution parameters estimated by ML using the current parameter set. The term prior is used since Eqn. (2.105) can be viewed using the MAP adaptation principle (Gauvain and Lee, 1994) and the dynamic ML distribution serves as the same role as the prior distribution in MAP. Similarly, for mixture weights and transition probabilities, Eqn. (2.94) is modified as

$$\hat{c}_{im} = \frac{\sum_{t=1}^{T}\gamma_{im}^{\mathrm{num}}(t) + k_{im}\,c_{im} + \tau^{\mathrm{ML}}\,c_{im}^{\mathrm{prior}}}{\sum_{m'=1}\left(\sum_{t=1}^{T}\gamma_{im}^{\mathrm{num}}(t) + k_{im}\,c_{im} + \tau^{\mathrm{ML}}\,c_{im}^{\mathrm{prior}}\right)}, \tag{2.108}$$

and τ^{ML} for Eqn. (2.108) is normally set to a different value from those for Eqns. (2.105) and (2.106), since they are derived differently as shown in Section 2.7.2. If MMI instead of ML is used to estimate the prior distributions, the technique is further referred to as the *dynamic MMI prior*, and is found to outperform the ML prior. Furthermore,

since I-smoothing is found to improve MMI (Povey, 2003), I-smoothing with an ML prior is often included in the dynamic MMI prior.

With I-smoothing and the dynamic MMI prior, MPE GMM-HMM training can generalise well on unseen test data. However, the WER does not always consistently reduce after each training epoch. Empirically, the use of a variance floor is beneficial to stabilise training, and in particular, the use of *percentile based variance floor* is useful (Povey, 2003; Young et al., 2015). It floors variances smaller than $\sigma_d^2(p\%)$ after every parameter update, where $\sigma_d^2(p\%)$ is the value ranked at $p\%$ among all variances of each dimension d. This requires ranking over all N variance values of each dimension with a complexity of $\mathcal{O}(N \log N)$.

2.8 LVCSR Acoustic Model Construction

This section briefly describes the CUED standard GMM-HMM acoustic model construction procedure as well as some related methods. CD modelling, discriminative training, and speaker adaptation are all involved.

2.8.1 HTK LVCSR silence modelling

HTK LVCSR silence models comprise two HMMs representing silence (denoted as sil) and short pause (referred to as sp) units separately. Since there are usually more silence samples in the training data, sil HMM has three states with each of them containing twice as many Gaussian components as those of other HMMs. The sil HMM allows transitions between any two states since some silence segments may be longer than normal and contain repeated fragments. Since the first or last state is the only entry or exit state, a minimum silence duration of two frames is guaranteed.

The sp HMM has similar transition probability constraints to the sil HMM, except for allowing the whole model to be skipped since short pause is not guaranteed to appear between words. The sp HMM states are tied to corresponding sil HMM states, and are not involved in context dependent unit modification. This silence model configuration is shown in Figure 2.4.

2.8.2 Embedded-unit training

When marking the triphone units in continuous speech, it is often observed that many triphones do not have clear boundaries. Therefore, rather than pre-segmenting the training utterances into triphone segments (as in Section 2.3.3), it is more sensible

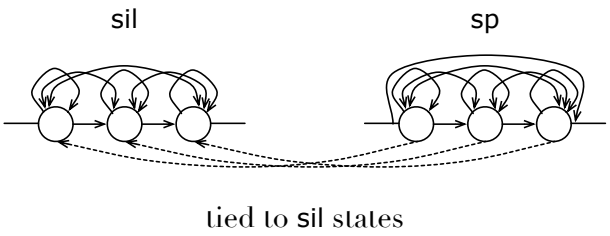


Figure 2.4 HTK LVCSR silence model structure.

to allow the segments to be probabilistic defined, which leads to the *embedded-unit training* approach (Young et al., 2015). The key idea is to synthesise a composite HMM for each training reference by concatenating the instances of the subword HMMs spanning the whole utterance according to the label sequence. Afterwards, the FB algorithm can be applied to the composite HMM to find the state occupancies for every frame, which provides "soft" boundaries for each speech segment.

2.8.3 Flat start initialisation

Besides the training method, another factor worth taking into account for acoustic model construction is the initialisation strategy. For different HMMs, their initial parameters can be either estimated separately using pre-segmented data, or copied from an initial HMM estimated with all data. The second strategy is termed as a *flat start*, which is equivalent to uniformly segment each utterance. Embedded-unit training can be applied to train flat start models by performing the FB algorithm on the composite HMMs to realign the data. Schematically, adopting a flat start implicitly assumes that many frames are reasonably aligned even with uniform segmentation, and therefore, the resulting models can generate better alignments to improve the following iteration of estimation with embedded-unit training. Such a procedure is repeated iteratively and can gradually refine the alignments. Empirically, a flat start is found to work no worse than more careful initialisation strategies.

2.8.4 GMM-HMM system construction

With the aforementioned methods, triphone GMM-HMMs can be constructed based on the following procedure, which is used in the standard CUED and HTK conventions (Evermann et al., 2005; Gales et al., 2006; Hain et al., 1999; Woodland, 2002; Woodland et al., 1997, 1995, 1994).

- Build a monophone GMM-HMM system with flat a start and embedded-unit training. An HMM is constructed for every monophone in the phone set including sil. Once the monophone system is trained, a new HMM is built for sp by sharing the sil HMM states.

- Constructing an untied triphone HMM system whose parameters are copied from single Gaussian monophone GMM-HMMs corresponding to their center phones. After re-estimation, decision tree based state tying is performed to cluster the triphone states. The generated models are then refined by adding more Gaussian components and re-estimating the parameters.

- The final ML tied state triphone system is trained using a *two-model re-estimation* method (Gales, 1995), which basically follows the same construction procedure as the initial triphone system except for using the FB "alignments" from the well-trained initial triphone system for decision tree clustering. Two-model re-estimation is found to produce a better tied state set.

- For PLP and MFCC, static coefficients with their first and second order derivatives are normally used in GMM-HMM system construction. ML GMM-HMMs are first extended to include additional dimensions from the third order derivatives and then projected back to the original number of dimensions using HLDA.

- SAT can be optionally applied to the ML system at this stage. CMLLRs for all training set speakers are estimated, followed by ML re-estimation based on the CMLLR normalised features. These steps are executed repeatedly for multiple iterations, and an additional set of CMLLRs is trained for the final ML models.

- Discriminative training is often applied as the last stage based on the lattices generated by the final ML GMM-HMM system. For MPE, phone marking is carried out once the word lattices are produced. It has been found that lattice regeneration is not important with a sufficient lattice density, and therefore, the same lattices are used by all training iterations.

Chapter 3

Artificial Neural Networks for Speech Recognition

3.1 An ANN Model

An artificial neural network (ANN) is a powerful model for both classification and regression tasks. The name originates from early attempts to find mathematical representations for biological information processing procedures (Bishop, 1995). There is a long term interest in using ANNs in ASR (Bourlard and Morgan, 1993; Robinson, 1989; Robinson and Fallside, 1987; Waibel et al., 1989), perhaps because it is a bio-inspired model for learning general purpose data mapping functions that makes no prior assumptions on the input features. This section briefly introduces some basics about ANNs.

In general, an ANN comprises a sequence of layers serving as nonlinear transformations to map an input vector $\mathbf{x}^{\text{in}}(t)$ to an output vector $\mathbf{y}^{\text{out}}(t)$ at time t. There is no generally accepted single definition of a layer, but it commonly contains a fixed number of artificial neurons. An artificial neuron j in layer l transforms its input value $a_j^{(l)}(t)$ to the jth dimension of the output vector, $y_j^{(l)}(t)$, through a nonlinear function $f(\cdot)$, i.e.,

$$y_j^{(l)}(t) = f(a_j^{(l)}(t)). \tag{3.1}$$

$a_j^{(l)}(t)$ and $f(\cdot)$ are called the *activation* and its *activation function* respectively. $a_j^{(l)}(t)$ is calculated as a linear combination of I_l basis functions, that is,

$$a_j^{(l)}(t) = \sum_{i=1}^{I_l} w_{ji}^{(l)} \phi_i^{(l)}(t) + b_j^{(l)}, \tag{3.2}$$

where $w_{ji}^{(l)}$ and $b_j^{(l)}$ are the learnable weights and biases associated with j, and $\phi_i^{(l)}(t)$ are the parametric basis functions representing the input to the layer. Figure 3.1a depicts the above-mentioned operations and the corresponding artificial neuron. $\phi_i^{(l)}(t)$ can either return an input value,

$$\phi_i^{(l)}(t) = x_d^{\text{in}}(t), \tag{3.3}$$

where $x_d^{\text{in}}(t)$ is the dth dimension of $\mathbf{x}^{\text{in}}(t)$, or return the output value from the dth artificial neuron of layer k at time $t^{'}$,

$$\phi_i^{(l)}(t) = y_d^{(k)}(t^{'}). \tag{3.4}$$

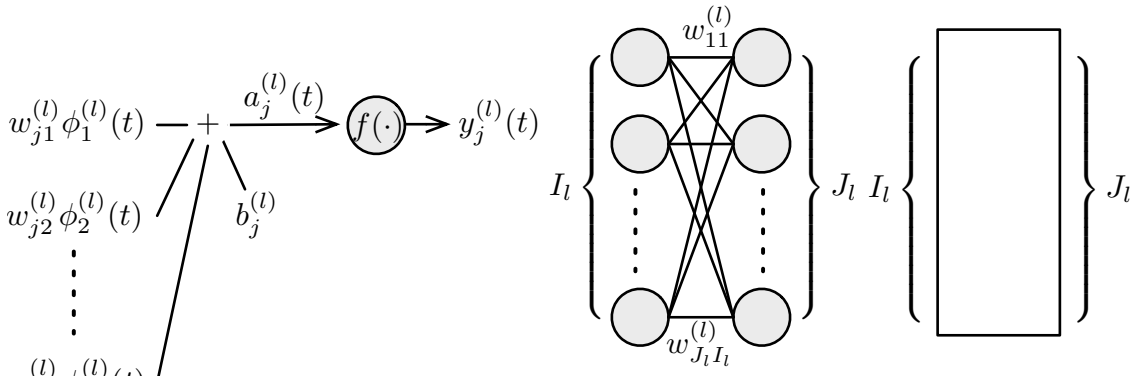
The layer producing $\mathbf{y}^{\text{out}}(t)$ is often called the *output layer* or *final layer*[1], and the rest layers are *hidden layers*. If the input vector to a layer contains only the elements from $\mathbf{x}^{\text{in}}(t)$, it is an *input layer*. It is obvious that the layers are connected through $\phi_i^{(l)}(t)$ that determines the ANN structure. In this section, only the most commonly seen ANN structure is considered, which is a chain where the output of each layer is only used as the input to its immediate succeeding layer, i.e., k equals to $l-1$ in Eqn. (3.4). In this chain structure, the first layer is the input layer, and the other ones are hidden layers except for the last one, which is the output layer.

If the layer has J_l artificial neurons, Eqns. (3.1) and (3.2) can be presented with matrices and vectors as

$$\mathbf{y}^{(l)}(t) = f\left(\mathbf{W}^{(l)}\mathbf{x}^{(l)}(t) + \mathbf{b}^{(l)}\right), \tag{3.5}$$

where $\mathbf{x}^{(l)}(t)$ and $\mathbf{y}^{(l)}(t)$ are I_l-dimensional input and J_l-dimensional output vectors; $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are a $J_l \times I_l$-dimensional weight matrix and a J_l-dimensional bias vector. In the literature, layers are often labelled inconsistently that can cause confusion, while in this thesis the layer count is defined as the weight matrix count. With this convention,

[1]Note that each ANN is assumed to have only one output layer since multiple output layers can be combined by using an extra layer with appropriate identity linear transforms.



(a) An artificial neuron j of layer l. (b) Artificial neurons related to layer l. (c) An ANN layer l.

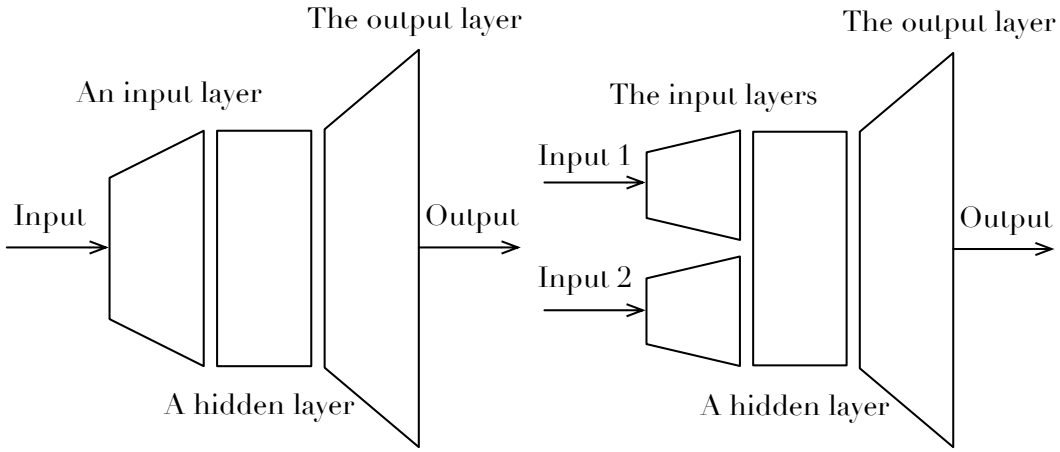
Figure 3.1 An ANN layer and artificial neurons.

a layer has two sets of artificial neurons separately shared with its immediate preceding and succeeding layers, as shown in Figure 3.1b. In Figure 3.1c, a block is used to represent a layer as a simplified depiction, whose left and right vertical edges are the input and output of the layer. Based on this convention, a 3-layer chain structured ANN model is shown in Figure 3.2a.

3.2 ANNs with Flexible Structures

In addition to the chain structure discussed in Section 3.1, an ANN model can have other structures by allowing k to be any valid layer index in Eqn. 3.4, which can be useful in learning more complicated feature transforms. For example, in Figure 3.2b, the input values to the hidden layer are from two input layers, which is useful in combining different input features. In general, an ANN model structure can be presented as a directed graph, with each graph node representing a layer and a directed arc from k to l representing $\phi_i^{(l)}(t) = y_d^{(k)}(t')$. Therefore, the input and output layers become the entry and exit graph nodes, and mapping $\mathbf{x}^{\text{in}}(t)$ to $\mathbf{y}^{\text{out}}(t)$ requires traversing all graph nodes from the entry to the exit, given that a node can be visited only when all of its immediate preceding nodes (the source nodes of the inbound arcs) have been visited (Diestel, 1997). ANN models can be categorised according to the graph types.

- If the graph is a DCG, it contains at least a cycle whose ending node relies on the output values from a succeeding node, which has been generated at a different time t'. These models are also known as recurrent ANNs. When the cycle is a self-loop with $t - t' = 1$, the associated graph node represents a *recurrent layer*



(a) A 3-layer chain structured ANN. (b) An ANN with a non-chain structure.

Figure 3.2 Examplar ANN models with different structures.

and the model is named as a recurrent neural network (RNN) (Pineda, 1987; Robinson and Fallside, 1987; Rumelhart et al., 1986). Such a structure is also sufficient to represent a bidirectional recurrent neural network (BRNN) (Schuster and Paliwal, 1997), which combines a standard RNN directed cycle and another cycle in the reverse time order.

- If the graph is a DAG, the graph traversal flows forward from the entry to the exit, since there is no node relying on its succeeding nodes. Therefore, such a model is called a feedforward neural network (FNN) (Bishop, 1995; Rosenblatt, 1961), and mapping its input to output is often termed as a *forward propagation*. The chain structured ANN in Figure 3.2a is an FNN.

$\mathbf{x}^{(l)}(t)$ consists of the returned values of $\phi_i^{(l)}(t)$, and a *feature mixture*, which comprises a number of *feature elements* representing fragments of $\mathbf{x}^{(l)}(t)$ from the same source (input features and specific layers), is used to represent it. That is,

$$\mathbf{x}^{(l)}(t) = \mathbf{x}_{e_1}^{(l)}(t), \mathbf{x}_{e_2}^{(l)}(t), \ldots, \mathbf{x}_{e_{E_l}}^{(l)}(t), \tag{3.6}$$

where $\mathbf{x}_{e_i}^{(l)}(t)$ is the fragment produced by the ith feature element e_i, and E_l is the feature element number in l. Let k be the source layer of e_i, whose output vector is stacked according to a *context shift set* $\mathbf{c}_{e_i}$, which contains C_i integers indicating the temporal context,

$$\mathbf{c}_{e_i} = \{c_1, c_2, \ldots, c_{C_i}\}. \tag{3.7}$$

$\mathbf{x}_{e_i}^{(l)}(t)$ is

$$\mathbf{x}_{e_i}^{(l)}(t) = \mathbf{y}^{(k)}(t+c_1), \mathbf{y}^{(k)}(t+c_2), \ldots, \mathbf{y}^{(k)}(t+c_{C_i}), \tag{3.8}$$

and l is an *immediate succeeding layer* of layer k. It is worth noting that different immediate succeeding layers l may require the same $\mathbf{y}^{(k)}(t+c)$, and we can save such redundant calculation by producing outputs based on a minimum context shift set $\mathbf{c}_{\min}^{(k)}$, which can be found through a backward graph traversal procedure from the exit to the entry as described in Algorithm 1.

Algorithm 1 Find layer specific minimum context shift sets

```
1:  procedure FindMinContexts()
2:      c_min^out ← {0}
3:      for each hidden layer l do
4:          c_min^(l) ← φ
5:      l ← final (output) layer
6:      while l ≠ 0 do
7:          for each input feature element e in l do
8:              if e is associated with a layer then
9:                  k ← the source layer of e
10:                 c_e ← the context shift set of e
11:                 c_min^(k) ← c_min^(k) ∪ c_e
12:         l ← l − 1
        return {c_min^(1), c_min^(2), ..., c_min^(L)}
```

Another important factor to the ANN layer type is parameter tying. A *time delay neural network* (TDNN) employs FNN layers to initially transform narrow temporal contexts, which are tied across time steps to learn temporal invariance (Peddinti et al., 2015; Waibel et al., 1989). More layers are stacked upon the initial transform to operate on a wider temporal context with a different temporal resolution. In order to learn spatial invariance, a convolutional neural network (CNN) performs convolution using a set of learnable filters along spatial axes of the input feature maps (Abdel-Hamid et al., 2014; LeCun et al., 1998a; Sainath et al., 2013b; Tóth, 2014). For acoustic modelling purposes, the input feature maps are often 2-dimensional whose spatial axes are related to time and frequency. The filters can be spatially unfolded into a conventional weight matrix with tied submatrices. From this perspective, a TDNN is a CNN that shares weights along a single temporal dimension. Moreover, one can have the RNN history truncated to a maximum time step and unfold the recurrent layer through time to many FNN layers with their parameters tied together (Saon et al., 2014).

Recently, besides standard weights and biases, more parameters are introduced into recurrent ANNs to memorise more history information and manage hidden states, such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) and gated recurrent unit (GRU) models (Chung et al., 2014). In addition to these variations based on Eqn. (3.5), sometimes other types of models that serve as continuous feature transformations are regarded as an ANN layer as well (Bishop, 1994; Goodfellow et al., 2013; LeCun et al., 1998a; Wu et al., 2016b). For instance, when a set of GMMs is used as an ANN output layer, it is sometimes named a GMM layer (Variani et al., 2015) and the whole model is a Gaussian mixture density neural network (MDNN). The use of local average, local sampling, and logarithm function can also be viewed as layers without any adaptive parameters.

3.3 Probabilistic Interpretations

3.3.1 Output activation function

To use ANNs in ASR for either acoustic or language modelling, an ANN model needs to be able to assign its input samples correctly to the classes and to estimate the corresponding likelihoods or probabilities. Each output layer artificial neuron, k, is associated with a class C_k, and is often referred to as an output target. A common solution is to use the *softmax* function as the output layer activation function to normalise the output activations into class-conditional probabilities (Bishop, 1995; Bridle, 1990b; Jacobs et al., 1991), i.e.,

$$y_k^{\text{out}}(t) = \frac{\exp(a_k^{\text{out}}(t))}{\sum_{k'} \exp(a_{k'}^{\text{out}}(t))}. \tag{3.9}$$

The softmax function is so named since it acts as a smoothed version of the "argmax" function. Since $\sum_k y_k^{\text{out}}(t) = 1$, $y_k^{\text{out}}(t)$ can be interpreted as $P(C_k|\mathbf{x}(t))$, the posterior probability of $\mathbf{x}(t)$ being classified as C_k. Eqn. (3.9) can be interpreted as a set of probabilistic generative models,

$$P(C_k|\mathbf{x}(t)) = \frac{p(\mathbf{x}(t)|C_k)P(C_k)}{\sum_{k'} p(\mathbf{x}(t)|C_{k'})P(C_{k'})}, \tag{3.10}$$

and the output activation $a_k^{\text{out}}(t)$ can be viewed as

$$a_k^{\text{out}}(t) = \ln p(\mathbf{x}(t)|C_k) + \ln P(C_k) + E, \tag{3.11}$$

where E is a constant shared by all classes.

3.3.2 Cross entropy criterion

The above generative models are often estimated by minimising the cross entropy (CE) criterion (Hopfield, 1987). Let Λ be the model to estimate, the CE between the output probability distributions of Λ and the true distributions produced by Λ_0 is defined as (MacKay, 2003)

$$-\sum_k P(C_k|\mathbf{x}(t), \Lambda_0) \ln P(C_k|\mathbf{x}(t), \Lambda), \tag{3.12}$$

which measures the average of information amount needed to identify the feature vectors generated by Λ_0 using Λ.

Define $\mathbf{r}_{k_0}(t)$ as

$$\mathbf{r}_{k_0}(t) = \{P(C_1|\mathbf{x}(t), \Lambda_0), P(C_2|\mathbf{x}(t), \Lambda_0), \ldots, P(C_K|\mathbf{x}(t), \Lambda_0)\}, \tag{3.13}$$

where K is the total number of classes and $\mathbf{x}(t)$ belongs to C_{k_0}. An alternative perspective exists when a $1-of-K$ target coding scheme is used, which makes $\mathbf{r}_{k_0}(t)$ the binary class labels where $r_{k_0k}(t) = 1$ if $k = k_0$ and $r_{k_0k}(t) = 0$ otherwise. Assume each class label is independently generated by a *Bernoulli distribution* (Gardiner, 1985), then the conditional distribution of the targets is

$$P(\mathbf{r}_{k_0}(t)|\mathbf{x}(t), \Lambda) = \prod_k P(C_k|\mathbf{x}(t), \Lambda)^{r_{k_0k}(t)} \left(1 - P(C_k|\mathbf{x}(t), \Lambda)\right)^{1-r_{k_0k}(t)}, \tag{3.14}$$

and the CE becomes the negative log-likelihood, $-\ln P(\mathbf{r}_{k_0}(t)|\mathbf{x}_t, \Lambda)$ (Bishop, 2006). Minimising the CE is equivalent to maximising the likelihood of generating the class labels using their associated features and the estimated model set Λ.

In practice, the minimum CE value

$$-\sum_k P(C_k|\mathbf{x}(t), \Lambda_0) \ln P(C_k|\mathbf{x}(t), \Lambda_0) \tag{3.15}$$

is often subtracted from the objective function to make it have a fixed minimum value of zero, which occurs when $\Lambda = \Lambda_0$. The final CE objective function $\mathcal{F}^{\text{CE}}$ is defined as

$$\begin{aligned}\mathcal{F}^{\text{CE}}(t) &= -\sum_k P(C_k|\mathbf{x}(t),\Lambda_0)\ln\frac{P(C_k|\mathbf{x}(t),\Lambda)}{P(C_k|\mathbf{x}(t),\Lambda_0)} \\ &= -\sum_k r_{k_0k}(t)\ln\frac{y_k^{\text{out}}(t)}{r_{k_0k}(t)}.\end{aligned} \tag{3.16}$$

However, according to the Kullback-Leibler divergence (KLD) definition in Section 2.3.5, $\mathcal{F}^{\text{CE}}$ is actually the KLD that measures the difference between two probability distributions $P(C_k|\mathbf{x}(t),\Lambda_0)$ and $P(C_k|\mathbf{x}(t),\Lambda)$, i.e.,

$$\mathcal{F}^{\text{CE}}(t) = \mathbb{D}(P(C_k|\mathbf{x}(t),\Lambda_0)||P(C_k|\mathbf{x}(t),\Lambda)), \tag{3.17}$$

and the difference between CE and KLD, the minimum CE value, is the entropy of the true distribution defined in Section 2.7.1 (MacKay, 2003).

When training acoustic models based on CE, since the $1-\text{of}-K$ scheme is often used in target coding, the ANN model is often seen as a classifier rather than a regression model. In addition to CE, MCE and minimum mean squared error (MMSE) are also commonly seen ANN training criteria (Golik et al., 2013; Juang and Katagiri, 1992). Similar to MCE for GMM-HMMs, the MCE objective function uses the sigmoid function to approximate a 0-1 classification error so that it can be differentiated. The MMSE criterion, on the other hand, learns real valued target vectors for regression purposes[1].

3.3.3 Sigmoid hidden activation function

In addition to the configurations discussed above, another important factor in ANN model design is the choice of hidden layer activation functions. Traditionally, the most commonly seen hidden activation function is the *sigmoid* function defined as (Rumelhart et al., 1986)

$$y_j^{(l)}(t) = \frac{1}{1+\exp(-a_j^{(l)}(t))}. \tag{3.18}$$

The name "sigmoid" means "S-shaped" that depicts its curve shape as shown in Figure 3.3. It can be seen that the sigmoid is a "smoothed" version of the 0-1 loss

[1]MMSE is sometimes used to train classifier as well, such as in (Golik et al., 2013).

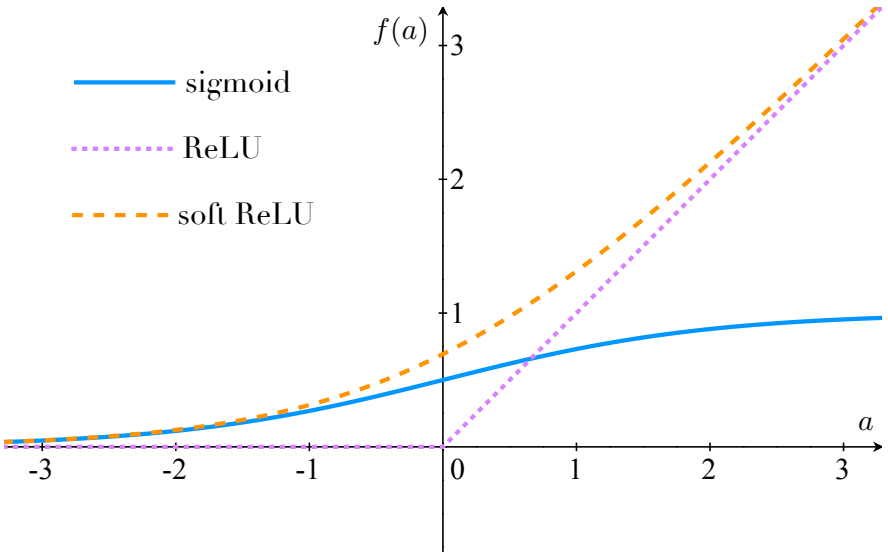


Figure 3.3 Sigmoid, ReLU, and soft ReLU functions.

function defined in Eqn. (2.57) that is differentiable to be learned through gradient based optimisation approaches. The function defined by Eqns. (3.1) and (3.2) with a linearly transformed 0-1 loss function is termed a *perceptron*[1]. Therefore, the chain structured FNN with $\mathbf{x}^{(1)}(t) = \mathbf{x}^{\text{in}}(t)$, $\mathbf{x}^{(l+1)}(t) = \mathbf{y}^{(l)}(t)$, and sigmoid hidden activation functions, is named a multi-layer perceptron (MLP)[2] (Minsky and Papert, 1969). It has been proven that any continuous mapping function can be represented arbitrarily accurately by a two layer MLP with enough hidden layer artificial neurons (Hornik et al., 1989). An MLP is illustrated in Figure 3.2a.

A biological interpretation of the sigmoid function relies on a *spike-rate coding* strategy that encodes the average spike rate of a single neuron within a time window (Ham and Kostanic, 2000). Let $a_j^{(l)}(t)$ be the accumulated stimulus intensity within a time step, then $f(a_j^{(l)}(t))$ is the cumulative spike rate. By assuming the spike rates follow *logistic distributions* with zero mean and $\pi^2/3$ variance, $f(\cdot)$ is sigmoid since it resembles the cumulative distribution function (CDF) of that distribution in shape (Gardiner, 1985). Alternatively, by viewing the activation as

$$\ln \frac{p(\mathbf{x}(t)|C_1)P(C_1)}{p(\mathbf{x}(t)|C_2)P(C_2)},$$

[1]For perceptron, $f(a)$ returns 1 if $a \geqslant 0$, and -1 otherwise, which equals to have the 0-1 loss multiplied by 2 and then subtracted by 1.

[2]Another commonly seen hidden activation function in MLP is the *hyperbolic tangent function*, which is equivalent to multiplying the sigmoid output by 2 and then subtracting by 1. These changes can be integrated into the weights and biases to generate an equivalent sigmoid MLP.

the sigmoid function becomes a two-class case of the softmax function,

$$P(C_1|\mathbf{x}(t)) = \frac{p(\mathbf{x}(t)|C_1)P(C_1)}{p(\mathbf{x}(t)|C_1)P(C_1) + p(\mathbf{x}(t)|C_2)P(C_2)}. \tag{3.19}$$

This interprets the sigmoid function output values as posterior probabilities.

Finally, an important property of the sigmoid can be seen by taking its derivatives with respect to the input activation

$$\begin{aligned}\frac{\partial y_j^{(l)}(t)}{\partial a_j^{(l)}(t)} =& \frac{\exp(-a_j^{(l)}(t))}{\left(1+\exp(-a_j^{(l)}(t))\right)^2} \\ =& y_j^{(l)}(t)\left(1-y_j^{(l)}(t)\right).\end{aligned} \tag{3.20}$$

Since $0 < y_j^{(l)}(t) < 1$, Eqn. (3.20) reveals that sigmoid can reduce the magnitude of the derivatives with respect to $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$, and $\mathbf{x}^{(l)}(t)$ by the chain rule.

3.3.4 ReLU hidden activation function

Recently, the rectified linear unit (ReLU) function,

$$y_j^{(l)}(t) = \begin{cases} a_j^{(l)}(t) & \text{if } a_j^{(l)}(t) \geqslant 0 \\ 0 & \text{otherwise} \end{cases}, \tag{3.21}$$

has become a widely used hidden activation function, which has a "hinge" like shape and keeps the positive part of the activations (Glorot et al., 2011; Hahnloser, 1998; Nair and Hinton, 2010; Salinas and Abbott, 1996). The ReLU activation function is also presented in Figure 3.3 to show a comparison with the sigmoid function. Instead of the class-conditional probabilities generated by the sigmoid, the ReLU function produces real values that can serve as approximations to the activation functions in some biological neurons (Dayan and Abott, 2001).

Taking the derivatives of the ReLU with respect to the input activation, we have

$$\frac{\partial y_j^{(l)}(t)}{\partial a_j^{(l)}(t)} = \begin{cases} 1 & \text{if } a_j^{(l)}(t) \geqslant 0 \\ 0 & \text{otherwise} \end{cases}. \tag{3.22}$$

Note that ReLU is not differentiable when $a_j^{(l)}(t) = 0$, and we set the derivative to one in Eqn. (3.22). In contrast to the sigmoid function, the ReLU function is shown to be

helpful to maintain the magnitudes of the derivatives with respect to the parameters in the layer. Furthermore, when the network learns to always produce a negative or zero activation as the input to a ReLU artificial neuron[1], its output $y_j^{(l)}(t)$ will always be zero, which learns a sparse representation of the data and is robust to small input changes (Glorot et al., 2011).

Finally, the *soft ReLU* function (Dugas et al., 2001; Glorot et al., 2011) is

$$y_j^{(l)}(t) = \ln\left(1+\exp(a_j^{(l)}(t))\right). \tag{3.23}$$

From Figure 3.3, it can be seen that the soft ReLU is actually a "smoothed" ReLU function that is differentiable when $a_j^{(l)}(t) = 0$. Taking the derivative of $y_j^{(l)}(t)$ with respect to $a_j^{(l)}(t)$, we have

$$\frac{\partial y_j^{(l)}(t)}{\partial a_j^{(l)}(t)} = \frac{1}{1+\exp(-a_j^{(l)}(t))}, \tag{3.24}$$

which is a sigmoid function.

3.4 Error Backpropagation

Since estimation of ANN parameters has no analytical solution, they are often learned through stochastic gradient descent (SGD), an iterative optimisation approach. In this section, the error backpropagation (EBP) algorithm is introduced for efficient gradient evaluation (Rumelhart et al., 1986; Werbos, 1974), which is modified to apply to ANNs with the generic structure defined in Section 3.2. The SGD method will be described in the next section.

When training an ANN model using gradient based approaches, the first order partial derivatives with respect to its parameters are required. For a layer defined by Eqn. (3.5), the partial derivatives of some objective function $\mathcal{F}$ are

$$\frac{\partial \mathcal{F}}{\partial w_{ji}^{(l)}} = \frac{\partial \mathcal{F}}{\partial a_j^{(l)}(t)} \frac{\partial a_j^{(l)}(t)}{\partial w_{ji}^{(l)}} \tag{3.25}$$

$$\frac{\partial \mathcal{F}}{\partial b_j^{(l)}} = \frac{\partial \mathcal{F}}{\partial a_j^{(l)}(t)} \frac{\partial a_j^{(l)}(t)}{\partial b_j^{(l)}}, \tag{3.26}$$

[1]This becomes true when, for example, a large negative bias is learned with the artificial neuron.

which rely on the partial derivatives with respect to the activations of the corresponding layer. These can be efficiently computed by the EBP algorithm. This section starts by explaining the terms *error* and *backpropagation* that give the algorithm its name.

Consider training a 2-layer MLP based on $\mathcal{F}^{\text{CE}}$. In the softmax output layer, the derivatives of $\mathcal{F}^{\text{CE}}$ with respect to the output of a target $k^{'}$ is

$$\begin{aligned}\frac{\partial \mathcal{F}^{\text{CE}}}{\partial y_{k'}^{\text{out}}(t)} =& -\frac{r_{k_0k'}(t)\ln y_{k'}^{\text{out}}(t) - r_{k_0k'}(t)\ln r_{k_0k'}(t)}{\partial y_{k'}^{\text{out}}(t)} \\ =& -\frac{r_{k_0k'}(t)}{y_{k'}^{\text{out}}(t)}.\end{aligned} \tag{3.27}$$

Now differentiating the softmax output of $k^{'}$ with respect to the input of k gives

$$\begin{aligned}\frac{y_{k'}^{\text{out}}(t)}{a_k^{\text{out}}(t)} =& \frac{1}{\sum_{k''}\exp(a_{k''}^{\text{out}}(t))}\frac{\exp(a_{k'}^{\text{out}}(t))}{a_k^{\text{out}}(t)} - \frac{\exp(a_{k'}^{\text{out}}(t))}{\sum_{k''}\exp(a_{k''}^{\text{out}}(t))}\frac{\exp(a_k^{\text{out}}(t))}{\sum_{k''}\exp(a_{k''}^{\text{out}}(t))} \\ =& \delta_{kk'}y_{k'}^{\text{out}}(t) - y_{k'}^{\text{out}}(t)y_k^{\text{out}}(t),\end{aligned} \tag{3.28}$$

where $\delta_{kk'}$ is the Kronecker delta function defined in Section 2.4.4. The signal generated by the objective function passing through k is

$$\begin{aligned}\frac{\partial \mathcal{F}^{\text{CE}}}{\partial a_k^{\text{out}}(t)} =& \sum_{k'}\frac{\partial \mathcal{F}^{\text{CE}}}{\partial y_{k'}^{\text{out}}(t)}\frac{y_{k'}^{\text{out}}(t)}{a_k^{\text{out}}(t)} \\ =& \sum_{k'} r_{k_0k'}(t)y_k^{\text{out}}(t) - \sum_{k'} r_{k_0k'}(t)\delta_{kk'} \\ =& y_k^{\text{out}}(t) - r_{k_0k}(t),\end{aligned} \tag{3.29}$$

which uses the the reference distribution property $\sum_{k'} r_{k_0k'}(t) = 1$. Eqn. (3.29) can be viewed as an "error signal" generated by subtracting the reference value from its corresponding output value. Actually training an output layer with a linear activation function

$$y_k^{\text{out}}(t) = a_k^{\text{out}}(t) \tag{3.30}$$

according to the MMSE criterion can result in error signals of the same form.

In the first layer, the error signal $\partial\mathcal{F}/\partial a_j^{(1)}(t)$ is calculated by the chain rule as

$$\begin{aligned}\frac{\partial\mathcal{F}}{\partial a_j^{(1)}(t)} &= \sum_k \frac{\partial\mathcal{F}}{\partial a_k^{\text{out}}(t)} \frac{\partial a_k^{\text{out}}(t)}{\partial a_j^{(1)}(t)} \\ &= \frac{\partial f(a_j^{(1)}(t))}{\partial a_j^{(1)}(t)} \sum_k \frac{\partial\mathcal{F}}{\partial a_k^{\text{out}}(t)} w_{kj}^{\text{out}}. \end{aligned} \tag{3.31}$$

That is, the first layer error signals rely on those from its succeeding layer, which is required to compute $\partial\mathcal{F}/\partial w_{ji}^{(1)}$ and $\partial\mathcal{F}/\partial b_j^{(1)}$ according to Eqn. (3.2). Since the error signals flow backward through the network during training, the algorithm is therefore referred to as backpropagation. Eqn. (3.31) also shows that $\partial\mathcal{F}/\partial a_k^{\text{out}}(t)$ and $\partial\mathcal{F}/\partial a_j^{(1)}(t)$ are common in calculating all $\partial\mathcal{F}/\partial w_{ji}^{(l)}$ and $\partial\mathcal{F}/\partial b_j^{(l)}$, and are stored to save redundant computation. This reduces the standard number of $\mathcal{O}((I+1)J^2(3K-1))$ calculations to $\mathcal{O}((I+2K)J+K)$ when computing $\partial\mathcal{F}/\partial w_{ji}^{(1)}$ and $\partial\mathcal{F}/\partial b_j^{(1)}$. This idea is the key point in EBP, with which all $\partial\mathcal{F}/\partial a_j^{(l)}(t)$ are computed only once and shared by all derivative calculations.

