Structured Deep Neural Networks for Speech Recognition

Chunyang Wu

Department of Engineering

University of Cambridge

This dissertation is submitted for the degree of

Doctor of Philosophy

Wolfson College March 2018

I would like to dedicate this thesis to my loving parents.

Declaration

This dissertation is the result of my own work carried out at the University of Cambridge and includes nothing which is the outcome of any work done in collaboration except where explicitly stated. It has not been submitted in whole or in part for a degree at any other university. Some of the work has been previously presented in international conferences (Ragni et al., 2017; Wu and Gales, 2015, 2017; Wu et al., 2016a,b) and workshops, or published as journal articles (Karanasou et al., 2017; Wu et al., 2017). The length of this thesis including footnotes, appendices and references is approximately 56100 words. This thesis contains 37 figures and 35 tables.

The work on multi-basis adaptive neural networks has been published in Karanasou et al. (2017); Wu and Gales (2015); Wu et al. (2016a). I was responsible for the original ideas, code implementation, experiments and paper writing of Wu and Gales (2015); Wu et al. (2016a). Penny Karanasou contributed to the i-vector extraction, discussion and paper writing of Karanasou et al. (2017).

The work on stimulated deep neural networks has been published in Ragni et al. (2017); Wu et al. (2016a, 2017). This is an extension inspired by Tan et al. (2015a). I was responsible for the ideas, code implementation, complete experiments on Wall Street Journal and broadcast news English, and paper writing of Wu et al. (2016a, 2017). For experiments on Babel languages, I was responsible for preliminary investigations on Javanese, Pashto and Mongolian, and module scripts related to stimulated systems for all languages in the option period 3. Anton Ragni was responsible for the paper writing of Ragni et al. (2017), and other team members in the project were responsible for the joint training systems and key-word-spotting performance for all Babel languages. Penny Karanasou contributed to discussions of the model in weekly meetings.

The work on deep activation mixture models has been published in Wu and Gales (2017). I was responsible for the original ideas, code implementation, experiments and paper writing.

Chunyang Wu
March 2018

Acknowledgements

First of all, I would love to express my sincere and utmost gratitude to my supervisor, Prof. Mark Gales, for his mentorship and support over the past four years. In the period of thesis writing, I clicked and read documents I prepared for weekly meetings. Looking back, I have learned a lot from his supervision and guidance. Particularly, I want to thank him for the great patience in teaching me how to think and organise research topics logically and thoroughly. His wisdom, insight and passion in research and projects have influenced me a lot. I believe the profound influence will be there with me for my whole life. Thanks you, Mark.

Special thanks go to the NST program (EPSRC funded), the Babel program (IARPA funded), the RATS program (DARPA funded) and research funding from Google and Amazon for the financial support, providing me an excellent opportunity to be involved in high-standard research and attend many international conferences and workshops.

I want to thank my advisor Prof. Pill Woodland for his constructive suggestions in my research. Also, I owe my thanks to my colleagues in the Machine Intelligence Laboratory. Particular thanks go to Dr. Xie Chen, Dr. Penny Karanasou, Dr. Anton Ragni, Dr. Chao Zhang, Dr. Kate Knill, Dr. Yongqiang Wang, Dr. Shixiong Zhang, Dr Rogier van Dalen, Dr. Yu Wang, Dr. Yanmin Qian, Dr. Pierre Lanchantin, Dr. Jingzhou Yang, Moquan Wan and Jeremy Wong, for scintillating discussions, whether it be speech recognition, machine learning, or subjects less directly related to our research. I also would like to thank Patrick Gosling and Anna Langley for their reliable support in maintaining the computer facilities.

The friends I made in Cambridge will be a treasure forever. It is my honour to meet so many kind people in the Department, University and Wolfson College. Especially, I would like to thank my housemates and friends met at Barton House and AA57 House of Academy. Their company brought countless joyful and unforgettable moments.

Finally, the biggest thanks go to my parents. For over twenty-seven years, they have offer everything possible to support me. This thesis is dedicated to them.

Abstract

Deep neural networks (DNNs) and deep learning approaches yield state-of-the-art performance in a range of machine learning tasks, including automatic speech recognition. The multi-layer transformations and activation functions in DNNs, or related network variations, allow complex and difficult data to be well modelled. However, the highly distributed representations associated with these models make it hard to interpret the parameters. The whole neural network is commonly treated a "black box". The behaviours of activation functions and the meanings of network parameters are rarely controlled in the standard DNN training. Though a sensible performance can be achieved, the lack of interpretations to network structures and parameters causes better regularisation and adaptation on DNN models challenging. In regularisation, parameters have to be regularised universally and indiscriminately. For instance, the widely used L^2 regularisation encourages all parameters to be zeros. In adaptation, it requires to re-estimate a large number of independent parameters. Adaptation schemes in this framework cannot be effectively performed when there are limited adaptation data.

This thesis investigates structured deep neural networks. Special structures are explicitly designed, and they are imposed with desired interpretation to improve DNN regularisation and adaptation. For regularisation, parameters can be separately regularised based on their functions. For adaptation, parameters can be adapted in groups or partially adapted according to their roles in the network topology. Three forms of structured DNNs are proposed in this thesis. The contributions of these models are presented as follows.

The first contribution of this thesis is the multi-basis adaptive neural network. This form of structured DNN introduces a set of parallel sub-networks with restricted connections. The design of restricted connectivity allows different aspects of data to be explicitly learned. Sub-network outputs are then combined, and this combination module is used as the speaker-dependent structure that can be robustly estimated for adaptation.

The second contribution of this thesis is the stimulated deep neural network. This form of structured DNN relates and smooths activation functions in regions of the network. It aids the visualisation and interpretation of DNN models but also has the potential to reduce over-fitting. Novel adaptation schemes can be performed on it, taking advantages of the smooth property that the stimulated DNN offer.

The third contribution of this thesis is the deep activation mixture model. Also, this form of structured DNN encourages the outputs of activation functions to achieve a smooth surface. The output of one hidden layer is explicitly modelled as the sum of a mixture model and a residual model. The mixture model forms an activation contour, and the residual model depicts fluctuations around this contour. The smoothness yielded by a mixture model helps to regularise the overall model and allows novel adaptation schemes.

Table of contents

List of figures

List of tables

Nomenclature

General Notations

a	a scalar is denoted by a plain lowercase letter
$\boldsymbol{a}$	a column vector is denoted by a bold lowercase letter
$\boldsymbol{A}$	a matrx is denoted by a bold uppercase letter
p	probability density function
P	probability mass distribution
t	frame index
l	hidden layer index
s	speaker index
u	utterance index
$\mathcal{L}$	training criterion
$\mathcal{F}$	overall training criterion
$\mathcal{R}$	regularisation function
$\mathcal{D}$	distance function
ϵ	learning rate
η	regularisation penalty

κ	hyper-parameter or constant
$\mathbb{D}$	training data
$\boldsymbol{x}$	input feature vector
$\boldsymbol{x}_{1:i}$	feature sequence of length i
ω	output class
$\boldsymbol{\omega}_{1:i}$	class sequence of length i
$\boldsymbol{z}^{(l)}$	input of layer l
$\boldsymbol{h}^{(l)}$	output of layer l
$\boldsymbol{H}^{(l)*}$	grid representation of output of layer l
$\boldsymbol{s}_i$	grid position of hidden unit i
$\boldsymbol{y}$	DNN output vector
$\boldsymbol{\phi}$	activation function
$\boldsymbol{W}^{(l)}$	matrix parameter of layer l
$\boldsymbol{b}^{(l)}$	bias vector parameter of layer l
$\boldsymbol{\theta}$	model parameter
$\mathcal{M}$	canonical model
$\boldsymbol{\Lambda}^{(s)}$	speaker-dependent transform for speaker s
$\boldsymbol{\psi}_{1:i}$	alignment sequence of length i
$\boldsymbol{\Psi}$	alignment sequence space
$\boldsymbol{\Omega}$	decoding hypothesis space
$\mathcal{N}$	density function of Gaussian distribution

$\boldsymbol{\mu}$	mean vector of Gaussian distribution
$\boldsymbol{\Sigma}$	covariance matrix of Gaussian distribution
$\boldsymbol{\sigma}$	unit variance vector of Gaussian distribution
ρ	correlation coefficient
$\boldsymbol{K}$	filter kernel

Acronyms / Abbreviations

AI	Artificial Intelligence
AM	Acoustic Model
ASR	Automatic Speech Recognition
BN	Broadcast News
CE	Cross Entropy
CMLLR	Constrained Maximum Likelihood Linear
CMN	Cepstral Mean Normalisation
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
CTS	Conversational Telephone Speech
CV	Cross Validation
CVN	Cepstral Variance Normalisation
DAMM	Deep Activation Mixture Model
DFT	Discrete Fourier Transform
DNN	Deep Neural Network

FLP	Full Language Package
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HLDA	Heteroscedastic Linear Discriminant Analysis
HMM	Hidden Markov Model
HPF	High-pass Filtering
HTK	Hidden Markov Model Toolkit
LM	Language Model
LSTM	Long Short Term Memory
MBANN	Multi-basis Adaptive Neural Network
MBR	Minimum Bayesian Risk
MCE	Minimum Classfication Error
MFCC	Mel-Frequency Cepstral Coeficients
ML	Maximum Likelihood
MLLR	Maximum Likelihood Linear Regression
MMI	Maximum Mutual Information
MPE	Minimum Phone Error
MSE	Mean Squared Error
MTWV	Maximum Term-weighted Value
PDF	Probability Density Function
PLP	Perceptual Linear Prediction

PMF	Probability Mass Function
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SD	Speaker Dependent
SGD	Stochastic Gradient Descent
SI	Speaker Independent
SNR	Signal-to-noise Ratio
WER	Word Error Rate
WSJ	Wall Street Journal
YTB	Youtube