Next, EBP is generalised to the flexible DCG ANN structure described in Section 3.2. First, let us consider only FNNs. For each layer k, let $\mathbf{s}^{(k)}$ be the collection constituting of all immediate succeeding layers of k. For each layer $l \in \mathbf{s}^{(k)}$, let $\mathbf{c}_k^{(l)}$ be the context shift set of the feature element associated with k in l, then the derivatives with respect to some parameter $\theta^{(k)}$ of layer k are computed by

$$\frac{\partial\mathcal{F}}{\partial\theta^{(k)}} = \sum_{l\in\mathbf{s}^{(k)}} \sum_{c\in\mathbf{c}_k^{(l)}} \sum_j \sum_i \frac{\partial\mathcal{F}}{\partial a_j^{(l)}(t+c)} \frac{\partial a_j^{(l)}(t+c)}{\partial y_i^{(k)}(t+c)} \frac{\partial y_i^{(k)}(t+c)}{\partial a_i^{(k)}(t+c)} \frac{\partial a_i^{(k)}(t+c)}{\partial\theta^{(k)}}, \tag{3.32}$$

where $\partial\mathcal{F}/\partial a_j^{(k)}(t+c)$ is found by EBP, $\partial y_i^{(k)}(t+c)/\partial a_i^{(l)}(t+c)$ and $\partial a_j^{(l)}(t+c)/\partial y_i^{(k)}(t+c)$ are computed by Eqns. (3.1) and (3.2), and $\partial y_i^{(k)}(t+c)/\partial\theta^{(k)}$ is computed by either Eqn. (3.25) or Eqn. (3.26) depending on whether θ is a weight or bias. Note that besides explicit ANN parameter tying, the flexible ANN structures can also give rise to shared structures, such as to use a layer's output in multiple layers. In these cases, the partial derivatives are summed over all contributions to the shared structures, and need to be normalised in SGD. Let $\mathbf{p}_\theta$ be the collection of layers that contain the same parameter θ, and the normalisation term is computed by

$$\sum_{k\in\mathbf{p}_\theta} \sum_{l\in\mathbf{s}^{(k)}} |\mathbf{c}_k^{(l)}|,$$

where $|\mathbf{c}_k^{(l)}|$ denotes the number of items in $\mathbf{c}_k^{(l)}$. For recurrent ANNs, the directed cycles are unfolded through time into an FNN before calculating the gradients using Eqn. (3.32), which is the common strategy used in the BPTT algorithm for standard RNN training (Pineda, 1987; Robinson and Fallside, 1987).

3.5 Gradient Descent

GD is the simplest iterative method for optimising differentiable variables (Gill et al., 1981). By viewing the training objective function $\mathcal{F}$ as a surface sitting above a model space Θ, an *error surface* is constructed which is a *scalar field* with each point in the surface serving as a particular value of the model parameters. In *vector calculus*, the gradient at a point is the vector of the first order partial derivatives with respect to the model parameters, which points in the direction of the steepest slope at that point. Therefore, iteratively updating the parameters by moving in the direction of the negative gradient can lead to a fast decrease of the objective function value.

3.5.1 Full batch based gradient descent

In practice, ANN training can be performed using the *full batch* based GD algorithm, which treats the entire training set as a full batch. For each update n, the averaged gradient values $\bigtriangledown\mathcal{F}$ are computed over all samples in the full batch. At each point $\Theta[n]$ in the error surface, a short distance, which is obtained by scaling the gradient $\bigtriangledown\mathcal{F}|_{\Theta[n]}$ by the learning rate $\eta_\Theta[n]$, is moved to get to the next point $\Theta[n+1]$, i.e.,

$$\Theta[n+1] = \Theta[n] - \eta_\Theta[n] \bigtriangledown \mathcal{F}|_{\Theta[n]}, \tag{3.33}$$

and the short distance is referred to as the *update value*. Each parameter update occurs at the end of one training epoch (i.e., using all training samples to calculate the gradients). If the learning rates are small enough, the objective function will decrease in each succeeding iteration and eventually converge to a local optimal point Θ^* that satisfies

$$\bigtriangledown\mathcal{F}|_{\Theta^*} = 0. \tag{3.34}$$

Interestingly, with some modifications, the GMM re-estimation formulas derived from the BW and EBW algorithms, Eqns. (2.37) – (2.39) and Eqns. (2.90) – (2.94), can be viewed in the GD framework (Schlüter et al., 2001; Sha and Saul, 2008). The

"learning rates" used in BW and EBW algorithms are carefully chosen to guarantee that the objective function value will not decrease after each epoch.

In addition, it is often useful to normalise the input features by 0-MN and 1-VN for GD based ANN training. For global normalisation, in principle it is a redundant linear transform that can be combined with standard model parameters by transforming weights and biases associated with the input. However, since feature normalisation can ensure all input and output dimensions to be of *order unity*, it can also help achieve order unified model parameters that are useful in applying unified random initialisation priors and learning rates *etc.* (Bishop, 1995; LeCun et al., 1998b). Note that variance normalisation is not as important in GMM training since BW and EBW are guaranteed to find almost equivalent variances with only a constant difference in log-likelihoods (see Eqn. (2.68)).

Alternative optimisation methods to GD often use more information from the derivatives (Gill et al., 1981), which include other first order methods, e.g., the *conjugate gradient method*, as well as second order methods, such as the *Newton's method* and the *Quasi-Newton method etc.* These methods are generally more powerful than GD that are often possible to converge to better local optima using fewer parameter updates (Bishop, 1995; Martens, 2010), while requiring more computation and storage.

3.5.2 Stochastic gradient descent

When the training set is large, the amount of computation required to process the entire set for each parameter update using the full batch based GD becomes too high. A more efficient modification is the *on-line version* of GD, also known as SGD, which has proven very useful in training ANNs on large data sets (Bottou, 2010; Robbins and Monro, 1951). Instead of updating the parameters once per epoch, SGD samples a small portion of data from the full batch known as a *minibatch*, and updates the parameters based on the averaged gradients computed over the samples in the minibatch. This process is repeated by cycling through the entire training set to accomplish a complete training epoch. That is, for a parameter $\theta \in \Theta$,

$$\theta[n+1] = \theta[n] + \delta_\theta[n], \tag{3.35}$$

where $\delta_\theta[n]$ is the update value applied to $\theta[n]$. $\delta_\theta[n]$ can be obtained by

$$\delta_\theta[n] = -\eta_\theta[n] \frac{\partial \mathcal{F}[n]|_{\Theta[n]}}{\partial \theta}, \tag{3.36}$$

and $\partial\mathcal{F}[n]/\partial\theta$ is the averaged partial derivative computed over the samples from the nth minibatch[1]. Therefore, compared to full batch based GD, SGD uses many more parameter updates along "less accurate" steepest slope directions on the error surface to accelerate optimisation. Another property of SGD is the possibility of escaping from local optima, since the optima with respect to the error surface for the whole data set may not be the optima for each individual minibatch (Bishop, 2006). Note that for speaker independent (SI) acoustic modelling, it is rather important to use appropriately sized minibatches and to randomly select their contents from the complete training set. As discussed in Section 2.6.1, speech samples contain unique characteristics from a particular speaker, and performing many SGD updates with a small minibatch or without shuffling the data from many speakers can cause the acoustic models to be biased to some speakers rather than modelling all speakers in general.

As shown in Eqn. (3.35), SGD requires each parameter update to be executed in a serial manner, and thus it is not easy to be parallelised to allow applying to very large scale problems. Some of recent studies have focused on removing this restriction in different ways (Chen and Huo, 2016; Dean et al., 2012; Povey et al., 2015; Seide et al., 2014). Finally, SGD has been shown to converge to a local optimum when certain conditions are satisfied (Robbins and Siegmund, 1971). In the following section, the choice of learning rates will be discussed.

3.6 Practical Solutions to SGD Issues

3.6.1 Learning rate scheduler

Generally speaking, the GD method is rather sensitive to appropriate learning rate values. If the learning rate is too large, the parameter update may always skip local optima and result in training divergence; if the learning rate is not sufficiently large, the training can be slow to converge. Actually the learning rates $\eta_\theta[n]$ can be both parameter dependent and adjusted individually at each parameter update, and various learning rate schedulers have been proposed to improve GD performance (Senior et al., 2013).

- The most naïve learning rate scheduler is to use a pre-determined learning rate shared by all parameters and fixed during an epoch or the entire training procedure, which is denoted as the *List* scheduler.

[1] In the experiments, if the minibatch is of size M, the learning rate is actually set to $\eta_\theta[n]/M$, since the gradients are accumulated but not averaged in HTK.

- A simple improvement to the List scheduler is to evaluate model performance after multiple updates (e.g., an epoch), and modify the learning rate accordingly. The *NewBob* scheduler is an example (see Section 3.10.1 for details) (Renals et al., 1992), and this kind of scheduling is sometimes called performance scheduling.
- Some researchers have reported that the shared learning rate can be gradually reduced after each update (Bottou, 2010; Senior et al., 2013; Xu, 2011).
- Alternative algorithms associate an individual learning rate with each parameter and adjust them according to various rules (Duchi et al., 2010; Riedmiller and Braun, 1993; Senior et al., 2013).

3.6.2 Momentum

In some cases, there exist some regions of *pathological curvature* on the error surface, where the steepest slope directions do not point towards any local optima, and SGD can cause oscillations in searching and result in slow convergence (Bishop, 1995). *Momentum* is a method that adds some inertia to the update direction on the error surface to smooth successive parameter updates, and can help avoid oscillations (Polyak, 1964). The classical form of momentum is implemented by modifying the update value defined in Eqn. (3.36) as

$$\delta_\theta[n] = -\eta_\theta[n]\frac{\partial \mathcal{F}[n]|_{\Theta[n]}}{\partial \theta} + \rho\delta_\theta[n-1], \tag{3.37}$$

where ρ is the *momentum coefficient*. Besides the classical momentum used in this thesis, there exist other types of momentum (Nesterov, 1983), and the momentum coefficient can be adjusted during training (Sutskever et al., 2013).

3.6.3 Gradient and update value clipping

When evaluating the SGD gradients, it is often observed that there exist some derivatives which are much larger than the others (Bengio et al., 1994). This can be caused by some abnormal data or data sampling noise in a minibatch. Such derivatives are calculated at some particular point on the error surface for some minibatches that have some dimensions with exceptionally large derivatives. The resulting large derivatives can even dominate a parameter update and make it ineffective.

This issue is termed as *gradient explosion* in the literature (Bengio et al., 1994), and can be addressed by a simple *gradient clipping* method, i.e. to clip the derivatives

whose absolute values are bigger than a threshold in each minibatch (Mikolov, 2012). In practice, it is often more effective to clip the update values instead, since they contain not only the gradient information, but also the influences from momentum and weight decay *etc.* More specifically, *update value clipping* can be presented as

$$\hat{\delta}_\theta[n] = \begin{cases} \text{sgn}(\delta_\theta[n])\upsilon & \text{if } |\delta_\theta[n]| > \upsilon \\ \delta_\theta[n] & \text{otherwise} \end{cases}, \tag{3.38}$$

where $\text{sgn}(\cdot)$ returns the sign and υ is the clipping threshold.

Furthermore, a problem related to gradient explosion is *gradient vanishing* (Bengio et al., 1994), which causes the derivative values to vanish. A solution is to scale up a vector formed from derivatives or update values, if its $L2$ norm falls below a threshold (Povey et al., 2011). This method is also useful in scaling down the exploding gradients.

3.6.4 Batch normalisation

It is known that the minibatch level mean and variance normalisations of the input to an ANN layer are often helpful in SGD (LeCun et al., 1998b; Wiesler and Ney, 2011; Wiesler et al., 2014b). Recently, a method called batch normalisation has drawn much attention (Ioffe and Szegedy, 2015). In this method, the input vector to layer l, $\mathbf{x}^{(l)}(t)$, is first normalised in each batch n,

$$\dot{\mathbf{x}}^{(l)}(t) = \frac{\mathbf{x}^{(l)}(t) - \mu^{(l)}[n]}{\sigma^{(l)}[n]^2}, \tag{3.39}$$

where $\mu^{(l)}[n]$ and $\sigma^{(l)}[n]^2$ are the mean and variance of $\mathbf{x}^{(l)}$ estimated by maximum likelihood (ML) in that batch. Afterwards, in order to maintain the input range, the input vector $\dot{\mathbf{x}}^{(l)}(t)$ is transformed linearly again by

$$\ddot{\mathbf{x}}^{(l)}(t) = \gamma^{(l)}\dot{\mathbf{x}}^{(l)}(t) + \beta^{(l)}. \tag{3.40}$$

$\ddot{\mathbf{x}}^{(l)}(t)$ is the input to the original layer defined by Eqn. (3.5), and $\gamma^{(l)}$ and $\beta^{(l)}$ are layer dependent parameters learned through network training. Similar to input feature normalisation, $\gamma^{(l)}$ and $\beta^{(l)}$ are again redundant parameters since they can not only be combined with weights and biases, but also cancel out the effect of batch normalisation

if

$$\gamma^{(l)} = \sigma^{(l)}[n]^2 \tag{3.41}$$

$$\beta^{(l)} = \frac{\mu^{(l)}[n]}{\sigma^{(l)}[n]^2}. \tag{3.42}$$

By alleviating the problem of layer specific input distribution changes, batch normalisation can be seen to smooth the error surface for each minibatch. It is found to reduce the difficulty of SGD based ANN training, which is reflected in enabling much larger learning rates, less careful initialisation, and less requirement for regularisation (Ioffe and Szegedy, 2015).

3.7 Regularisation

In machine learning, *regularisation* can refer to any modifications to the learning algorithm that help alleviate over-fitting (Goodfellow et al., 2016). A common regularisation strategy is to smooth the non-linear mapping function learned by the model, for example, by adding an extra term to the training objective function to penalise mappings that are not smooth, such as the I-smoothing method (see Section 2.7.4).

Here $L2$ *regularisation* is mainly discussed, which adds a squared $L2$ *norm* term $\varepsilon/2\sum_\theta \theta^2$ to the objective function $\mathcal{F}$, and ε is the $L2$ *coefficient*. Assume that the parameters θ have a Gaussian prior $\mathcal{N}(\theta|0,\sigma^2)$, and it can be shown that $L2$ regularisation can be obtained by imposing such Gaussian prior, and $\varepsilon = 1/\sigma^2$ (Bishop, 2006).

An alternative view of $L2$ regularisation is to check the modified update value based on Eqn. (3.36), which is

$$\delta_\theta[n] = -\eta_\theta[n]\frac{\partial \mathcal{F}[n]|_{\Theta[n]}}{\partial \theta} - \eta_\theta[n]\varepsilon\theta. \tag{3.43}$$

Since the extra term $-\eta_\theta[n]\varepsilon\theta$ reduces the increase of $|\theta|$ and can cause θ to decay in the long term, $L2$ regularisation is also termed as *weight decay* when applied to ANNs[1]. An empirical explanation is that relatively large weights are more likely to generate mapping functions with large curvatures that over-fit (Bishop, 1995; Woodland, 1989), and therefore, weight decay can help alleviate over-fitting.

[1]The bias parameter can be seen as an extended weight value, and the input value to the associated dimension is constant one.

In addition to explicitly adding a regularisation term as used later in this thesis, there are other machine learning methods that take the regularisation effect without changing objective functions (Goodfellow et al., 2016). Examples include the *early stopping* method that suggests model training should stop early to avoid over-fitting (Bishop, 1995), the *dropout* method that regularises by randomly inhibiting some hidden artificial neurons in training (Scrivastava et al., 2014), and *multi-task learning* (Caruana, 1993; Goodfellow et al., 2016) *etc.* Multi-task learning is a method that requires some generic model parameters shared across several tasks, and it pools the task related samples to put more pressure on the parameters towards values that generalise well. For acoustic modelling, the tasks are often either training criteria of the same targets (Kim et al., 2016; Povey et al., 2016) or the same criterion with different targets (Ghoshal et al., 2013; Heigold et al., 2013; Huang et al., 2013; Swietojanski et al., 2012).

3.8 Deep Learning

In machine learning, a narrow sense of *deep learning* often refers to a class of approaches that use many ANN layers to extract high-level, abstract features from raw data, which belongs to a broader class of *representation learning* (Goodfellow et al., 2016). Each layer provides a simple concept of things serving as a step for the computer to capture more complex concepts. Examples include using deep ANN models for image classification (Hinton et al., 2006), language modelling (Mikolov, 2012), and acoustic modelling (Hinton et al., 2012). Though the development of deep learning is perhaps the most important trend in machine learning and artificial intelligence in recent years, rather than a new technology, it is actually a resurgence of a classic technology that used to be named *cybernetics* and *connectionism*[1]. Many important concepts, ideas, and methods can be traced back several decades, but now they are used with more layers or at a larger scale, which is the reason why this resurgence is termed "deep" (Goodfellow et al., 2016). There are two key issues to deep learning:

- How to design a model to learn better data presentations for the task.
- How to design an ANN to better model the data for a specific task.

Meanwhile, nowadays the term "deep learning" also goes beyond the scope of ANN models. It also refers to a general principle of learning *multiple levels of composition*

[1]Note cybernetics mainly concerned with systems with feedback loops while connectionism systems are usually EBP trained MLPs or RNNs.

in frameworks based on a single objective function (Goodfellow et al., 2016). For instance, LeCun et al. (1998a) introduced the training of an entire system composed by multiple heterogeneous modules using EBP, and Bahdanau et al. (2015) implicitly and jointly learnt the different components in a machine translation system using an encoder-decoder model, and the same ideas have been applied to ASR as well (Bahdanau et al., 2016; Lu et al., 2015). In this thesis, deep learning is used in both senses. In Chapter 4 and Chapter 5, it refers to a DNN model, while in Chapter 6, it means the joint learning of both feature extraction and acoustic model modules. In the following section, some deep learning methods will be briefly introduced.

3.8.1 Deep ANN models

Intuitively, ANNs with more layers are able to produce more abstracted high-level features that are often useful in modelling. Early researchers worked on RNNs in the late 80s that could be viewed as a deep FNN with many layers unfolded through time and tied together (Robinson, 1989; Rumelhart et al., 1986). Meanwhile, with increasing device computational power and knowledge, researchers also gradually used more hidden layers in MLPs (with up to four hidden layers) (Ellis and Morgan, 1999; Grézl et al., 2007; Zhu et al., 2005). However, it was often found that adding more layers is not only computational intensive at that time but also unstable in training. Until a decade ago, there was no known commonly used training procedure for deep neural networks (DNNs), which referred to deep MLPs with more than four hidden layers (Bengio, 2009). The standard random initialisation and an SGD training setup often result in convergence to very poor local optima (Glorot and Bengio, 2010).

The current resurgence of deep learning started from the successful training of DNNs, which relied on a two stage training procedure comprising of layer-wise pretraining (PT) and fine-tuning (FT) (Hinton and Salakhutdinov, 2006). PT is actually a sensible parameter initialisation method that gradually trains more complicated mapping functions with more hidden layers, and FT is the normal training with the final DNN structure. Initially, PT was carried out by stacking restricted Boltzmann machines (RBMs) (Hinton, 2010; Hinton et al., 2006). Shortly afterwards, it was found that this procedure could be altered to stacking MLPs (Bengio et al., 2007). Analysis showed that the difficulty of DNN training was caused by the positive mean value of the sigmoid which could drive hidden layers into *saturation* (Glorot and Bengio, 2010). In other words, sigmoid outputs are stuck around zero or one and inhibit the training due to the vanishing gradients probably caused by the quickly increasing bias values (based on Eqn. (3.20)). Glorot and Bengio (2010) has also shown that PT can be replaced

by a more appropriate random initialisation method without much performance loss, which is

$$\mathbf{W}^{(l)} \sim U\left[-\frac{4\sqrt{6}}{\sqrt{I_l+J_l}}, \frac{4\sqrt{6}}{\sqrt{I_l+J_l}}\right] \tag{3.44}$$

$$\mathbf{b}^{(l)} = \mathbf{0} \tag{3.45}$$

for the sigmoid activation function, where U denotes the uniform distribution. This method is named after its author as *Xavier's initialisation*, and is used throughout this thesis by all sigmoid Actually a very similar parameter initialisation approach was found more than a decade earlier (LeCun et al., 1998b). More solutions were proposed in later studies, such as second order optimisation (Martens, 2010), better momentum setting (Sutskever et al., 2013), and batch normalisation (Ioffe and Szegedy, 2015). Moreover, ReLU DNNs were found to be easier to optimise and had no need to pretrain. Comparing Eqn. (3.20) with Eqn. (3.22), the ReLU does not decrease the derivatives and is more useful to overcome the gradient vanishing problem in training very deep models.

Furthermore, RNNs and deep CNNs can be converted to special DNNs by unfolding their tied parameters (see Section 3.1 and 3.2 for details). Both RNN and CNN have been applied to different tasks for a long time and recent research has improved their performance and training stability (Bengio et al., 2012; Graves and Schmidhuber, 2005; He et al., 2015b; Pascanu et al., 2013; Simonyan and Zisserman, 2015). Nowadays, researchers are paying a great deal of attention to even deeper models, such as RNNs with internal memory functions, ReLU based very deep CNNs, along with their combinations, and significant improvements have been observed by using these models (Bi et al., 2016; He et al., 2015a; Qian and Woodland, 2016; Saon et al., 2016; Sercu et al., 2016; Xiong et al., 2016). These models are called *deep ANNs* in this thesis, while the term DNN only refers to deep MLPs. More interestingly, RNNs are extended to manage external memory to mimic some of the human brain's short-term working memory (Graves et al., 2014).

Although the PT and FT approach is no longer necessary in DNN construction, it is reviewed next in the section since it is still useful in small scale tasks and is related to some proposed methods in this thesis.

3.8.2 Generative PT

The RBM is a generative probabilistic model that consists of a set of disconnected hidden units $\mathbf{h}$ and a set of visible units $\mathbf{v}$, where $\mathbf{v}$ and $\mathbf{h}$ are random variables (Hinton, 2010). Each hidden unit is connected and only connected with all visible units. The parameter set Θ is $\{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$, where $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{W}$ are the visible unit bias vector, hidden unit bias vector, and weight matrix, respectively. The probabilities of the input feature is calculated by summing over the joint distribution of $\mathbf{v}$ and $\mathbf{h}$,

$$\begin{aligned} p(\mathbf{v}|\Theta) &= \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}|\Theta) \\ &= \sum_{\mathbf{h}} \frac{\exp(-E(\mathbf{v}, \mathbf{h}|\Theta))}{\sum_{\mathbf{v}'} \sum_{\mathbf{h}'} \exp(-E(\mathbf{v}', \mathbf{h}'|\Theta))}, \end{aligned} \tag{3.46}$$

where for real-valued input data, the energy function $E(\mathbf{v}, \mathbf{h}|\Theta)$ is

$$E(\mathbf{v}, \mathbf{h}|\Theta) = -||\mathbf{v} - \mathbf{a}||^2 - \mathbf{h}^{\mathrm{T}}\mathbf{b} - \mathbf{h}^{\mathrm{T}}\mathbf{W}\mathbf{v}, \tag{3.47}$$

and $||\mathbf{v} - \mathbf{a}||$ is the 2-norm of the vector $\mathbf{v} - \mathbf{a}$. An RBM is often trained by maximising $p(\mathbf{v}|\Theta)$ using an iterative *contrastive divergence* algorithm (Hinton et al., 2006), which is an unsupervised training approach since no labels are required.

The generative PT approach is carried out by stacking RBMs. Initially an RBM is trained on the input features, and its output values from the hidden units are

$$p(\mathbf{h}|\mathbf{v}, \Theta) = f(\mathbf{W}\mathbf{v} + \mathbf{b}), \tag{3.48}$$

where $f(\cdot)$ is sigmoid. Next, in each step a new RBM is trained by taking the outputs from the last RBM as the visible unit inputs, until the desired number of hidden layers is reached, and each new RBM can improve the variational bounds on the log probability. Finally, a randomly initialised output layer is put on top of all RBMs to complete the PT procedure (Hinton and Salakhutdinov, 2006). This procedure follows a *greedy strategy* that each RBM is estimated by freezing all preceding RBMs. It restricts parameter searching only to a subset of the model space and thus can only lead to sub-optimal solutions. After PT, the resulting model is seen as an initial DNN and all of its parameters are optimised jointly through EBP and SGD, which is the FT stage. Note that when viewing the pretrained model as a DNN, all visible biases $\mathbf{a}$ are discarded; each hidden unit vector $\mathbf{h}$ becomes the artificial neurons of each layer and no longer represents any random variables.

3.8.3 Discriminative PT

In the literature, *discriminative PT* often refers to PT by stacking 2-layer MLPs (Bengio et al., 2007). Analogous to the layer-wise procedure in generative PT, in each step a 2-layer MLP is discriminatively trained for multiple epochs to convergence and added on top of the existing model. Each MLP is trained to discriminate the same set of targets as the final DNN, and its output layer is removed when stacking the next MLP. It is obvious that this PT approach is also a greedy algorithm since the parameter search is still restricted to a subset of the model space.

An improved algorithm is proposed in (Seide et al., 2011a) that all existing layers, instead of just the newly added 2-layer MLP, are fully trained to convergence at each step, which allows more dimensions in the model space to be searched and alleviates the sub-optimal problem. Note that in PT when the hidden layer number grows to more than four, it is no longer a shallow MLP but a DNN. Later experiments have shown lower WERs can be obtained from the final DNN if the model is only trained for one epoch in each discriminative PT step (Hinton et al., 2012). This is perhaps because training the model to convergence can over train the hidden layer parameters with the intermediate model error surfaces and cause the FT stage to find sub-optimal solutions. Since the last version in (Hinton et al., 2012) was found to work consistently well, it is used as the default discriminative PT algorithm thereafter in the thesis without any distinction.

Besides the aforementioned advantages, replacing generative PT with discriminative PT can simplify DNN training from both conceptual and implementation perspectives, since only one class of models and one training approach are required. Furthermore, discriminative PT is often based on a criterion $\mathcal{F}$ that better matches the FT criterion. However, this can also generate a side-effect that the discriminatively pretrained DNN more easily over-fits the objective function, especially when the data set is small. A solution to this issue will be discussed later in Section 4.4.1.

3.9 Integrating ANNs into ASR

Though ASR can be constructed solely using ANNs (Bahdanau et al., 2016; Graves and Jaitly, 2014; Lu et al., 2015; Miao et al., 2015; Robinson, 1989; Waibel et al., 1989; Woodland, 1992; Zhang et al., 2016), ANNs can also be integrated into different modules of HMM based ASR systems, which provides a convenient way to combine their advantages. When ANNs are used to extract features for GMM-HMM acoustic models or to produce HMM output probabilities, the resulting systems are called

tandem systems or *hybrid systems*, respectively. ANNs can also be applied to language modelling (Liu et al., 2016; Mikolov, 2012), which is beyond the scope of this thesis and is not discussed here. Particularly, training hybrid systems using a sequence level objective function is also briefly reviewed here, which is a key difference between ANN acoustic modelling and other applications. Note that none of the tandem, hybrid, or ANN LM approaches is new, but all of them benefit from the development of deep learning by simply replacing traditional shallow ANNs with deep ANNs and produce state-of-the-art results.

3.9.1 Tandem system

Originally, a tandem system comprised an MLP system and a GMM-HMM system (Hermansky et al., 2000). The MLP is trained to classify phonetic units whose posterior probabilities are used as the input to the following GMM-HMMs. The word "tandem" means that the MLP and GMM-HMM systems work together at the same time. Since the MLP output vector dimension is often too high to be modelled by GMMs, it is first compressed by Karhunen-Loève transform (KLT) or LDA and then combined with standard acoustic observations such as MFCC or PLP. These transforms linearly decorrelate the posterior features at the same time, in order to model them with GMMs with diagonal covariance matrices. The final input features to GMM-HMMs are referred to as tandem features in this thesis.

An alternative tandem system configuration is to train a bottleneck (BN) MLP, which has a reduced dimension hidden layer, i.e., a BN layer (Grézl et al., 2007). Since the BN layer is normally much smaller in size than the other hidden layers, its output vector $\mathbf{y}^{\text{bn}}(t)$ is very compact and suitable to be used as features in GMMs. The training procedure for BN MLPs is usually the same as for normal MLPs. Once the model is trained, the activation function of the BN layer is changed to a linear activation function defined by Eqn. (3.30), making its output values $\mathbf{y}^{\text{bn}}(t)$. There are different ways of using the BN features. Following the standard CUED approach (Park et al., 2011), conventional tandem features in this thesis employ a single class STC to decorrelate $\mathbf{y}^{\text{bn}}(t)$ and concatenate it with `PLP_D_A_T` projected by HLDA. Therefore, the tandem feature $\mathbf{z}(t)$ is

$$\mathbf{z}(t) = \begin{bmatrix} \mathbf{A}^{\text{HLDA}}\mathbf{o}(t) \\ \mathbf{A}^{\text{STC}}\mathbf{y}^{\text{bn}}(t) \end{bmatrix}. \tag{3.49}$$

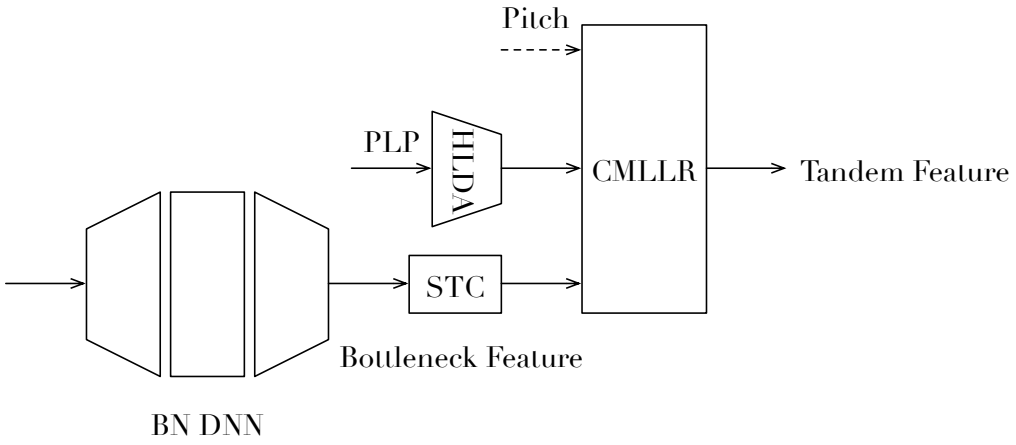


Figure 3.4 The tandem feature generation procedure. The pitch feature can be included in the SI tandem features.

The tandem GMM-HMM training uses $\mathbf{z}(t)$ instead of $\mathbf{o}(t)$ in likelihood calculations, ML and discriminative training, as well as speaker transform adaptation, and $\mathbf{z}(t)$ replaces $\mathbf{o}(t)$ in the HLDA transform estimation stage in the procedure in Section 2.8. The tandem feature generation procedure is illustrated in Figure 3.4.

Deep ANNs have been applied to tandem systems by taking the place of MLPs. It was shown BN DNN tandem features outperformed posterior DNN tandem features (Tüske et al., 2012), and therefore, only the BN tandem system configuration is studied in this thesis. Yan et al. (2013) also found KLT projected normal hidden layer outputs could be used by high performance tandem systems. Other commonly seen deep models such as CNN, RNN, and LSTM have been used to produce BN features as well (Sainath et al., 2015a).

3.9.2 Hybrid system

Traditional acoustic models use GMMs to model HMM output probabilities. Therefore, replacing GMMs with MLPs is seen as "hybrid" of ANNs and HMMs (Bourlard and Morgan, 1993). In such hybrid systems, MLPs are used directly to provide state log-likelihoods for the HMM acoustic models, as shown in Figure 3.5, and the MLP posterior probabilities $P(C_k|\mathbf{x}^{\text{in}}(t))$ are converted to the log-likelihood of $\mathbf{x}^{\text{in}}(t)$ generated by the HMM state k relevant to C_k using Bayes' rule as

$$\ln p(\mathbf{x}^{\text{in}}(t)|k) = \ln P(C_k|\mathbf{x}^{\text{in}}(t)) + \ln p(\mathbf{x}^{\text{in}}(t)) - \ln P(k), \tag{3.50}$$

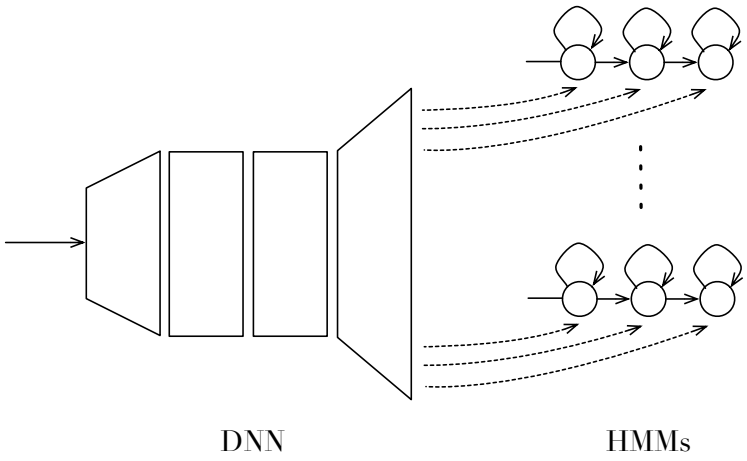


Figure 3.5 A sketch map of a hybrid system.

where,

$$P(k) = \text{count}(k) / \sum_{k'} \text{count}(k'), \tag{3.51}$$

and count(k) is the number of frames aligned to the state k, $P(k)$ and $p(\mathbf{x}^{\text{in}}(t))$ are the prior probabilities of the output target k and the input frame respectively, and $p(\mathbf{x}^{\text{in}}(t))$ is independent of the HMM state. For triphone hybrid systems, GMM-HMM decision tree tying is used to generate the tied triphone states. An MLP which uses the resulting tied states as the output targets is denoted as a context-dependent (CD) MLP since triphones are the only type of CD units considered in this thesis. The MLP is often trained based on the frame level criterion $\mathcal{F}^{\text{CE}}$ using the state to frame alignments produced by an existing system. It is worth noting that unlike GMM-HMMs, the MLP and HMMs are trained separately due to the fact that the HMM transition probabilities make minor differences to hybrid system performance (Dahl et al., 2012).

Besides using a single MLP, the hybrid system can also use individually trained MLPs stacked in different ways (Morgan, 2012). Using a hybrid system with a DNN model was the first successful deep learning instance in ASR (Dahl et al., 2012; Seide et al., 2011b). Separately trained DNNs can be stacked to form a stacked DNN hybrid system (Knill et al., 2013). Deep hybrid systems can also be constructed with deep CNNs, RNNs, LSTMs, and even the convolutional long short-term memory deep neural network (CLDNN) constructed by stacking a CNN, an LSTM, and a DNN together (Sainath et al., 2015a). Furthermore, recent progress has shown that hybrid systems can be trained using the FB algorithm and without relying on any fixed alignments (Graves et al., 2006; Povey et al., 2016).

3.9.3 Sequence training for hybrid systems

As introduced in Chapter 2, training acoustic models based on a sequence level criterion is often very important in high performance ASR construction. There is a long-term interest in applying sequence training to ANN acoustic models, often through a connection to HMMs (Bengio, 1991; Bridle, 1990a; Niles and Silverman, 1990). "Alpha-nets" was an RNN architecture that incorporates the alignment step based on the forward variable $\alpha(t)$s from the forward-backward algorithm (FB) (see Section 2.3.2), which has an exact interpretation in terms of HMMs (Bridle, 1990a). Similarly, Niles and Silverman (1990), Bengio (1991), Bengio et al. (1992), and Yan et al. (1997) directly calculated the FB variables based on HMMs and trained a set of ANNs for output probability estimation with these variables through EBP. Note that the ANN output distributions are often single Gaussians, and the resulting ANN-HMM hybrid system can be viewed as a tandem system as well (Bengio, 1991; Bengio et al., 1992; Bridle, 1990a). Furthermore, an RNN-HMM hybrid system was developed for handwritting recognition, and the entire model was trained at the sequence level also using FB (Senior, 1994).

For LVCSR, Bourlard et al. (2016) performed an interleaved procedure of maximum a posteriori (MAP) based alignment and the hybrid acoustic model training. With the increasing of computational power, lattice based sequence training for MLP-HMMs based on maximum mutual information (MMI) or minimum Bayes' risk (MBR) criterion also becomes possible (Kingsbury, 2009; Valtchev, 1995). For DNN-HMMs, Kingsbury et al. (2012) extended the latticed based MMI/MBR training to use the second order optimisation method; alternatively, Veselý et al. (2013), Su et al. (2013), Wiesler et al. (2015), and Zhang and Woodland (2015a) investigated the same training methods using SGD. The detailed approach will be discussed later in Section 6.1. Nowadays, it has been demonstrated that lattice free MMI training for LVCSR is also possible (Povey et al., 2016).

3.10 Baseline Configurations

In this section, some configurations for building tandem and hybrid systems are presented, which include the learning rate scheduler, silence modelling, the tandem system BN layer size and position, hybrid system input features, and a tandem and hybrid system combination approach, known as joint decoding.

3.10.1 An improved NewBob scheduler

The improved NewBob scheduler, NewBob$^+$, modifies a learning rate shared by all model parameters based on the changes of a criterion $\mathcal{F}$. For training ANN acoustic models based on CE, $\mathcal{F}$ is normally the frame classification accuracy. To have better evaluation of $\mathcal{F}$, a cross-validation (CV) set is often held out from the training set[1], and $\mathcal{F}$ is calculated on the CV set with the parameters frozen at the end of each training epoch.

The actual NewBob$^+$ procedure is presented as Algorithm 2. N and $\eta[N]$ are the current epoch index and learning rate, and *ramp* is a Boolean flag indicating whether the algorithm is in the ramp state. At the beginning, the current criterion value change $\Delta\mathcal{F}[N]$ is calculated. If the criterion is not improved, the previous best model is reloaded. The training stops if the algorithm is in the ramp state and $\Delta\mathcal{F}[N]$ is smaller than the threshold $\Delta\mathcal{F}_{\text{stop}}$. If $\Delta\mathcal{F}[N]$ is smaller than the ramp state threshold $\Delta\mathcal{F}_{\text{ramp}}$, the learning rate is reduced by half since $\mathcal{F}[N]$ decreases more slowly than expected and may indicate that the model is close to convergence; the algorithm enters the ramp state if the training has lasted for more epochs than a minimum number, N_{min}. Once it is in the ramp state, to avoid the local optimum being skipped by using a large step size, the learning rate is reduced by half in every consequent epoch.

Algorithm 2 NewBob$^+$ learning rate scheduler

procedure NewBob$^+$(N, $\eta[N]$,*ramp*)
  $\Delta\mathcal{F}[N] = \mathcal{F}[N] - \mathcal{F}_{\max}[N-1]$
  if $\Delta\mathcal{F}[N] \leqslant 0.0$ **then**
    reload the model that produced $\mathcal{F}_{\max}[N-1]$
    $\mathcal{F}_{\max}[N] = \mathcal{F}_{\max}[N-1]$
  else
    $\mathcal{F}_{\max}[N] = \mathcal{F}[N]$
  if *ramp* and $\Delta\mathcal{F}[N] < \Delta\mathcal{F}_{\text{stop}}$ **then**
    stop training
  if *ramp* **then**
    $\eta[N+1] = 0.5 \cdot \eta[N]$
  else if $\Delta\mathcal{F}[N] < \Delta\mathcal{F}_{\text{ramp}}$ **then**
    $\eta[N+1] = 0.5 \cdot \eta[N]$
    if $N \geqslant N_{\text{min}}$ **then**
      $ramp = \text{TRUE}$

[1] It is really a *held-out* validation rather than CV, since it learns and tests on only one data division.

The NewBob$^+$ is an improvement to the standard NewBob algorithm, which has been very successful in MLP acoustic model training (Renals et al., 1992). The difference between the NewBob$^+$ and NewBob is the introduction of the minimum epoch number, N_{min}. In the standard NewBob, the algorithm can enter the ramp state as soon as the criterion value change is smaller than expected ($\Delta\mathcal{F}_{\text{ramp}}$). Then in the ramp state, the learning rate decreases by half at the end of each epoch and the training normally stops quite quickly. This is supported by the early stopping strategy (see Section 3.7). However, in acoustic modelling, since even a small training set can have millions of frames, and is usually sufficient to allow a more aggressive training strategy. This issue is solved in this thesis by adding $N_{\text{min}} > 0$ to delay the entering of the ramp state. (Wiesler et al., 2014b) changed NewBob for the same reason, and made the training more aggressive by using a smaller learning rate reduction rate.

An experiment was carried out to compare NewBob$^+$ with NewBob schedulers on the Babel Cantonese FLP data set (see Appendix A.1 for details). Minimum phone error (MPE) tandem systems were constructed with 1000 tied states and 16 Gaussian components/state on the full training set[1]. Both tandem features were generated by monophone BN DNNs with a structure of $468 \times 1000 \times 1000 \times 1000 \times 1000 \times 26 \times 126$, trained separately using the NewBob and NewBob$^+$. The learning rate settings for PT and FT are listed in Table A.1. From Table 3.1, NewBob$^+$ used $N_{\text{min}} = 12$ and the training stopped at the 16th epoch, while the NewBob training ended with only 10 epochs. It is clear that NewBob$^+$ outperformed NewBob in both frame classification accuracy and CER[2].

Table 3.1 Comparison between the NewBob and NewBob$^+$ schedulers on Babel Cantonese FLP data set. The table show the frame level classification correctness and the recognition CER with a trigram LM.

Scheduler	n Epoch	%Corr.		%CER
		Train	CV	Test
NewBob	10	78.6	77.2	56.1
NewBob$^+$	16	80.2	78.3	55.0

[1]Since the CV set is held-out from training, all training data and training data apart from the CV data are denoted as the full training set and training set, respectively. The CV set is formed by randomly selecting 10% of the data from the full training set throughout this thesis.

[2]Note that though *confusion network* decoding (Evermann and Woodland, 2000; Mangu et al., 2000) is widely used in many CUED HTK systems, none of the results presented in this thesis used it.

Table 3.2 Babel Cantonese FLP BN GMM-HMM system trigram LM CERs with different BN CD-DNN silence modelling methods.

DNN Structure	`sil`	%CER
$468 \times 1000 \times 1000 \times 1000 \times 1000 \times 26 \times 429$	30	55.5
$468 \times 1000 \times 1000 \times 1000 \times 1000 \times 26 \times 402$	3	54.5
$468 \times 1000 \times 1000 \times 1000 \times 1000 \times 26 \times 400$	1	54.0

3.10.2 Tandem baseline system

Silence modelling

When extending the monophone BN DNN system in Section 3.10.1 to use triphone output targets, as shown in the 1st system in Table 3.2, the CER increased rather than decreased, compared to the 2nd system in Table 3.1. The issue lies in the non-speech model (that is seen as the silence model for convenience) used by the BN DNNs, which consists of 30 HMM states and is used by some recent CUED GMM-HMM systems (Gales et al., 2015; Knill et al., 2013; van Dalen et al., 2015; Wang et al., 2015), where different types of non-speech exist in the data. Generally speaking, the difference between the frames aligned with different `sil` states should be smaller than those aligned with the phone states. Therefore, in the training, there can be many silence frames assigned to incorrect `sil` states (according to the training alignments), which causes the DNN parameters more focused on correctly classifying the `sil` frames. For the monophone BN DNN, it has 126 targets associated with 90 monophone HMM states and 30 `sil` states. The monophones are rather different from each other and easier to distinguish. For the triphone BN DNN, however, the 399 triphone tied states are more similar to each other and require more training effort, but the 30 `sil` states distract the training from this goal. Two alternative `sil` models are compared, one is 3 `sil` targets corresponding to the 3 states of the HTK LVCSR silence model in Section 2.8, and the other is to keep only one `sil` target. From the results presented in Table 3.2, one `sil` target worked best that matches the analysis presented above. Therefore, the single `sil` target is used by all BN DNNs thereafter.

BN layer settings

When training a BN DNN for use in a tandem system, the BN layer position and size also matter. Different BN layer sizes have been investigated in previous studies, and two configurations, 26-dimensional and 39-dimensional, were commonly used (Grézl and Fousek, 2008; Park et al., 2011). (Grézl et al., 2007) suggested to put the BN

Table 3.3 Babel Cantonese FLP BN GMM-HMM system trigram LM CERs with different BN layer positions and sizes.

DNN Structure	`sil` targets	%CER
$468 \times 1000 \times 1000 \times 26 \times 1000 \times 1000 \times 126$	30	55.9
$468 \times 1000 \times 1000 \times 1000 \times 26 \times 1000 \times 126$	30	55.1
$468 \times 1000 \times 1000 \times 1000 \times 1000 \times 26 \times 126$	30	55.0
$468 \times 1000 \times 1000 \times 1000 \times 1000 \times 39 \times 126$	30	54.7

Table 3.4 MGB 200h BN GMM-HMM system 64k word 4-gram LM WERs on Dev.sub with different BN layer positions and sizes.

DNN Structure	%WER
$468 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 26 \times 6027$	33.2
$468 \times 1000 \times 1000 \times 1000 \times 1000 \times 26 \times 1000 \times 6027$	32.8
$468 \times 1000 \times 1000 \times 1000 \times 1000 \times 39 \times 1000 \times 6027$	32.2

layer as the middle hidden layer. This BN layer position is compared with alternative positions using the Babel Cantonese FLP data, and the configuration used is the same as in Section 3.10.1. The results in Table 3.3 show that it is better to use the BN layer as the last or second to the last hidden layer, which is probably because these layers are closer to the output layer and are more discriminantive for classifying the monophone targets. Furthermore, 39-dimensional BN features worked better than 26-dimensional, but also gave rise to more parameters in both BN DNN and GMMs. Further increasing the BN layer size resulted in no obvious improvements.

Similar experiments were conducted using the multi-genre broadcast (MGB) 200h tandem systems. The systems were tested on the Dev.sub test set with a trigram LM based on the 64k vocabulary, and the GMM-HMMs had 6052 tied states and 16 Gaussian mixture components/state. Details of the MGB 200h data set, test set, LM, dictionary, and DNN training configuration are listed in Appendix A.2. From the results in Table 3.4, it can be seen for the tied state targets, it is better to keep a hidden layer after the BN layer, which is probably because classifying 6000 targets requires more information than that can be represented by a 26-dimensional vector. Note that the BN layer position change resulted in a large increase in the number of DNN parameters, from 4.7M to 9.5M. When increasing the BN layer size from 26 to 39, like in Table 3.3, the WER is also decreased. Therefore, the BN layer setting of the 3rd system in Table 3.4 is used in the rest of the thesis.

Table 3.5 Babel Cantonese FLP CD-DNN-HMM system trigram LM CERs with different non-speech modelling methods.

DNN Structure	`sil` targets	%CER
$504 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 6082$	30	46.8
$504 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 6055$	3	46.5
$504 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 6053$	1	47.4

3.10.3 Hybrid baseline system

Silence modelling

In Section 3.10.2, it is shown that the choice of silence modelling is important to CD BN DNN training. Here, the same question is investigated with Babel Cantonese FLP hybrid systems. The CD DNNs were trained with 30, 3, and 1 `sil` targets, respectively. The 30 `sil` targets and 3 `sil` targets are associated with the HMM states in the 30 state silence model and the 3 state HTK LVCSR silence model. The 1 `sil` target DNN is used with a modified HTK LVCSR silence model with the 3 `sil` states tied together. Table 3.5 shows that the DNN with 3 `sil` targets had the lowest CER. It is different to the BN DNN conclusion for tandem systems in Section 3.10.2. This is perhaps because maintaining 3 distinct states is important to both `sil` and `sp` HMMs to model different types of non-speech events. It is not an issue to BN DNNs since it is handled by the tandem GMM-HMMs. In the rest of this thesis, all DNN acoustic models have 3 `sil` targets. It is worth noting that in Table 3.5, the DNN input layer size is 504 rather than 468 since it is based on tandem features. These systems are called *stacked hybrid systems* since they stack a BN DNN for feature extraction and a CD DNN acoustic model.

Input features

Next, hybrid systems with different input features are investigated on the Babel Cantonese FLP data set. In Table 3.6, both PLP input hybrid systems and stacked hybrid systems are compared. It is not surprising that stacked hybrid system worked better than using PLP input features since tandem features were shown to be more useful than PLP features for GMM-HMMs. Furthermore, using CMLLR to normalise the speaker variations resulted in clear improvements to both features[1], and the BN

[1]When CMLLR is applied to the hybrid systems, HLDA and STC are also used to decorrelate the PLP and BN features first.

Table 3.6 Babel Cantonese FLP CD-DNN-HMM system trigram LM CERs with different input feature transforms.

Input Feature			%CER
$\mathbf{o}(t)$	$\mathbf{y}^{\text{bn}}(t)$	CMLLR	
√	×	×	50.8
√	√	×	48.9
√	×	√	47.0
√	√	√	46.5

Table 3.7 MGB 200h 64k word 4-gram LM WERs on Dev.sub, produced by BN GMM-HMMs, CD-DNN-HMMs, and their combination via joint decoding.

System		%WER
BN GMM-HMMs	MPE	29.5
CD-DNN-HMMs	MPE	28.6
BN GMM-HMMs ⊗ CD-DNN-HMMs	$(0.4, 1.0)$	27.4

DNN and CMLLR transforms brought reductions in CER since the hybrid system with tandem speaker adaptive training (SAT) features had the minimum CER.

3.10.4 Joint decoding system

A *joint decoding* system has been developed to linearly combine the log-likelihoods from many acoustic models. For example, when combining a tandem acoustic model, Λ^{tan}, and a hybrid acoustic model, Λ^{hyb}, the combined score is

$$c^{\text{tan}} \ln p(\mathbf{O}|\Lambda^{\text{tan}}) + c^{\text{hyb}} \ln p(\mathbf{O}|\Lambda^{\text{hyb}}), \tag{3.52}$$

where $(c^{\text{tan}}, c^{\text{hyb}})$ is the pre-determined combination weight pair[1]. The score is treated as the acoustic model log-likelihood in the normal first pass decoding. The same method has been applied to multi-level LM combination (Liu et al., 2010).

The results of an example of SI tandem and SI hybrid system combination are presented in Table 3.7. Both systems were built with 200h MGB data. The system construction configuration is listed in Appendix A.2. The decoding uses the Dev.sub test set with a 4-gram LM built on a 64k vocabulary.

[1] In joint decoding, the weighted sums according to $(c^{\text{tan}}, c^{\text{hyb}})$ of the grammar scaling factors and pruning beam widths of the component systems are used.

3.11 Joint Training Methods

3.11.1 ASR system joint training

As the final part of this chapter, ASR training is reviewed from a deep learning perspective. To train an ASR system, the feature extraction module is often built first. Afterwards, the acoustic models are trained based on the pre-generated features which is equivalent to freezing all feature extraction module parameters in training. Next, independent from the other ASR modules, the LM is often trained based on its own text-only corpus, in order to use as much data as possible. Based on this procedure, three weaknesses of the classical ASR approach can be drawn.

- The three modules are individually trained with different criteria and approaches, which is often not a good match to the ASR evaluation criterion, e.g., WER.
- The acoustic model is trained based on features extracted from a pretrained model. Similar to generative PT, this is again a greedy strategy and can result in sub-optimal solutions.
- It is assumed that there is no dependency between the low level acoustic information and high level language information in modelling, which is obviously not true since the decoder combines them both to make decisions.

In order to overcome these issues, an *ASR system level joint training* approach is proposed to optimise the feature extraction and acoustic model modules together in an established system, and MPE is used as the joint training criterion to reduce the mismatch between the training and evaluation criteria. In joint training the feature extraction models and acoustic models are viewed as a single deep ANN with flexible structures, as proposed in Section 3.2 and 3.4. Analogous to DNN training, the conventional module-by-module ASR construction procedure serves as the PT stage while the joint training is the FT stage. Note that the LM module is not involved in joint training in this thesis and is proposed for future work.

3.11.2 DNN acoustic model joint training

Joint training can be applied within a module as well, which removes factors that impose biases in searching the model space and count against finding the optimal solution. Considering a standard CD-DNN acoustic model, the tied state targets are normally derived from the decision trees of a GMM-HMM system, and the training

data alignments are produced by another system. In Chapter 4, standalone training is proposed as an attempt to remove dependencies on the additional systems for decision tree and alignments. Furthermore, DNNs usually have pre-determined forms of hidden activation functions, which is also a factor that results in biased model training. As proposed in Chapter 5, this factor can be removed by introducing additional parameters into a DNN to control the hidden activation function shapes. Both of these methods can be seen as joint training instances.

3.11.3 Related work

There are many other studies related to ASR system level joint training. Pioneering work based on GMM-HMM acoustic models included applying MCE to train both FBANK scaling factors and acoustic model parameters (Biem et al., 2002), MPE for feature transform learning (Povey et al., 2005), as well as SAT with CMLLR (Gales, 1998). For ANN based approaches, (Veselý et al., 2011) trained two MLPs for feature extraction and acoustic model jointly based on the CE criterion. (Gao et al., 2015) applied a similar approach to DNNs, with the feature extraction DNN initialised to enhance noisy speech. CLDNN-HMM acoustic model could be seen as to use the CNN layers to extract better features for succeeding layers (Sainath et al., 2015a,b). Furthermore, (Sainath et al., 2015b; Tüske et al., 2014) showed the CNN layers could even learn data representations similar to the standard filterbank features directly from the raw waveform input. More recently, efforts have been made to jointly train more than feature extraction and acoustic model modules. CTC end-to-end training uses RNNs with character output targets to attempt to remove the dictionary and LM (Graves and Jaitly, 2014). Alternatively, the *encoder-decoder approach* "encodes" acoustic information into segment level context vectors using an encoder RNN, and then decodes the information into words or characters with a decoder RNN, and the phonetic and language knowledge are implicitly modelled by the decoder (Bahdanau et al., 2016; Lu et al., 2015). Note that these end-to-end approaches in some cases are beyond the stochastic ASR framework. Moreover, instead of a single criterion, training towards multiple objectives multi-task learning has also been applied in joint training. (Qian et al., 2016; Yin et al., 2016) generated factors to represent speaker and environment *etc.*, and trained the factor extractors together with the acoustic model on different objectives. In the encoder-decoder approach, (Kim et al., 2016) smoothed the encoder training by CTC to improve the alignments in noisy data.

It should be noted that, in this thesis, ASR system level joint training relies on initial values from the traditional individually constructed ASR modules. It is

debatable whether having such an initialisation stage is important. On one hand, (Sainath et al., 2015b) has reported that with thousands of hours of training data that, random initialisation can be effective for deep ANN training. Actually, in a similar way to the difference between the second and third discriminative PT methods mentioned in Section 3.8.3, initialisation based on over trained sub-optimal models can give rise to smaller joint training improvements. On the other hand, the importance of initialisation for training using a small amount of data should be noted (Glorot and Bengio, 2010; LeCun et al., 1998b; Sutskever et al., 2013). Some parameters, such as the CD states, are even hard to initialise randomly. Therefore, the parameters obtained through standard ASR construction procedures can be used as initial values for joint training. Furthermore, real world ASR systems normally rely on rather complicated pipelines comprising of extra pre-processing and post-processing stages, and it is possible to view an existing pipeline as an initialised system and apply joint training to it. Therefore, the conventional ASR construction stage is used prior to joint training for the aforementioned reasons, but each module is not individually trained to maximise its performance to avoid reducing the effectiveness of subsequent joint training.

Chapter 4

DNN Acoustic Model Standalone Training

This section introduces a method for cross entropy (CE) DNN training that can produce the data alignments and tied context-dependent (CD) state targets itself, and is therefore termed *standalone training*. The motivation for standalone training is to remove the reliances on additional systems to make the DNN training self-contained. Compared to other alignment integrated training methods (Graves et al., 2006; Povey et al., 2016), standalone training is easy to implement as it depends on separate Viterbi alignment procedures rather than an integrated forward-backward algorithm (FB), but will perhaps result in worse WERs. Actually alignment integrated training approaches often require a large amount of training data to outperform the traditional method with the alignments from a high performance system, due to the difficulty in alignment learning (Pundak and Sainath, 2016).

This section is organised as follows: At first, context-independent (CI) target DNN training is presented that starts with uniformly segmented alignments and then gradually improves them during the entire training procedure. For CD standalone training, the GMM based decision tree tying is modified to fit in with DNN models and produce the desired tied CD states. As seen in Section 3.11.2, these proposed algorithms remove the dependencies on additional systems in CD-DNN construction, and can be seen as a DNN acoustic model joint training approach. Moreover, a method that initialises CD-DNN with a CI-DNN is found useful in standalone training, which is then investigated and applied to standard CD-DNN construction as an alternative PT method.

4.1 CI-DNN-HMM Standalone Training

CE DNN training depends on the availability of labels for all frames, often acquired by forced alignment with a high performance GMM-HMM system (Dahl et al., 2012; Hinton et al., 2012). To eliminate such reliance, DNN-HMMs should be able to align the reference transcriptions themselves.

4.1.1 Initial alignment refinement

To train CI-DNN-HMMs, the CI state level transcriptions are generated from the word transcriptions. This is done by expanding every word to CI phones according to its first pronunciation in the dictionary, and then replacing every CI phone with the corresponding HMM states.

In order to align the CI state transcriptions without relying on an existing GMM-HMM system, an idea analogous to the flat-start initialisation strategy used in GMM-HMM training (see Section 2.8) is employed, i.e., every state in an utterance is assigned an equal duration in the initial alignment. During the DNN-HMM training procedure, the data are repeatedly realigned based on the word transcriptions. These initial uniformly segmented transcriptions are called *flat initial alignments*.

Since the states and frames are usually poorly aligned in the flat initial alignments, the alignments are refined by the following steps:

1. train a 2-layer MLP with flat initial alignments for one epoch using error back-propagation (EBP);
2. use the current MLP to realign the training set;
3. use the realignments to train a new 2-layer MLP from scratch for one epoch using EBP;
4. repeat steps 2-4 for a number of iterations.

The above steps are similar to those used to obtain iterative Viterbi alignments in (Bourlard and Morgan, 1993). A major difference is that 2-layer MLPs are trained from scratch in order to avoid the problem caused by bad initial alignments (Trentin and Gori, 2003). After refinement, the alignments are used for discriminative PT.

4.1.2 Discriminative PT with realignment

Instead of the conventional layer-wise discriminative PT (as described in Section 3.8.3), *discriminative PT with realignment* is proposed to build CI-DNNs. With this method,

the data are realigned each time a new hidden layer is trained with EBP, to refine the training labels and to increase their match with the specific hidden layers. The steps are:

1. train a 2-layer MLP with the initial alignments for one epoch, and use the MLP to realign the data;
2. replace the current output layer with a hidden layer along with a new output layer;
3. train the modified MLP with the latest alignments for one epoch;
4. use the MLP to realign the training data transcriptions;
5. repeat steps 2-5 until the planned DNN structure is realised.

After PT, all DNN layers are jointly trained by EBP to fine-tune the model parameters. After these steps, the required CI-DNN is trained. It was found that realigning the data and retraining new CI-DNN-HMMs from scratch with conventional PT and FT methods could further improve the performance (Dahl et al., 2012).

4.2 Target Clustering for CD-DNN-HMMs

After the estimation of CI-DNN-HMMs, the resulted models are used to produce HMM state to frame alignments, whose monophone state labels can be modified to triphone state labels. In order to construct a CD-DNN acoustic model with tied state targets without relying on any additional systems, the CD states, either seen or unseen in the training set, should be tied based on DNN-HMMs rather than GMM-HMMs.

4.2.1 Class-conditional distribution interpretation

Since decision tree tying clusters the output probability density functions of the states, to modify the algorithm for DNN-HMMs, the equivalent class-conditional distributions from the DNN are needed. Like the original GMM-HMM based algorithm described in Section 2.5, here the class-conditional distributions $p(\mathbf{x}(t)|C_k)$ are assumed to be Gaussian[1]. If all untied Gaussian distributions have the same covariance matrix, i.e.,

[1]Note that $\mathbf{x}(t)$ in this subsection specifies the input vector to the output layer, $\mathbf{x}^{\text{out}}(t)$, and the superscript is omitted to save space.

$p(\mathbf{x}(t)|C_k) = \mathcal{N}(\mathbf{x}(t)|\mu_k, \Sigma)$, from Eqn. (3.10), we have

$$\begin{aligned} p(C_k|\mathbf{x}(t)) =& \frac{\exp\left(-0.5\mathbf{x}(t)^{\mathrm{T}}\Sigma^{-1}\mathbf{x}(t) + \mu_k^{\mathrm{T}}\Sigma^{-1}\mathbf{x}(t) - 0.5\mu_k^{\mathrm{T}}\Sigma^{-1}\mu_k + \ln P(C_k)\right)}{\sum_{k'}\exp\left(-0.5\mathbf{x}(t)^{\mathrm{T}}\Sigma^{-1}\mathbf{x}(t) + \mu_{k'}^{\mathrm{T}}\Sigma^{-1}\mathbf{x}(t) - 0.5\mu_{k'}^{\mathrm{T}}\Sigma^{-1}\mu_{k'} + \ln P(C_{k'})\right)} \\ =& \frac{\exp\left(\mu_k^{\mathrm{T}}\Sigma^{-1}\mathbf{x}(t) - 0.5\mu_k^{\mathrm{T}}\Sigma^{-1}\mu_k + \ln P(C_k)\right)}{\sum_{k'}\exp\left(\mu_{k'}^{\mathrm{T}}\Sigma^{-1}\mathbf{x}(t) - 0.5\mu_{k'}^{\mathrm{T}}\Sigma^{-1}\mu_{k'} + \ln P(C_{k'})\right)}. \end{aligned} \tag{4.1}$$

Meanwhile, by taking Eqn. (3.2) into Eqn. (3.9), we can get

$$p(C_k|\mathbf{x}(t)) = \frac{\exp\left(\mathbf{w}_k^{\mathrm{T}}\mathbf{x}(t) + b_k\right)}{\sum_{k'}\exp\left(\mathbf{w}_{k'}^{\mathrm{T}}\mathbf{x}(t) + b_{k'}\right)}. \tag{4.2}$$

Consequently, the relationship between parameters of the Gaussian distributions (means and variances) and those of the DNN output layer can be obtained as

$$\alpha\,\mathbf{w}_k^{\mathrm{T}} = \mu_k^{\mathrm{T}}\Sigma^{-1} \tag{4.3}$$

$$\alpha\,b_k = -0.5\mu_k^{\mathrm{T}}\Sigma^{-1}\mu_k + \ln P(C_k), \tag{4.4}$$

where α is any non-zero real number. Eqns. (4.3) and (4.4) can be used to generate a DNN layer from known distributions. This relation between equal covariance Gaussian distributions and the linear discriminant function was first described in (Duda and Hart, 1973).

Since the output densities are estimated based on $\mathbf{x}^{\text{out}}$, the DNN-HMM based method clusters in the space of $\mathbf{x}^{\text{out}}$, $\Omega_{\mathbf{x}^{\text{out}}}$, while the GMM-HMM based decision tree state tying method clusters in the space of the original observations $\mathbf{o}$, $\Omega_{\mathbf{o}}$.

4.2.2 DNN-HMM based decision tree target clustering

With the CI-DNN-HMM system obtained in Section 4.1, the CD states are clustered in the space of $\mathbf{x}^{\text{out}}(t)$ generated by the last hidden layer of the CI-DNN, $\Omega^{\text{CI}}_{\mathbf{x}^{\text{out}}}$. The major steps of the modified method include

1. Realign the training set with CI-DNN-HMMs.
2. Estimate Gaussian distributions for the CD targets based on the output vectors $\mathbf{x}^{\text{out}}(t)$ from the last CI-DNN hidden layer.

3. Convert the Gaussians obtained in the last step to a new CD output layer using Eqns. (4.3) and (4.4), and add it onto the CI-DNN hidden layers.

4. Train the new CD output layer and use the resulted model to realign the training set.

5. Perform FT over the CD-DNN obtained in step 4 according to the new alignments it produced.

The other parts of the method follow the standard GMM-HMM based state tying, as introduced in Section 2.5. Compared to the traditional GMM decision tree state tying based on the standard acoustic feature, the DNN based decision trees perform clustering based on the output vectors from the last CI-DNN hidden layer. Though the output vectors from the last hidden layer of the CI-DNN still mismatch with those from the last CD-DNN hidden layer due to the difference in the targets and training alignments, but they might be better approximations than the acoustic feature vectors since the CI and CD hidden layers are similarly configured. Approaches to estimate the Gaussian distributions in step 2 are going to be presented next.

4.2.3 Distribution estimation based on hidden activations

To obtain the input to the decision tree clustering, the output densities for the untied states are required. The untied states together with their training labels are obtained by expanding the CI states with their neighbouring phones, using the alignments generated by the CI-DNN-HMMs. Then the parameters of the distributions, i.e., the mean vectors and the common covariance matrix, are estimated based on the maximum likelihood (ML) criterion. Eqns. (2.38) and (2.39) are modified to

$$\mu_k = \frac{\sum_t \gamma_k(t)\mathbf{x}^{\text{out}}(t)}{\sum_t \gamma_k(t)} \tag{4.5}$$

$$\Sigma = \frac{\sum_k \sum_t \gamma_k(t)(\mathbf{x}^{\text{out}}(t) - \mu_k)(\mathbf{x}^{\text{out}}(t) - \mu_k)^{\text{T}}}{\sum_k \sum_t \gamma_k(t)}. \tag{4.6}$$

Note that for alignments, $\gamma_k(t)$ are 0-1 valued hard probabilities which are assigned according to the frame label, and the statistics for Σ are summed over all targets.

Since $\mathbf{\Sigma}$ is usually a large full matrix, its determinant, used to get the log-likelihood by Eqn. (2.62), is hard to compute. Therefore, Σ is transformed to a diagonal matrix using a rotation, i.e., the orthonormal matrix $\mathbf{A}$ whose columns are the eigenvectors of Σ. According to Eqn. (2.68), $\mathbf{A}$ transforms the untied Gaussians to have a common

diagonal covariance matrix by

$$p(\mathbf{x}^{\text{out}}(t)|C_k) \propto \mathcal{N}(\mathbf{A}^{\text{T}}\mathbf{x}^{\text{out}}(t); \mathbf{A}^{\text{T}}\mu_k, \mathbf{A}^{\text{T}}\Sigma\mathbf{A}). \tag{4.7}$$

Furthermore, to reduce the dimension of $\mathbf{A}^{\text{T}}\Sigma\mathbf{A}$ and speed up computation, some columns of $\mathbf{A}$ with very small eigenvalues can be discarded.

4.2.4 Statistics collection and CD-DNN construction

After the parameters of the class-conditional distributions have been determined, they are converted into an output layer with untied state targets using Eqns. (4.3) and (4.4). This new output layer is then added in place of the original output layer of the CI-DNN-HMM, and used to collect the statistics $\sum_t P(C_k|\mathbf{x}^{\text{in}}(t))$, serving as the term of $\sum_t \gamma_k(t)$ used for decision tree clustering in Eqns. (2.62) and (2.63). If the new output layer is converted from diagonal Gaussians transformed by $\mathbf{A}$, then to take decorrelated inputs, $\mathbf{A}^{\text{T}}$ with a zero bias vector should be treated as an extra layer with the linear hidden activation function and interposed between the output layer and the existing hidden layers.

After clustering, the output layer with the newly clustered targets is added to the hidden layers of the CI-DNN-HMMs. The hidden layer weights are fixed and only the new output layer is trained. After this step, the resulting CD-DNN-HMMs are used to realign the training set, and FT is applied according to the realignments. The resulting CD-DNN-HMMs are the required models.

If we denote $\Omega^{\text{CD}}_{\mathbf{x}^{\text{out}}}$ as the space of input vectors to the output layer of the final CD-DNN-HMMs, the DNN-HMM based target clustering makes predictions about the best targets of $\Omega^{\text{CD}}_{\mathbf{x}^{\text{out}}}$ in $\Omega^{\text{CI}}_{\mathbf{x}^{\text{out}}}$. In contrast, the GMM-HMM based state tying predicts CD state targets in $\Omega_{\mathbf{o}}$.

4.3 Standalone Training Experiments

This section presents comparisons between the standard CE DNN training and standalone training methods. The experiments are conducted using the WSJ corpus with the SI-284 setup. Details of the corpus and DNN system configurations are listed in Appendix A.3.

Table 4.1 WSJ SI-284 setup baseline systems with a 65k word trigram LM.

ID	System	Alignments	%WER	
			Dev	Eval
G1	ML GMM-HMMs		9.1	9.5
G2	MPE GMM-HMMs		8.0	8.7
I1	CI-DNN-HMMs	G2	10.5	12.0
I2	CI-DNN-HMMs	I1	10.7	13.7
D1	CD-DNN-HMMs	G2	6.7	8.1

4.3.1 Baseline system performance

CI-DNN-HMM (I1) and CD-DNN-HMM (D1) baseline systems were built with conventional discriminative PT using the labels derived from the alignments generated by an HLDA minimum phone error (MPE) GMM-HMM system (G2), which have the DNN structures of $351 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 138$ and $351 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 5981$, respectively. The triphone tied state targets were generated by the GMM-HMM based decision tree tying approach, and the GMM-HMM system had 5,981 tied states and 12 Gaussian components/state. I1 was used to realign the data and another CI-DNN-HMM baseline (I2) with the same configuration was trained from scratch based on the realignments. The baseline performance is listed in Table 4.1. Results of GMM-HMMs are also included as a comparison to previous work (Povey, 2003; Woodland et al., 1995).

4.3.2 CI system standalone training

The flat initial alignments were first refined for 20 iterations. Afterwards, several CI-DNN-HMM systems were trained with different PT and conventional FT based on the alignments generated. One system (I3) was built using discriminative PT with realignment, and another (I4) was later built with conventional discriminative PT based on the alignments generated by I3. For comparison, one CI-DNN-HMM system (I5) was trained with conventional discriminative PT, whose realignments were used to build another set of CI-DNN-HMMs (I6) from scratch. The performance of above systems is presented in Table 4.2.

Comparing I3 with I5 and I4 with I6, we can see that systems with discriminative PT with realignment gave on average a 2.3% and a 4.0% relative reduction in WER (on Dev and Eval combined). Retraining the systems from scratch for more passes caused the performance to fluctuate. As for I4 and I1, although I4 performed more poorly than

Table 4.2 WSJ SI-284 system results of CI-DNN-HMMs with conventional PT or PT with realignment, using a 65k word trigram LM. The "PT Procedure" column listed the PT methods used in system construction.

ID	PT Procedure	%WER	
		Dev	Eval
I3	Realignment	12.2	14.3
I4	Realignment + Conventional	11.7	13.8
I5	Conventional	12.2	15.0
I6	Conventional + Conventional	12.0	14.6

I1, its results were achieved without information from _T features, HLDA transforms, and CD modelling embedded in the alignments. This conclusion is also supported by system I2. I2 suffered from an 8% averaged relative WER increase compared to I1, since it excluded the above information.

4.3.3 CD system standalone training

The difference between GMM-HMM and DNN-HMM based decision tree state tying is now investigated. The experiments started from the best standalone CI-DNN-HMMs, I4. A total of 68,172 untied triphone states occurred in the alignments generated by I4, in contrast to 68,034 untied states involved in GMM-HMM based state tying. The new output layer converted from the Gaussians estimated using the I4 alignments gave a CD frame classification accuracy of 35.3% on the combination of the training and held out sets. The common covariance matrix was decorrelated with a rotation and kept 300 dimensions (accounting for 96% of the variance) with the largest eigenvalues in the transformed diagonal covariance matrix. 5,996 tied states were generated by DNN-HMM based decision tree clustering. These clustered targets were added to the I4 hidden layers. The effect of different clusterings was examined by using EBP either through all layers or through the output layer only, based on the training labels derived from I4 alignments. The results are given in Table 4.3, where the structures of the CD-DNNs clustered by the GMM-HMM and DNN-HMM based approaches are $351 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 5981$ and $351 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 5996$, respectively.

From the results, D2 slightly outperformed G3 (1% averaged relative WER reduction), which indicates that the tied states clustered in $\Omega^{\text{CI}}_{\mathbf{x}^{\text{out}}}$ (of I4) match the existing I4 hidden layers better than those clustered in $\Omega_{\mathbf{o}}$. Meanwhile, if all layers were trained

Table 4.3 WSJ SI-284 system comparison between GMM-HMM and DNN-HMM based state tying using a 65k word trigram LM.

ID	Clustering	EBP Layers	%WER	
			Dev	Eval
G3	GMM-HMM	Output Layer	7.6	9.0
G4	GMM-HMM	All Layers	6.8	7.9
D2	DNN-HMM	Output Layer	7.7	8.7
D3	DNN-HMM	All Layers	6.8	7.8

by EBP, D3 only outperformed G4 by 0.6% relative WER. The performance difference between G4 and D3 is reduced, compared to that between G3 and D2, due to the power of FT, which not only changes the output layer weights but also changes their input features $\mathbf{x}^{\text{out}}(t)$. The small WER difference is possibly the reason why GMM instead of DNN based decision trees are still commonly used, since the GMM based decision tree software is widely used and it is an essential component for building initial CD-GMM-HMM.

Compared to the baseline CD-DNN-HMMs D1, the CD-DNN-HMMs trained in a standalone fashion, D3, performed 1.5% relatively poorer on Dev but 3.7% relatively better on Eval in terms of WER. As a result, the task of training a state-of-the-art CD-DNN-HMM system without relying on any GMM-HMMs has been accomplished. In addition, the proposed training procedure is efficient since training and aligning the data with CI-DNN-HMMs can be much faster than with CD-DNN-HMMs.

4.4 CI Discriminative PT

In this section, an approach to pretrain CD-DNN models discriminatively with CI state targets is proposed, which serves as a regulariser to prevent the DNN hidden layers from over-fitting to a specific CD target set. Furthermore, the CI state classification accuracy based on CD state conversion is used to compare DNNs with different target sets at the frame level.

4.4.1 CI initialisation for CD-DNNs

Studies have shown that a key advantage of using deep models is to have multiple layers of non-linear feature transforms with different functions (Mohamed et al., 2012). For speech recognition, it is found that lower DNN layers model low level characteristics,

for example, partially eliminating the speaker dependencies from phones, while higher layers capture highly non-linear structures, for instance, those useful for phoneme discrimination (Mohamed et al., 2012). Therefore, when training a DNN, it is desirable to keep lower layers more general and higher layers more target specific. One way to achieve this is to use hybrid PT, which can be seen as a linear interpolation between jointly optimised generative and discriminative RBM stacks (Sainath et al., 2013a). Alternatively, one can use a CI-DNN to initialise a CD-DNN, which is termed as the *CI initialisation* method.

The steps of using CI initialisation for CD-DNNs are as follows:

1. Discriminative PT with CI Targets
   Discriminative PT to generate an initial DNN with CI state targets;

2. FT with CI Targets
   Train the initial CI-DNN for multiple epochs;

3. Target Swap
   Replace the CI state output layer with a randomly initialised output layer with desired CD targets;

After PT, the initial CD-DNN model is trained to full convergence by FT as usual. In the above procedure, step 1 and 2 are standard CI-DNN discriminative PT and FT, which results in far more updates of the initial models compared to CD discriminative PT. This is appropriate since a CI-DNN usually has far fewer parameters and is less likely to over-fit to the training data. Actually, as shown later in the experiments in Section 4.5.1, step 2 is not essential for CI initialisation to outperform the standard methods.

It is worth noting that the idea of using layers from a CI-DNN to initialise a CD-DNN is similar to the hidden layer sharing approach used by multi-lingual tandem and hybrid systems (Ghoshal et al., 2013; Heigold et al., 2013; Huang et al., 2013; Swietojanski et al., 2012). However, the multi-lingual studies share the limited training data across different languages by using common hidden layers and language specific output layers, while CI initialisation is to ensure that the hidden layers are generic enough for low level data transformations. All these methods actually fall into the multi-task learning framework, which has been introduced in Section 3.7.

4.4.2 CI state classification accuracy

A problem arising from CI initialisation is that the commonly used DNN acoustic model performance indicator, the frame classification accuracy, is not available during the complete DNN training process since different target sets are involved. To solve this problem, each CD state is associated with a CI state by stripping its contexts, and the CD state results and labels are converted at the CI state level. Let $\mathbf{C}_l$ be a set of CD states, and $C_k \in \mathbf{C}_l$ means that C_k's center phone and state index constitute the CI state that $\mathbf{C}_l$ represents. Therefore, the posterior probability for the CI state $\mathbf{C}_l$ is found by summing over all of the CD instances C_k

$$P(\mathbf{C}_l|\mathbf{x}^{\text{in}}(t)) = \sum_{C_k \in \mathbf{C}_l} P(C_k|\mathbf{x}^{\text{in}}(t)). \tag{4.8}$$

Then in classification, the CI state that gives the maximum accumulated posterior probability is chosen and compared to its reference label.

Applying Bayes' rule to Eqn. (4.8) leads to,

$$p(\mathbf{x}^{\text{in}}(t)|\mathbf{C}_l) = \sum_{C_k \in \mathbf{C}_l} \frac{P(C_k)}{P(\mathbf{C}_l)} p(\mathbf{x}^{\text{in}}(t)|C_k). \tag{4.9}$$

As in Section 4.2.1, both $p(\mathbf{x}^{\text{in}}(t)|\mathbf{C}_l)$ and $p(\mathbf{x}^{\text{in}}(t)|C_k)$ are generated through softmax functions that have equivalent Gaussian distributions. Therefore, the conversion actually regards the single Gaussian model for each CD state as a mixture component in the GMM for its corresponding CI state, and $P(C_k)/P(\mathbf{C}_l)$ is the mixture weight with

$$\sum_{C_k \in \mathbf{C}_l} \frac{P(C_k)}{P(\mathbf{C}_l)} = 1. \tag{4.10}$$

4.5 CI Initialisation Experiments

In this section, experiments are first carried out on the WSJ data set to compare CI initialisation with the standard generative and discriminative PT methods. On the Aurora-4 data set, the regularisation effect of CI initialisation is studied by comparing it to weight decay.

Table 4.4 WSJ SI-84 DNN-HMM system recognition and classification results with a 65k word trigram LM.

ID	System	PT	%WER		CI State
			Dev	Eval	CV %Acc.
S1	CI-DNN-HMMs	CI	14.6	16.6	67.2
S2	CD-DNN-HMMs	RBM	9.4	10.9	68.9
S3	CD-DNN-HMMs	CD	9.6	11.3	68.7
S4	CD-DNN-HMMs	CI (no FT)	8.9	10.3	69.7
S5	CD-DNN-HMMs	CI	8.4	10.0	70.2

4.5.1 WSJ SI-84 DNN-HMM system performance

The first experiments were conducted with the WSJ SI-84 setup, which has a fairly limited number of training samples. The CI-DNN, S1, had 138 CI states, which were associated with 46 phones, while all CD-DNN-HMMs, S2-S5, had 3,007 tied states produced by the GMM-HMM based decision tree state tying approach. The DNNs had 5 hidden layers all with 1,000 artificial neurons. Baseline CD-DNN-HMM systems, S2 and S3, were initialised by generative and traditional CD discriminative PT, respectively. S5 was the model initialised by the proposed CI discriminative PT, whose starting point was actually S1. S4 differed from S5 by removing the CI-DNN FT step from CI discriminative PT (step 2 listed in Section 4.4.1). Performance of these systems is presented in Table 4.4.

From Table 4.4, S5, the CD-DNN-HMMs initialised by the proposed CI initialisation, had the lowest WER among all SI-84 systems. Compared to the baselines with generative and CD discriminative PT, S2 and S3, S5 gave on average a 9.4% and a 12.0% relative reduction in WER (on Dev and Eval combined). Furthermore, by comparing S4 to S5, removing the CI-DNN FT step degraded the performance, but S4 still outperformed S2 and S3, which had the same total number of epochs in training, by an average of 5.7% and 8.1% relative reduction in WER. These results reveal that although the CI-DNN FT step could help reduce WER, performing more epochs of training is not the only reason for CI initialisation to outperform CD discriminative PT. Moreover, the CI state accuracies of the CD-DNNs, computed by Eqn. (4.8) on the CV set, are also included. These numbers are consistent with WERs across all SI-84 systems.

Table 4.5 Standard deviations of DNN layer output values on the WSJ SI-84 CV set.