Chapter 1

Introduction

1.1 Deep Neural Network

In recent years, deep neural networks (DNNs) and deep learning (LeCun et al., 2015) approaches have yielded state-of-the-art performance in a wide range of tasks in machine learning and artificial intelligence (AI), including speech recognition (Dahl et al., 2012), natural language processing (Sutskever et al., 2014), and computer vision (Krizhevsky et al., 2012). These models introduce multiple layers of non-linear processing units, which allow complex data to be well modelled. Deep learning can yield automatic representations in a multi-layer configuration from raw data representation. Therefore, raw features, such as signal spectrograms and image pixels, can be directly used in this general-purpose learning algorithm. In comparison, traditional machine learning approaches highly depend on careful choices of data representation, referred to as features. Domain-specific knowledge and engineering (Chiang et al., 2009; Forman, 2003) are commonly used to design effective features for a specific task. Feature engineering could yield gains in practice; however, intensive labour on feature design limits the scope of AI application.

The concept of training multi-layer networks to replace hand-designed features was investigated at the end of the 1950s (Rosenblatt, 1958; Selfridge, 1958). In the 1980s, the invention of error back-propagation algorithms (Rumelhart et al., 1988) enabled a simple stochastic gradient descend scheme to train multi-layer models. Owning to constraints on computing resources, early DNN models (LeCun et al., 1990; Waibel et al., 1989) were often evaluated in simple and small configurations. Recently, benefiting from recent advances in computing resources, particularly graphical processing units (GPUs), large DNN models on large datasets can be optimised fast and efficiently, which is different from those in the early studies. Very deep neural networks, including the VGG (Simonyan and Zisserman, 2014) and residual networks (He et al., 2016), have been investigated, and they can consist of tens of layers. In addition, to model different types of data, a range of network variations have been proposed, including convolutional neural networks (Krizhevsky et al., 2012; LeCun et al., 1998) (CNNs) and recurrent neural networks (Bengio et al., 2003; Mikolov et al., 2010) (RNNs).

Although DNN models have achieved promising performance, there are several issues with them. One is that DNNs are likely to over-fit to training data, which limits their generalisation to unseen ones. Usually, regularisation approaches are used during training to reduce over-fitting. These include the weight decay and dropout (Srivastava et al., 2014) methods. In addition, natively trained DNNs are usually treated as "black boxes", and the highly distributed representations are difficult to interpret directly. This issue restricts the potential in further network regularisation and model post-modification.

1.2 Automatic Speech Recognition

Automatic speech recognition (ASR), sometimes referred to as speech to text, was one of the earliest tasks in AI research. Speech, or spoken language, is the most natural way to communicate between people. ASR systems can provide a more convenient and user-friendly platform for human-computer interaction. Also, automatic processing and understanding of speech by machines contribute to the ultimate goal of artificial intelligence.

The first ASR system, a digit recogniser, was implemented in 1952 by Bell Laboratories (Davis et al., 1952). Early studies in speech recognition concentrated on rule-based and knowledge-based heuristic approaches, such as the acoustic phonetic approach (Hemdal and Hughes, 1967) and pattern matching (Itakura, 1975). In the

1970s, hidden Markov models (HMMs), particularly Gaussian mixture model HMMs (GMM-HMMs), were introduced to speech recognition (Baker, 1975; Jelinek, 1976). Since then, statistical method have dominated this research area. Mathematically, given a sequence of features $\boldsymbol{x}_{1:T}$, of length T, extracted from raw speech signal,

$$\boldsymbol{x}_{1:M} = \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T, \tag{1.1}$$

according to Bayes' decision rule, the most likely decoding hypothesis $\hat{\boldsymbol{\omega}}$, is given as

$$\hat{\boldsymbol{\omega}} = \arg\max_{\boldsymbol{\omega}} P(\boldsymbol{\omega}|\boldsymbol{x}_{1:T}). \tag{1.2}$$

This hypothesis can be expressed as a word sequence of length M,

$$\boldsymbol{\omega}_{1:M} = \omega_1, \omega_2, \ldots, \omega_M. \tag{1.3}$$

Using Bayes' rule, the decision formula can be rewritten as

$$\hat{\boldsymbol{\omega}} = \arg\max_{\boldsymbol{\omega}} p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega})P(\boldsymbol{\omega}) \tag{1.4}$$

where $p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega})$ is referred to as the acoustic model, and $P(\boldsymbol{\omega})$ is the language model. In this configuration, the ASR system is described in a generative framework. HMMs are applied to estimate the acoustic model, $p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega})$. A range of effective extensions were proposed for the HMM framework in the following decades, including state tying (Young et al., 1994), discriminative training (Povey and Woodland, 2002), and speaker/noise adaptation (Gales, 1998; Leggetter and Woodland, 1995). Meanwhile, ASR systems have also evolved from recognising isolated words to large-vocabulary continuous speech, from handling clean environments to complex scenarios such as telephone conversation (Godfrey et al., 1992). Recent progress in integrating deep learning to HMMs has significantly improved the performance of ASR systems (Dahl et al., 2012; Deng et al., 2013; Hinton et al., 2012; Seide et al., 2011b). Instead, discriminative models (Cho et al., 2014; Graves et al., 2006; Sutskever et al., 2014), also known as end-to-end models, has been investigated as well. In these models, neural networks are used to model the conditional probability $P(\boldsymbol{\omega}|\boldsymbol{x}_{1:T})$, which is directly related to the decision rule.

Efforts in the past half century have greatly improved the technology in speech recognition. Such technology has started to change the lifestyle of human being, and promote the progress of civilisation. Nevertheless, there are still a number of issues remaining in such techniques. For instance, effective and rapid adaptation methods to specific accents and corruptions, such as speech distorted by noise and reverberation, remain major challenges in modern ASR systems, particularly DNN-based schemes. They requires further exploration in the realm of speech recognition as well as artificial intelligence.

1.3 Thesis Organisation

Common neural network configurations introduce a large number of parameters, and hidden units are treated as independent components, not groups based on functional similarities. One crutial concern in learning algorithms is how well a model can perform on unseen data rather than the training data. For DNNs, the large number of parameters are likely to be over-fitted to the training data. In network training, methods such as regularisation are often used to improve the generalisation on unseen data. The lack of interpretability can cause issues, and possible limitations, in further network regularisation and adaptation. One example is speaker adaptation that it is difficult to robustly estimate a large number of independent parameters when there is limited adaptation data.

This thesis presents three forms of structured deep neural networks to address the issues above. Several structures, for either activation functions or parameters, are explicitly imposed to the network topology, making specific aspects of the data well modelled. The major contribution of this thesis is that structured DNNs help to aid interpretation and improve regularisation and adaptation for DNN models. The proposed models can be applied to various tasks. In the discussion of the framework, this thesis mainly focuses on its application to speech recognition. The rest of this thesis is organised as follows.

Chapter 2 provides an overview of deep neural networks and deep learning. It covers the description of conventional network architectures, training, and regularisation schemes.

Chapter 3 presents deep learning approaches in speech recognition. It consists of key techniques of ASR systems, followed by various forms of deep learning in both generative and discriminative models for sequential tasks such as speech recognition.

Chapter 4 describes multi-basis adaptive neural networks. This form of structured DNN modifies the network topology and introduces a set of parallel sub-networks with restricted connectivity. The restricted connectivity causes different aspects of data to be explicitly learned. Sub-network outputs are then combined, and this combination module can be used as speaker-dependent parameters, which can be robustly estimated for adaptation.

Chapter 5 describes stimulated deep neural networks. Traditional DNN configurations treat activation functions to as independent components. Instead, this structured DNN models activation function outputs to be smoothed and related in regions of the network. It aids visualisation and interpretations of the network, but also has the potential to reduce over-fitting. In addition, novel techniques for speaker adaptation can be applied to it, taking advantages of the smooth property that stimulated DNNs offer.

Chapter 6 describes deep activation mixture models. Similar to stimulated DNNs, this form of structured neural network encourages activation function outputs to achieve a smooth surface. The output of one hidden layer is explicitly modelled as the sum of a mixture and residual models. The mixture model forms an activation contour, and the residual model depicts fluctuations around this contour. The smoothness yielded by a mixture model helps to regularise the DNN and allows novel adaptation schemes.

Chapter 7 evaluates the structured deep neural networks on the broadcast news English and Babel languages.

Chapter 8 concludes with a summary of the thesis and a discussion of future work.

Chapter 2

Deep Neural Network

In recent years, deep neural networks have been widely used in supervised learning tasks (LeCun et al., 2015). The goal of supervised learning is to infer prediction models that are able to learn and generalise from a set of training data,

$$\mathbb{D} = \{(\boldsymbol{x}_1, \omega_1), (\boldsymbol{x}_2, \omega_2), \ldots, (\boldsymbol{x}_N, \omega_N)\} \tag{2.1}$$

where $(\boldsymbol{x}_t, \omega_t)$ is a training instance: $\boldsymbol{x}_t$ is a feature vector, and ω_t is the corresponding class. This thesis concentrates on classification tasks, where the class ω_t is defined in a discrete space,

$$\omega_t \in \{\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_K\}. \tag{2.2}$$

The prediction model maps the input feature vector $\boldsymbol{x}_t$ onto the output vector $\boldsymbol{y}_t$, representing the predicted "scores" for all classes. Commonly, the scores are described in a probabilistic fashion, that is,

$$P(\omega = \mathscr{C}_k | \boldsymbol{x}_t) = y_{tk}. \tag{2.3}$$

The class with the highest score is picked as the desired prediction. This framework is usually formalised and described using Bayes' decision rule, that is, the most likely class $\hat{\omega}$ is given by

$$\hat{\omega} = \arg\max_{\omega} P(\omega | \boldsymbol{x}_t). \tag{2.4}$$

Deep neural networks are an effective form of prediction model. The multiple layers of non-linear processing units in DNNs allow complex data to be well modelled.