ID	Averaged Standard Deviation	
	1st Hidden Layer	Output Layer
S2	4.19×10^{-1}	9.71×10^{-3}
S3	4.23×10^{-1}	9.46×10^{-3}
S4	4.09×10^{-1}	1.08×10^{-2}

4.5.2 Investigation of DNN layer output values

When classifying CI rather than CD targets, there would be only CI errors generated and backpropagated to the first layer. Therefore, only the CI low level characteristics are modelled. Specifically, if the first layer handles the task of, for instance, low-level feature normalisation, these features can be learned from CI speaker characteristics. After swapping CI targets with CD targets for FT purpose, the CD low level characteristics, e.g., CD pronunciation changes caused by the accent of a particular speaker, will not be modelled by the first layer and can be used by higher layers for better CD target discrimination. Intuitively, if this assumption is true, then for the final CD-DNN with CI discriminative PT, the first layer should produce more general features with smaller variances, while the output layer posterior probabilities should be more discriminative. This assumption is validated using SI-84 trained systems and the corresponding CV set, as shown in Table 4.5.

In Table 4.5, for each system, the standard deviations of the output of each node in the first hidden layer and the output layer were calculated and averaged. First, the baseline with generative PT, S2, had smaller first hidden layer and larger output layer standard deviations than S3 with CD discriminative PT. This matches the fact that generative PT has more general first hidden layers than CD discriminative PT (Hinton et al., 2012; Mohamed et al., 2012). S5 had the smallest first layer and the largest output layer standard deviations, which indicated that CI discriminative PT produced the most generic first hidden layer and the most discriminative output layer, among the three methods.

4.5.3 WSJ SI-284 DNN-HMM system performance

Finally, CI initialisation was applied to the larger data set with the WSJ SI-284 configuration, which contains about 4.5 times more data than SI-84. The CD output layer size is increased from 3,007 to 5,981, making the DNNs have 1.5 times more

Table 4.6 WSJ SI-284 DNN-HMM system recognition and classification results using a 65k word trigram LM. The time cost in terms of seconds of different PT methods is also included.

ID	System	PT			%WER		CI State
		Method	n Epoch	Time	Dev	Eval	CV %Acc.
S11	CI-DNN-HMMs	CI			11.6	12.6	70.5
S12	CD-DNN-HMMs	RBM	8	4562.6	6.9	8.1	73.4
S13[1]	CD-DNN-HMMs	CD	4	8617.1	6.7	8.1	72.5
S14	CD-DNN-HMMs	CI (no FT)	4	1703.1	6.3	7.4	73.4
S15	CD-DNN-HMMs	CI	16	9794.5	6.3	7.4	72.9

parameters. As a result, every DNN parameter has three times more training samples on average. The SI-284 DNN-HMM systems and their performance are presented in Table 4.6.

From Table 4.6, CD-DNN-HMMs with CI initialisations, S14 and S15, still outperformed the baseline systems S12 and S13 by a margin of 8.7% and 7.4% relative reduction in WER. To make the CI state accuracies comparable between Table 4.4 and 4.6, the CI state accuracies in Table 4.6 were still on the SI-84 CV set, since it was included in the SI-284 CV set. The CI state CV accuracies of S12-S15 are appropriate compared to the accuracy of S11, but are not all consistent with their WERs. Furthermore, S14 and S15 had identical WERs, which reveals that CI-DNN FT with sufficient training samples is not as important as when using less data, since the first few hidden layers may have been updated a sufficient number of times to model the low level characteristics well during the CI-DNN PT phase. In this case, CI initialisation not only improves the CD-DNN performance, but also considerably reduces the required amount of training time, as shown in Table 4.6. The experiments in Table 4.6 used a single NVIDIA K20c GPU card, and the RBM time cost was not strictly comparable to the others, since it was trained with a different software, TNet (BUT, 2013).

4.5.4 Aurora-4 DNN-HMM system performance

Next, the regularisation effect of CI initialisation is investigated on the Aurora-4 task. The data set description and system configuration are presented in Appendix A.4. All CD-DNN-HMM systems have a structure of $720 \times 2000 \times 2000 \times 2000 \times 2000 \times 2000 \times 3066$. The CE training alignments were produced by a tandem system trained

Table 4.7 Aurora-4 CD-DNN-HMMs results with different regularisation setups and a 5k bigram LM.

PT	Weight Decay	%WER				
		A	B	C	D	Avg.
CD	0.0	4.2	7.7	9.0	20.2	12.9
CD	0.001	3.6	7.3	7.4	18.3	11.8
CI	0.0	3.8	7.2	8.3	19.5	12.3
CI	0.001	3.8	7.2	7.3	18.0	11.6

with multi-condition data. In Table 4.7, systems were constructed with different combinations of weight decay and discriminative PT. From the results, we can see that regularisation, either by weight decay or by CI initialisation, is necessary for DNN-HMMs to perform well on this data set. The weight decay coefficient was set to 0.001, which was tuned to give the lowest WER and outperformed CI initialisation. In addition, CI initialisation is complementary to weight decay to some degree.

4.6 Summary and Conclusions

Three algorithms were proposed in this chapter: DNN-HMM standalone training based on discriminative PT with realignment, a DNN-HMM based decision tree tying approach, as well as the CI initialisation for CD-DNNs.

CI-DNN-HMM standalone training has the CI-DNN trained in an interleaved fashion by updating the model parameters with the reference labels and updating the labels by realigning the training set. DNN-HMM decision tree tying adapts GMM-HMM decision tree tying by estimating a Gaussian distribution with a common covariance matrix for every untied CD state based on the CI-DNN's last hidden layer output vectors. The clustered CD states can be converted to a CD-DNN output layer. Experiments on the SI-284 training setup for the WSJ corpus have shown that the combined CD-DNN-HMM standalone training procedure gives comparable performance to the standard approach. CI initialisation pretrains the CD-DNNs discriminatively with CI state targets, and serves as a regulariser complementary to the weight decay method. Experiments on WSJ and Aurora-4 data sets have shown that CI initialisation resulted in systems with lower WERs than CD discriminative PT with the training data size up to 65h, and is also more efficient in computation.

An interesting finding worth more discussion lies in the decision tree tying methods. In Table 4.3, it was shown though the DNN-HMM based method outperformed the

traditional GMM-HMM based one, their performance was actually quite close. This is a bit surprising since the traditional method clusters states without taking any factors from DNNs into account. This indicates perhaps the power of CD-DNNs mainly comes from additional parameters and dependencies with the surrounding phones, compared to the CI-DNN. The specific state tying relationship is not crucial as long as the clustering is appropriate and the resulted tied state sets are similar in size, since later DNN training can adjust the hidden layers to fit the output targets. Meanwhile, there are also invalid assumptions in the method that could be further improved. When the decision tree nodes are split, the covariances of the Gaussian distributions are individually estimated while the softmax output layer requires one covariance matrix shared by all Gaussians. Furthermore, the alignments are not fixed in the clustering procedure and ML may not be the most suitable criterion. In general, it should be better to increase the matching degree between the CD targets and the final DNN model, which can probably be achieved by integrating tree construction into the DNN training procedure (Jernite et al., 2016).

4.7 Related Work

There are different DNN-HMM based decision tree tying methods (Bacchiani and Rybach, 2014; Gosztolya et al., 2015; Li and Wu, 2014a; Senior et al., 2014; Zhu et al., 2015). In (Imseng et al., 2013), minimum KLD was used to cluster MLP-HMM states. Zhang and Woodland (2014) and Senior et al. (2014) proposed to use the DNN hidden layer output vectors to estimate Gaussian distributions for ML decision tree tying at the same time. Zhang and Woodland (2014) used full covariance Gaussians connected with the softmax output layer, while Senior et al. (2014) used diagonal Gaussians for both data-driven (Chou, 1991) and manually generated question sets (Young et al., 1994). Shortly afterwards, minimum KLD (Gosztolya et al., 2015), entropy (Zhu et al., 2015), and mean squared error (Li and Wu, 2014a) criteria were also used for DNN decision tree construction, which removes the need for explicit assumptions of the output distributions of the states. Zhu et al. (2015) also found DNN decision trees outperformed GMM decision trees when producing more than 10k clustered states.

Recently, sequence deep ANN training without relying on any pre-existing alignments has drawn much attention. Senior et al. (2014) used the same flat-start and realignment strategy as the method proposed in this chapter, but applied realignment at each minibatch without a PT method. Like GMM-HMMs, Yan et al. (1997), Li and Wu (2014b), and Tüske et al. (2015a) have applied the FB algorithm to find the

state occupancies for ML training from scratch. Alternatively, CTC training maximises posterior probabilities and has been extended to take the hypotheses and their error rates into account (Graves et al., 2006; Graves and Jaitly, 2014). Similar ideas have also been applied to maximum mutual information (MMI) training as well (Gosztolya et al., 2016; Povey et al., 2016; Zhou et al., 2014), and the results obtained were superior to those of the traditional training methods on both small and large training sets (Povey et al., 2016). Moreover, CI initialisation has been extended to a DNN training method with both CI and CD classification (Bell and Renals, 2015a,b). Besides the normal CD target output layer, an additional CI target output layer was aslo added upon the final hidden layer, and both output layers were used in CE training in an interleaved fashion in each minibatch, which trains the shared hidden layers for both CI and CD classification as a multi-task learning. The same method was also applied to speaker adaptation (Swietojanski et al., 2015).

Chapter 5

Learning Hidden Activation Functions

As reviewed in Section 3.8.1, the activation function plays an important role in deep learning, since it has a significant influence on the model optimisation procedure and thus impacts on deep ANN design. Besides the very successful application of sigmoid and ReLU functions in DNN acoustic modelling (Dahl et al., 2013, 2012; Hinton et al., 2012; Maas et al., 2013; Seide et al., 2011b; Tóth, 2013; Zeiler et al., 2013), some more recent research work has studied learning the speaker independent (SI) parameters associated with activation functions (Delcroix et al., 2016; He et al., 2015b; Siniscalchi et al., 2010; Yoshioka et al., 2016; Zhang and Woodland, 2015b). This section proposes generic parameterised forms of sigmoid and ReLU functions, namely the p-Sigmoid and p-ReLU functions, and applies them to construct SI DNN acoustic models. As discussed in Section 3.11.2, this can be seen as jointly training the hardwired activation functions in standard training along with other DNN parameters. Furthermore, both p-Sigmoid and p-ReLU functions are also used for speaker adaptation later in this chapter (Zhang et al., 2016), which falls into the learning hidden unit contribution (LHUC) framework that applies speaker dependent (SD) activation function output amplifiers transformed by various functions to both adaptation and adaptive training (Swietojanski et al., 2016; Swietojanski and Renals, 2014).

5.1 Training Parameterised Activation Functions with error backpropagation (EBP)

In order to distinguish hidden activation functions with differing parameters, Eqn. (3.1) is rewritten as

$$y_i^{(k)}(t) = f_i^{(k)}(a_i^{(k)}(t)), \tag{5.1}$$

for artificial neuron i of layer k, and $\Theta_i^{(k)}$ is the parameter set of $f_i^{(k)}(\cdot)$. Like other DNN parameters, activation function parameters can be shared in different ways, although this has not been investigated in this thesis.

As for other ANN parameters, SGD is used for training activation function parameters. Therefore, calculations of the associated gradient values using the EBP algorithm are required. If $\theta^{(k)}$ is a standard DNN parameter, the derivatives of $\mathcal{F}$ with respect to $\theta^{(k)}$ are obtained by taking Eqn. (5.1) into Eqn. (3.32), and

$$\frac{\partial y_i^{(k)}(t+c)}{\partial a_i^{(k)}(t+c)} = \frac{\partial f_i^{(k)}(a_i^{(k)}(t+c))}{\partial a_i^{(k)}(t+c)} \tag{5.2}$$

needs to be calculated; if $\theta^{(k)}$ is an activation function parameter, we can get

$$\frac{\partial \mathcal{F}}{\partial \theta^{(k)}} = \sum_{l \in \mathbf{s}^{(k)}} \sum_{c \in \mathbf{c}_k^{(l)}} \sum_j \sum_i \frac{\partial \mathcal{F}}{\partial a_j^{(l)}(t+c)} \frac{\partial a_j^{(l)}(t+c)}{\partial y_i^{(k)}(t+c)} \frac{\partial f_i^{(k)}(a_i^{(k)}(t+c))}{\partial \theta^{(k)}}, \tag{5.3}$$

and

$$\frac{\partial y_i^{(k)}(t+c)}{\partial \theta^{(k)}} = \frac{\partial f_i^{(k)}(a_i^{(k)}(t+c))}{\partial \theta^{(k)}} \tag{5.4}$$

is needed, in order to obtain the gradient values with respect to the activation function parameters.

It should be noted that from Eqn. (5.1), activation function parameters can be more effective than weights and biases of the same layer, since they directly impact on $y_i^{(k)}(t)$. However, this also indicates they are more sensitive than other parameters as their value changes may have a bigger influence on the rest of the network. One solution is to constrain and control the learning by non-linearly transforming these parameters (Swietojanski and Renals, 2014). Alternatively, update value clipping can be used (see Section 3.6) to prevent gradient explosion causing training instability.

5.2 Parameterised Activation Functions

5.2.1 Parameterised sigmoid functions

Functions from the sigmoid family have been widely investigated as ANN hidden activation functions for many ASR tasks. Example uses include smoothed forms of the 0-1 classification error (Juang et al., 1997) and for making soft decisions on data selection according to the noise estimation error (Barker et al., 2000).

In this section, the form of sigmoid functions is generalised to become

$$f(a) = \alpha \cdot \frac{1}{1+\exp(-\beta a+\gamma)}, \tag{5.5}$$

which is actually the *logistic function*, and α, β, and γ are the learnable parameters. This generalised form is denoted as defining the p-Sigmoid(α, β, γ).

In the p-Sigmoid function, α, β, and γ have different effects on the curve $f(a)$.

- Among the three parameters, α can make the largest change to the function by scaling $f(a)$ linearly. $|\alpha|$ is the maximum value of $|f(a)|$. Since there is not a constraint imposed on the parameters, α can be any real number. If $\alpha > 0$, α indicates the contribution of the corresponding hidden layer artificial neuron; an artificial neuron is disabled if $\alpha = 0$. It is possible that an artificial neuron can make a negative contribution by allowing $\alpha < 0$.

- β controls the steepness of the curve. When $|\beta|$ increases, $f(a)$ can arbitrarily approximate a step function. When $\beta \to 0$, $f(a)$ makes less difference to the input around 0, and outputs constant values if $\beta = 0$.

- γ applies a horizontal displacement to $f(a)$. γ/β is the x-value of the midpoint of the p-Sigmoid function curve, if β is non-zero.

It is well-known that the p-Sigmoid, or logistic functions, can be used for piecewise approximations to other functions. Figure 5.1 illustrates different shapes of the p-Sigmoid functions for $a \in [-3,3]$, by varying α, β, and γ. p-Sigmoid$(1,30,0)$ is similar to the 0-1 loss function. The illustrated fragment of p-Sigmoid$(4,1,2)$ has a similar shape to that of the soft ReLU activation function. p-Sigmoid$(3,-2,3)$ approximates ReLU, although the approximation is poor around zero. Therefore, if all parameters are learned appropriately, p-Sigmoid units can behave like other commonly used activation functions.

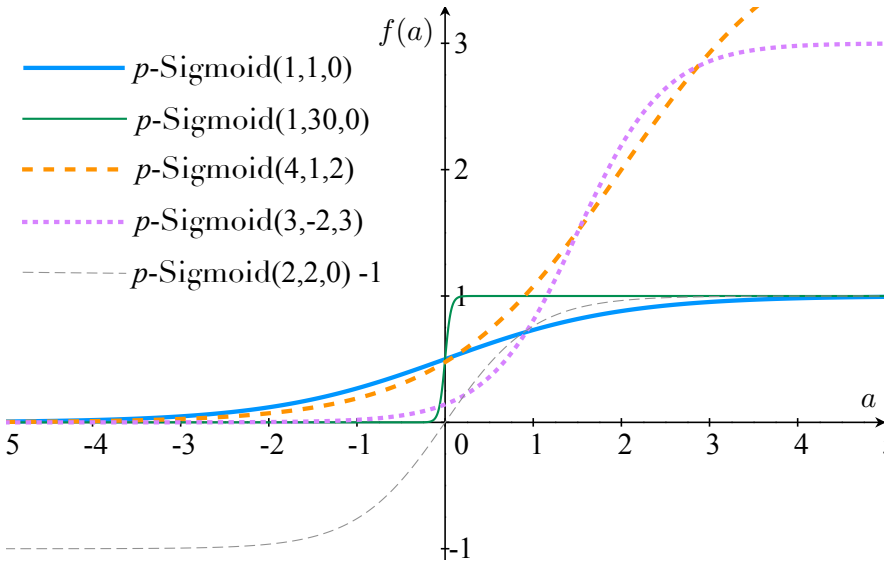


Figure 5.1 Piecewise approximation by p-Sigmoid functions.

An alternative viewpoint is based on the spike-rate coding framework introduced in Section 3.3.3, which views each activation value as the accumulated stimulus intensity within a time step, and the corresponding output value of that artificial neuron can be seen as the cumulative spike rate. By using a sigmoid activation function, it is assumed that the spike rates follow logistic distributions. p-Sigmoid(1,β,γ) can be rearranged as

$$f(a) = \frac{1}{1+\exp(-\frac{a-\gamma/\beta}{1/\beta})}, \tag{5.6}$$

which is the CDF of a logistic distribution with the mean and variance equal to γ/β and $\pi^2/(3\beta^2)$. Scaling the artificial neuron output values by α changes the mean and variance of the logistic distribution of the following sigmoid layer. Therefore, if an individual set of parameters is associated with each hidden layer artificial neuron, using the hidden activation function p-Sigmoid($\alpha_i^{(k)}, \beta_i^{(k)}, \gamma_i^{(k)}$) allows each artificial neuron k to use a different distribution to model its spike rates.

Note that p-Sigmoid$(2,2,0)-1$ is the hyperbolic tangent activation function (Bishop, 1995), as shown by the grey dashed line in Figure 5.1. However, there is not a learnable parameter of the p-Sigmoid function that controls the vertical shift in this thesis.

As seen in Section 5.1, it is necessary to calculate the partial derivatives of $f_i^{(k)}(\cdot)$ with respect to its input activation $a_i^{(k)}$, and also to each of its parameters $\alpha_i^{(k)}$, $\beta_i^{(k)}$,

and $\gamma_i^{(k)}$. For the p-Sigmoid, the derivatives are computed by

$$\frac{\partial f_i^{(k)}(a_i^{(k)})}{\partial a_i^{(k)}} = \begin{cases} 0 & \text{if } \alpha_i^{(k)} = 0 \\ \beta_i^{(k)} f_i^{(k)}(a_i^{(k)}) \left(1 - f_i^{(k)}(a_i^{(k)})/\alpha_i^{(k)}\right) & \text{if } \alpha_i^{(k)} \neq 0 \end{cases}, \tag{5.7}$$

$$\frac{\partial f_i^{(k)}(a_i^{(k)})}{\partial \alpha_i^{(k)}} = \begin{cases} \left(1 + \exp(-\beta_i^{(k)} a_i^{(k)} + \gamma_i^{(k)})\right)^{-1} & \text{if } \alpha_i^{(k)} = 0 \\ f_i^{(k)}(a_i^{(k)})/\alpha_i^{(k)} & \text{if } \alpha_i^{(k)} \neq 0 \end{cases}, \tag{5.8}$$

$$\frac{\partial f_i^{(k)}(a_i^{(k)})}{\partial \beta_i^{(k)}} = \begin{cases} 0 & \text{if } \alpha_i^{(k)} = 0 \\ a_i^{(k)} f_i^{(k)}(a_i^{(k)}) \left(1 - f_i^{(k)}(a_i^{(k)})/\alpha_i^{(k)}\right) & \text{if } \alpha_i^{(k)} \neq 0 \end{cases}, \tag{5.9}$$

$$\frac{\partial f_i^{(k)}(a_i^{(k)})}{\partial \gamma_i^{(k)}} = \begin{cases} 0 & \text{if } \alpha_i^{(k)} = 0 \\ -f_i^{(k)}(a_i^{(k)}) \left(1 - f_i^{(k)}(a_i^{(k)})/\alpha_i^{(k)}\right) & \text{if } \alpha_i^{(k)} \neq 0 \end{cases}. \tag{5.10}$$

From Eqns. (5.8) and (5.9), the input activation, $a_i^{(k)}$, needs to be used in calculating the gradients. This is a different requirement from the sigmoid function in Eqn. (3.20), where $a_i^{(k)}$ does not need to be kept for EBP, and it gives rise to extra cost for both computation and storage. The computational cost is negligible if a GPU is used in training. Furthermore, when $\beta_i^{(k)}$ is not involved in training, it has been found that using $\partial f_i^{(k)}(a_i^{(k)})/\partial \alpha_i^{(k)} = 0$ for the $\alpha_i^{(k)} = 0$ case in Eqn. (5.9) makes only a little difference[1], which can simplify the implementation.

5.2.2 Parameterised ReLU functions

For the ReLU function with a hinge-like shape, the most straightforward way to make it a parameterised form is to individually associate a scaling factor with either part of the function, i.e., to enable the two ends of the hinge to rotate separately around the "pin", as illustrated in Figure 5.2. This function is denoted as p-ReLU,

$$f(a) = \begin{cases} \alpha \cdot a & \text{if } a > 0 \\ \beta \cdot a & \text{if } a \leqslant 0 \end{cases}, \tag{5.11}$$

where α, and β are real-valued scaling factors of the positive and negative parts. The parametric ReLU function proposed in (He et al., 2015b) is a simplified form of p-ReLU(α, β), by fixing $\alpha = 1$. Similarly, another simplified form of interest is p-ReLU$(\alpha, 0)$.

[1]Note that this approximation was not used in the experiments reported in this thesis.

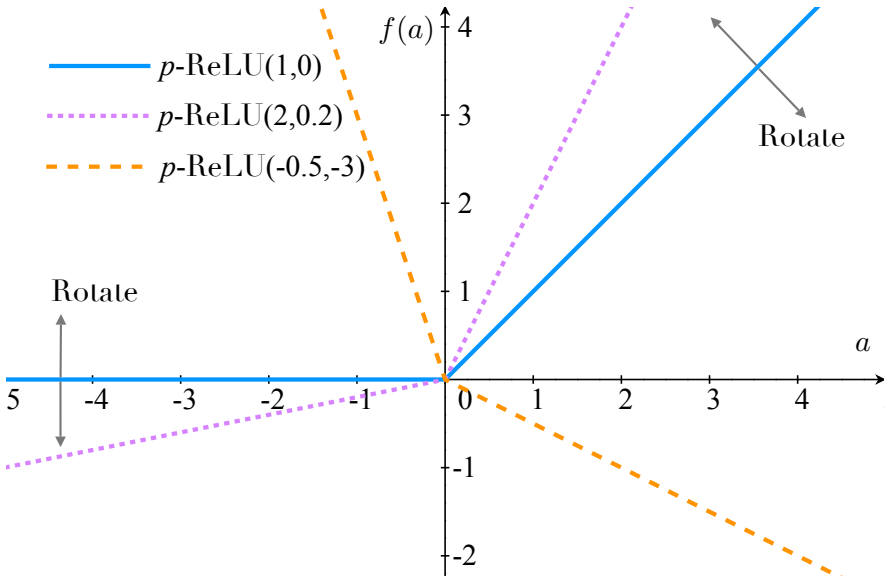


Figure 5.2 The rotation of the positive and negative parts of p-ReLU functions around the origin of the coordinate system.

In a similar way to the p-Sigmoid, the function parameters depend on each hidden layer artificial neuron. Therefore, the derivatives of p-ReLU with respect to $a_i^{(k)}$, $\alpha_i^{(k)}$, and $\beta_i^{(k)}$ for EBP are

$$\frac{\partial f_i^{(k)}(a_i^{(k)})}{\partial a_i^{(k)}} = \begin{cases} \alpha_i^{(k)} & \text{if } a_i^{(k)} > 0 \\ \beta_i^{(k)} & \text{if } a_i^{(k)} \leqslant 0 \end{cases}, \tag{5.12}$$

$$\frac{\partial f_i^{(k)}(a_i^{(k)})}{\partial \alpha_i^{(k)}} = \begin{cases} a_i^{(k)} & \text{if } a_i^{(k)} > 0 \\ 0 & \text{if } a_i^{(k)} \leqslant 0 \end{cases}, \tag{5.13}$$

$$\frac{\partial f_i^{(k)}(a_i^{(k)})}{\partial \beta_i^{(k)}} = \begin{cases} 0 & \text{if } a_i^{(k)} > 0 \\ a_i^{(k)} & \text{if } a_i^{(k)} \leqslant 0 \end{cases}. \tag{5.14}$$

Since the values of $\alpha_i^{(k)}$ and $\beta_i^{(k)}$ are not constrained, the sign of $a_i^{(k)}$ cannot be inferred from $f_i^{(k)}(a_i^{(k)})$. Therefore, it is necessary to keep $a_i^{(k)}$ with extra memory usage for backpropagation. As for the p-Sigmoid case, for p-ReLU($\alpha_i^{(k)}$,0), $a_i^{(k)}$ does not need to be kept by not updating the zero-valued $\alpha_i^{(k)}$.

Moreover, it may be argued that, for unbounded activation functions like ReLU, scaling them with factors of absolute values bigger than one is unsafe. However, in the experiments performed in this thesis, it has not been found to cause any issues. This indicates that though scaling up ReLU outputs may cause gradient explosion (Bengio et al., 1994), it is also possible that the automatically learned scaling factors

can prevent explosion from occurring by scaling down ReLU outputs. In addition, even if the gradient does get overly large, this issue can be fixed using gradient clipping, as mentioned in Section 5.1, or ReLU function output clipping.

5.3 Speaker Independent Modelling Experiments

The use of the p-Sigmoid and p-ReLU for standard SI DNN acoustic model training is investigated in this section. All CD-DNN-HMMs were constructed using the Mandarin CTS data following the configurations described in Appendix A.5. If not explicitly defined, for p-Sigmoid, $\alpha_i^{(k)}$, $\beta_i^{(k)}$, and $\gamma_i^{(k)}$ were initialised as 1.0, 1.0, and 0.0, respectively; the initial values of $\alpha_i^{(k)}$ and $\beta_i^{(k)}$ for p-ReLU were set to 1.0, and 0.25, respectively. There are two things of interest: which activation function parameters to select and how to train these parameters appropriately. For clarity, only the Dev04 test set was used for investigation, and the other test sets were kept for validation in the end.

5.3.1 Training p-Sigmoid and p-ReLU parameters

For the p-Sigmoid CD-DNN-HMM systems, care needs to be taken in training activation function parameters from the PT step, since some parameters, e.g., $\alpha_i^{(k)}$, can change the p-Sigmoid outputs rapidly and can affect the convergence of other model parameters. Starting to train p-Sigmoid parameters from the FT step is preferred since all weights and biases are already reasonably initialised. Therefore, training of $\alpha_i^{(k)}$, $\beta_i^{(k)}$, and $\gamma_i^{(k)}$ individually were compared when either PT and FT, or only FT is applied. The results are listed in Table 5.1.

Comparing S1$^+$, S2$^+$, and S3$^+$ to S1, S2, and S3 respectively, we can see all p-Sigmoid systems that started to learn their parameters from FT only outperformed their counterparts that learned all the parameters together since PT. These results coincide with our above mentioned inference. This was also tested with the p-ReLU CD-DNN-HMM systems. Since PT was not used in ReLU DNN training, the p-ReLU parameters were kept fixed in the first epoch to prevent them affecting other parameters. The resulting systems were R1$^-$ and R2$^-$. From Table 5.2, it is clear that training $\alpha_i^{(k)}$ from the first epoch makes R1 outperform R1$^-$, while doing this for $\beta_i^{(k)}$ has no impact on WERs. This reveals that since ReLU DNNs have unbounded activation functions, their parameters can be more robust to the fluctuation of activation function outputs, and it is not necessary to avoid the impact on other parameters at the beginning.

Table 5.1 Dev04 WERs with a trigram LM produced by the p-Sigmoid Mandarin CTS CD-DNN-HMM systems.

ID	Activation Function	PT	FT	%WER
S1$^+$	p-Sigmoid($\alpha_i^{(k)}$,1,0)	$\surd$	$\surd$	27.6
S2$^+$	p-Sigmoid(1,$\beta_i^{(k)}$,0)	$\surd$	$\surd$	27.7
S3$^+$	p-Sigmoid(1,1,$\gamma_i^{(k)}$)	$\surd$	$\surd$	27.7
S1	p-Sigmoid($\alpha_i^{(k)}$,1,0)	$\times$	$\surd$	27.1
S2	p-Sigmoid(1,$\beta_i^{(k)}$,0)	$\times$	$\surd$	27.5
S3	p-Sigmoid(1,1,$\gamma_i^{(k)}$)	$\times$	$\surd$	27.4
S4	p-Sigmoid($\alpha_i^{(k)}$,$\beta_i^{(k)}$,0)	$\times$	$\surd$	27.2
S5	p-Sigmoid(1,$\beta_i^{(k)}$,$\gamma_i^{(k)}$)	$\times$	$\surd$	27.2
S6	p-Sigmoid($\alpha_i^{(k)}$,1,$\theta_i^{(k)}$)	$\times$	$\surd$	27.4
S7	p-Sigmoid($\alpha_i^{(k)}$,$\beta_i^{(k)}$,$\gamma_i^{(k)}$)	$\times$	$\surd$	27.3

Table 5.2 Dev04 WERs with a trigram LM produced by the p-ReLU Mandarin CTS CD-DNN-HMM systems.

ID	Activation Function	First Epoch	Rest Epochs	%WER
R1	p-ReLU($\alpha_i^{(k)}$,0)	$\surd$	$\surd$	26.8
R2	p-ReLU(1,$\beta_i^{(k)}$)	$\surd$	$\surd$	27.0
R3	p-ReLU($\alpha_i^{(k)}$,$\beta_i^{(k)}$)	$\surd$	$\surd$	27.1
R1$^-$	p-ReLU($\alpha_i^{(k)}$,0)	$\times$	$\surd$	27.4
R2$^-$	p-ReLU(1,$\beta_i^{(k)}$)	$\times$	$\surd$	27.0

In summary, it was found to be helpful to keep the p-Sigmoid parameters frozen at the beginning of training, but unnecessary or even harmful for p-ReLU systems.

5.3.2 Selecting p-Sigmoid and p-ReLU parameters to train

DNN-HMM WERs with all choices of p-Sigmoid and p-ReLU functions are shown in Table 5.1 and 5.2. From the results, none of the p-Sigmoid and p-ReLU systems with multiple learned parameters has outperformed S1 and R1, the two systems with only the linear scaling factors estimated. Though the WER difference between p-Sigmoid and p-ReLU with different parameters are not obvious, output value amplifier $\alpha_i^{(k)}$ are selected to produce the full results on all test sets in Table 5.3 since it not only results in the lowest WERs for both p-Sigmoid and p-ReLU, but also has been proved

Table 5.3 Mandarin CTS CD-DNN-HMM system WERs with a trigram LM on Eval97, Eval03, and Dev04 test sets.

ID	Activation Function	Eval97	Eval03	Dev04
S0	sigmoid	34.1	29.7	27.9
S1	p-Sigmoid($\alpha_i^{(k)}$,1,0)	32.9	28.6	27.1
R0	ReLU	33.3	29.1	27.6
R1	p-ReLU($\alpha_i^{(k)}$,0)	32.7	28.7	26.8

useful as both SI and SD parameters for CNN, DNN, and RNN (Goh and Mandic, 2003; Swietojanski et al., 2016; Swietojanski and Renals, 2014; Trentin, 2001). Other p-Sigmoid and p-ReLU parameters may also be the choice according to the situation, such as in (He et al., 2015b).

Full results on the three test sets are presented in Table 5.3. S0 and R0 are the baseline systems with standard sigmoid and ReLU functions. Comparing S1 to S0, and R1 to R0, p-Sigmoid and p-ReLU resulted in on average 3.4% and 2.0% relative WER reduction, by increasing the number of parameters by only 0.06% (6,000 among 10M) in training, which illustrates the important role of parameterised activation functions. Of course, an equivalent model without any activation function parameters can be obtained by replacing p-Sigmoid($\alpha_i^{(k)}$,1,0) or p-ReLU($\alpha_i^{(k)}$,0) with sigmoid or ReLU, and scaling the corresponding weights of the following layer by $\alpha_i^{(k)}$s. Furthermore, although on this task, p-ReLU outperformed p-Sigmoid, the relative improvement is small. This shows that improvement from p-Sigmoid using $\alpha_i^{(k)}$ may already contain some benefits from making sigmoid more similar to ReLU, but more improvements were from the extra flexibility to allow the contribution of each hidden artificial neurons to be automatically and individually weighted.

5.4 p-Sigmoid and p-ReLU for Speaker Adaptation

This section introduces another application of the p-Sigmoid and p-ReLU functions: speaker adaptation. Both functions are first modified to use SD parameters; then the adaptation criterion and procedure used in this thesis are introduced.

5.4.1 SD p-Sigmoid and p-ReLU parameters

Adaptation can be viewed as a special model training approach using two data sets that are asymmetric in size. That is, a relatively small set of adaptation data is available

for either training or testing, which has a mismatch with the existing model and is insufficient to construct a new model. Adaptation aims to reduce such mismatch by retraining existing model parameters using the adaptation data.

Like other training problems, the choice of parameters, training criteria, and optimisation methods are important for adaptation. It is worth taking into account the properties of the adaptation data when designing an adaptation method. For example, in the MLLR/CMLLR approach described in Section 2.6.1, the use of linear transforms as parameters, the maximum likelihood (ML) criterion, as well as the regression trees to cluster transforms, are all designed to use the limited adaptation data more efficiently.

Applying the p-Sigmoid and p-ReLU to the adaptation task requires adapting their parameters. This is an appropriate choice since activation function parameters can be more efficient in training than standard weights and biases, as mentioned in Section 5.1. The p-Sigmoid and p-ReLU parameters can be combined with various different criteria and optimisation procedures for adaptation, but in this thesis only one possible solution presented later in Section 5.4.2 is used. Furthermore, SD parameter estimation can rely on either true reference word labels or ASR output hypotheses, often called the *supervised* or *unsupervised* adaptation mode. All experiments with p-Sigmoid and p-ReLU adaptations in this thesis use test-time adaptation (see Section 2.6.1) with the unsupervised mode.

The best configurations found for SI acoustic model training in Table 5.1 and 5.2 were used for adaptation, which scale the sigmoid and ReLU output values linearly. The scaling factor is made to be dependent on each speaker s, i.e., $\alpha_i^{(k,s)}$, and the functions in use are p-Sigmoid$(\alpha_i^{(k,s)},1,0)$ and p-ReLU$(\alpha_i^{(k,s)},0)$. Both functions can be rewritten as

$$f_i^{(k,s)}(a_i^{(k)}) = \alpha_i^{(k,s)} \cdot f(a_i^{(k)}), \tag{5.15}$$

which fall into the LHUC speaker adaptation framework (Swietojanski and Renals, 2014). The EBP training uses Eqns. (5.7) and (5.8), or (5.12) and (5.13), by replacing SI linear scaling with SD scaling. All $\alpha_i^{(k,s)}$s are initialised to 1.0 so that the initial SD model is equivalent to the SI model. A major difference between this work and the initial LHUC adaptation work (Swietojanski and Renals, 2014) is that a linear scaling factor $\alpha_i^{(k,s)}$ rather than a sigmoid function constraint scaling factor is used. It falls into the LHUC framework extended later that allows different functions (Swietojanski et al., 2016), including a linear function, to be applied to transform the amplifiers.

5.4.2 Adaptation criteria and the layer-wise scheme

In this thesis, the $\alpha_i^{(k,s)}$s for all p-Sigmoid and p-ReLU functions are trained with the standard cross entropy (CE) criterion using the EBP algorithm and speaker specific adaptation data. If the supervision is provided as frame-state alignments, each class label follows the Bernoulli distribution and Eqn. (3.16) becomes

$$\begin{aligned} \mathcal{F}^{\text{CE}}(t) &= -\ln P(C_{k_0}|\mathbf{x}^{\text{in}}(t)) \\ &\propto -\ln p(\mathbf{x}^{\text{in}}(t)|C_{k_0}), \end{aligned} \tag{5.16}$$

where k_0 is the reference target at time t. Therefore, minimising CE maximises the per frame log posterior probability, and is equivalent to maximising the per frame log-likelihood. This follows a similar idea to both MAP and ML adaptation methods (Gauvain and Lee, 1994; Woodland, 2001). Alternatively, other criteria and learning methods can also be applied to adapt the p-Sigmoid and p-ReLU parameters (Huang et al., 2015; Liao, 2013; Yu et al., 2013).

It should be noted that learning $\alpha_i^{(k,s)}$ is equivalent to using standard activation functions with all SD weights associated with node i in the following layer scaled by $1/\alpha_i^{(k,s)}$ (if $\alpha_i^{(k,s)} \neq 0$), which if learned directly would require more parameters to be changed. This gives an insight of the power of the parameterised sigmoid and ReLU methods.

While adaptation with an unconstrained scaling factor may be unsafe, in the experiments performed, a layer-wise adaptation approach that simulates the discriminative PT and FT method (see Section 3.8.3), has been found sufficient to stabilise adaptation. That is, once adaptation of a hidden layer has completed for a single epoch, the following hidden layer is jointly adapted with all $\alpha_i^{(k,s)}$ associated with previous layers. This procedure starts from the input layer and can be continued until a desired number of hidden layers have been adapted. The final step is FT of all the $\alpha_i^{(k,s)}$s in the whole network which are adapted together for multiple epochs.

5.5 Speaker Adaptation Experiments

The SD DNN parameters were updated once per adaptation utterance, with the updates averaged over all frames in the utterance. Based on the $\alpha_i^{(k,s)}$, the parameters of both p-Sigmoid and p-ReLU were learned using the layer-wised adaptation method described in Section 5.4.2. The FT step in adaptation was performed for 6 epochs. A

fixed learning rate was used throughout adaptation with 1.0×10^{-2} for p-Sigmoid and 2.5×10^{-3} for p-ReLU.

Three different sets of experiments were performed to evaluate the proposed adaptation methods on the 200h MGB data set (see Appendix A.2). First, the standard sigmoid and ReLU DNN-HMMs were adapted using p-Sigmoid and p-ReLU. Second, adaptation was applied to bottleneck (BN) DNN features for use in GMM-HMMs. Finally, BN DNN and DNN-HMM adaptation methods were combined based on stacked DNN-HMM systems. The supervision hypotheses for adaptation were generated by joint decoding of SI tandem and DNN-HMM systems. For p-Sigmoid and p-ReLU methods, the DNNs had sigmoid and ReLU activation functions accordingly, and the resulting WERs of the hypothesis supervisions were 28.2% and 27.8%. Furthermore, the proposed methods are compared with baseline adaptation methods, such as DNN layer bias adaptation, MLLR, and CMLLR. Note that only test-time p-Sigmoid and p-ReLU adaptation is used in this section, so it is not used in any DNN or GMM-HMM training. At the end of this section, 150h TED CD-DNN-HMM systems were adapted with the supervision produced by the target systems themselves.

5.5.1 MGB hybrid system adaptation

Sigmoid and ReLU SI DNN-HMM systems were built to evaluate the hidden activation function adaptation approaches. The DNN structure was $720 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 6001$. Configurations with different hidden layers adapted were evaluated in Table 5.4, to show the role of each hidden layer in speaker adaptation. With the comparison of different adaptation configurations, more hidden layers were progressively involved in adaptation, from the input to the output of the DNN. The SD parameters of all different layers involved in adaptation, the $\alpha_i^{(k,s)}$s, are jointly adapted.

From Table 5.4, although adapting more hidden layers with p-Sigmoid and p-ReLU consistently reduces the WER, the improvement from extra layers decreases as the newly adapted hidden layer is closer to the DNN output layer. This coincides with previous findings that the low level characteristics of speech, such as speaker dependencies, are mainly modelled by the input layer, which makes them more important for speaker adaptation (Mohamed et al., 2012; Swietojanski and Renals, 2014). For the MGB data set, both p-Sigmoid and p-ReLU gave the lowest WERs by adapting all hidden layers. Meanwhile, compared to adapting the bias values of each hidden layer, p-Sigmoid and p-ReLU gave consistently better performance since their parameters were more effective as discussed in Section 5.4.1. Adapting hidden layer biases is also a commonly used

Table 5.4 MGB 200h CD-DNN-HMM system WERs with a 160k word 4-gram LM on Dev.sub test set. The parameters of p-Sigmoid, p-ReLU, and the biases of sigmoid and ReLU systems from the first n layers were adapted. The 6th layer is the output layer.

n Layers	p-Sigmoid	Bias (sigmoid)	p-ReLU	Bias (ReLU)
0	30.6	30.6	29.9	29.9
1	28.9	30.1	28.2	29.3
2	28.5	29.4	28.0	29.1
3	28.4	28.9	28.0	29.0
4	28.3	28.8	27.8	29.0
5	28.3	28.8	27.8	28.7
6		29.0		28.4

method (Yao et al., 2012), which adapts the same number of parameters as p-Sigmoid and p-ReLU.

5.5.2 MGB tandem system adaptation

The use of the p-Sigmoid and p-ReLU in tandem BN DNN adaptation for GMM-HMMs is explored in this section. The input features to BN DNNs were `FBANK_D`, and the DNN structure was $720 \times 1000 \times 1000 \times 1000 \times 1000 \times 39 \times 1000 \times 6001$. For BN DNNs, only hidden layers preceding the BN layer were involved in p-Sigmoid and p-ReLU speaker adaptation, as the SD parameters of the last two hidden layers were not involved in BN feature generation.

Two BN DNNs were trained separately for the sigmoid and ReLU. The BN DNN systems were constructed following the standard recipe described in Section 3.9.1 and their adaptation results are listed in Table 5.5. If CMLLR was used, BN GMM-HMMs were trained by applying CMLLR-based speaker adaptive training (SAT), which had the GMM-HMMs and the per speaker CMLLR transforms estimated iteratively. Both SAT and non-SAT GMM-HMMs finally used minimum phone error (MPE) training. Extra model-based MLLR transforms were also used for SAT-based models. Note that once BN features were changed due to BN DNN adaptation, CMLLR and MLLR transforms were re-estimated to accommodate these changes.

Table 5.5 gives the results of applying p-Sigmoid and p-ReLU for test-time adaptation of BN DNNs. It can be seen that the BN GMM-HMM system performance was improved, as it improved the overall quality of testing features. Comparing p-Sigmoid and p-ReLU adaptation with the joint use of CMLLR and MLLR, although p-Sigmoid adaptation resulted in smaller WER reductions, it only adapted BN features at test

Table 5.5 MGB 200h BN GMM-HMM system WERs with a 160k word 4-gram LM on Dev.sub using different BN DNN and adaptation methods.

p-Sigmoid	CMLLR	MLLR	%WER
×	×	×	29.3
√	×	×	28.3
×	√	√	27.8
√	√	√	27.6
p-ReLU	CMLLR	MLLR	%WER
×	×	×	29.5
√	×	×	27.6
×	√	√	27.6
√	√	√	27.3

time, while CMLLR transforms were also estimated during GMM-HMM training. Meanwhile, CMLLR transformed features matched GMM-HMMs better due to the CMLLR based SAT, which was more computationally expensive than test-time only adaptation. In addition, it can also be seen that both p-Sigmoid and p-ReLU adaptation were complementary to CMLLR and MLLR transforms. It should be noted that if BN features are adapted, it is better to re-compute the mean and variance normalisation, as BN features can have a large change in range caused by adaptation.

5.5.3 MGB stacked hybrid system adaptation

A combination of the adaptation methods is finally investigated based on stacked DNN-HMM systems. The BN tandem features described in Section 5.5.2 were used to train the stacked hybrid DNN-HMMs. The DNN acoustic model structure was the same as that used in Section 5.5.1, except for the input vector size. If the decorrelation matrix and CMLLRs were used to transform the BN tandem features, the DNN input size was 702; otherwise, it was 819.

Table 5.6 contains results of the stacked DNN-HMM systems. The 1st, 2nd, and 3rd columns in the table show if adaptation was applied to the BN DNN, the BN tandem features, or to the DNN-HMM acoustic models. From the results, it is clear that all of the three adaptation methods can reduce the WER. It is not surprising that for BN DNN adaptation, p-Sigmoid and p-ReLU clearly performed better than CMLLR, since adaptation was applied to multiple hidden layers and should be more powerful than using just linear transforms. Meanwhile, p-Sigmoid and p-ReLU resulted in slightly lower WERs when they were used for BN DNN adaptation than for DNN-

Table 5.6 MGB 200h stacked DNN-HMM system WERs with a 160k word 4-gram LM on Dev.sub using different BN features and adaptation methods.

BN p-Sigmoid	CMLLR	p-Sigmoid	%WER
×	×	×	28.9
×	×	√	28.1
×	√	×	28.2
√	×	×	28.0
√	√	×	27.9
√	√	√	28.1
BN p-ReLU	CMLLR	p-ReLU	%WER
×	×	×	28.6
×	×	√	27.4
×	√	×	27.8
√	×	×	27.1
√	√	×	26.8
√	√	√	27.4

HMM adaptation. Perhaps ideally, the opposite would be expected since both BN and PLP features can be involved in acoustic model adaptation. Therefore, this indicates that the stacked DNN adaptation is more prone to over-fitting than the BN DNN. Furthermore, p-Sigmoid and p-ReLU are complementary to CMLLR, as the joint use of these methods further improved the performance. Finally, if the input vectors are already well adapted, test-time only DNN-HMM adaptation is no longer useful, perhaps due to over-fitting. However, SAT training may improve performance further, as in Table 5.5, p-Sigmoid with CMLLR based SAT achieves a WER of 27.6%, which is better than the 27.9% shown in Table 5.6.

5.5.4 TED hybrid system adaptation

The speaker adaptation experiments in all previous sections used supervisions from other systems. Though this does not affect comparisons between different adaptation methods with the same supervisions and is closer to common real word situations, the absolute improvements are not a perfect measure of how much the proposed methods could gain from the speaker specific data, since system complementarity embedded in the supervision may also contribute to the improvements. Therefore, all TED system adaptation experiments used supervisions produced by the same SI systems. Details of the TED data and the system configurations can be found in Appendix A.6.

Table 5.7 TED CD-DNN-HMM system 4-gram LM WERs on Dev2010, Tst2010, and Tst2011 test sets.

Activation Function	Adaptation	Dev2010	Tst2010	Tst2011
sigmoid	None	16.4	15.0	12.5
sigmoid	p-Sigmoid	15.9	13.9	11.8
ReLU	None	16.2	14.4	12.3
ReLU	p-ReLU	15.4	13.1	11.2

The TED CD-DNN structure was $720 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 6024$, and CE hybrid system results are listed in Table 5.7. From the results, p-Sigmoid and p-ReLU adaptation can consistently reduce WERs with the supervision produced by corresponding SI systems. On Dev2010, Tst2010, and Tst2011, p-Sigmoid adaptation gave a relative WER reduction over the SI sigmoid system of 3.0%, 7.3%, and 5.6%, respectively, while p-ReLU adaptation gave WER reductions by 4.9%, 9.0%, and 8.9%, respectively.

The results in Table 5.7 were obtained by adaptation using all test data, which is more than generally could be collected in practice. Therefore, it is interesting to see if these methods can be applied to the case when fewer speaker specific data are available for SD parameter estimation. The p-Sigmoid and p-ReLU adaptation performance with different amounts of speaker specific data on Tst2010 is presented in Figure 5.3. From the figure, it can be observed that both methods require far fewer than 600 seconds per speaker to take effect, but more speaker specific data is often helpful to further improve performance.

5.6 Summary and Conclusions

In this chapter, the use of general parameterised forms of sigmoid and ReLU activation functions was studied. It has been found that a linear scaling factor $\alpha_i^{(k)}$ with no constraint imposed is the most useful for both sigmoid and ReLU. Experimental results on a challenging CTS Mandarin task showed that DNNs trained with parameterised sigmoid and ReLU functions resulted in 3.4% and 2.0% relative reductions in WER, respectively. This reduction in WER requires an increase in the number of parameters in training by only 0.06%.

Once an SI DNN with p-Sigmoid$(\alpha_i^{(k)}, \beta_i^{(k)}, \gamma_i^{(k)})$ is trained, $\alpha_i^{(k)}$, $\beta_i^{(k)}$, and $\gamma_i^{(k)}$ can be implemented by scaling the outbound weights, inbound weights, and bias value associated with artificial neuron i by $\alpha_i^{(k)}$, $\beta_i^{(k)}$, and $(b_i^{(k)} + \beta_i^{(k)})/b_i^{(k)}$. Similarly, a

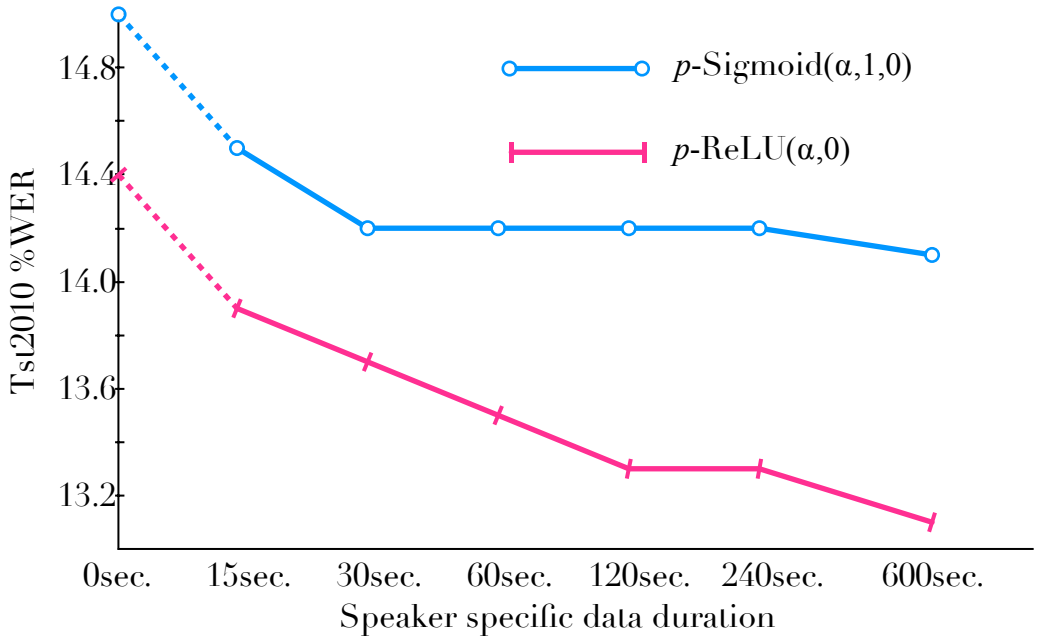


Figure 5.3 p-Sigmoid and p-ReLU adaptation results on TED Tst2010 test set with different amounts of speaker specific data.

p-ReLU($\alpha_i^{(k)}$,0) DNN can also be converted to an equivalent ReLU DNN, by scaling the outbound weights of i by $\alpha_i^{(k)}$. These reveal that extra parameters (except for $\beta_i^{(k)}$ for p-ReLU) do not change the form of functions learned through training. It may be argued whether these redundant activation function parameters are useful. However, common deep learning techniques, such as using more hidden layers (e.g., DNNs) and sharing parameters in different ways (e.g., CNNs), are not about "what mappings to learn", but about "how to learn the mappings", since a 2-layer MLP with the sigmoid activation function can arbitrarily accurately represent any continuous mapping function, as mentioned in Section 3.3.3. Actually these p-Sigmoid and p-ReLU parameters, such as $\alpha_i^{(k)}$ and $\beta_i^{(k)}$, scale a vector of weights together by the same factor and impose correlations within each such vector, which cannot be achieved by normal GD or SGD training.

A technique that has a similar effect to p-Sigmoid and p-ReLU SI modelling is the batch normalisation method introduced in Section 3.6.4, which employs two vectors to linearly transform the input to each layer normalised at the batch level. These vectors have the same effect as $\alpha_i^{(k-1)}$ and $\gamma_i^{(k)}$ for p-Sigmoid, and $\alpha_i^{(k-1)}$ for p-ReLU. The original explanation for adding these two vectors is to maintain the input value range changed by batch level normalisation (Ioffe and Szegedy, 2015). From a parameterised activation function perspective, the improvements from batch normalisation comprises

gains from both reducing layer specific input distribution changes and adding activation function parameters.

The p-Sigmoid($\alpha_i^{(s,k)}, 1, 0$) and p-ReLU($\alpha_i^{(s,k)}, 0$) were also applied to speaker adaptation in this chapter. Several thousands of parameters from DNN-HMM, BN GMM-HMM, and stacked DNN-HMM systems were adapted for each speaker using the proposed methods at test time. A layer-wise adaptation scheme was used to stabilise the multi-layer SD hidden activation function parameter learning. It has been found that learning activation function parameters is an effective method for speaker adaptation, which is not only complementary to standard CMLLR feature transforms, but also more powerful in DNN adaptation scenarios.

5.7 Related Work

There has been a long-term interest in studying sigmoid activation function parameters. The widely used hyperbolic tangent function actually associates an input value scale 2 and output value bias -1 to the sigmoid (Bishop, 1995). Han and Moraga (1995) further investigated different logistic functions with fixed parameters, and found some of them could lead to faster convergence. Trentin (2001) and Goh and Mandic (2003) showed output value amplifiers could be learned in training. Since then, there have been several studies on learning parameterised activation functions. In (Siniscalchi et al., 2013, 2010), Hermite polynomial activation functions with a number of coefficients were learned in SI MLP acoustic models, and the coefficients were made SD by adapting to a particular speaker during testing. Similarly, Swietojanski and Renals (2014) associated each hidden artificial neuron with a weight constraint by sigmoid functions, and the DNN was adapted by LHUC, learning a sigmoid constraint scaling factor for each hidden artificial neuron. An advantage of LHUC is that it does not rely on the choice of hidden activation functions. The p-Sigmoid and p-ReLU with only $\alpha_i^{(k)}$ became variations of LHUC (Zhang and Woodland, 2016). Zhao et al. (2015) also used parameterised sigmoid functions for adaptation, which did not contain the output value scaling factor. The parametric ReLU was proposed as a general activation function, and applied to a very deep CNN for image classification (He et al., 2015b). This method had the coefficients of its negative part adaptively learned and allowed automatic learning of rectification. Shortly afterwards, Sivadas et al. (2015) and Yoshioka et al. (2016) applied the same method to DNN and SI CNN acoustic modelling for Aurora-4 noisy speech recognition. Moreover, the softmax function was parameterised as well,

which had equivalent terms to GMMs with a shared covariance matrix (compared to equivalent terms to the normal softmax function in Section 3.3.1) (Tüske et al., 2015a).

DNN adaptation is also a problem drawing more attention nowadays, since it is less mature compared to GMM-HMM adaptation methods. A convenient way of adapting DNNs is to use existing GMM-HMM adaptation methods such as CMLLR to create SD input features (Karafiát et al., 2014; Knill et al., 2013; Seide et al., 2011b). Alternatively, different DNN oriented adaptation techniques have also been developed, which usually use a discriminative criterion, sometimes with additional regularisation (Huang et al., 2015; Liao, 2013; Swietojanski et al., 2015; Wu et al., 2016a; Yu et al., 2013). Besides the hidden activation parameters (Siniscalchi et al., 2013; Swietojanski et al., 2016; Swietojanski and Renals, 2014; Woodland et al., 2015; Zhang and Woodland, 2016; Zhao et al., 2015), there are other choices as to which DNN parameters to adapt, which include the weights and biases of standard DNN layers (Doddipatla et al., 2014; Gemello et al., 2007; Li and Sim, 2010; Liao, 2013; Neto et al., 1995; Ochiai et al., 2014; Yao et al., 2012; Yu et al., 2013), extra input transforms (Cui and Goel, 2015; Li and Sim, 2010; Seide et al., 2011b), and combination weights of several speaker cluster based DNNs (Tan et al., 2015; Wu and Gales, 2015). Another set of approaches is to supply the DNN with additional inputs to make a speaker-conditioned model such as the use of additional i-vectors (Karanasou et al., 2014; Miao et al., 2014; Saon et al., 2013; Senior and Lopez-Moreno, 2014) and speaker codes (Abdel-Hamid and Jiang, 2013; Ström, 1996) as input features.

Chapter 6

Hybrid and Tandem System Joint Training

In this chapter, ASR system level joint training, which is introduced in Section 3.11.1, is investigated with applications to both hybrid and tandem configurations. For hybrid systems, DNN-HMMs are constructed based on CMLLR normalised features, and CMLLR initialised transforms and DNN-HMMs are optimised together according to the minimum phone error (MPE) criterion. For tandem systems, the bottleneck (BN) DNN and GMMs combine to form a Gaussian mixture density neural network (MDNN) model, and the joint training is actually MDNN-HMM discriminative sequence training, still based on MPE. Note that both applications jointly train the feature extraction and acoustic model modules, while the LM and decoding parameters are kept fixed during the procedure.

6.1 Hybrid System MPE Training

MPE training for standard DNN-HMM acoustic models is reviewed in this section, as a precondition for hybrid system joint training. To calculate the gradients using error backpropagation (EBP) according to Eqn. (3.32), $\partial \mathcal{F}^{\text{MPE}}(\mathbf{O})/\partial a_i^{\text{out}}(t)$ is required. Applying the chain rule to Eqn. (2.101) yields

$$\begin{aligned}\frac{\partial \mathcal{F}^{\text{MPE}}(\mathbf{O})}{\partial \ln y_{k'}^{\text{out}}(t)} =& \frac{\partial \mathcal{F}^{\text{MPE}}(\mathbf{O})}{\partial \ln P(q_t = k'|\mathbf{o}(t), \Lambda)} \frac{\ln p(\mathbf{o}(t)|q_t = k', \Lambda) + \ln P(q_t = k'|\Lambda) - \ln p(\mathbf{o}(t)|\Lambda)}{\ln p(\mathbf{o}(t)|q_t = k', \Lambda)} \\ =& \kappa \gamma_{k'}^{\text{MPE}}(t),\end{aligned} \tag{6.1}$$

and therefore,

$$\begin{aligned}\frac{\partial \mathcal{F}^{\mathrm{MPE}}(\mathbf{O})}{\partial a_k^{\mathrm{out}}(t)} =& \sum_{k'} \frac{\partial \mathcal{F}^{\mathrm{MPE}}(\mathbf{O})}{\partial \ln y_{k'}^{\mathrm{out}}(t)} \frac{1}{y_{k'}^{\mathrm{out}}(t)} \frac{\partial y_{k'}^{\mathrm{out}}(t)}{\partial a_k^{\mathrm{out}}(t)} \\ =& \kappa \gamma_k^{\mathrm{MPE}}(t) - y_k^{\mathrm{out}}(t) \kappa \sum\nolimits_{k'} \gamma_{k'}^{\mathrm{MPE}}(t), \end{aligned} \tag{6.2}$$

according to Eqn. (3.28). Using Eqn. (2.99), $\sum_{k'} \gamma_{k'}^{\mathrm{MPE}}(t)$ can be arranged as

$$\begin{aligned}&\sum_{k'} \gamma_{k'}^{\mathrm{MPE}}(t) = \kappa \sum_{\mathbf{W}'} \frac{p(\mathbf{O}|\mathbf{W}')^{\kappa} P(\mathbf{W}') \mathrm{PhoneAcc}(\mathbf{W}, \mathbf{W}')}{\sum_{\mathbf{W}''} p(\mathbf{O}|\mathbf{W}'')^{\kappa} P(\mathbf{W}'')} \sum_{k'} \gamma_{k'|\lambda_{\mathbf{W}'}}(t) - \\ &\kappa \frac{\sum_{\mathbf{W}'} p(\mathbf{O}|\mathbf{W}')^{\kappa} P(\mathbf{W}') \mathrm{PhoneAcc}(\mathbf{W}, \mathbf{W}')}{\sum_{\mathbf{W}'} p(\mathbf{O}|\mathbf{W}')^{\kappa} P(\mathbf{W}')} \sum_{\mathbf{W}'} \frac{p(\mathbf{O}|\mathbf{W}')^{\kappa} P(\mathbf{W}')}{\sum_{\mathbf{W}''} p(\mathbf{O}|\mathbf{W}'')^{\kappa} P(\mathbf{W}'')} \sum_{k'} \gamma_{k'|\lambda_{\mathbf{W}'}}(t), \end{aligned} \tag{6.3}$$

which equals to zero using $\sum_{k'} \gamma_{k'|\lambda_{\mathbf{W}'}}(t) = 1$, and hence Eqn. (6.2) becomes

$$\frac{\partial \mathcal{F}^{\mathrm{MPE}}(\mathbf{O})}{\partial a_k^{\mathrm{out}}(t)} = \kappa \gamma_k^{\mathrm{MPE}}(t). \tag{6.4}$$

Similarly, based on Eqns. (2.86), (2.87), and (3.28), we have $\sum_{k'} \gamma_{k'}^{\mathrm{MMI}}(t) = 0$,

$$\frac{\partial \mathcal{F}^{\mathrm{MMI}}(\mathbf{O})}{\partial a_k^{\mathrm{out}}(t)} = \kappa \gamma_k^{\mathrm{MMI}}(t) \tag{6.5}$$

for maximum mutual information (MMI) sequence training, and

$$\frac{\partial \mathcal{F}^{\mathrm{ML}}(\mathbf{O})}{\partial a_k^{\mathrm{out}}(t)} = \kappa \left(\gamma_k(t) - y_k^{\mathrm{out}}(t) \right) \tag{6.6}$$

for maximum likelihood (ML) sequence training.

Unlike frame level training such as cross entropy (CE), sequence training processes each whole utterance $\mathbf{O}$ to calculate $\gamma_k(t)$, from Eqns. (6.4) – (6.6). Therefore, it is less efficient to conduct SGD based sequence training with frame level data shuffling, since it requires too much utterance processing for each minibatch. Utterance level data shuffling is often used in sequence training where only one utterance is processed in each minibatch, and each parameter update can be based on the gradients accumulated on multiple utterances. In this thesis, all parameters are updated once per minibatch, and thus sequence training usually uses a smaller learning rate than CE training to

avoid overly large parameter changes when the data in the minibatch are not well sampled.

6.2 MPE Training Experiments

This section only contains DNN-HMM MPE training results, since MMI and ML training are not directly applied to joint training later in this chapter. The results on both a small data set, 15h Babel Tamil LLP CTS data (see Appendix A.1), and a large data set, 200h MGB broadcast data (see Appendix A.2), are presented.

The Tamil LLP context-dependent (CD) DNN acoustic model structure is $504 \times 1000 \times 500 \times 500 \times 500 \times 500 \times 1002$. The CE DNN training configuration is listed in Table A.1. The first experiment investigated the use of denominator lattices with different densities. The learning rates were fixed to 2.0×10^{-5} for four epochs in these experiments. From the results in Table 6.1, the denominator lattice density APS= 2000.0 (see Section 2.4.3) is sufficient for DNN-HMM MPE training, and is used throughout the thesis. Next, the impact of learning rates on MPE training was investigated. From Table 6.1, it can be seen that a sufficiently large learning rate is important for the model to converge to a good local optimum, but the performance will dramatically decrease once the learning rate exceeds a limit. In line with the discussion in Section 6.1, this learning rate limit is much smaller than those for normal frame level training. Note that the optimal learning rate found in Table 6.1 is much larger than those reported in other publications (Kingsbury, 2009; McDermott et al., 2014; Su et al., 2013; Veselý et al., 2013; Wiesler et al., 2015). This is perhaps partly because Tamil LLP contains only 15h data and is insufficient to use smaller learning rates. However, in practice, it is often observed that using large learning rates are helpful in improving MPE training performance, and all results presented later in this chapter were based on learning rates tuned for each individual system to perform well. DNN-HMM MPE training can also be conducted without tuning the learning rates, for instance, by using second order optimisation methods (Kingsbury et al., 2012).

The MGB 200h CD-DNN acoustic model structure is $720 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 5973$, and the learning rate used for MPE training is 4.0×10^{-5}. The WER for each MPE epoch is shown in Figure 6.1. The WER is consistently reduced across all four MPE epochs, and in the end a 9.2% relative WER reduction is obtained. Lattice regeneration is also evaluated. Initially, lattices were produced by the CE DNN-HMMs and used in all epochs, which were assumed to be a good approximation to the hypothesis space regardless of model changes. This assumption can be improved

Table 6.1 Babel Tamil LLP DNN-HMM system CERs on Dev set with a trigram LM, and different lattice pruning settings.

Criterion	Learning Rate	APS	%CER
CE	2.0×10^{-3}		79.1
MPE	2.0×10^{-5}	1000.0	78.3
MPE	2.0×10^{-5}	2000.0	78.1
MPE	2.0×10^{-5}	4000.0	78.1
MPE	2.0×10^{-5}	6000.0	78.1
MPE	4.0×10^{-5}	2000.0	77.9
MPE	8.0×10^{-5}	2000.0	77.9
MPE	1.6×10^{-4}	2000.0	77.5
MPE	3.2×10^{-4}	2000.0	77.4
MPE	6.4×10^{-4}	2000.0	78.4

by regenerating lattices using the model produced by the first epoch, and a 0.2% absolute WER reduction was acquired. Lattice regeneration by later models did not result in any performance improvement. Though lattice regeneration can improve MPE training by a small amount, it is not used in following experiments in this chapter, since it is very expensive for both computation and storage.

6.3 Discriminative Joint SAT

As introduced in Section 2.6.1, speaker adaptive training (SAT) applies speaker adaptation in both acoustic model training and testing, to obtain acoustic models that are more focused on modelling phonetically relevant variations. Meanwhile, instead of MAP or MLLR, GMM-HMM adaptation can be conducted based on MPE or MMI (Povey et al., 2003a; Wang and Woodland, 2008), which can generate more discriminative speaker dependent (SD) parameters. This section combines both ideas by performing SAT for DNN-HMMs to optimise a discriminative sequence criterion. Specifically, the DNN-HMM is constructed based on CMLLR normalised features, and both CMLLR initialised input feature transforms and CD-DNN acoustic model parameters are concurrently updated in MPE training, which falls into the ASR system level joint training framework for both feature extraction and acoustic model modules. Therefore, the proposed method is denoted as discriminative joint SAT.

Figure 6.2 depicts the DNN model structure for discriminative joint SAT. Since the CMLLR initialised input transform is used for each frame $\mathbf{o}(t)$ which is stacked

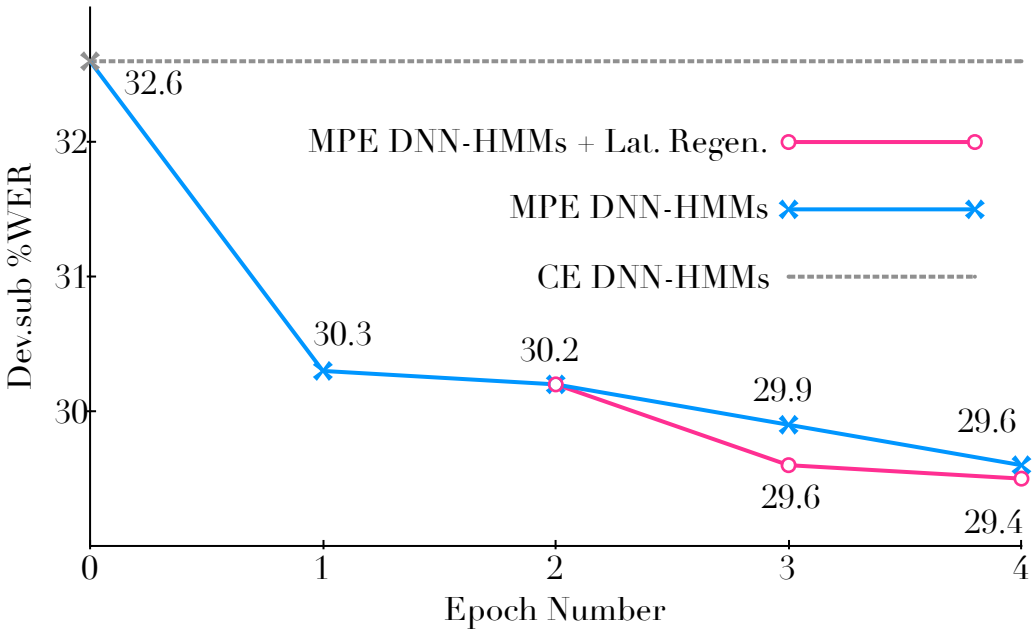


Figure 6.1 MGB 200h DNN-HMM MPE training performance on Dev.full test set with a 64k word 4-gram LM. CE DNN-HMMs generated the initial lattices for MPE; lattices can be regenerated after 1st MPE epoch.

according to a particular context shift set ($\mathbf{c} = \{-4,-3,-2,-1,0+1,+2,+3,+4\}$ in this thesis) to form the DNN acoustic model input $\mathbf{x}^{\text{in}}(t)$, the extended DNN structure is different from the standard one. Not only does the original first layer take inputs from multiple layers, but also those layers have their parameters tied together. Calculating the gradients with EBP for such a structure requires the extended formulae in Eqn. (3.32), with the shared input transform replaced according to the current speaker.

For supervised adaptation, some data from a testing speaker are available with their true reference, possibly generated through an enrolment session. In this case, it is possible to model the speaker specific characteristics better with discriminative training. For unsupervised adaptation, discriminative sequence adaptation can cause SD parameters to over-fit the hypotheses that serve as the supervision rather than the correct references. Since this thesis focuses only on unsupervised adaptation, a regularisation method is required for discriminative sequence training. The training and decoding procedure is as follows.

- On the training data, the standard MPE criterion is used in joint SAT. The CMLLR initialised transforms are concurrently updated with the CD-DNN acoustic model based on true references.

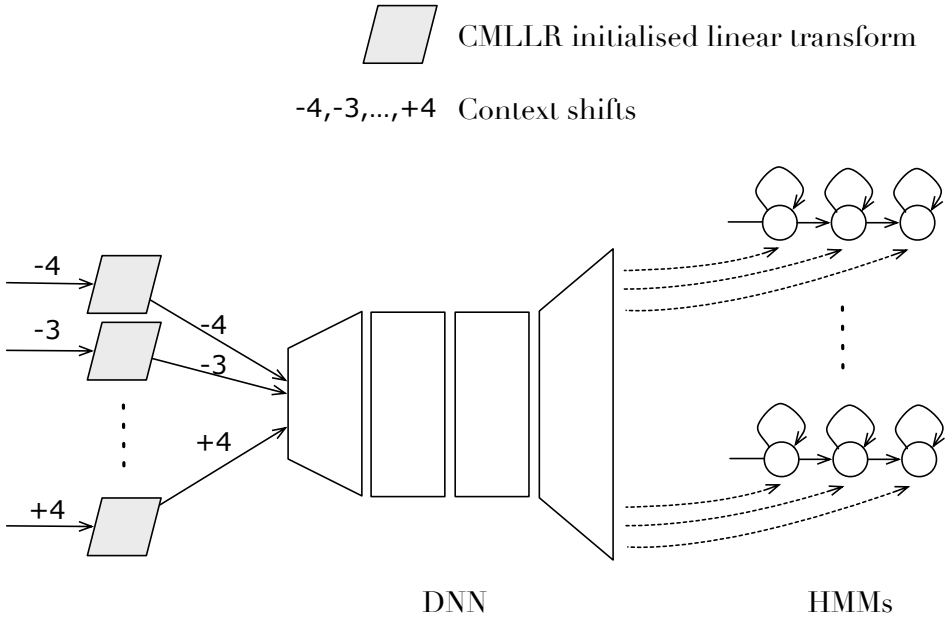


Figure 6.2 The DNN structure for discriminative joint SAT. A CMLLR transform is used to initialise an input layer shared by all context shifts of the frames from the same speaker.

- At test time, the original CD-DNN acoustic model parameters are kept frozen and only CMLLR transforms of testing speakers are trained through the discriminative sequence training according to the supervision.

One possible way to regularise unsupervised discriminative sequence training is to use the *F-smoothing* method in testing SD parameter estimation, which is a special case of the *H-criterion* (Gopalakrishnan et al., 1991) smoothing a discriminative sequence objective function with frame level CE (Su et al., 2013),

$$\mathcal{F}^{\text{MPE}}(\mathbf{O}) + H \sum\nolimits_t \mathcal{F}^{\text{CE}}(t), \tag{6.7}$$

since CE training is found to be effective in unsupervised adaptation (see Section 5.4). This method has also been applied to discriminative sequence adaptation by other researchers (Huang et al., 2015). Alternatively, we can assume that the difference between all discriminatively trained SD transforms and their CMLLR initialisations can be modelled by a linear transform. An extra linear transform is imposed between CMLLR transforms and the first hidden layer, which is learned through joint SAT and applied to testing speakers without modification. Note that no CMLLR initialised transform is updated in this joint training procedure. This method is denoted as

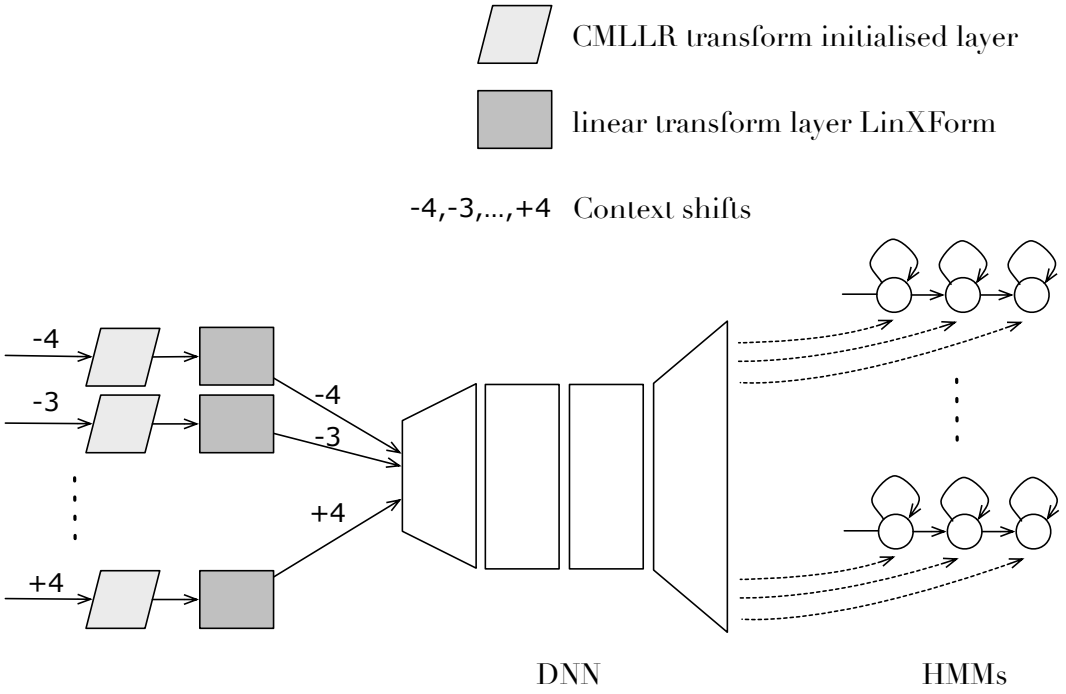


Figure 6.3 The DNN structure for discriminative joint SAT with LinXForm. A CMLLR transform is viewed as an input layer shared by all context shifts of the frames from the same speaker. The LinXForm layer is shared by all frames.

LinXForm and does not need test-time discriminative training. The DNN structure is shown in Figure 6.3.

6.4 Discriminative Joint SAT Experiments

This section starts with experiments on the WSJ0 SI-84 setup (see Appendix A.3) and the resulting method is applied to Mandarin CTS data (see Appendix A.5) as validation later. The SI-84 system results are listed in Table 6.2. The H parameter associated with the MPE criterion is the F-smoothing coefficient in Eqn. (6.7), which has been tuned to generate the lowest WER. When $H = 0.0$, the standard MPE criterion is used. The CD-DNN structure is $351 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 3007$, the same as the SI-84 speaker independent (SI) CD-DNNs used in Section 4.5.1. The SAT CE DNN-HMM system J0 was constructed using context-independent (CI) initialisation, which reduced the WER on average by 9.1% relative compared to the SI system S5 in Table 4.4. Four epochs of MPE training with a learning rate of 1.6×10^{-4} improved the performance with a further 3.0% relative WER reduction, which is similar to that obtained from the Tamil LLP system in Table 6.1. J2 was constructed using

Table 6.2 SI-84 MPE SAT DNN-HMM system performance with a 65k word trigram LM.

ID	Criterion	Parameters to Update			%WER	
		CD-DNN	LinXForm	CMLLR	Dev	Eval
J0	CE	√		×	7.8	8.9
J1	MPE $H = 0.0$	√		×	7.6	8.6
J2	MPE $H = 0.0$	√		√	7.4	8.3
J3	MPE $H = 0.2$	√		√	7.2	8.3
J4	MPE $H = 0.0$	√	√	×	7.5	8.5

Table 6.3 72h Mandarin CTS MPE SAT DNN-HMM system performance with a trigram LM on Eval97, Eval03, and Dev04 test sets.

ID	Criterion	Parameters to Update		%WER		
		CD-DNN	CMLLR	Eval97	Eval03	Dev04
S0[1]	CE (Baseline)			34.1	29.7	27.9
J5	MPE $H = 0.0$	√	×	31.2	27.3	25.7
J6	MPE $H = 1.2$	√	√	31.2	27.0	25.5

discriminative joint SAT based on the standard MPE criterion, and performed better than the CE and MPE SAT baseline system J0 and J1. A further improved system, J3, was obtained by using F-smoothing in joint SAT with $H = 0.2$ that outperformed J1 by 4.4%. However, it was found that F-smoothing did not improve the SI DNN-HMM MPE training performance, perhaps due to differences in lattice design and silence modelling compared to those used in (Su et al., 2013). The LinXForm method, J4, also produced the lowest WER when no F-smoothing was used. J4 performed better than the baseline J1 but worse than both J2 and J3.

Four epochs of MPE training with a learning rate of 4.0×10^{-3} were applied to the CMLLR based CE DNN-HMMs in Table 5.3, and the results are presented in Table 6.3. Compared to the sigmoid baseline system S0 in Table 5.3, the MPE SAT DNN-HMM system J5 produced lower WER by 8.2% relative on average. The discriminative joint SAT system, J6, only outperformed J5 by 0.5%, and the improvements were not consistent across all three test sets. This is perhaps because the over-fitting issue by discriminative adaptation is more severe on this data set than on WSJ, and therefore, it requires a fairly large F-smoothing coefficient $H = 1.2$ to regularise the unsupervised discriminative input transform learning, which is not helpful in training SI parameters.

6.5 Training GMMs using SGD

Tandem system joint training is investigated in following sections of this chapter, which has the BN DNN for feature extraction and GMM acoustic models concurrently updated through SGD based MPE training. First, the GMM log-likelihood and gradient calculation methods are modified to fit the GPU hardware. Next, the tandem system configuration is revisited to simplify the build procedure and the parameters used more efficiently. Third, techniques required by the traditional GMM-HMM MPE training, I-smoothing and percentile based variance floor, are adapted from the EBW to SGD optimisation framework. Finally, the tandem system joint training approach with techniques to stabilise and improve training performance is proposed.