This chapter reviews the basic methodology used in deep neural network and deep learning algorithms. It includes typical network architectures, training and optimisation schemes, regularisation methods, network visualisation, and interpretation[1].

2.1 Neural Network Architecture

To meet the requirements for specific data and tasks, the architecture of a neural network can be organised in different ways. This section presents three typical network architectures: feed-forward neural networks, convolutional neural networks, and recurrent neural networks. Feed-forward neural networks (Rumelhart et al., 1988) are a fundamental and widely used network architecture. Convolutional neural networks (LeCun et al., 1998) are designed to model specific data with array-like internal structures, such as pixels in an image. Recurrent neural networks (Williams and Zipser, 1989) are a common network architecture to model sequential data, such as audio and sentences.

2.1.1 Feed-forward Neural Network

Feed-forward neural networks (Rumelhart et al., 1988), also known as multi-layer perceptrons, are a basic DNN architecture and widely used in a variety of machine learning tasks. Inspired by the biological neural systems that constitute animal brains (Rumelhart et al., 1986), a feed-forward DNN imitates signal transmissions among a collection of artificial neurones, referred to as units. The units are organised in a chain of layers, referred to as hidden layers. This chain design naturally structures a specific task in multiple levels, similar to the process of brain reasoning.

The topology of a feed-forward DNN is illustrated in Figure 2.1. It consists of an input layer, a series of hidden layers, and an output layer. The input layer receives input features. Then, the signals are transformed and forwarded across the hidden layers. Each hidden layer includes a number of hidden units (or nodes), and an activation

[1]To simplify the explanation, some of later sections take the feed-forward neural network as an example.

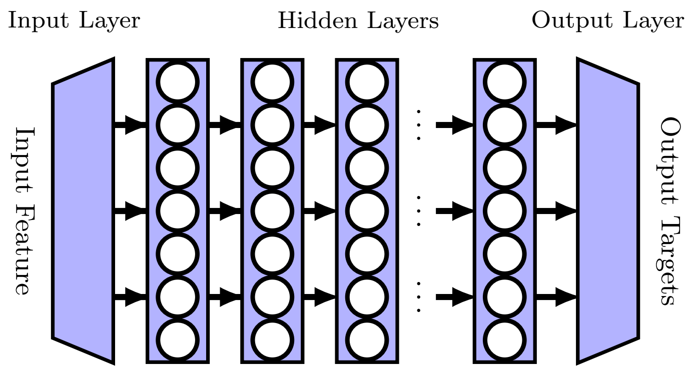


Fig. 2.1 Feed-forward neural network.

function for each unit. The unit receives signals from the previous layer, processes the information, and forwards the transformed signals to units in the next layer. Usually, the units between two successive layers are fully connected. Finally, after a series of hidden layers, the output layer yields scores for a range of classes.

Formally, given an input feature vector $\boldsymbol{x}_t$, activation function inputs $\boldsymbol{z}_t^{(l)}$ and outputs $\boldsymbol{h}_t^{(l)}$ of hidden layers are recursively defined as

$$\boldsymbol{z}_t^{(l)} = \boldsymbol{W}^{(l)\mathrm{T}}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}^{(l)} \quad 1 \leq l \leq L, \tag{2.5}$$

$$\boldsymbol{h}_t^{(l)} = \boldsymbol{\phi}\left(\boldsymbol{z}_t^{(l)}\right) \quad 1 \leq l < L, \tag{2.6}$$

$$\boldsymbol{h}_t^{(0)} = \boldsymbol{x}_t, \tag{2.7}$$

where L denotes the total number of layers, and $\boldsymbol{\phi}(\cdot)$ represents activation functions, operating on each unit. An affine transformation is applied between two successive layers associated with parameters $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$. The activation function specifies the output signal of a hidden unit. It is often modelled as a nonlinear function to allow the network to derive meaningful feature abstractions. This function can take a range of forms, and they are discussed in detail in Section 2.2.

The output of neural network is denoted by $\boldsymbol{y}_t$. For classification tasks, a softmax function is usually used , which can directly be interpreted as the conditional probability of class ω given input feature $\boldsymbol{x}_t$ (Bishop, 1995; Bridle, 1990),

$$P(\omega = \mathscr{C}_i | \boldsymbol{x}_t) = y_{ti} = \frac{\exp\left(z_{ti}^{(L)}\right)}{\sum_j \exp\left(z_{tj}^{(L)}\right)}. \tag{2.8}$$

The output vector $\boldsymbol{y}_t$ can be viewed as a soft version of 1-ok-K coding, which assigns the prediction scores for different classes. In practice, the total number of classes may vary in a large range. Simple tasks such as handwritten digit recognition contain only a few classes. For modern speech recognition systems, there can be thousands of classes involved (Dahl et al., 2012).