6.5.1 GPU based GMM calculations

From Eqn. (2.24), the log of $c_{im}\mathcal{N}(\mathbf{z}(t)|\mu_{im},\sigma^2_{im})$ is calculated by

$$\log c_{im} - \frac{D}{2}\ln(2\pi) - \sum_{d=1}^{D}\log\sigma_{imd} - \frac{1}{2}\sum_{d=1}^{D}\frac{(z_d(t)-\mu_{imd})^2}{\sigma^2_{imd}}, \tag{6.8}$$

and the log-likelihood of the GMM is obtained by summing $\log(c_{im}\mathcal{N}(\mathbf{z}(t)|\mu_{im},\sigma_{im}))$ using the *log add* algorithm presented in Algorithm 3 below (Young et al., 2015), in which the constants are determined empirically. Most calculations happen in computing the fourth term in Eqn. (6.8), which requires $T\times S\times M\times D$ summations and $2\times T\times S\times M\times D$ multiplications given the total number of tied states S, the number of Gaussians in each state M, and the samples in a minibatch T.

Algorithm 3 Calculate $\log(\exp(x)+\exp(y))$ approximately

procedure LogAdd(x,y)
  $v^{\min} \leftarrow \min(x,y)$
  $v^{\max} \leftarrow \max(x,y)$
  $v^{\text{diff}} \leftarrow v^{\min} - v^{\max}$
  if $v^{\text{diff}} < -23.0258$ **then**
    if $v^{\max} < -0.5\times 10^{10}$ **then**
      return -1.0×10^{10}
    else
      return $v^{\max}$
  else
    return $v^{\max} + \log\left(1+\exp\left(v^{\text{diff}}\right)\right)$

A speed-up method used in this thesis for log-likelihood calculations is based on[1]

$$\begin{aligned}&\sum_{t=1}^{T}\sum_{i=1}^{S}\sum_{m=1}^{M}\sum_{d=1}^{D}(z_d(t)-\mu_{imd})^2/\sigma_{imd}^2 \\ =&\sum_{t=1}^{T}\sum_{i=1}^{S}\sum_{m=1}^{M}\sum_{d=1}^{D}\frac{z_d^2(t)}{\sigma_{imd}^2}-2\sum_{t=1}^{T}\sum_{i=1}^{S}\sum_{m=1}^{M}\sum_{d=1}^{D}\frac{z_d(t)\mu_{imd}}{\sigma_{imd}^2}+\sum_{t=1}^{T}\sum_{i=1}^{S}\sum_{m=1}^{M}\sum_{d=1}^{D}\frac{\mu_{imd}^2}{\sigma_{imd}^2}.\end{aligned} \tag{6.9}$$

By collecting the mean and variance vectors of all Gaussians in all tied states as a mean matrix and a variance matrix both of dimension $SM \times D$, and collecting the T samples as two $T \times D$-dimensional sample matrices containing $z_d(t)$ and $z_d^2(t)$, the terms

$$\sum_{t=1}^{T}\sum_{i=1}^{S}\sum_{m=1}^{M}\sum_{d=1}^{D}\frac{z_d^2(t)}{\sigma_{imd}^2}$$
$$\sum_{t=1}^{T}\sum_{i=1}^{S}\sum_{m=1}^{M}\sum_{d=1}^{D}\frac{z_d(t)\mu_{imd}}{\sigma_{imd}^2}$$

can be computed efficiently using the highly optimised general matrix multiplication (GEMM) functions in the basic linear algebra subprograms (BLAS) library (Dongarra et al., 1990).

A difficulty in SGD based GMM training is to update c_{im} and σ_{im} under the constraints $c_{im} > 0$, $\sum_{m=1}^{M} c_{im} = 1$, and $\sigma_{im} > 0$, while SGD is an unconstrained optimisation method. These can be ensured by the following parameter transformations (Juang et al., 1997; Young, 1990),

$$c_{im} = \frac{\exp(\tilde{c}_{im})}{\sum_{m'=1}^{M}\exp(\tilde{c}_{im'})} \tag{6.10}$$

$$\sigma_{imd} = \exp(\tilde{\sigma}_{imd}), \tag{6.11}$$

[1] $1/\sigma_{imd}^2$ instead of σ_{imd}^2 is actually stored and used in the calculation, in order to convert divisions to multiplications.

where $\tilde{\sigma}_{imd}$ and $\tilde{c}_{im'}$ are the actual unconstrained parameters updated by SGD. Therefore, by applying the chain rule, the partial derivatives of $\bar{\mathcal{F}}(\mathbf{O})$ are

$$\frac{\partial \bar{\mathcal{F}}(\mathbf{O})}{\partial \tilde{c}_{im}} = \sum_{t=1}^{T} \bar{\gamma}_{im}(t)\,(1 - c_{im}) \tag{6.12}$$

$$\frac{\partial \bar{\mathcal{F}}(\mathbf{O})}{\partial \mu_{imd}} = \sum_{t=1}^{T} \bar{\gamma}_{im}(t)\,\frac{z_d(t) - \mu_{imd}}{\sigma_{imd}^2} \tag{6.13}$$

$$\frac{\partial \bar{\mathcal{F}}(\mathbf{O})}{\partial \tilde{\sigma}_{imd}} = \sum_{t=1}^{T} \bar{\gamma}_{im}(t)\,\frac{z_d^2(t) - 2z_d(t)\mu_{imd} + \mu_{imd}^2 - \sigma_{imd}^2}{\sigma_{imd}^2} \tag{6.14}$$

$$\frac{\partial \bar{\mathcal{F}}(\mathbf{O})}{\partial z_d(t)} = \sum_{s=1}^{S} \sum_{m=1}^{M} \bar{\gamma}_{im}(t)\,\frac{z_d(t) - \mu_{imd}}{\sigma_{imd}^2}, \tag{6.15}$$

where $\bar{\gamma}_{im}(t)$ is the posterior probability of being in Gaussian mixture component m of state i at time t calculated by multiplying $\bar{\gamma}_i(t)$ with $\pi_{im}(t)$, and $\pi_{im}(t)$ is defined by Eqn. (2.31). When $\bar{\mathcal{F}}(\mathbf{O})$ is the ML, MMI, or MPE objective function, $\bar{\gamma}_i(t)$ is the ML state occupancy $\gamma_i(t)$, scaled MMI state occupancy $\kappa\gamma_i^{\text{MMI}}(t)$, or scaled MPE state occupancy $\kappa\gamma_i^{\text{MPE}}(t)$, which has been defined in Eqn. (2.87), (2.86), or (2.101), respectively.

In order to speed up the gradient value calculation in Eqns. (6.13) and (6.14), similar to the log-likelihood calculation, all $\bar{\gamma}_{im}(t)$ are arranged as an $SM \times T$-dimensional occupancy matrix, and the terms $\sum_{t=1}^{T} \bar{\gamma}_{im}(t) z_d(t)$ and $\sum_{t=1}^{T} \bar{\gamma}_{im}(t) z_d^2(t)$ are calculated by multiplying the occupancy matrix with the sample matrices $z_d(t)$ and $z_d^2(t)$ using the GEMM functions in the BLAS library. Meanwhile, it is observed that many $\pi_{im}(t)$ values are very small. This causes the related gradient values to be very small and can be ignored in calculation. Therefore, a threshold is used as a lower limit, and any $\pi_{im}(t)$ smaller than the threshold is set to 0.0, which makes the occupancy matrix more sparse and can speed up training further.

6.5.2 Revisiting tandem system construction

From the results in Table 3.7, although performance of the SI tandem system with 39-dimensional BN features is rather close to the SI hybrid system trained in the same way, the tandem system used both FBANK and PLP features while the hybrid system used only FBANK features, and PLP features were actually derived from FBANK features. Such feature redundancy causes a less efficient parameter use, for example, the tandem system in Table 3.7 has about 17M parameters, while the hybrid system

only has about 10M parameters. The tandem system configuration is revised in this section, in order to use the parameters more efficiently.

The revised tandem system build procedure is presented below.

1. Train a BN DNN based on the alignments generated by pretrained models.
2. Construct a monophone GMM-HMM system using BN features $\mathbf{y}^{\text{bn}}(t)$ derived from the BN DNN without any modification, i.e., $\mathbf{z}(t) = \mathbf{y}^{\text{bn}}(t)$. The GMM-HMMs are estimated using the normal ML training.
3. Expand the monophone BN GMM-HMM system to an initial tied state triphone BN GMM-HMM system using ML training, and the tied states are generated by the standard decision tree tying method using $\mathbf{y}^{\text{bn}}(t)$.
4. The final ML triphone BN GMM-HMM system is trained using the two-model re-estimation method, with alignments for decision tree tying produced by the well-trained initial triphone system. The final ML system can be further refined by the conventional GMM-HMM MPE training.

The key differences between the revised tandem configuration presented above and the previous one presented in Section 3.9.1 lie in two aspects:

- First, only BN features are used for GMM construction in the revised configuration, and the features are not decorrelated for GMM training, while the previous configuration used both HLDA projected PLPs and STC transformed BN features.
- Second, the previous procedure reused the tied state decision trees produced based on PLP features, while the revised configuration builds new decision trees for BN features.

All tandem systems in the rest of this chapter were built with the revised configuration, and main differences between tandem and hybrid systems include the use of the BN layer and the output layer with diagonal covariance GMMs rather than a normal DNN layer with the softmax function.

6.6 MPE Training for GMMs with SGD

This section adapts the traditional EBW based GMM-HMM MPE training to the SGD based optimisation framework. The SGD based GMM training uses the formulas in

Section 6.5.1. From Section 2.7.4, two techniques need to be adapted to SGD for MPE, namely I-smoothing and the use of a percentile based variance floor.

6.6.1 Parameter smoothing and weight decay

Recall the I-smoothing method introduced in Section 2.7.4, which applies a data dependent interpolation between MPE and ML objective functions. The data availability of each Gaussian component is taken into account with the Gaussian mixture component dependent interpolation coefficient, $\tau_{im}(\mathbf{O})$, which causes each $\gamma_{im}^{\text{MPE}}(t)$ to be increased by a constant τ according to the definition in Eqn. (2.103). When applying a dynamic MMI prior, the change to $\gamma_{im}^{\text{MPE}}(t)$ is the same as for ML based I-smoothing, and the key difference between the two methods lies in the use of MMI rather than ML estimated prior parameters in the EBW update formulas Eqns. (2.105) – (2.108). However, in the SGD based update formulas in Eqns. (3.35) and (3.36), the objective function can only influence parameter changes through the term $\partial\mathcal{F}[n]|_{\Theta[n]}/\partial\theta$, which is calculated using EBP according to $\gamma_{im}^{\text{MPE}}(t)$. As a result, ML based I-smoothing and the MMI dynamic prior have identical effects in SGD. To simulate I-smoothing with a dynamic MMI prior in the SGD framework, the H-criterion is used to interpolate $\mathcal{F}^{\text{MPE}}$ with $\mathcal{F}^{\text{MMI}}$ using a constant weighting coefficient τ^{MMI}, and $\mathcal{F}^{\text{MMI}}$ is pre-smoothed by I-smoothing with $\tau_{im}^{\text{ML}}(\mathbf{O})$. Thus, the objective function is

$$\mathcal{F}^{\text{MPE}}(\mathbf{O}) + \tau^{\text{MMI}} \left(\mathcal{F}^{\text{MMI}}(\mathbf{O}) + \tau_{im}^{\text{ML}}(\mathbf{O}) \mathcal{F}^{\text{ML}}(\mathbf{O}) \right). \tag{6.16}$$

An interesting question to investigate through SGD based MPE training is the cause of the over-fitting issue in traditional GMM-HMM MPE training. If SGD based GMM-HMM MPE training can improve test set WERs without I-smoothing, the over-fitting is caused by the EBW algorithm; otherwise, it is due to GMM distributions with individual diagonal covariance matrices, since MPE training for DNN-HMMs with softmax output distributions does not require any regularisation (as seen in Section 6.2). Furthermore, since it is easy to apply other types of regularisation methods to SGD, such as the weight decay method (see Section 3.6), it is also possible to compare the effects from I-smoothing and weight decay. As a result, the full objective function used for SGD based GMM-HMM MPE training is

$$\mathcal{F}^{\text{MPE}}(\mathbf{O}) + \tau^{\text{MMI}} \left(\mathcal{F}^{\text{MMI}}(\mathbf{O}) + \tau_{im}^{\text{ML}}(\mathbf{O}) \mathcal{F}^{\text{ML}}(\mathbf{O}) \right) + \frac{\varepsilon}{2} \sum\nolimits_{\theta} \theta^2, \tag{6.17}$$

where θ is a parameter associated with the mth Gaussian in the ith state, and ε is the weight decay coefficient. When differentiating the objective function, $\tau_{im}^{\mathrm{ML}}(\mathbf{O})$ is viewed as a constant, and $\partial\mathcal{F}^{\mathrm{MPE}}(\mathbf{O})/\partial\theta$, $\partial\mathcal{F}^{\mathrm{MMI}}(\mathbf{O})/\partial\theta$, and $\partial\mathcal{F}^{\mathrm{ML}}(\mathbf{O})/\partial\theta$ are calculated using Eqns. (6.4) – (6.6).

6.6.2 The percentile based variance floor

As mentioned in Section 2.7.4, the use of a percentile based variance floor is beneficial to stabilise training after each parameter update in MPE training with EBW. To apply this method to the SGD framework, two issues need to be solved.

- How frequently should the variance floor be applied? Though using it only once at the end of each epoch, as in the EBW framework, is insufficient to prevent variance reduction since the SGD update occurs after each utterance, using it over frequently can increase variance values as the relative threshold $\sigma_d^2(p\%)$ is obtained based on current parameters.
- How can $\sigma_d^2(p\%)$ be calculated more efficiently? The traditional method is based on a sorting algorithm, since it is executed once per epoch and hence the efficiency is not very important. To use this method more frequently during the entire SGD training procedure, $\sigma_d^2(p\%)$ needs to be obtained with fewer calculations.

To solve the first issue, experiments were conducted to find out an appropriate application frequency. It is found that flooring the variance after every 10 updates is helpful to maintain the variance value range, and thus is used in all experiments later in Section 6.8. To save the cost from computing the exact $\sigma_d^2(p\%)$ by sorting algorithms,

$$\sigma_d^2(p\%) \approx \mu(\sigma_d^2) + \Phi^{-1}(\frac{p}{100}) \cdot \sigma(\sigma_d^2) \tag{6.18}$$

is used as an approximate threshold, where $\mu(\sigma_d^2)$ and $\sigma(\sigma_d^2)$ are the mean and standard deviation of all variance values of dimension d, and $\Phi(\cdot)$ is the CDF for the standard Gaussian distribution. This reduces the computation complexity from $\mathcal{O}(N \log N)$ to $\mathcal{O}(N)$, where $N = SM$ is the number of Gaussian components, by applying an assumption that variance values associated with the same dimension follow a Gaussian distribution.

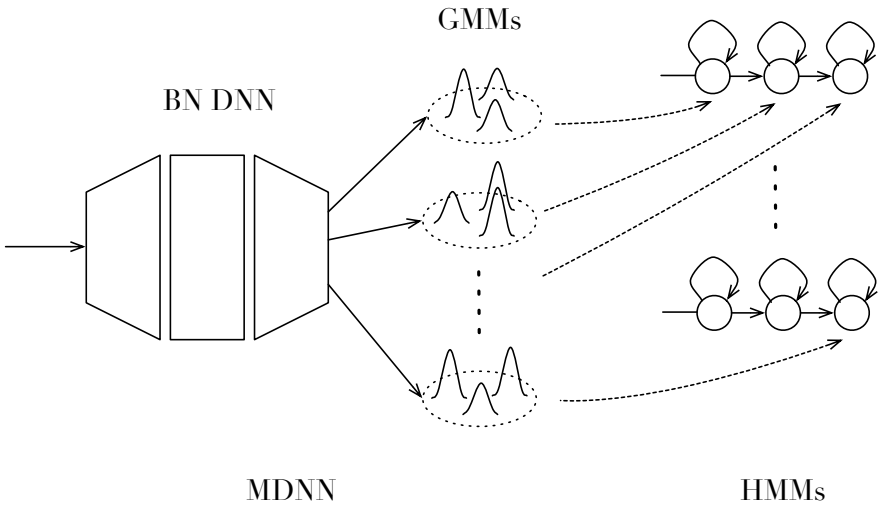


Figure 6.4 MDNN-HMMs used for tandem system joint training.

6.7 Tandem System Joint Training

The tandem system can be seen as MDNN-HMM models by combining the BN DNN feature extraction module and GMM-HMM acoustic model module together, which is shown in Figure 6.4, and the tandem system joint training is carried out as MDNN-HMM MPE training in this thesis. This section addresses various issues in this approach. It should be noted that MDNN, or mixture density neural network, refers to an ANN model with a GMM layer rather a softmax layer for the output in this thesis, i.e., each output target distribution is modelled by a GMM. A very similar term, *mixture density network* or MDN, often refers to a different regression model that generates real-valued outputs using a GMM whose parameters are the output from ANN models (Bishop, 1994). MDN has been applied to speech processing for both speech synthesis (Zen and Senior, 2014) and acoustic-articulatory inversion (Richmond, 2002, 2006) tasks.

6.7.1 Use of ReLU to replace linear activation functions

In practice, it is observed that the use of linear activation functions in the BN layer can cause a stability issue in training. This happens when the average of $\partial\mathcal{F}/\partial\mathbf{y}^{\text{bn}}(t)$ over a minibatch moves from positive to negative and the parameters can become stuck at a very poor solution. To prevent this phenomenon from happening, the ReLU function can be used to take the place of linear activation functions to keep the average of $\partial\mathcal{F}/\partial\mathbf{y}^{\text{bn}}(t)$ positive. Furthermore, in order to avoid the information loss caused by

rectification, $\mathbf{y}^{\text{bn}}(t)$ is transformed to

$$\tilde{\mathbf{y}}^{\text{bn}}(t) = \mathbf{y}^{\text{bn}}(t) - \mu(\mathbf{y}^{\text{bn}}) + 6\,\sigma(\mathbf{y}^{\text{bn}}) \tag{6.19}$$

by modifying the BN layer bias vector $\mathbf{b}^{\text{bn}}$, where $\mu(\mathbf{y}^{\text{bn}})$ and $\sigma(\mathbf{y}^{\text{bn}})$ are the mean and standard deviation vectors of $\mathbf{y}^{\text{bn}}$ estimated over the training set. This guarantees that 99.99966% of $\mathbf{y}^{\text{bn}}$ samples are rectified without information loss, assuming $\mathbf{y}^{\text{bn}}(t)$ follows a multivariate Gaussian distribution. Finally, the Gaussian components constructed on $\mathbf{y}^{\text{bn}}(t)$ features, $\mathcal{N}(\mathbf{y}^{\text{bn}}(t)|\mu_{im}, \sigma^2_{im})$, can be transformed to fit the transformed BN feature $\tilde{\mathbf{y}}^{\text{bn}}(t)$ as

$$\mathcal{N}(\tilde{\mathbf{y}}^{\text{bn}}(t)|\mu_{im} - \mu(\mathbf{y}^{\text{bn}}) + 6\,\sigma(\mathbf{y}^{\text{bn}}), \sigma^2_{im})$$

without any retraining.

6.7.2 Relative update value clipping

As seen in Table 6.1, in SGD training, a fairly large learning rate, which is necessary for fast convergence, can sometimes cause severe performance degradation. This can be due to large inaccurate gradients generated due to various reasons such as poor acoustic conditions and erroneous reference labels *etc.* A possible solution is the batch normalisation method introduced in Section 3.6, but this was found to cause DNN models to over-fit on small training sets (e.g., 15h). A widely used alternative solution that prevents the parameters changing too much in a single update is the update value clipping (see Section 3.6). However, as the standard method requires specific clipping thresholds, it is tedious to use it here as the MDNN has both GMM and DNN parameters, the values of which are rather different in range.

Here a method to find a relative threshold for clipping a particular collection of parameters, Θ, is proposed. Let $\delta_\theta[n]$ be the proposed change of θ at the nth update, the mean and standard deviation of $|\delta_\theta[n]|$ for all $\theta \in \Theta$ are $\mu(|\delta_\Theta[n]|)$ and $\sigma(|\delta_\Theta[n]|)$, and then $|\delta_\theta[n]|$ is clipped according to a threshold

$$\upsilon = \mu(|\delta_\Theta[n]|) + m\,\sigma(|\delta_\Theta[n]|), \tag{6.20}$$

where m is the relative threshold. The obtained threshold υ is then used in the update value clipping method in Eqn. (3.38). Θ can be $\{c_{im}\}$, $\{\mu_{imd}\}$, and $\{\sigma_{imd}\}$ for all i and m, or $\mathbf{W}^{(l)}$ or $\mathbf{b}^{(l)}$ for a particular layer l.

6.7.3 Amplified GMM learning

The MDNN output layer has a rather different functional form from other DNN layers. As shown later in Section 6.8.1, the learning rate suitable for GMM parameters is significantly larger than a normal DNN learning rate. Thus, in MDNN-HMM sequence training, different learning rates, η and $\alpha\eta$ are used for the BN DNN and GMMs separately, where $\alpha > 1$ is the amplification factor. To regularise training appropriately, the weight decay factor ε for GMMs is also scaled by α.

6.7.4 Parameter updating schemes

Three different parameter update schemes for tandem system joint training are investigated in this thesis:

1. Update GMMs and hidden layers in an interleaved manner, which may also be useful as a regulariser; or
2. Update all parameters concurrently without restriction; or
3. Update all MDNN parameters concurrently, then update GMMs only to fit BN features better.

These updating schemes are compared later in Section 6.8.2.

6.8 Tandem System Joint Training Experiments

The tandem system joint training experiments were first carried out on the MGB 50h data set. The DNN acoustic model and BN DNN feature extraction structures are $720 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 3984$ and $720 \times 1000 \times 1000 \times 1000 \times 1000 \times 39 \times 1000 \times 3982$, respectively. The configurations were later validated on the MGB 200h data set, and the related DNN acoustic model and BN DNN feature extraction structures are $720 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 6029$ and $720 \times 1000 \times 1000 \times 1000 \times 1000 \times 39 \times 1000 \times 6027$, respectively. The DNN layer MPE training used a fixed learning rate of 1.0×10^{-4} and a relative update value clipping method threshold of $m = 3$, whereas a threshold of $m = 9$ for GMMs in joint training. Note that the BN features were not decorrelated for the diagonal coveriance GMMs, which is discussed later in Section 6.8.3.

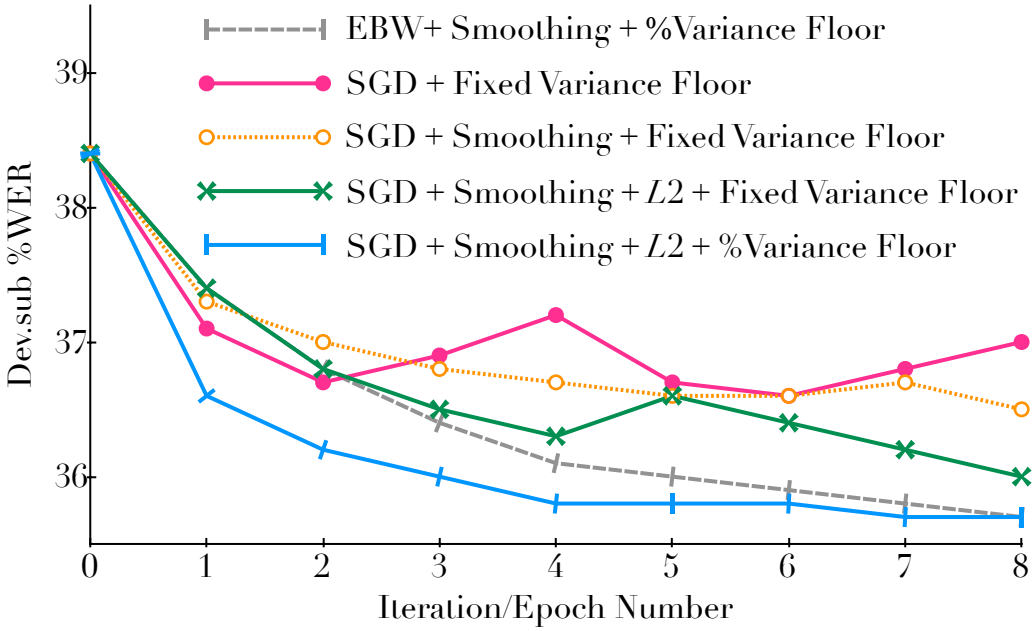


Figure 6.5 MGB 50h GMM-HMM MPE system %WERs on Dev.sub with a 160k word trigram LM.

6.8.1 GMM-HMM MPE training

EBW and SGD based GMM-only training were performed on the 50h training set, and compared in Figure 6.5. Both EBW and SGD MPE training started from a baseline ML BN GMM-HMM system with a WER of 38.4%. EBW MPE training with I-smoothing, a dynamic MMI prior, and a percentile based variance floor, can reduce the WER to 36.1% after four iterations. Unlike EBW, SGD based MPE GMM training with a learning rate of 5.0×10^{-3} and no regularisation can also reduce WER, though the results fluctuate over the eight epochs. This reveals the over-fitting issue in traditional GMM-HMM MPE training was caused by the EBW algorithm.

Next, the parameter smoothing method (described in Section 6.6) and weight decay were added. τ^{MMI} and τ^{ML} per frame were set to 3.0×10^{-5} and 2.0×10^{-6}, and ε was 4.0×10^{-4}. It can be seen that both parameter smoothing and weight decay help stabilise and improve the performance. When the percentile based variance floor is finally applied, SGD based MPE training consistently reduced the WER with every epoch and gave a WER of 35.8% after the 4th epoch. It can be seen from Figure 6.5 that the final SGD based MPE GMM training works at least well as the EBW based method.

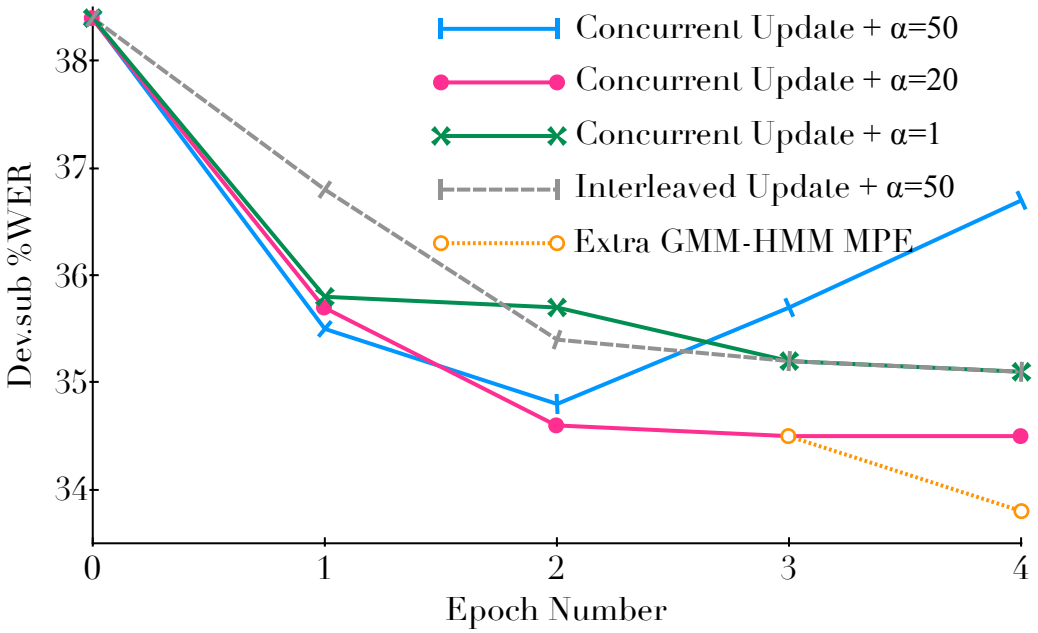


Figure 6.6 MGB 50h jointly trained tandem system %WERs on Dev.sub with a 160k word trigram LM.

6.8.2 MDNN-HMM MPE training

From the experiments in Section 6.8.1, the learning rates suitable for the GMM layer are 50 times larger than those for the hidden layers. Here, different parameter update schemes and different GMM learning amplification factors were compared. The BN DNN learning rate and weight decay factor were set to $\eta = 1.0 \times 10^{-4}$ and $\varepsilon = 4.0 \times 10^{-5}$, respectively. The results are shown in Figure 6.6. For concurrent updates, it can be seen that $\alpha = 1$ and 20 gave a consistent WER reduction across epochs, and a WER of 34.5% was obtained when $\alpha = 20$. If α is further increased to 50, a WER of 34.6% was found at the 2nd epoch, but severe over-fitting occurred thereafter. In the interleaved update scheme, the GMM layer was updated first, and a WER of 35.1% after updating GMMs and hidden layers each for two epochs was achieved. If three epochs of the concurrent update and one epoch of SGD based GMM-HMM MPE training were applied, both with $\alpha = 20$, the best 50h SI tandem system WER of 33.8% was obtained. Compared to the use of concurrent update throughout all four epochs, the performance improvement was acquired by freezing the hidden layer parameters in the last epoch, which makes the GMMs fit BN features after the 3rd epoch better.

6.8.3 Further experiments

Table 6.4 contains results on the 50h training set. H_0^{50h} is the baseline CE DNN-HMM hybrid system, which has a relative lower WER of 3.9% than the ML trained BN GMM-HMM system; after MPE training, the WER is further reduced by 7.3% and H_1^{50h} is produced. T_2^{50h} outperforms H_1^{50h} since tandem system joint training gives a larger improvement than DNN-HMM MPE training. Note that H_1^{50h} and T_2^{50h} also have similar numbers of parameters (8.7M and 8.8M, respectively). By using alignments from T_2^{50h} for DNN-HMM training, a 0.6% absolute WER reduction was obtained, which was slightly better than using alignments produced by the MPE DNN-HMM system H_1^{50h}. If the training targets were derived from the tied states of T_2^{50h}, another 0.4% absolute WER reduction was acquired, as T_2^{50h} decision trees were constructed on BN features that are more suitable for clustering DNN output targets than PLPs.

Compared with the tandem system structure used in this section (shown in Figure 6.4), the CUED traditional SI tandem system structure (presented in Figure 3.4) uses addtional HLDA projected PLP features in the GMMs, and also decorrelates the BN features by a STC transform. A single class STC transform was also used to decorrelate the ML BN GMM-HMM system T_0^{50h}, by integrating the transform into the BN layer using the method introduced in Section 2.6.3, and the resulting ML system performance was improved by 0.2% absolute WER reduction, but the resulting MPE MDNN-HMM system has the same WER as T_2^{50h}, which shows that the joint training could adjust features to fit GMMs with diagonal covariance matrices. The CUED traditional SI MPE tandem system constructed based on the same 50h setup produced a WER of 35.2% on the Dev.sub test set, which is lower than that produced by T_0^{50h} due to the PLP features in the GMMs, but this also obviously increased the number of parameters, and is therefore not a fair comparison with the hybrid system setup.

The proposed approach was then validated on the larger 200h training set. All MDNN-HMM MPE training parameters are the same as the 50h systems', except for the learning rate of 2.5×10^{-5}. Based on the results presented in Table 6.5, the jointly trained MPE MDNN-HMM system, T_1^{200h}, is comparable to the MPE trained DNN-HMM system, H_1^{200h}, both in performance and size. This is consistent with the 50h system results. Finally, the use of traditional GMM-HMM techniques, such as MLLR and joint decoding, was studied for MPE MDNN-HMMs. With test-time unsupervised MLLR adaptation based on the hypotheses produced by T_1^{200h}, the SD system T_2^{200h} outperformed the SI system T_1^{200h} by 4.0% relative WER reduction. Joint decoding was used to combine H_2^{200h} with either T_1^{200h} or T_2^{200h}, and the resulting systems J_1^{200h}

Table 6.4 MGB 50h system performance with a 160k word trigram LM on Dev.sub.

ID	System	%WER
T_0^{50h}	ML BN-GMM-HMMs	38.4
T_1^{50h}	MPE BN-GMM-HMMs	36.1
T_2^{50h}	MPE MDNN-HMMs	33.8
H_0^{50h}	CE DNN-HMMs	36.9
H_1^{50h}	MPE DNN-HMMs	34.2
H_2^{50h}	MPE DNN-HMMs + H_1^{50h} alignments	33.7
H_3^{50h}	MPE DNN-HMMs + T_2^{50h} alignments	33.6
H_4^{50h}	MPE DNN-HMMs + T_2^{50h} alignments & trees	33.2

Table 6.5 MGB 200h system performance with a 160k word trigram LM on Dev.sub.

ID	System	%WER
T_0^{200h}	ML BN-GMM-HMMs	33.7
T_1^{200h}	MPE MDNN-HMMs	29.8
T_2^{200h}	MPE MDNN-HMMs + MLLR	28.6
H_0^{200h}	CE DNN-HMMs	31.9
H_1^{200h}	MPE DNN-HMMs	29.6
H_2^{200h}	MPE DNN-HMMs + T_1^{200h} alignments & trees	29.0
J_1^{200h}	$T_1^{200h} \otimes H_2^{200h}$ joint decoding	28.3
J_2^{200h}	$T_2^{200h} \otimes H_2^{200h}$ joint decoding	27.4

and J_2^{200h} outperformed their constituent systems, which showed the complementarity between DNN-HMMs and MDNN-HMMs. For J_1^{200h}, the complementarity only came from the difference between a normal softmax output layer and GMMs with a BN hidden layer. For J_2^{200h}, speaker adaptation was also involved by the MDNN-HMMs and the lowest WER in this section, 27.4%, was achieved. These examples demonstrate the benefit of reusing standard GMM-HMM methods for MDNN-HMMs.

6.9 Summary and Conclusions

This chapter studies ASR system level joint training using both hybrid and tandem system configurations. In the hybrid configurations, DNN-HMM MPE training was first investigated as a direct extension to MLP discriminative sequence training (Kingsbury, 2009). The improvements from our systems are similar to those reported by other researchers (McDermott et al., 2014; Su et al., 2013; Veselý et al., 2013; Wiesler

et al., 2015). Next, MPE training was applied to joint SAT. In the example studied in this chapter, the CMLLR initialised input transforms for feature extraction and CD-DNN acoustic models were concurrently updated in MPE training, and a frame level CE training smoothed MPE training method was used to update testing speaker SD parameters in an unsupervised manner. This method resulted in 4.4% and 0.5% relative WER reductions with the 15h WSJ SI-84 setup and 72h Mandarin CTS setup, respectively. The smaller improvement to Mandarin CTS systems is perhaps due to over-fitting to the hypotheses used as supervision.

Afterwards, in the tandem system configuration, the tandem system build procedure was revisited. Conventional EBW based GMM-HMM MPE training was extended to the SGD framework and applied to MDNN-HMM MPE training for the tandem system joint training. A set of methods were modified or proposed to improve training performance, which include I-smoothing, a dynamic MMI prior, a percentile based variance floor, linear to ReLU activation function conversion, relative update value clipping, amplified GMM learning, and different parameter update schemes. WERs obtained from the refined tandem system were comparable to those from MPE trained hybrid systems. The resulting tandem system was found useful for hybrid system construction and system combination. Finally, experiments have also shown that jointly trained tandem systems can also benefit from existing GMM based approaches, such as MLLR and joint decoding, which can further reduce system WERs. Note that although MPE was used as the examplar criterion in this chapter, other discriminative sequence training criteria, such as MMI and state level minimum Bayes' risk (MBR), can be used with the same methods.

Although hybrid systems are simpler in construction, separately trained tandem systems sometimes have lower WER than the hybrid systems. Yan et al. (2013) showed MMI DNN-HMM had higher WERs than the GMM-HMMs trained on the features produced by the same DNN and projected by a Karhunen-Loève transform (KLT). In IEEE ASRU 2015 MGB Challenge (Bell et al., 2015), the best performance single ASR system was a SAT tandem system constructed based on the the CUED traditional recipe (Woodland et al., 2015). There are several further reasons that make the tandem approach, especially the jointly trained one, still of interest. First, DNN and GMMs can be combined to form an MDNN, which is a general framework for modelling non-Gaussian conditional probability distributions. This is in contrast to the distributions generated by a conventional DNN acoustic model with a softmax output function, which have equivalent terms to single Gaussians with a shared covariance matrix, as shown in Section 3.3.1. Second, it is straightforward to improve the performance of tandem

systems by applying techniques developed for GMMs to MDNN such as adaptation and decorrelation methods. Finally, tandem and hybrid systems are known to produce complementary errors, and hence, performance improvements can be obtained by system combination. Such complementarity is due to the use of the BN layer and different output distributions.

6.10 Related Work

Discriminative adaptation was widely explored using GMM-HMM models (Povey et al., 2003a,b; Wang and Woodland, 2008; Yu et al., 2009) based on the MPE and MMI criteria. Using DNN-HMM models, almost all adaptation researches mentioned in Section 5.7, such as (Abdel-Hamid and Jiang, 2013; Doddipatla et al., 2014; Gemello et al., 2007; Huang et al., 2015; Li and Sim, 2010; Liao, 2013; Neto et al., 1995; Ochiai et al., 2014; Seide et al., 2011b; Siniscalchi et al., 2013; Swietojanski et al., 2015, 2016; Swietojanski and Renals, 2014; Tan et al., 2015; Woodland et al., 2015; Wu and Gales, 2015; Wu et al., 2016a; Yao et al., 2012; Yu et al., 2013; Zhang and Woodland, 2016; Zhao et al., 2015), used different discriminative criteria. Swietojanski et al. (2016) found CE based LHUC test-time adaptation can be applied to MBR trained hybrid systems. Huang et al. (2015) used sequence training while the others used frame level training. Seide et al. (2011b) jointly learned randomly initialised input transforms together with the CD-DNN acoustic model through CE training, which required constructing a large block diagonal input matrix to hold the input transforms associated with each frame in every minibatch. Ochiai et al. (2014) used a similar approach to make any of the hidden layers SD. Tüske et al. (2015b) used the last hidden layer as SD and set them to CMLLR transforms. Such SD transforms were kept frozen in joint training.

For the joint tandem system training, Bengio (1991) and Bengio et al. (1992) trained ANN-HMMs based on the MMI criterion for phoneme recognition, which can be regarded as a simple tandem system since the ANNs have an output layer with each state having a single Gaussian distribution. For LVCSR, both CE and ML training have been applied to MDNN (Variani et al., 2015) and standard DNN with a parameterised softmax output function (Tüske et al., 2015a,b). MPE training has also been applied to the task with GMMs and BN DNN interleavingly optimised by EBW and SGD (Paulik, 2013).

Chapter 7

Conclusions and Future Work

7.1 Contributions and Conclusions

This thesis investigates joint training methods for the statistical HMM based ASR approach. Two types of joint training are proposed, namely DNN acoustic model joint training and ASR system level joint training. DNN acoustic model joint training aims at removing dependencies on other ASR systems or pre-determined activation functions in the DNN acoustic model design. ASR system level joint training, on the other hand, has different ASR modules concurrently optimised after their construction in the traditional way. Joint training of the feature extraction and acoustic model modules using the widely used tandem and hybrid system configurations is studied in this thesis. This relies on combining the various modules involved as a single deep ANN model, and applying the error backpropagation (EBP) algorithm for flexible architectures proposed in Section 3.4. Actually all methods developed in this thesis benefit from the advances in deep learning research. The major contributions in this thesis include the items listed below:

- Proposed a feature mixture structure and its corresponding forward and back-propagation algorithms, which is used to provide a generic support to flexible ANN architectures.
- Proposed a context-dependent (CD) standalone training method that comprises of context-independent (CI) standalone training and DNN based decision tree state tying.
- Proposed the use of parameterised sigmoid and ReLU hidden activation functions for both speaker independent (SI) acoustic modelling and speaker adaptation.

- Proposed a DNN discriminative sequence training based speaker adaptive training (SAT) framework, and applied it to train CMLLR initialised input transforms.
- Proposed a tandem system joint training approach by MDNN-HMM discriminative sequence training, and a set of techniques to improve and stabilise the training.

These contributions are summarised in detail in the rest of this section.

In Chapter 4, a standalone training method that builds a CD-DNN without relying on any existing system was proposed. The method could be divided into two parts: discriminative PT with integrated realignment to first train context independent DNNs without relying on previously generated alignments; and CD-DNN decision tree target clustering, which is a modification of the standard decision tree state tying based on explicitly estimating approximations of equivalent terms to CD-DNN output distributions. Experiments showed that the proposed techniques yielded comparable WER performance to CD-DNNs that rely on GMM-HMMs. Furthermore, a CI initialisation method is proposed to initialise CD DNN FT with hidden layers trained to classify a small number of CI states. CI initialisation has been found to serve as an effective regulariser which is complementary to the standard weight decay method. Experiments on 15h and 50h corpora have shown that CI initialisation can significantly reduce resulting WERs over the baselines with generative and CD discriminative PT, while saving a large amount of time for PT.

Chapter 5 proposes to use the parameterised forms of sigmoid and ReLU as hidden activation functions in SI DNN acoustic modelling. Three parameters are used with the sigmoid: the curve's maximum value, the curve's steepness, and a scaled horizontal displacement. In this way, the parameterised sigmoid function can perform piecewise approximations to other activation functions. Meanwhile, both positive and negative parts of the ReLU are separately associated with a linear scaling factor, which allows an unconstrained trade-off between the positive and negative activations. The role of the parameters has been investigated in the context of Mandarin CTS data experiments and using an adaptive output value amplifier for each artificial neuron has been found to be the most effective configuration for both parameterised sigmoid and ReLU. Later in the chapter, this type of configuration is applied to the speaker adaptation task by using the amplifiers as speaker dependent (SD) parameters, and DNN adaptation becomes re-weighting the importance of different hidden artificial neurons for each speaker. This adaptation scheme is applied to both DNN acoustic models in hybrid systems and bottleneck (BN) DNN feature extraction in tandem systems. Unsupervised

adaptation experiments using MGB data show that the technique is effective in directly adapting DNN acoustic models and the BN features, and combines well with other DNN adaptation techniques. Reductions in WER are consistently obtained using parameterised sigmoid and ReLU activation functions for multiple hidden layer adaptation.

ASR system level joint training is studied in Chapter 6. DNN-HMM discriminative sequence training is first investigated as a cornerstone method for joint training, and the results on both 15h and 200h data sets show obvious and consistent WER reduction from minimum phone error (MPE) trained DNN-HMMs over the baselines. Discriminative joint SAT is proposed as a hybrid system joint training instance, and DNN-HMMs with CMLLR input transform normalised features are the case studied in discriminative joint SAT. Once the DNN-HMMs have been constructed using cross entropy (CE), MPE sequence training is applied to both the DNN acoustic model and the CMLLR initialised input transforms. Input transforms associated with testing speakers are estimated in an unsupervised manner with the DNN parameters kept frozen. It is found that using the F-smoothing method by interpolating the MPE with the CE objective function is useful to alleviate the discriminatively learned SD parameters from over-fitting to the hypotheses served as supervision. The use of discriminative joint SAT has been found to bring considerable improvement by systems trained on the WSJ SI-84 setup, but improvement from the Mandarin CTS system is marginal, perhaps due to over-fitting to the supervision.

The second part of Chapter 6 proposes to jointly optimise the separately trained GMM acoustic model and BN DNN feature extraction in a traditional tandem system. Such an acoustic model and feature extraction can be viewed as an MDNN model that is initialised with a conventional tandem system, which is then refined by lattice based MPE training. The tandem system construction procedure is first revisited, and the related parameter smoothing and percentile based variance floor methods are modified for use with SGD. It is found through comparative study that the over-fitting issue in traditional GMM-HMM MPE training has been caused in part by the EBW algorithm. Next, a number of methods were used to improve the system performance, which include linear to ReLU conversion, relative update value clipping, amplified GMM learning, and different parameter update schemes, which were found to stabilise and improve MDNN-HMM MPE training. The final combined method yields comparable performance to MPE trained DNN-HMMs on MGB systems, and further experiments show that the jointly trained tandem system is useful in DNN-HMM construction and system combination.

The aforementioned methods validate that the proposed joint training is an effective way to improve statistical HMM based ASR, by having more parameters jointly optimised together. On one hand, DNN acoustic model joint training comprises several methods that can improve the DNN training procedure and the resulting system performance. ASR system level joint training, on the other hand, overcomes one weakness lying in the divide and conquer ASR construction strategy that the acoustic model built by greedily optimising an objective function based on pre-extracted features is not guaranteed to have optimal parameters. The idea of first initialising system components carefully and then training them jointly is possible to be applied to more complicated systems or pipelines.

7.2 Future Work

The most straightforward extension to the methods developed in this thesis is perhaps to replace DNN models by other deep ANN models, such as CNNs, LSTMs, and CLDNN *etc.* As presented in Appendix B, the ANN extension to HTK used by this thesis supports all these deep models.

Furthermore, as mentioned in Section 3.11.3, commonly used alternative methods to DNN training without alignments include CTC and lattice-free maximum mutual information (MMI) training. Similar ideas can be applied to MPE training as well, which can generate MPE hybrid systems while skipping the frame level training stage. By using such methods in SAT, the training set SD parameters, such as input feature transforms or p-Sigmoid/p-ReLU amplifiers, can be learned more naturally from the beginning of acoustic model training and according to the final objective function.

Both p-Sigmoid and p-ReLU based SAT training can be used in tandem system joint training as well. If the BN DNN model is adapted for training set speakers first, all following tandem system construction stages can be conducted in an SAT fashion, such as SAT decision tree tying, which is possible to produce better tandem systems and leads to better joint training.

Finally, it would be very interesting to use p-Sigmoid and p-ReLU in unfolded RNN models, which can produce the output values by the shared recurrent layer re-weighted at each time step. This is possibly an alternative way to enable RNN to hold longer memory. Actually even the RNNs layer, or other LSTM or GRU parameters, can be shared differently at different time steps.

Appendix A

Data Sets and System Setup

A.1 Babel Conversational Telephone Speech

The original task objective of the IARPA Babel program (Harper, 2011) is to find all the exact matches of a specific word/phrase query in given speech data, which uses a speech recognition stage followed by a word-spotting stage from lattices. However, the focus here is only the speech to text conversion accuracy, so speech recognition error rate is used as the evaluation standard. This thesis involves some experiments conducted on the CTS data from the Cantonese full language pack (FLP) and the Tamil limited language pack (LLP). The Cantonese FLP full training set, Cantonese test set, Tamil LLP full training set, and Tamil test set consist of 145.5, 16.5, 14.2, and 11.7 hours of audio data from 80,152, 9,134, 11,809, and 10,862 utterances respectively. The audio was sampled at a 8kHz sampling rate.

The Cantonese and Tamil ASR systems were constructed based on a phone set consisting of 32 and 33 phones apart from `sil` and `sp` respectively, and the input feature was 52-dimensional `PLP_D_A_T`, unless expressly stated. The DNN input feature vector was expanded according to a context shift set $\mathbf{c} = \{-4, -3, -2, -1, 0, +1, +2, +3, +4\}$, and normalised by 0-MN and 1-VN based on the statistics collected from all data in each conversation side. Discriminative PT and FT were used for all DNN training, and the detailed configurations are listed in Table A.1. The bigram LMs applied during Cantonese and Tamil decoding were constructed based on 27k words and 21k vocabularies respectively, and were estimated on the training data transcriptions only.

Table A.1 The Babel DNN CE training configuration.

Option	Value
minibatch size	800
ρ	0.5
data shuffling	frame level
PT scheduler	List
$\eta[1]$	1.0×10^{-3}
FT scheduler	NewBob or NewBob$^+$
$\eta[1]$	2.0×10^{-3}
$\Delta\mathcal{F}_{\text{ramp}}$	0.005
$\Delta\mathcal{F}_{\text{stop}}$	0.005
N_{min} (if applicable)	12

A.2 Multi-Genre Broadcast Task

The ASRU 2015 Multi-Genre Broadcast (MGB) challenge data (Bell et al., 2015) are used in this thesis to evaluate the proposed techniques. The audio consists of seven weeks of BBC television programmes with a raw total duration of 1,600 hours, and was sampled at a 16kHz sampling rate. The data covers a full range of genres, e.g. news, comedy, drama, sports, quiz shows, documentaries *etc.* The audio was pre-processed using a lightly supervised decoding process (Chan and Woodland, 2004; Lamel et al., 2002; Lanchantin et al., 2016), and 200 hours of data from 2,180 shows were randomly selected for which the subtitles and the lightly supervised output had a phone matched error rate of less than 20% (Lanchantin et al., 2015). The 200 hour training set consists of 115,932 utterances that were automatically clustered into 10,930 speaker clusters (Karanasou et al., 2015; Wang et al., 2016). A 50 hour subset was evenly sampled from the 200 hour set as the 50 hour training set, which includes 32,771 utterances and 3,272 speaker clusters.

Apart from `sil`, there are 47 phones in the phone set. Two vocabularies were used in decoding, one has 64k words and the other has 160k words. Trigram and 4-gram LMs were estimated using either vocabulary. All LMs were trained on 650M words of audio transcriptions and additional MGB subtitles, and pruned with an entropy based beam of 1.0e-9. The full test set, **Dev.full**, has 28 hours of audio data from 47 different shows and is the official full MGB transcription development set (Bell et al., 2015). The manual segmentation was processed by automatic speaker clustering that resulted in 30,690 uttterances and 285 speaker clusters. The official subset of Dev.full,

Dev.sub, is also used in the experiments, which contains 5.5 hours data from 12 shows. There are 8,713 utterances and 285 speaker clusters in Dev.sub, which has 69 seconds data per speaker on average. More details about the dictionary, LMs, and test sets can be found in (Woodland et al., 2015).

Two types of acoustic features were used, 40-dimensional FBANK and 13-dimensional PLP coefficients, which were further expanded to 80-dimensional `FBANK_D` and 52-dimensional `PLP_D_A_T`. The inputs to all DNNs were produced by stacking the features according to $\mathbf{c} = \{-4, -3, -2, -1, 0, +1, +2, +3, +4\}$, and normalised by 0-MN and 1-VN based on the statistics calculated with all valid speech segments in each show. The overlapping segments were ignored in scoring. All GMM-HMMs have 16 Gaussians per state, except for the `sil` states. All CE DNN training used the training configuration listed in Table A.2.

Table A.2 The MGB DNN CE training configuration.

Option	Value
minibatch size	800
ρ	0.5
data shuffling	frame level
PT scheduler	List
$\eta[1]$	1.0×10^{-3}
FT scheduler	NewBob$^+$
$\eta[1]$	2.0×10^{-3}
$\Delta\mathcal{F}_{\text{ramp}}$	0.0005
$\Delta\mathcal{F}_{\text{stop}}$	0.0001
N_{min}	16

A.3 Wall Street Journal Read Speech

The read speech corpora created by DARPA based on the Wall Street Journal (WSJ) news text have been widely used for acoustic modelling since the 1990s (Paul and Baker, 1992). There are two corpora for training, known as WSJ0 and WSJ1, for which the texts to be read were selected within the WSJ text corpus. There are two training set configurations, SI-84 and SI-284. SI-84 consists of 15 hours of audio from 7,185 utterances from 84 different speakers in WSJ0. SI-284 includes 36,493 utterances from 284 different speakers from both WSJ0 and WSJ1, and the total duration is about 66 hours. The sampling rate of the audio was 16kHz. The LIMSI phone set was used in

all system construction, with 45 phones plus the silence units. The input feature vector is 52-dimensional `PLP_D_A_T` projected to 39-dimensional by HLDA. The DNN input feature is still stacked based on $\mathbf{c} = \{-4, -3, -2, -1, 0, +1, +2, +3, +4\}$, with utterance level 0-MN and global 1-VN. The CE DNN training setup follows that in Table A.3.

Two test sets were used in this thesis, namely the 1994 H1-dev (denoted as **Dev**) and November 1994 H1-eval (denoted as **Eval**), both of which had text prompts from a variety of North American Business News types. The Dev set contains 310 utterances from 20 speakers, while the Eval set has 316 utterances from 20 different speakers. The LIMSI derived dictionary with 65k words, along with a trigram LM were used for all related decoding (Woodland et al., 1995).

Table A.3 The WSJ DNN CE training configuration.

Option	Value
minibatch size	800
ρ	0.5
data shuffling	frame level
PT scheduler	List
$\eta[1]$	1.0×10^{-3}
FT scheduler	List
$\eta[1] - \eta[6]$	1.0×10^{-3}
$\eta[7] - \eta[12]$	2.0×10^{-3}

A.4 Aurora-4 Multi-Condition Read Speech

The Aurora-4 multi-condition corpus was obtained by adding multiple noise conditions to the WSJ0 SI-84 training set (Pearce, 2002). Six different types of noises between 10dB to 20dB were used. The test set was constructed based on a subset of the Nov92 NIST evaluation set with 330 utterances from 8 speakers, with 14 test conditions created from different recording channels and noises. These 14 conditions were further grouped into 4 subsets of the Aurora-4 test set, denoted as A, B, C, and D. All decoding used a modified version of the CMU dictionary with 5k words and a bigram LM for the standard WSJ0 SI-84 setup (Woodland et al., 1994).

The GMM-HMM input feature is also 52-dimensional `PLP_D_A_T`. The DNN input is 80-dimensional `FBANK_D` stacked based on $\mathbf{c} = \{-4, -3, -2, -1, 0, +1, +2, +3, +4\}$. Utterance level 0-MN and global 1-VN were applied. The CE DNN training setup is shown in Table A.4.

Table A.4 The Aurora-4 DNN CE training configuration.

Option	Value
minibatch size	200
ρ	0.9
ε	0.001
υ (update value)	0.32
data shuffling	frame level
PT scheduler	List
$\eta[1]$	1.0×10^{-3}
FT scheduler	NewBob$^+$
$\eta[1]$	2.0×10^{-3}
$\Delta\mathcal{F}_{\text{ramp}}$	0.001
$\Delta\mathcal{F}_{\text{stop}}$	0.001
N_{min}	16

A.5 Mandarin Conversational Telephony Speech

A 72 hour CTS data set is also used in this thesis. It contains 50 hours of LDC 2004 CTS Mandarin data, as well as 22 hours of LDC Call Home Mandarin and Call Friend Mandarin data. The training set consists of 786 conversation sides, and was used in building the 2004 Cambridge University HTK-based Mandarin system (Gales et al., 2005). Three test sets were involved in performance evaluation: a two hour development set **Dev04** containing 24 conversations and 48 speakers, a 1.1 hours 2003 evaluation data set with 12 conversations and 24 speakers taken from the Call Friend data, **Eval03**, as well as **Eval97**, a 1.5 hours 1997 NIST Hub4 Mandarin evaluation set with 20 conversations and 49 speakers.

The base phone set contains 46 toneless phones or 124 tonal phones, and the recognition word list contains 63k words, comprising of about 52k multi-character Chinese words, 5k single character Chinese words, and an additional 5k frequent English words (Liu et al., 2015). Decoding was performed with a trigram LM trained using a total of one billion words of text data.

The acoustic feature vector consists of 52-dimensional `PLP_D_A_T` coefficients normalised by vocal length normalisation (Lee and Rose, 1996) and projected to 39-dimensional using HLDA. Pitch features extracted by the Kaldi toolkit (Ghahremani et al., 2014; Povey et al., 2011), along the their first and second order differentials were appended to the projected vector. The 42-dimensional augmented vector was further normalised by speaker level 0-MN and 1-VN, and then used to build SAT

GMM-HMMs for CMLLR transform estimation. After transformation by CMLLR, the feature vector was normalised again to 0-MN and 1-VN, and stacked according to $\mathbf{c} = \{-4, -3, -2, -1, 0, +1, +2, +3, +4\}$ to form the DNN input vector. The CE DNN training configuration is presented in Table A.5

Table A.5 The Mandarin CTS DNN CE training configuration.

Option	Value
minibatch size	800
ρ	0.5
data shuffling	frame level
PT scheduler	List
$\eta[1]$	1.0×10^{-3}
FT scheduler	NewBob$^+$
$\eta[1]$	2.0×10^{-3}
$\Delta\mathcal{F}_{\text{ramp}}$	0.001
$\Delta\mathcal{F}_{\text{stop}}$	0.005
N_{min}	12

A.6 TED Talks

The TED (*Technology, Entertainment, and Design*) talk corpus is a publicly available English data set consisting of 143 hours of speech from 813 TED talks (speakers) (Cettolo et al., 2012). Three test sets are used for this task, namely **Dev2010**, **Tst2010**, and **Tst2011**, which contain 8, 11, and 8 10-minute talks, respectively. A 4-gram LM estimated from 751M words is used during decoding. Details of the data sets, phone set, dictionary, and LM can be found in (Bell and Renals, 2015a).

The input feature vector to the GMM-HMMs is a 52-dimensional `PLP_D_A_T` feature. The DNN input vector is 80-dimensional `FBANK_D` stacked based on $\mathbf{c} = \{-4, -3, -2, -1, 0, +1, +2, +3, +4\}$ and normalised at the speaker level. The DNN CE training configuration is the same as the MGB systems as in Table A.2.

Appendix B

A General ANN Extension for HTK

HTK is a research source code toolkit designed primarily for ASR with more than 100,000 users around the world. Previously, the most recent version is 3.4.1 which was released in 2008, and contains many commonly used HMM based ASR techniques, such as GMM decision tree tying, MLLR and CMLLR adaptation, and lattice-based discriminative training. These have allowed the construction of ASR systems for both research and commercial deployment. Beyond the official HTK release, there are a number of extensions, and the most well-known is the HTS system for parametric speech synthesis (Zen et al., 2007). However, previous versions of HTK did not support ANNs, and therefore the use of ANNs in HTK-based systems relied on external ANN tools such as QuickNet (Johnson, 2012; Knill et al., 2013; Zhang and Woodland, 2014). This appendix describes the recently developed ANN extension to HTK (HTK-ANN), which has been widely used inside CUED and partly released in HTK version 3.5.

B.1 Design Principles

Three principles were applied to the design of HTK-ANN.

1. In order to accommodate new models and methods easily, without sacrificing efficiency, the design should be as generic as possible. HTK-ANN supports ANNs with flexible input feature configurations and model architectures, and relies on a universal definition of ANN layer input features.

2. The new ANN modules should be compatible with as many existing functions in HTK as possible, which minimises the effort to reuse previous HTK related

source code and tools in the new framework and simplifies the transfer of many techniques, for instance, sequence training and speaker adaptation, from the GMM to the ANN framework.

3. It should be "research friendly" so that further extensions and modifications can be created. To promote ease of reuse and future extensions, the functions are designed to be fine-grained and loosely coupled.

B.2 Implementation Details

HTK includes many widely used speech processing technologies, covering the entire ASR pipeline. Many of these features including front-end feature extraction, HMM state clustering, feature transforms, sequence training, large vocabulary decoders, as well as lattice generation and processing, are all used in the design of ANN based ASR systems. Therefore, implementing native support of ANNs in HTK can simplify the use of all of these established approaches and allow ANN-based systems to benefit from the HTK infrastructure. This is achieved by developing HTK-ANN as a number of new HTK modules (libraries) and tools, and extending other libraries and tools to be compatible with them. An overview of the newly added and extended HTK modules and tools is shown in Table B.1 and B.2.

Table B.1 A list of the ANN related HTK modules.

Name	Type	Function Descriptions/Updates
`HANNet`	new	ANN structures and core algorithms
`HArc`	extended	CUDA based lattice FB algorithm
`HCUDA`	new	CUDA based math kernel functions
`HFBLat`	extended	Lattice based $\gamma_i^{\mathrm{MPE}}(t)$ and $\gamma_i^{\mathrm{MMI}}(t)$ computation
`HMath`	extended	ANN related math kernel functions
`HModel`	extended	ANN model reading/writing interface
`HNCache`	new	Data cache for data random access
`HParm`	extended	Using ANN output values as GMM input features
`HShell`	extended	ANN related user interface changes

B.3 ANN Support and Training Methods

HTK-ANN supports DCG based generic ANN structures defined in Section 3.2, and all parameters can be SD and shared at the vector and matrix level. Apart from

Table B.2 A list of the ANN related HTK tools.

Name	Type	Function Descriptions/Updates
`HCompV`	extended	Tandem feature mean and variance estimation
`HDecode`	extended	Tandem/hybrid system LVCSR decoder
`HDecode.joint`	new	Joint decoding of HTK AMs
`HDecode.mod`	extended	Tandem/hybrid system model marking
`HERest`	extended	Tandem system ML training
`HHEd`	extended	ANN model creation and editing
`HMMIRest`	extended	Tandem system discriminative sequence training
`HNForward`	new	ANN evaluation and output generation
`HNTrainSGD`	new	SGD based tandem/hybrid system joint training
`HVite`	extended	Tandem/hybrid decoder and alignment

Eqn. (3.6) that the values from the feature elements are concatenated, the values from the elements can be combined in other ways, such as a weighted summation and multiplication *etc*, and the combination weights can be any input values, parameters, or output values from another layer. All activation functions mentioned in this thesis, such as softmax, linear, sigmoid, p-Sigmoid, ReLU, p-ReLU, soft ReLU, and subsampling *etc* has been implemented. Using these mechanisms, CNN, DNN, RNN layers, and their combination models are supported. Other toolkits that support similar types of models include Theano (Bastien et al., 2012), RWTH-ASR (Wiesler et al., 2014a), CNTK (Yu et al., 2014), TensorFlow (Abadi et al., 2015), and Kaldi nnet3 (Povey et al., 2011). GMMs with diagonal covariance matrices are re-implemented in HTK-ANN using Eqns. (6.12) – (6.15), which can be seen as a special layer and reused together with other common layers.

Training ANN models with DAG structures (FNNs) use Eqn. (3.32), and recurrent ANN training requires to convert the model structures to DAG first by unfolding and truncating the recurrent layers. Both frame level criteria such as CE and MMSE, and discriminative sequence criteria including MMI, MPE, and MWE are supported. For different training modes, the supervision used for ANN training can come from label files with frame to label alignments (for CE and MMSE training), lattice files (for lattice based MMI, MPE, and MWE), feature data files, and the output values from some ANN layer. HTK-ANN supports many commonly used training techniques introduced in Section 3.6, such as momentum, gradient/update value clipping and scaling, weight decay, dropout, and batch normalisation. For learning rate schedulers, examples from every type mentioned in Section 3.6.1 were implemented, including the List and NewBob$^+$ schedulers used in this thesis.

B.4 Data Cache

As mentioned in Section 3.5.2, it is important to access the data in a randomised order in SGD training. In order to minimise the I/O cost from random data access, loading data into the memory through a cache is usually crucial. To make the cache sufficiently large, HTK was extended to have the full 64bit support. Three different data rearrangements are available in the data cache to support different types of models and training modes.

1. Frame level randomisation/shuffling. All frames from all utterances in the cache are shuffled, which is a commonly used approach for frame level ANN training. However, in HTK-ANN, in order to process FNNs with any architecture, context shifts for every frame in the minibatch should be available at the same time. This also paves the way for training unfolded RNNs.

2. Utterance based shuffling. This is useful in sequence training and SAT.

3. Batch of utterance level shuffling. A batch of utterances is randomly selected and processed in parallel. Once an utterance is finished, a new utterance will be loaded to the empty position in the batch. This kind of cache is sometimes used for folded RNN training *etc.*

Since efficiency is a key factor in cache design, an extra thread can be enabled to load data into the cache, while the main thread is working on ANN forward/backward propagation.

B.5 Interfacing ANNs with HMMs

In hybrid configurations, each ANN output target is associated with an HMM state, and the output probabilities are converted to log-likelihoods using Eqn. (3.50). This enables the HMM based techniques in HTK, such as lattice generation, rescoring, and decoding, to be reused by ANN-HMMs.

In tandem configurations, BN features can be written out and transformed into normal HTK feature data files using the `HNForward` tool, and the tandem system processing becomes normal GMM-HMM processing based on such features. An alternative way that saves additional data file generation is to feed the ANN layer output values to GMMs directly through the traditional HTK HMM feature processing module `HParm`.

Such direct ANN features can be normalised using a parameterised linear activation function

$$f_i^{(l)}(a_i^{(l)}(t)) = \alpha_i^{(l)} a_i^{(l)}(t) + \beta_i^{(l)}, \tag{B.1}$$

where $\alpha_i^{(l)}$ and $\beta_i^{(l)}$ are the parameters associated with AN i. When using data dependent $\alpha_i^{(l)}$ and $\beta_i^{(l)}$ with

$$\alpha_i^{(l)} = \frac{1}{\sigma_i^{(l)}} \tag{B.2}$$

$$\beta_i^{(l)} = -\frac{\mu_i^{(l)}}{\sigma_i^{(l)}}, \tag{B.3}$$

where $\mu_i^{(l)}$ and $\sigma_i^{(l)}$ are the mean and standard deviation of $a_i^{(l)}(t)$ calculated over the corresponding data samples, Eqn. (B.1) acts as the on-the-fly 0-MN and 1-VN to $a_i^{(l)}(t)$. `HCompV`, `HERest`, `HMMIRest`, and the decoders can use this function to build tandem systems. In `HNTrainSGD`, a set of GMMs can be implicitly converted to an ANN layer, which makes the tandem system an MDNN-HMM for the joint training proposed in Section 6.5.

B.6 Other Key Features

B.6.1 Math kernels

A set of math kernel functions required by ANN processing was added. All new kernels have standard CPU, Intel MKL, and CUDA based implementations. Both single and double precision float numbers are supported.

B.6.2 Input transforms and speaker adaptation

In HTK, ANNs can directly utilise many types of SI and SD input transforms estimated by GMM-HMMs, including HLDA, STC, and CMLLR *etc.* Meanwhile, since all ANN parameters are stored as either matrices or vectors, a new light-weight speaker adaptation mechanism was added to sequence level training, which can swap some small-grained ANN parameter units, represented by matrices and vectors, according to speaker ids.

B.6.3 Model editing

Like previous versions of HTK, the HTK model editor tool, `HHEd`, is used to edit the structure of ANN models. Current edit operations include insertion, removal or initialisation of a layer; changing the activation functions or dimensions of a layer; modifying a feature mixture by adding or removing a feature element, or changing its associated context shift set. These operations can be used to generate ANNs with any DCG equivalent architectures, and any intermediate models for PT. `HHEd` can also associate an isolated ANN with a GMM-HMM set by assigning each ANN output target an HMM state.

B.6.4 Decoders

Both the standard HTK Viterbi decoder `HVite` and the LVCSR decoder `HDecode` were updated to support tandem and hybrid system decoding. In ANN-HMM training, `HVite` is also used to produce the frame to label alignments. Model marking on word lattices with ANN-HMMs has been implemented by both `HVite` and `HDecode.mod`, a variant of `HDecode`, which is required by MPE. Another more recent `HDecode` variant, `HDecode.joint`, performs joint decoding of multiple HTK AMs with a shared decision tree, as defined in Section 3.10.4.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. http://tensorflow.org.

Abdel-Hamid, O. and Jiang, H. (2013). Fast speaker adaptation of hybrid NN/HMM model for speech recognition based on discriminative learning of speaker code. In *Proc. ICASSP*, Vancouver, Canada.

Abdel-Hamid, O., Mohamed, A.-R., Jiang, H., Deng, L., Penn, G., and Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22:1533–1545.

Anastasakos, T., McDonough, J., Schwartz, R., and Makhoul, J. (1996). A compact model for speaker adaptive training. In *Proc. ICSLP*, Philadelphia, PA, USA.

Atal, B. and Hanauer, S. (1971). Speech analysis and synthesis by linear prediction of the speech wave. *Journal of the Acoustical Society of America*, 50:637–655.

Bacchiani, M. and Ostendorf, M. (1998). Using automatically-derived acoustic subword units in large vocabulary speech recognition. In *Proc. ICSLP*, Sydney, Australia.

Bacchiani, M. and Rybach, D. (2014). Context dependent state tying for speech recognition using deep neural network acoustic models. In *Proc. ICASSP*, Florence, Italy.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*, San Diego, CA, USA.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016). End-to-end attention-based large vocabulary speech recognition. *arXiv 1508.04395*.

Bahl, L., Bakis, R., Jelinek, F., and Mercer, R. (1980). Language-model/acoustic-channel-model balance mechanism. *IBM Technical Disclosure Bulletin*, 23:3464–3465.

Bahl, L., de Souza, P., Gopalakrishnan, P., Nahamoo, D., and Picheny, M. (1991). Decision trees for phonological rules in continuous speech. In *Proc. ICASSP*, Toronto, Canada.

Baker, J. (1975). The DRAGON system – An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23:24–29.

Barker, J., Josifovski, L., Cooke, M., and Green, P. (2000). Soft decisions in missing data techniques for robust automatic speech recognition. In *Proc. Interspeech*, Beijing, China.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, Lake Tahoe, CA, USA.

Baum, L. and Egon, J. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8.

Bell, P., Gales, M., Hain, T., Kilgour, J., Lanchantin, P., Liu, X., McParland, A., Renals, S., Saz, O., Wester, M., and Woodland, P. (2015). The MGB challenge: Evaluating multi-genre broadcast media transcription. In *Proc. ASRU*, Scottsdale, AZ, USA.

Bell, P. and Renals, S. (2015a). Complementary tasks for context-dependent deep neural network acoustic models. In *Proc. Interspeech*, Dresden, Germany.

Bell, P. and Renals, S. (2015b). Regularization of context-dependent deep neural networks with context-independent multi-task training. In *Proc. ICASSP*, Brisbane, Australia.

Bengio, Y. (1991). *Artificial Neural Networks and Their Application to Sequence Tecognition*. PhD thesis, McGill University.

Bengio, Y. (2009). *Learning Deep Architectures for AI (Foundations and Trends in Machine Learning)*. Now Publishers Inc, Boston, MA, USA.

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2012). Advances in optimizing recurrent networks. *arXiv 1212.0901*.

Bengio, Y., De Mori, R., Flammia, G., and Kompe, R. (1992). Phonetically motivated acoustic parameters for continuous speech recognition using artificial neural networks. *Speech Communication*, 11:261–271.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, 19:153–160.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166.

Berndt, D. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *Proc. AAAI*, Seattle, WA, USA.

Beulen, K. and Ney, H. (2000). Automatic question generation for decision tree based state tying. In *Proc. ICASSP*, Istanbul, Turkey.

Bi, M., Qian, Y., and Yu, K. (2016). Very deep convolutional neural networks for LVCSR. In *Proc. ICASSP*, Shanghai, China.

Biem, A., Katagiri, S., McDermott, E., and Juang, B.-H. (2002). An application of discriminative feature extraction to filter-bank-based speech recognition. *IEEE Transactions on Speech and Audio Processing*, 9:96–110.

Bishop, C. (1994). Mixture density networks. Technical report, NCRG 4288, Neural Computing Research Group, Aston University.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer, New York, NY, USA.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proc. COMPSTAT*, Paris, France.

Bourlard, H., Konig, Y., and Morgan, N. (2016). REMAP: Recursive estimation and maximization of a posteriori probabilities in connectionist speech recognition. In *Proc. Interspeech*, San Francisco, CA, USA.

Bourlard, H. and Morgan, N. (1993). *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, MA, USA.

Bridle, J. (1990a). Alpha-nets: A recurrent 'neural network architecture with a hidden Markov model interpretation. *Speech Communication*, 9:83–92.

Bridle, J. (1990b). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Fogelman Soulié, F. and Hérault, J., editors, *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236. Springer-Verlag, New York, NY, USA.

Brown, P. (1987). *The Acoustic-Modeling Problem in Automatic Speech Recognition*. PhD thesis, Carnegie Mellon University.

BUT (2013). TNet. http://speech.fit.vutbr.cz/software/neural-network-trainer-tnet.

Byrne, W., Beyerlein, P., Huerta, J., Khudanpur, S., Marthi, B., Morgan, J., Peterek, N., Picone, J., Vergyri, D., and Wang, W. (2000). Towards language independent acoustic modeling. In *Proc. ICASSP*, Istanbul, Turkey.

Caruana, R. (1993). Multitask connectionist learning. In *Proc. Connectionist Models Summer School*, Boulder, CO, USA.

Cettolo, M., Girardi, C., and Federico, M. (2012). Wit^3: Web inventory of transcribed and translated talks. In *Proc. EAMT*, Trento, Italy.

Chan, H. and Woodland, P. (2004). Improving broadcast news transcription by lightly supervised discriminative training. In *Proc. ICASSP*, Montreal, Canada.

Chen, K. and Huo, Q. (2016). Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering. In *Proc. ICASSP*, Shanghai, China.

Chesta, C., Laface, P., and Ravera, F. (1997). Bottom-up and top-down state clustering for robust acoustic modeling. In *Proc. Eurospeech*, Rhodes, Greece.

Chou, P. (1991). Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:340–354.

Choukri, K. and Chollet, G. (1986). Adaptation of automatic speech recognizers to new speakers using canonical correlation analysis techniques. *Computer Speech and Langauge*, 1:95–107.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv 1412.3555*.

Church, K. and Gale, W. (1991). A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5:19–54.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (1990). *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA.

Cox, S. and Bridle, J. (1989). Unsupervised speaker adaptation by probabilistic spectrum fitting. In *Proc. ICASSP*, Glasgow, UK.

Cui, X. and Goel, V. (2015). Maximum likelihood nonlinear transformations based on deep neural networks. In *Proc. ICASSP*, Brisbane, Australia.

Dahl, G., Sainath, T., and Hinton, G. (2013). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Proc. ICASSP*, Vancouver, Canada.

Dahl, G., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20:30–42.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28:357–366.

Dayan, P. and Abott, L. (2001). *Theorectical Neuroscience*. MIT Press, Cambridge, MA, USA.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. (2012). Large scale distributed deep networks. *Advances in Neural Information Processing Systems*, 25:1223–1231.

Delcroix, M., Kinoshita, K., Yu, C., Ogawa, A., Yoshioka, T., and Nakatani, T. (2016). Context adaptive deep neural networks for fast acoustic model adaptation in noisy conditions. In *Proc. ICASSP*, Shanghai, China.

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–38.

Demuynck, K., Duchateau, J., Van Compernolle, D., and Wambacq, P. (2000). An efficient search space representation for large vocabulary continuous speech recognition. *Speech Communication*, 30:37–53.

Diestel, R. (1997). *Graph Theory*. Springer-Verlag, New York, NY, USA.

Digalakis, V., Rtischev, D., and Neumeyer, L. (1995). Speaker adaptation using constrained estimation of Gaussian mixtures. *IEEE Transactions Speech and Audio Processing*, 3:357–366.

Doddipatla, R., Hasan, M., and Hain, T. (2014). Speaker dependent bottleneck layer training for speaker adaptation in automatic speech recognition. In *Proc. Interspeech*, Singapore.

Dongarra, J., du Croz, J., and Duff, I. (1990). A set of level 3 basic linear algebra subprograms. *IEEE Transactions on Mathematical Software*, 16:1–17.

Doumpiotis, V. and Byrne, W. (2004). Pinched lattice minimum Bayes risk discriminative training for large vocabulary continuous speech recognition. In *Proc. Interspeech*, Lisbon, Portugal.

Duchi, J., Hazan, E., and Singer, Y. (2010). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*. Wiley, New York, NY, USA.

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., and Garcia, R. (2001). Incorporating second-order functional knowledge for better option pricing. *Advances in Neural Information Processing Systems*, 13:472–478.

Ellis, D. and Morgan, N. (1999). Size matters: An empirical study of neural network training for large vocabulary continuous speech recognition. In *Proc. ICASSP*, Phoenix, AZ, USA.

Evermann, G., Chan, H., Gales, M., Jia, B., Mrva, D., Woodland, P., and Yu, K. (2005). Training LVCSR systems on thousands of hours of data. In *Proc. ICASSP*, Philadelphia, PA, USA.

Evermann, G. and Woodland, P. (2000). Large vocabulary decoding and confidence estimation using word posterior probabilities. In *Proc. ICASSP*, Istanbul, Turkey.

Frankel, J., Wester, M., and King, S. (2007). Articulatory feature recognition using dynamic bayesian networks. *Computer Speech and Language*, 21:620–640.

Fung, P., Byrne, W., Zheng, T., Kamm, T., Liu, Y., Song, Z., Venkataramani, V., and Ruhi, U. (2000). Pronunciation modeling of Mandarin casual speech. Technical report, Workshop 2000 for Language Engineering for Students and Professionals Integrating Research and Education.

Furui, S. (1986). Speaker-independent isolated word recognition based on emphasized spectral dynamics. In *Proc. ICASSP*, Tokyo, Japan.

Gales, M. (1995). *Model-based Techniques for Robust Speech Recognition*. PhD thesis, University of Cambridge.

Gales, M. (1998). Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech and Language*, 12:75–98.

Gales, M. (1999). Semi-tied covariance matrices for hidden Markov models. *IEEE Transactions on Speech and Audio Processing*, 7:272–281.

Gales, M. (2000). Cluster adaptive training of hidden Markov models. *IEEE Transactions on Speech and Audio Processing*, 8:417–428.

Gales, M. (2002). Maximum likelihood multiple subspace projections for hidden Markov models. *IEEE Transactions on Speech and Audio Processing*, 10:37–47.

Gales, M., Jia, B., Liu, X., Sim, K., Woodland, P., and Yu, K. (2005). Development of the CUHTK 2004 Mandarin conversational telephone speech transcription system. In *Proc. ICASSP*, Philadelphia, PA, USA.

Gales, M., Knill, K., and Ragni, A. (2015). Unicode-based graphemic systems for limited resource languages. In *Proc. ICASSP*, Brisbane, Australia.

Gales, M. and Woodland, P. (1996). Mean and variance adaptation within the MLLR framework. *Computer Speech and Language*, 10:249–264.

Gales, M., Woodland, P., Chan, H., Mrva, D., Sinha, R., and Tranter, S. (2006). Progress in the CU-HTK broadcast news transcription system. *IEEE Transactions on Audio, Speech, and Language Processing*, 14:1513–1525.

Gao, T., Du, J., Dai, L.-R., and Lee, C.-H. (2015). Joint training of front-end and back-end deep neural networks for robust speech recognition. In *Proc. ICASSP*, Brisbane, Australia.

Gardiner, C. (1985). *Handbook of Stochastic Methods*. Springer Berlin, Berlin, Germany.

Gauvain, J.-L. and Lee, C.-H. (1994). Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains. *IEEE Transactions on Speech and Audio Processing*, 2:291–298.

Gemello, R., Mana, F., Scanzio, S., Laface, P., and De Mori, R. (2007). Linear hidden transformations for adaptation of hybrid ANN/HMM models. *Speech Communication*, 49:827–835.

Gemmeke, J., Virtanen, T., and Hurmalainen, A. (2011). Exemplar-based sparse representations for noise robust automatic speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 19:2067–2080.

Ghahremani, P., BabaAli, B., Povey, D., Riedhammer, K., Trmal, J., and Khudanpur, S. (2014). A pitch extraction algorithm tuned for automatic speech recognition. In *Proc. ICASSP*, Florence, Italy.

Ghoshal, A., Swietojanski, P., and Renals, S. (2013). Multilingual training of deep neural networks. In *Proc. ICASSP*, Vancouver, Canada.

Gibson, M. and Hain, T. (2006). Hypothesis spaces for minimum Bayes risk training in large vocabulary speech recognition. In *Proc. Interspeech*, Pittsburgh, PA, USA.

Gill, P., Murray, W., and Wright, M. (1981). *Practical Optimization*. Academic Press, Cambridge, MA, USA.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proc. AISTATS*, Sardinia, Italy.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier networks. In *Proc. AISTATS*, Ft. Lauderdale, FL, USA.