2.1.2 Convolutional Neural Network

Convolutional neural networks (LeCun et al., 1998) (CNNs) can be viewed as a special type of feed-forward DNNs. It was inspired by research on mammalian vision neurone systems (Hubel and Wiesel, 1965), in which individual cortical neurons respond to activation only in a restricted area of the visual field. Similarly, the CNN architecture introduces restricted connections, not fully connected, to process data with a grid-like intrinsic structure. Examples include image data, which can be viewed as 2D pixel grids, and fixed-length audio data, which can be viewed as 1D grids, sequentially sampled at a fixed rate. CNNs provide a way to specialise DNNs to explicitly work on this type of organised data.

The configuration of a hidden layer in CNNs is illustrated in Figure 2.2. It includes three processing stages: the convolution stage, detector stage, and pooling stage.

- **Convolution:** The convolution stage is a core part of CNN models. For example, in image data, one pixel is highly correlated with its neighbours in the image grid. Thus, properties in local regions are desired to be captured. In contrast with fully connected network, the convolution stage of one CNN layer introduces highly restricted connections to model local properties. It uses m_l kernels for layer l, $\boldsymbol{K}_1^{(l)}, \boldsymbol{K}_2^{(l)}, \ldots, \boldsymbol{K}_{m_l}^{(l)}$ with trainable parameters to perform convolution

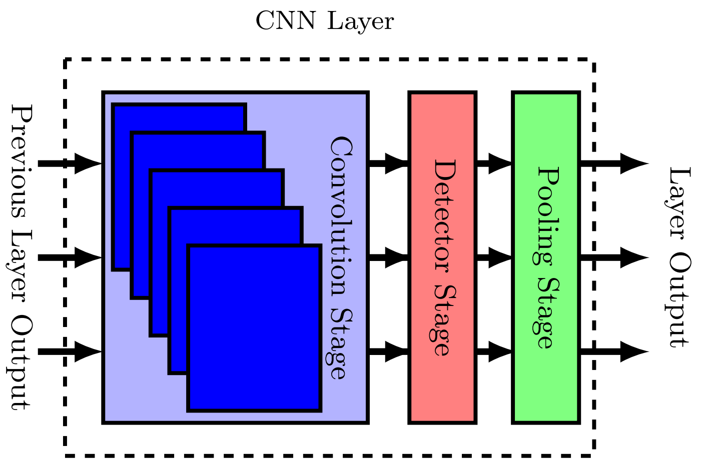


Fig. 2.2 Typical layer configuration for convolutional neural networks.

operations. The results of the j-th convolution are expressed as

$$\boldsymbol{Z}_j^{(l)} = \boldsymbol{H}^{(l-1)*} * \boldsymbol{K}_j^{(l)} \tag{2.9}$$

where $*$ stands for the convolution operation and $\boldsymbol{H}^{(l)*}$ is the grid representation of previous-layer outputs $\boldsymbol{h}^{(l)}$. Usually, $\boldsymbol{H}^{(l)*}$ is modelled as a 3D tensor. The convolution outputs $\boldsymbol{Z}_1^{(l)}, \boldsymbol{Z}_2^{(l)}, \ldots, \boldsymbol{Z}_{m_l}^{(l)}$ are then treated as "slides" and compounded to form a large tensor $\boldsymbol{Z}^{(l)*}$.

Similar to the configuration of feed-forward DNN, the convolution stage of CNN can also be viewed as applying an affine transformation,

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)\mathrm{T}} \boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{2.10}$$

where $\boldsymbol{z}^{(l)}$ is the original vector representation of $\boldsymbol{Z}^{(l)*}$. However, compared with fully connected layers in feed-forward DNNs, the transformation matrix $\boldsymbol{W}^{(l)}$ of a CNN layer is a very sparse matrix, where most elements are zero. Only a restricted number of tied parameters are introduced to focus on local regions. This allows CNN models to be robustly and efficiently trained.

- **Detector:** The outputs after convolution operations are then run through non-linear activation functions:

  $$\tilde{\boldsymbol{h}}^{(l)} = \boldsymbol{\phi}\left(\boldsymbol{z}^{(l)}\right). \tag{2.11}$$

  This stage is usually referred to as the detector stage, which generates a range of high-level feature detectors. The outputs of activation functions, $\tilde{\boldsymbol{h}}^{(l)}$, are subsequently used in the final pooling stage.

- **Pooling:** A pooling operation is used in the last processing stage to further improve activation function outputs from the detector stage. Pooling is designed to combine the outputs, and obtain summarised signals for different local regions. This operation can take a range of forms. For example, the max pooling function (Zhou et al., 1988) generates the maximal output from activation functions in a rectangular region,

  $$h_i^{(l)} = \max_{j \in \mathbb{G}_i} \left\{ \tilde{h}_j^{(l)} \right\}, \tag{2.12}$$

  where $\mathbb{G}_i$ stands for the index set of units of a rectangular region in the grid. Other popular pooling functions include the average, L^2 norm, and Gaussian blur functions, which can be performed similarly on a rectangular neighbourhood.

  Pooling can be viewed as enforcing representation to be more robust and invariant to small fluctuations from input candidates. It summarises signals from local regions, thus generating fewer outputs than those from the detector stage. This reduces both time and space computational complexities. Also, the smoothness achieved by pooling can help to improve the network regularisation.

CNN layers are usually combined with fully connected layers to obtain powerful DNN models. For instance, Alexnet (Krizhevsky et al., 2012) introduces five CNN layers and three fully connected layers; GoogLeNet (Szegedy et al., 2015) consists of 22 hidden layers of different types.

2.1.3 Recurrent Neural Network

Recurrent neural networks (Williams and Zipser, 1989) (RNNs) are a family of DNN architecture designed for sequential data, such as sentences and audio waveforms. The

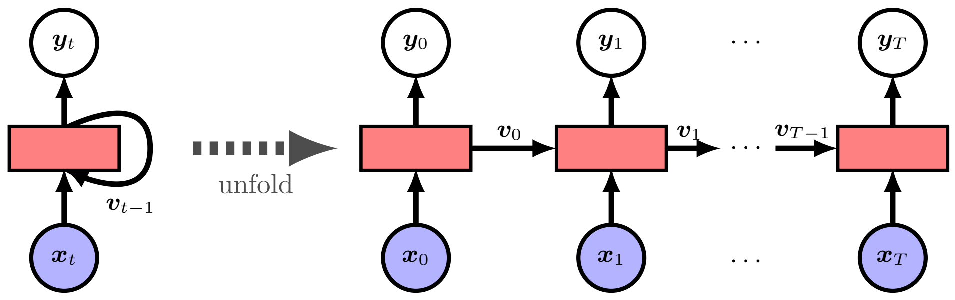


Fig. 2.3 Recurrent neural network. The loop design allows the network to be unfolded to process sequential data of variable length.

challenge of such data is the flexible length of data, which means that a fixed-length input layer cannot be utilised directly. To resolve this issue, an internal "memory" mechanism is modelled in RNNs, which can process sequences of variable lengths.

Usually, the RNN is used to model $P(\boldsymbol{\omega}_{1:T}|\boldsymbol{x}_{1:T})$, where the input feature $\boldsymbol{x}_{1:T}$ and the output class $\boldsymbol{\omega}_{1:T}$ are of variable length (denoted by T). The probability $P(\boldsymbol{\omega}_{1:T}|\boldsymbol{x}_{1:T})$ is approximated by

$$P(\boldsymbol{\omega}_{1:T}|\boldsymbol{x}_{1:T}) \simeq \prod_{t=1}^{T} P(\omega_t|\boldsymbol{x}_{1:t}), \tag{2.13}$$

where $P(\omega_t|\boldsymbol{x}_{1:t})$ is recursively commuted by the RNN. An example of RNN with one hidden layer is illustrated in Figure 2.3. In contrast to feed-forward neural networks, a connection loop is designed on the RNN architecture. This design allows the network to be unfolded to handle the variable-length issue. The loop in the hidden layer recurrently feeds a history vector $\boldsymbol{v}_{t-1}$, i.e. the delayed activation outputs $\boldsymbol{h}_{t-1}^{(1)}$ at time $t-1$, into the input layer at time t. In this way, the hidden layer can represent information both from the current input feature and the history vectors:

$$\boldsymbol{h}_t^{(1)} = \boldsymbol{\phi}(\boldsymbol{z}_t^{(1)}), \tag{2.14}$$

$$\begin{aligned} \boldsymbol{z}_t^{(1)} &= \boldsymbol{W}^{(1)T}\boldsymbol{x}_t + \boldsymbol{R}^{(1)T}\boldsymbol{v}_{t-1} + \boldsymbol{b}^{(1)} \\ &= \boldsymbol{W}^{(1)T}\boldsymbol{x}_t + \boldsymbol{R}^{(1)T}\boldsymbol{h}_{t-1}^{(1)} + \boldsymbol{b}^{(1)} \end{aligned} \tag{2.15}$$

where $\boldsymbol{R}^{(1)}$ strands for additional parameters for recurrent connections. The history vector $\boldsymbol{v}_{t-1}$ encodes a temporal representation for all past inputs, so effective history information can be preserved across time. At time t, the probabilistic interpretation of RNN output can be viewed as

$$P(\omega = \mathscr{C}_i | \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t) \simeq P(\omega = \mathscr{C}_i | \boldsymbol{x}_t, \boldsymbol{v}_{v-1}) = y_{ti}, \tag{2.16}$$

which approximates the probability condition on the full past history $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{t-1}$ by the history vector $\boldsymbol{v}_{v-1}$.

The concept of recurrent units can be implemented in a variety of ways. Deep RNNs (Pascanu et al., 2013) introduce recurrent units in multiple hidden layers. Rather than capturing information only from past history, bidirectional RNNs (Graves et al., 2013b; Schuster and Paliwal, 1997) combine information both moving forward and backward through time to yield a prediction depending on the whole input sequence. Another RNN generalisation is recursive neural networks (Socher et al., 2011). Instead of sequential data with chain dependencies, it is designed to process tree-structured data, such as syntactic trees.