Goh, S. and Mandic, D. (2003). Recurrent neural networks with trainable amplitude of activation functions. *Neural Networks*, 16:1095–1100.

Golik, P., Doetsch, P., and Ney, H. (2013). Cross-entropy vs. squared error training: A theoretical and experimental comparison. In *Proc. Interspeech*, Lyon, France.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *Proc. ICML*, Atlanta, GA, USA.

Gopalakrishnan, P., Kanevsky, D. Nádas, A., and Nahamoo, D. (1991). An inequality for rational functions with applications to some statistical estimation problems. *IEEE Transactions on Information Theory*, 37:814–817.

Gosztolya, G., Grósz, T., and Tóth, L. (2016). GMM-free flat start sequence-discriminative DNN training. *arXiv 1610.03256*.

Gosztolya, G., Grósz, T., Tóth, L., and Imseng, D. (2015). Building context-dependent DNN acoustic models using Kullback-Leibler divergence-based state tying. In *Proc. ICASSP*, Brisbane, Australia.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proc. ICML*, Pittsburgh, PA, USA.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *Proc. ICML*, Beijing, China.

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18:602–610.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural Turing machines. *arXiv 1410.5401*.

Grézl, F. and Fousek, P. (2008). Optimizing bottle-neck features for LVCSR. In *Proc. ICASSP*, Las Vegas, NV, USA.

Grézl, F., Karafiát, M., Kontár, S., and Černocký, J. (2007). Probabilistic and bottle-neck features for LVCSR of meetings. In *Proc. ICASSP*, Honolulu, HI, USA.

Hahnloser, R. (1998). On the piecewise analysis of networks of linear threshold neurons. *Neural Networks*, 11:691–697.

Hain, T., Woodland, P., Niesler, T., and Whittaker, E. (1999). The 1998 HTK system for transcription of conversational telephone speech. In *Proc. ICASSP*, Phoenix, AZ, USA.

Ham, F. and Kostanic, I. (2000). *Principles of Neurocomputing for Science and Engineering*. McGraw-Hill Higher Education, New York, NY, USA.

Han, J. and Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. In *Proc. IWANN*, Malaga-Torremolinos, Spain.

Harper, M. (2011). IARPA Babel program. http://www.iarpa.gov/Programs/ia/Babel/babel.html.

Haton, J.-P. (1985). Knowledge-based and expert systems in automatic speech recognition. In De Mori, R. and Suen, C. Y., editors, *New Systems and Architectures for Automatic Speech Recognition and Synthesis*, pages 249–269. Springer Berlin, Berlin, Germany.

He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Deep residual learning for image recognition. *arXiv 1512.03385*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv 1502.01852*.

Heigold, G. (2010). *A Log-Linear Discriminative Modeling Framework for Speech Recognition*. PhD thesis, RWTH Aachen University.

Heigold, G., Deselaers, T., Schlüter, R., and Ney, H. (2008). Modified MMI/MPE: A direct evaluation of the margin in speech recognition. In *Proc. ICML*, Helsinki, Finland.

Heigold, G., Vanhoucke, V., Senior, A., Nguyen, P., Ranzato, M., Devin, M., and Dean, J. (2013). Multilingual acoustic models using distributed deep neural networks. In *Proc. ICASSP'13*, Vancouver, Canada.

Hermansky, H. (1990). Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, 87:1738–1752.

Hermansky, H., Ellis, D., and Sharma, S. (2000). Tandem connectionist feature extraction for conventional HMM systems. In *Proc. ICASSP*, Istanbul, Turkey.

Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. Technical report, UTML TR 2010-003, Department of Computer Science, University of Toronto.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kinsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, pages 2–17.

Hinton, G., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554.

Hinton, G. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313:504–507.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780.

Hopfield, J. (1987). Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proceedings of the National Academy of Sciences*, 84:8429–8433.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Huang, J.-T., Li, J., Yu, D., Deng, L., and Gong, Y. (2013). Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *ICASSP*, Vancouver, Canada.

Huang, X. (1989). *Semi-Continuous Hidden Markov Models for Speech Recognition*. PhD thesis, University of Edinburgh.

Huang, X., Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice Hall, Upper Saddle River, NJ, USA.

Huang, Z., Siniscalchi, S., Chen, I.-F., Li, J., Wu, J., and Lee, C.-H. (2015). Maximum a posteriori adaptation of network parameters in deep models. In *Proc. ICASSP*, Dresden, Germany.

Hwang, M.-Y., Huang, X., and Alleva, F. (1996). Predicting unseen triphones with senones. *IEEE Transactions on Speech and Audio Processing*, 4:412–419.

Imseng, D., Bourlard, H., Dines, J., Garner, P., and Magimai.-Doss, M. (2013). Applying multi- and cross-lingual stochastic phone space transformations to non-native speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 21:1713–1726.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv 1502.03167*.

Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3:79–87.

Jelinek, F. (1991). Up from trigrams! The struggle for improved language models. In *Proc. ECSCT*, Genoa, Italy.

Jelinek, F. (1998). *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, MA, USA.

Jernite, Y., Choromanska, A., Sontag, D., and LeCun, Y. (2016). Simultaneous learning of trees and representations for extreme classification, with application to language modeling. *arXiv 1610.04658*.

Johnson, D. (2012). Quicknet. http://www1.icsi.berkeley.edu/speech/qn.html.

Juang, B.-H. (1985). Maximum-likelihood estimation for mixture multivariate stochastic observations of Markov chains. *AT&T Technical Journal*, 64:1235–1249.

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

Juang, B.-H. and Katagiri, S. (1992). Discriminative learning for minimum error classification. *IEEE Transactions on Signal Processing*, 40:3043–3054.

Kaiser, J., Horvat, B., and Kačič, Z. (2002). Overall risk criterion estimation of hidden Markov model parameters. *Speech Communication*, 38:383–398.

Kanthak, S. and Ney, H. (2002). Context-dependent acoustic modelling using graphemes for large-vocabulary speech recognition. In *Proc. ICASSP*, Orlando, FL, USA.

Karafiát, M., Veselý, J., Szokem, I., and Černocký, J. (2014). BUT 2014 Babel system: Analysis of adaptation in NN based systems. In *Proc. Interspeech*, Singapore.

Karanasou, P., Gales, M., Lanchantin, P., Liu, X., Qian, Y., Wang, L., Woodland, P., and Zhang, C. (2015). Speaker diarisation and longitudinal linking in multi-genre broadcast data. In *Proc. ASRU*, Scottsdale, AZ, USA.

Karanasou, P., Wang, Y., Gales, M., and Woodland, P. (2014). Adaptation of deep neural network acoustic models using factorised i-vectors. In *Proc. Interspeech*, Singapore.

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35:400–401.

Kim, S., Hori, T., and Watanabe, S. (2016). Joint CTC-attention based end-to-end speech recognition using multi-task learning. *arXiv 1609.06773*.

King, S., Stephenson, T., Isard, S., Taylor, P., and Strachan, A. (1998). Speech recognition via phonetically featured syllables. In *Proc. ICSLP*, Sydney, Australia.

Kingsbury, B. (2009). Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *Proc. ICASSP*, Taipei, Taiwan.

Kingsbury, B., Sainath, T., and Soltau, H. (2012). Scalable minimum Bayes risk training of deep neural network acoustic models using distributed Hessian-free optimization. In *Proc. Interspeech*, Portland, OR, USA.

Kirchhoff, K. (1999). *Robust Speech Recognition using Articulatory Information*. PhD thesis, University of Bielefeld.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Proc. ICASSP*, Detroit, MI, USA.

Knill, K., Gales, M., Rath, S., Woodland, P., Zhang, C., and Zhang, S.-X. (2013). Investigation of multilingual deep neural networks for spoken term detection. In *Proc. ASRU*, Olomouc, Czech Republic.

Kumar, N. (1997). *Investigation of Silicon-Auditory Models and Generalization of Linear Discriminant Analysis for Improved Speech Recognition*. PhD thesis, John Hopkins University.

Lamel, L., Gauvain, J.-L., and Adda, G. (2002). Lightly supervised and unsupervised acoustic model training. *Computer Speech and Language*, 16:115–129.

Lanchantin, P., Gales, M., Karanasou, P., Liu, X., Qian, Y., Wang, L., Woodland, P., and Zhang, C. (2015). The development of the Cambridge university alignment systems for the multi-genre broadcast challenge. In *Proc. ASRU*, Scottsdale, AZ, USA.

Lanchantin, P., Gales, M., Karanasou, P., Liu, X., Qian, Y., Wang, L., Woodland, P., and Zhang, C. (2016). Selection of multi-genre broadcast data for the training of automatic speech recognition systems. In *Proc. Interspeech*, San Francisco, CA, USA.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324.

LeCun, Y., Bottou, L., Orr, G., and Müller, K.-R. (1998b). Efficient backprop. In G.B. and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer-Verlag Berlin Heidelberg, Berlin, Germany.

Lee, C.-H. (2004). From knowledge-ignorant to knowledge-rich modeling: A new speech research paradigm for next generation automatic speech recognition. In *Proc. Interspeech*, Jeju Island, Korea.

Lee, L. and Rose, R. (1996). Speaker normalization using efficient frequency warping procedures. In *Proc. ICASSP*, Atlanta, GA, USA.

Leggetter, C. and Woodland, P. (1995). Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech and Language*, 9:171–185.

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710.

Levinson, S. (1986). Continuously variable duration hidden Markov models for automatic speech recognition. *Computer Speech and Language*, 1:29–45.

Li, B. and Sim, K. (2010). Comparison of discriminative input and output transformations for speaker adaptation in the hybrid NN/HMM systems. In *Proc. Interspeech*, Makuhari, Japan.

Li, J., Yuan, M., and Lee, C.-H. (2006). Soft margin estimation of hidden Markov model parameters. In *Proc. Interspeech*, Pittsburgh, PA, USA.

Li, X. and Wu, X. (2014a). Decision tree based state tying for speech recognition using DNN derived embeddings. In *Proc. ISCSLP*, Singapore.

Li, X. and Wu, X. (2014b). Labeling unsegmented sequence data with DNN-HMM and its application for speech recognition. In *Proc. ISCSLP*, Singapore.

Liao, H. (2013). Speaker adaptation of context dependent deep neural networks. In *Proc. ICASSP*, Vancouver, Canada.

Liu, X., Chen, X., Wang, Y., Gales, M., and Woodland, P. (2016). Two efficient lattice rescoring methods using recurrent neural network language models. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24:1438–1449.

Liu, X., Flego, F., Wang, L., Zhang, C., Gales, M., and Woodland, P. (2015). The Cambridge University 2014 BOLT conversational telephone Mandarin Chinese LVCSR system for speech translation. In *Proc. Interspeech*, Dresden, Germany.

Liu, X., Gales, M., Hieronymus, J., and Woodland, P. (2010). Language model combination and adaptation using weighted finite state transducers. In *Proc. ICASSP*, Dallas, TX, USA.

Liu, X., Gales, M., Hieronymus, J., and Woodland, P. (2011). Investigation of acoustic units for LVCSR systems. In *Proc. ICASSP*, Prague, Czech Republic.

Liu, X., Gales, M., and Woodland, P. (2003). Automatic complexity control for HLDA systems. In *Proc. ICASSP*, Hong Kong.

Liu, X., Gales, M., and Woodland, P. (2013). Use of contexts in language model interpolation and adaptation. *Computer Speech and Language*, 27:301–321.

Ljolje, A., Pereira, F., and Riley, M. (1999). Efficient general lattice generation and rescoring. In *Proc. Eurospeech*, Budapest, Hungary.

Lowerre, B. (1976). *The HARPY Speech Recognition System*. PhD thesis, Carnegie Mellon University.

Lu, L., Zhang, X., Cho, K., and Renals, S. (2015). A study of the recurrent neural network encoder-decoder for large vocabulary speech recognition. In *Proc. Interspeech*, Dresden, Germany.

Maas, A., Hanun, A., and Ng, A. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, Atlanta, GA, USA.

MacKay, D. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, UK.

Mangu, L., Brill, E., and Stolcke, A. (2000). Finding consensus in speech recognition: word error minimization and other applications of confusion networks. *Computer Speech and Language*, 14:373–400.

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Martens, J. (2010). Deep learning via Hessian-free optimization. In *Proc. ICML*, Haifa, Israel.

McDermott, E., Heigold, G., Moreno, P., Senior, A., and Bacchiani, M. (2014). Asynchronous stochastic optimization for sequence training of deep neural networks: Towards big data. In *Proc. Interspeech*, Singapore.

Miao, Y., Gowayyed, M., and Metze, F. (2015). EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In *Proc. ASRU*, Scottsdale, AZ, USA.

Miao, Y., Zhang, H., and Metze, F. (2014). Towards speaker adaptive training of deep neural network acoustic models. In *Proc. Interspeech*, Singapore.

Mikolov, T. (2012). *Statistical Language Models based on Neural Networks*. PhD thesis, Burno University of Technology.

Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge, MA, USA.

Mohamed, A., Hinton, G., and Penn, G. (2012). Understanding how deep belief networks perform acoustic modelling. In *Proc. ICASSP*, Kyoto, Japan.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16:69–88.

Morgan, N. (2012). Deep and wide: Multiple layers in automatic speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20:7–13.

Myers, C., Rabiner, L., and Rosenberg, A. (2003). Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28:623–635.

Nádas, A. (1983). A decision theoretic formulation of a training problem in speech recognition and a comparison of training by unconditional versus conditional maximum likelihood. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31:814–817.

Nair, V. and Hinton, G. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proc. ICML*, Haifa, Israel.

Neal, R. (2000). Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9:249–265.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $(1/\mathrm{sqr}(k))$. *Soviet Mathematics Doklady*, 27:372–376.

Neto, J., Almeida, L., Hochberg, M., Martins, C., Nunes, L., Renals, S., and Robinson, T. (1995). Speaker-adaptation for hybrid HMM-ANN continuous speech recognition system. In *Proc. Eurospeech*, Madrid, Spain.

Ney, H. and Aubert, X. (1994). A word graph algorithm for large vocabulary. In *Proc. ICSLP*, Yokohama, Japan.

Niles, L. and Silverman, H. (1990). Combining hidden Markov model and neural network classifiers. In *Proc. ICASSP*, Albuquerque, NM, USA.

Normandin, Y. (1991). *Hidden Markov Models, Maximum Mutual Information Estimation, and the Speech Recognition Problem.* PhD thesis, McGill University.

Ochiai, T., Matsuda, S., Lu, X., Hori, C., and Katagiri, S. (2014). Speaker adaptive training using deep neural networks. In *Proc. Eurospeech*, Florence, Italy.

Odell, J. (1995). *The Use of Context in Large Vocabulary Speech Recognition.* PhD thesis, University of Cambridge.

Odell, J. (1999). Network and language models for use in a speech recognition system. US Patent 6668243 B1.

Odell, J., Valtchev, V., Woodland, P., and Young, S. (1994). A one pass decoder design for large vocabulary recognition. In *Proc. HLT*, Plainsboro, NJ, USA.

Oppenheim, A. and Schafer, R. W. (1975). *Discrete-Time Signal Processing*. Prentice Hall, Upper Saddle River, NJ, USA.

Park, J., Diehl, F., Gales, M., Tomalin, M., and Woodland, P. (2011). The efficient incorporation of MLP features into automatic speech recognition systems. *Computer Speech and Language*, 25:519–634.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *arXiv 1211.5063*.

Paul, D. (1997). Extensions to phone-state decision-tree clustering: single tree and tagged clustering. In *Proc. ICASSP*, Munich, Germany.

Paul, D. and Baker, J. (1992). The design for the Wall Street Journal based CSR corpus. In *Proc. DARPA SLS Workshop*, Pacific Grove, CA, USA.

Paulik, M. (2013). Lattice-based training of bottleneck feature extraction neural networks. In *Proc. Interspeech*, Lyon, France.

Pearce, D. (2002). Aurora working group: DSR front end LVCSR evaluation. Technical report, AU/384/02, Aurora working group.

Peddinti, V., Povey, D., and Khudanpur, S. (2015). A time delay neural network architecture for efficient modeling of long temporal contexts. In *Proc. Interspeech*, Dresden, Germany.

Pineda, F. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59:2229–2232.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17.

Povey, D. (2003). *Discriminative Training for Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge.

Povey, D., Burget, L., Agarwal, M., Akyazi, P., Feng, K., Ghoshal, A., Glembek, O., Goel, N., Karafiát, M., Rastrow, A., Rose, R., Schwarz, P., and Thomas, S. (2010). Subspace Gaussian mixture models for speech recognition. In *Proc. ICASSP*, Dallas, TX, USA.

Povey, D., Gales, M., Kim, D., and Woodland, P. (2003a). MMI-MAP and MPE-MAP for acoustic model adaptation. In *Proc. Interspeech*, Geneva, Switzerland.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlíček, P., Qian, Y., Schwarz, P., Silovský, J., Stemmer, G., and Veselý, K. (2011). The Kaldi speech recognition toolkit. In *Proc. ASRU*, Waikoloa, HI, USA.

Povey, D., Hannemann, M., Boulianne, G., Burget, L., Ghoshal, A., Janda, M., Karafiát, M., Kombrink, S., Motliček, P., Qian, Y., Riedhammer, K., Veselý, K., and Thang Vu, N. (2012). Generating exact lattices in the WFST framework. In *Proc. ICASSP*, Kyoto, Japan.

Povey, D., Kanevsky, D., Kingsbury, B., Ramabhadran, B., Saon, G., and Visweswariah, K. (2008). Boosted MMI for model and feature-space discriminative training. In *Proc. ICASSP*, Las Vegas, NV, USA.

Povey, D. and Kingsbury, B. (2007). Evaluation of proposed modifications to MPE for large scale discriminative training. In *Proc. ICASSP*, Honolulu, HI, USA.

Povey, D., Kingsbury, B., Mangu, L., Saon, G., Soltau, H., and Zweig, G. (2005). fMPE: discriminatively trained features for speech recognition. In *Proc. ICASSP*, Philadelphia, PA, USA.

Povey, D., Vijayaditya, P., Galves, D., Ghahremani, P., Manohar, V., Na, X., Wang, Y., and Khudanpur, S. (2016). Purely sequence-trained neural networks for ASR based on lattice-free MMI. In *Proc. Interspeech*, San Francisco, CA, USA.

Povey, D. and Woodland, P. (1999). An investigation of frame discrimination for continuous speech recognition. Technical report, CUED/F-INFENG/TR332, University of Cambridge.

Povey, D. and Woodland, P. (2002). Minimum phone error and I-smoothing for improved discriminative training. In *Proc. ICASSP*, Orlando, FL, USA.

Povey, D., Woodland, P., and Gales, M. (2003b). Discriminative MAP for acoustic model adaptation. In *Proc. ICASSP*, Hong Kong.

Povey, D., Zhang, X., and Khudanpur, S. (2015). Parallel training of DNNs with natural gradient and parameter averaging. In *Proc. ICLR*, San Diego, CA, USA.

Pundak, G. and Sainath, T. (2016). Lower frame rate neural network acoustic models. In *Proc. Interspeech*, San Francisco, CA, USA.

Pye, D. and Woodland, P. (1997). Experiments in speaker normalisation and adaptation for large vocabulary speech recognition. In *Proc. ICASSP*, Munich, Germany.

Qian, Y., Tan, T., Yu, D., and Zhang, Y. (2016). Integrated adaptation with multi-factor joint-learning for far-field speech recognition. In *Proc. ICASSP*, Shanghai, China.

Qian, Y. and Woodland, P. (2016). Very deep convolutional neural networks for robust speech recognition. In *Proc. SLT*, San Diego, CA, USA.

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286.

Rabiner, L. and Juang, B.-H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall, Upper Saddle River, NJ, USA.

Reichl, W. and Chou, W. (2000). Robust decision tree state tying for continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 8:555–566.

Renals, S., Morgan, N., Cohen, M., and Franco, H. (1992). Connectionist probability estimation in the DECIPHER speech recognition system. In *Proc. ICASSP*, San Francisco, CA, USA.

Richmond, K. (2002). *Estimating Articulatory Parameters from the Acoustic Speech Signal*. PhD thesis, University of Edinburgh.

Richmond, K. (2006). A trajectory mixture density network for the acoustic-articulatory inversion mapping. In *Proc. Interspeech*, Pittsburgh, PA, USA.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster back-propagation learning: The RPROP algorithm. In *Proc. ICNN*, San Francisco, CA, USA.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22:400–407.

Robbins, H. and Siegmund, D. (1971). A convergence theorem for non negative almost supermartingales and some applications. In Rustagi, J., editor, *Optimizing Methods in Statistics*, pages 233–257. Academic Press, New York, NY, USA.

Robinson, A. (1989). *Dynamic Error Propagation Networks*. PhD thesis, University of Cambridge.

Robinson, A. and Fallside, F. (1987). The utility driven dynamic error propagation network. Technical report, CUED/F-INFENG/TR1, University of Cambridge.

Rosenblatt, F. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, DC, USA.

Rumelhart, D., McClelland, J., and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volumn 1: Foundations*. MIT Press, Cambridge, MA, USA.

Russell, M. and Moore, R. (1985). Explicit modeling of state occupancy in hidden Markov models for automatic speech recognition. In *Proc. DARPA Speech Recognition Workshop*, Tampa, FL, USA.

Sainath, T., Kingsbury, B., Soltau, H., and Ramabhadran, B. (2013a). Optimization techniques to improve training speed of deep neural networks for large speech tasks. *IEEE Transactions on Audio, Speech, and Language Processing*, 21:2267–2276.

Sainath, T., Mohamed, A., Kingsbury, B., and Ramabhadran, B. (2013b). Deep convolutional neural networks for LVCSR. In *Proc. ICASSP*, Vancouver, Canada.

Sainath, T., Vinyals, O., Senior, A., and Sak, H. (2015a). Convolutional, long short-term memory, fully connected deep neural networks. In *Proc. ICASSP*, Brisbane, Australia.

Sainath, T., Weiss, R., Senior, A., Wilson, K., and Vinyals, O. (2015b). Learning the speech front-end with raw waveform CLDNNs. In *Proc. Interspeech*, Dresden, Germany.

Salinas, E. and Abbott, L. (1996). A model of multiplicative neural responses in parietal cortex. *Neurobiology*, 93:11956–11961.

Saon, G., Sercu, T., Rennie, S., and Kuo, H.-K. (2016). The IBM 2016 English conversational telephone speech recognition system. *arXiv 1604.08242*.

Saon, G., Soltau, H., Emami, A., and Picheny, M. (2014). Unfolded recurrent neural networks for speech recognition. In *Proc. Interspeech*, Singapore.

Saon, G., Soltau, H., Nahamoo, D., and Picheny, M. (2013). Speaker adaptation of neural network acoustic models using i-vectors. In *Proc. ASRU*, Olomouc, Czech Republic.

Schlüter, R., Macherey, W., Müller, B., and Ney, H. (2001). Comparison of discriminative training criteria and optimization methods for speech recognition. *Speech Communication*, 34:287–310.

Schultz, T. and Waibel, A. (2001). Language-independent and language adaptive acoustic modeling for speech recognition. *Speech Communication*, 35:31–35.

Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681.

Schwartz, R., Chow, Y., Kimball, O., Roucos, S., Krasner, M., and Makhoul, J. (1985). Context-dependent modeling for acoustic-phonetic recognition of continuous speech. In *Proc. ICASSP*, Tampa, FL, USA.

Scrivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. (2014). 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *Proc. Interspeech*, Singapore.

Seide, F., Li, G., Chen, X., and Yu, D. (2011a). Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Proc. ASRU*, Waikoloa, HI, USA.

Seide, F., Li, G., and Yu, D. (2011b). Conversational speech transcription using context-dependent deep neural networks. In *Proc. Interspeech*, Florence, Italy.

Selkirk, E. (1986). *Phonology and Syntax: The Relationship between Sound and Structure*. MIT Press, Cambridge, MA, USA.

Senior, A. (1994). *Off-line Cursive Handwriting Recognition using Recurrent Neural Networks*. PhD thesis, University of Cambridge.

Senior, A., Heigold, G., Bacchiani, M., and Liao, H. (2014). GMM-free DNN training. In *Proc. ICASSP*, Florence, Italy.

Senior, A., Heigold, G., Ranzato, M., and Yang, K. (2013). An empirical study of learning rates in deep neural networks for speech recognition. In *Proc. ICASSP*, Vancouver, Canada.

Senior, A. and Lopez-Moreno, I. (2014). Improving DNN speaker independence with i-vector inputs. In *Proc. ICASSP*, Florence, Italy.

Sercu, T., Puhrsch, C., Kingsbury, B., and LeCun, Y. (2016). Very deep multilingual convolutional neural networks for LVCSR. In *Proc. ICASSP*, Shanghai, China.

Sha, F. and Saul, L. (2006). Large margin Gaussian mixture modeling for phonetic classification and recognition. In *Proc. ICASSP*, Toulouse, France.

Sha, F. and Saul, L. (2008). Generalization of extended Baum-Welch parameter estimation for discriminative training and decoding. In *Proc. Interspeech*, Brisbane, Australia.

Shannon, C. and Weaver, W. (1949). *The Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL, USA.

Shinoda, K. and Watanabe, T. (2000). MDL-based context-dependent subword modeling for speech recognition. *Journal of Acoustic Society of Japan*, 21:79–86.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional neural networks for large-scale image recognition. In *Proc. ICLR*, San Diego, CA, USA.

Siniscalchi, S., Li, J., and Lee, C.-H. (2013). Hermitian polynomial for speaker adaptation of connectionist speech recognition systems. *IEEE Transactions on Audio, Speech, and Language Processing*, 21:2152–2161.

Siniscalchi, S., Svendsen, T., Sorbello, S., and Lee, C.-H. (2010). Experimental studies on continuous speech recognition using neural architectures with "adaptive" hidden activation functions. In *Proc. ICASSP*, Dallas, TX, USA.

Sivadas, S., Wu, Z., and Ma, B. (2015). Investigation of parametric rectified linear units for noise robust speech recognition. In *Proc. Interspeech*, Dresden, Germany.

Ström, N. (1996). Speaker adaptation by modeling the speaker variation in a continuous speech recognition system. In *Proc. ICSLP*, Philadelphia, PA, USA.

Su, H., Li, G., Yu, D., and Seide, F. (2013). Error back propagation for sequence training of context-dependent deep neural networks for conversational speech transcription. In *Proc. ICASSP*, Vancouver, Canada.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialisation and momentum in deep learning. In *Proc. ICML*, Atlanta, GA, USA.

Swietojanski, P., Bell, P., and Renals, S. (2015). Structured output layer with auxiliary targets for context-dependent acoustic modelling. In *Proc. Interspeech*, Dresden, Germany.

Swietojanski, P., Ghoshal, A., and Renals, S. (2012). Unsupervised cross-lingual knowledge transfer in DNN-based LVCSR. In *Proc. IWSLT*, Hong Kong.

Swietojanski, P., Li, J., and Renals, S. (2016). Learning hidden unit contributions for unsupervised acoustic model adaptation. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 24:1450–1463.

Swietojanski, P. and Renals, S. (2014). Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *Proc. SLT*, Lake Tahoe, CA, USA.

Tan, T., Qian, Y., Yin, M., Zhuang, Y., and Yu, K. (2015). Cluster adaptive training for deep neural network. In *Proc. ICASSP*, Brisbane, Australia.

Tóth, L. (2013). Phone recognition with deep sparse rectifier neural networks. In *Proc. ICASSP*, Vancouver, Canada.

Tóth, L. (2014). Combining time- and frequency-domain convolution in convolutional neural network-based phone recognition. In *Proc. ICASSP*, Florence, Italy.

Trentin, E. (2001). Networks with trainable amplitude of activation functions. *Neural Networks*, 14:471–493.

Trentin, E. and Gori, M. (2003). Robust combination of neural networks and hidden Markov models for speech recognition. *IEEE Transactions on Neural Networks*, 14:1519–1531.

Tüske, Z., Golik, P., Schlüter, R., and Ney, H. (2014). Acoustic modeling with deep neural networks using raw time signal for LVCSR. In *Proc. Interspeech*, Singapore.

Tüske, Z., Golik, P., Schlüter, R., and Ney, H. (2015a). Integrating Gausssian mixtures into deep neural networks: Softmax layer with hidden variables. In *Proc. ICASSP*, Brisbane, Australia.

Tüske, Z., Golik, P., Schlüter, R., and Ney, H. (2015b). Speaker adaptive joint training of Gaussian mixture models and bottleneck features. In *Proc. ASRU*, Scottsdale, AZ, USA.

Tüske, Z., Sundermeyer, M., Schlüter, R., and Ney, H. (2012). Tandem system adaptation using multiple linear feature transforms. In *Proc. ICASSP*, Portland, OR, USA.

Valtchev, V. (1995). *Discriminative Methods in HMM-based Speech Recognition*. PhD thesis, University of Cambridge.

van Dalen, R., Yang, J., Wang, H., Ragni, A., Zhang, C., and Gales, M. (2015). Structured discriminative models using deep neural-network features. In *Proc. ASRU*, Scottsdale, AZ, USA.

Vapnik, V. (1998a). *Statistical Learning Theory*. John Wiley, New York, NY, USA.

Vapnik, V. (1998b). The support vector method of function estimation. In Suykens, J. and Vandewalle, J., editors, *Nonlinear Modeling: Advanced Black-Box Techniques*, pages 55–85. Kluwer Academic Publishers, Boston, MA, USA.

Variani, E., McDermott, E., and Heigold, G. (2015). A Gaussian mixture model layer jointly optimized with discriminative features within a deep neural network architecture. In *Proc. ICASSP*, Brisbane, Australia.

Veselý, K., Ghoshal, A., Burget, L., and Povey, D. (2013). Sequence-discriminative training of deep neural networks. In *Proc. Interspeech*, Lyon, France.

Veselý, K., Karafiát, M., and Grézl, F. (2011). Convolutive bottleneck network features for LVCSR. In *Proc. ASRU*, Waikoloa, HI, USA.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269.

Wachter, M. D., Matton, M., Demuynck, K., Wambacq, P., Cools, R., and Compernolle, D. V. (2007). Template based continuous speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 15:1377–1390.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37:328–339.

Wang, H., Ragni, A., Gales, M., Knill, K., Woodland, P., and Zhang, C. (2015). Joint decoding of tandem and hybrid systems for improved keyword spotting on low resource languages. In *Proc. Interspeech*, Dresden, Germany.

Wang, L. and Woodland, P. (2008). MPE-based discriminative linear transforms for speaker adaptation. *Computer Speech and Language*, 22:256–272.

Wang, L., Zhang, C., Woodland, P., Gales, M., Karanasou, P., Lanchantin, P., Liu, X., and Qian, Y. (2016). Improved DNN-based segmentation for multi-genre broadcast audio. In *Proc. ICASSP*, Shanghai, China.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences*. PhD thesis, Harvard University.

Wiesler, S., Golik, P., Schlüter, R., and Ney, H. (2015). Investigations on sequence training of neural networks. In *Proc. ICASSP*, Brisbane, Australia.

Wiesler, S. and Ney, H. (2011). A convergence analysis of log-linear training. *Advances in Neural Information Processing Systems*, 24:657–665.

Wiesler, S., Richard, A., Golik, P., Schlüter, R., and Ney, H. (2014a). RASR/NN: The RWTH neural network toolkit for speech recognition. In *Proc. ICASSP*, Florence, Italy.

Wiesler, S., Richard, A., Schlüter, R., and Ney, H. (2014b). Mean-normalized stochastic gradient for large-scale deep learning. In *Proc. ICASSP*, Florence, Italy.

Woodland, P. (1989). Weight limiting, weight quantisation and generalisation in multi-layer perceptrons. In *Proc. ICANN*, London, UK.

Woodland, P. (1992). Spoken alphabet recognition using multilayer perceptrons. In Linggard, R., Myers, D., and Nightingale, C., editors, *Neural Networks for Vision, Speech and Natural Language*, pages 135–147. Springer Netherlands, Dordrecht, Netherlands.

Woodland, P. (2001). Speaker adaptation for continuous density HMMs: A review. In *ISCA ITR-Workshop*, Sophia Antipolis, France.

Woodland, P. (2002). The development of the HTK broadcast news transcription system: An overview. *Speech Communication*, 37:47–67.

Woodland, P., Gales, M., Pye, D., and Young, S. (1997). Broadcast news transcription using HTK. In *Proc. ICASSP*, Munich, Germany.

Woodland, P., Leggetter, C., Odell, J., Valtchev, V., and Young, S. (1995). The 1994 HTK large vocabulary speech recognition system. In *Proc. ICASSP*, Detroit, MI, USA.

Woodland, P., Liu, X., Qian, Y., Zhang, C., Gales, M., Karanasou, P., Lanchantin, P., and Wang, L. (2015). Cambridge University transcription systems for the multi-genre broadcast challenge. In *Proc. ASRU*, Scottsdale, AZ, USA.

Woodland, P., Odell, J., Valtchev, V., and Young, S. (1994). Large vocabulary continuous speech recognition using HTK. In *Proc. ICASSP*, Adelaide, Australia.

Woodland, P. and Povey, D. (2002). Large scale discriminative training of hidden Markov models for speech recognition. *Computer Speech and Language*, 16:25–47.

Wu, C. and Gales, M. (2015). Multi-basis adaptive neural network for rapid adaptation in speech recognition. In *Proc. ICASSP*, Brisbane, Australia.

Wu, C., Karanasou, P., Gales, M., and Sim, K. (2016a). Stimulated deep neural network for speech recognition. In *Proc. Interspeech*, San Francisco, CA, USA.

Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. (2016b). On multiplicative integration with recurrent neural networks. *arXiv 1606.06630*.

Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. (2016). The Microsoft 2016 conversational speech recognition system. *arXiv 1609.03528*.

Xu, W. (2011). Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv 1107.2490*.

Yan, Y., Fanty, M., and Cole, R. (1997). Speech recognition using neural networks with forward-backward probability generated targets. In *Proc. ICASSP*, Munich, Germany.

Yan, Z., Huo, Q., and Xu, J. (2013). A scalable approach to using DNN-derived features in GMM-HMM based acoustic modeling for LVCSR. In *Proc. Interspeech*, Lyon, France.

Yang, J., Zhang, C., Ragni, A., Gales, M., and Woodland, P. (2016). System combination with log-linear models. In *Proc. ICASSP*, Shanghai, China.

Yao, K., Yu, D., Seide, F., Su, H., Deng, L., and Gong, Y. (2012). Adaptation of context-dependent deep nerual networks for automatic speech recognition. In *Proc. IWSLT*, Hong Kong.

Yin, M., Sivadas, S., Yu, K., and Ma, B. (2016). Discriminatively trained joint speaker and environment representations for adaptation of deep neural network acoustic models. In *Proc. ICASSP*, Shanghai, China.

Yoshimura, T., Tokuda, K., Masuko, T., Kobayashi, T., and Kitamura, T. (1998). Duration modeling for HMM-based speech synthesis. In *Proc. ICSLP*, Sydney, Australia.

Yoshioka, T., Ohnishi, K., Fang, F., and Nakatini, T. (2016). Noise robust speech recognition using recent developments in neural networks for computer vision. In *Proc. ICASSP*, Shanghai, China.

Young, S. (1990). Competitive training in hidden Markov models. In *Proc. ICASSP*, Albuquerque, NM, USA.

Young, S. (1996). Large vocabulary continuous speech recognition: A review. *IEEE Signal Processing Magazine*, 13:45–57.

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., Ragni, A., Valtchev, V., Woodland, P., and Zhang, C. (2015). *The HTK Book (for HTK version 3.5)*. Cambridge University Engineering Department, Cambridge, UK.

Young, S., Odell, J., and Woodland, P. (1994). Tree-based state tying for high accuracy acoustic modelling. In *Proc. HLT*, Plainsboro, NJ, USA.

Young, S., Russel, N., and Thornton, J. (1989). Token passing: A simple conceptual model for connected speech recognition systems. Technical report, CUED/F-INFENG/TR38, University of Cambridge.

Young, S. and Woodland, P. (1993). The use of state tying in continuous speech recognition. In *Proc. Eurospeech*, Berlin, Germany.

Yu, D., Deng, L., He, X., and Acero, A. (2006). Use of incrementally regulated discriminative margins in MCE training for speech recognition. In *Proc. Interspeech*, Pittsburgh, PA, USA.

Yu, D., Eversole, A., Seltzer, M., Yao, K., Huang, Z., Guenter, B., Kuchaiev, O., Zhang, Y., Seide, F., Wang, H., Droppo, J., Zweig, G., Rossbach, C., Currey, J., Gao, J., May, A., Peng, B., Stolcke, A., and Slaney, M. (2014). An introduction to computational networks and the computational network toolkit. Technical report, MSR-TR-2014-112, Microsoft Corporation.

Yu, D., Yao, K., Su, H., Li, G., and Seide, F. (2013). KL-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In *Proc. ICASSP*, Vancouver, Canada.

Yu, K., Gales, M., and Woodland, P. (2009). Unsupervised adaptation with discriminative mapping transforms. *IEEE Transactions on Audio, Speech, and Language Processing*, 17:714–723.

Zeiler, M., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q., Nguyen, P., Senior, A., Vanhoucke, V., Dean, J., and Hinton, G. (2013). On rectified linear units for speech processing. In *Proc. ICASSP*, Vancouver, Canada.

Zen, H. and Gales, M. (2011). Decision tree-based context clustering based on cross validation and hierarchical priors. In *Proc. ICASSP*, Prague, Czech Republic.

Zen, H., Nose, T., Yamagishi, J., Sako, S., Masuko, T., Black, A., and Tokuda, K. (2007). The HMM-based speech synthesis system (HTS) version 2.0. In *Proc. 6th ISCA Workshop on Speech Synthesis*, Bonn, Germany.

Zen, H. and Senior, A. (2014). Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *Proc. ICASSP*, Florence, Italy.

Zhang, C. and Woodland, P. (2014). Standalone training of context-dependent deep neural network acoustic models. In *Proc. ICASSP*, Florence, Italy.

Zhang, C. and Woodland, P. (2015a). A general artificial neural network extension for HTK. In *Proc. Interspeech*, Dresden, Germany.

Zhang, C. and Woodland, P. (2015b). Parameterised sigmoid and ReLU hidden activation functions for DNN acoustic modelling. In *Proc. Interspeech*, Dresden, Germany.

Zhang, C. and Woodland, P. (2016). DNN speaker adaptation using parameterised sigmoid and ReLU hidden activation functions. In *Proc. ICASSP*, Shanghai, China.

Zhang, C. and Woodland, P. (2017). Joint optimisation of tandem systems using Gaussian mixture density neural network discriminative sequence training. In *Proc. ICASSP*, New Orleans, LA, USA.

Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Laurent, C., Bengio, Y., and Courville, A. (2016). Towards end-to-end speech recognition with deep convolutional neural networks. In *Proc. Interspeech*, San Francisco, CA, USA.

Zhao, Y., Li, J., Xue, J., and Gong, Y. (2015). Investigating online low-footprint speaker adaptation using generalized linear regression and click-through data. In *Proc. ICASSP*, Brisbane, Australia.

Zhou, P., Dai, L., and Jiang, H. (2014). Sequence training of multiple deep neural networks for better performance and faster training speed. In *Proc. ICASSP*, Florence, Italy.

Zhu, L., Kilgour, K., Stüker, S., and Waibel, A. (2015). Gaussian free cluster tree construction into deep neural network. In *Proc. Interspeech*, Dresden, Germany.

Zhu, Q., Chen, B., Grézl, F., and Morgan, N. (2005). Improved MLP structures for data-driven feature extraction for ASR. In *Proc. Interspeech*, Lisbon, Portugal.

Zue, V. and Lamel, L. (1986). An expert spectrogram reader: A knowledge-based approach to speech recognition. In *Proc. ICASSP*, Tokyo, Japan.