Gated RNN and Long Short-term Memory

RNN models are usually trained using gradient-based algorithms (see Section 2.3.2), which require the calculation of parameter gradients. One challenge in training RNNs is that long-term dependencies (Bengio et al., 1994) cause vanishing gradients, i.e. the magnitude of gradients turns to be very small in long sequences, making recurrent network architectures difficult to optimise. Gated RNNs, such as long short-term memory (Hochreiter and Schmidhuber, 1997) and gated recurrent units (Chung et al., 2014), were developed to handle the issue of long-term dependency.

Long short-term memory (LSTM) has shown good performance in many practical applications, including speech recognition (Graves and Jaitly, 2014; Graves et al., 2013b). A diagram of an LSTM layer (also known as an LSTM block) is shown in Figure 2.4. A key modification in the of LSTM is the memory cell, which maintains history information through time. The gates (marked in red) are explicitly designed to control inward/outward information on the cell. The cell input is scaled by the input gate, while the forget cell controls what history should be retained. The cell output is

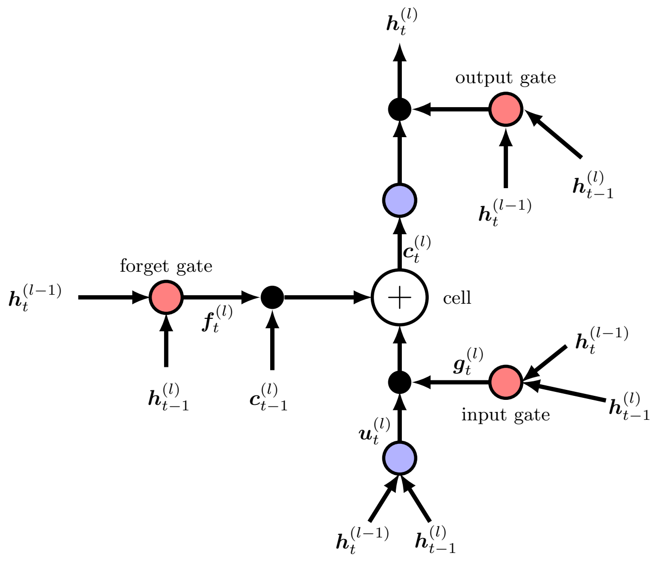


Fig. 2.4 Long short-term memory. Red circles are gate activation functions, blue circles are input/output activation functions, and black circles stand for element-wise multiplication.

also dynamically scaled by an output gate. This process can be expressed as follows:

$$\boldsymbol{u}_t^{(l)} = \boldsymbol{\phi}^{\text{in}}\left(\boldsymbol{W}_i^{(l)T}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{R}_i^{(l)T}\boldsymbol{h}_{t-1}^{(l)} + \boldsymbol{b}_i^{(l)}\right), \qquad \text{block input} \qquad (2.17)$$

$$\boldsymbol{g}_t^{(l)} = \boldsymbol{\phi}^{\text{gate}}\left(\boldsymbol{W}_g^{(l)T}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{R}_g^{(l)T}\boldsymbol{h}_{t-1}^{(l)} + \boldsymbol{b}_g^{(l)}\right) \qquad \text{input gate} \qquad (2.18)$$

$$\boldsymbol{f}_t^{(l)} = \boldsymbol{\phi}^{\text{gate}}\left(\boldsymbol{W}_f^{(l)T}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{R}_f^{(l)T}\boldsymbol{h}_{t-1}^{(l)} + \boldsymbol{b}_f^{(l)}\right) \qquad \text{forget gate} \qquad (2.19)$$

$$\boldsymbol{c}_t^{(l)} = \boldsymbol{g}_t^{(l)} \otimes \boldsymbol{u}_t^{(l)} + \boldsymbol{u}_t^{(l)} \otimes \boldsymbol{c}_{t-1}^{(l)} \qquad \text{cell state} \qquad (2.20)$$

$$\boldsymbol{o}_t^{(l)} = \boldsymbol{\phi}^{\text{gate}}\left(\boldsymbol{W}_o^{(l)T}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{R}_o^{(l)T}\boldsymbol{h}_{t-1}^{(l)} + \boldsymbol{b}_o^{(l)}\right) \qquad \text{output gate} \qquad (2.21)$$

$$\boldsymbol{h}_t^{(l)} = \boldsymbol{o}_t^{(l)} \otimes \boldsymbol{\phi}^{\text{out}}(\boldsymbol{c}_t^{(l)}) \qquad \text{block output} \qquad (2.22)$$

where $\otimes$ stands for element-wise multiplication, $\boldsymbol{\phi}^{\text{gate}}(\cdot)$, $\boldsymbol{\phi}^{\text{in}}(\cdot)$, and $\boldsymbol{\phi}^{\text{out}}(\cdot)$ are activation functions, respectively, on the gate, block input, and output units. Usually, $\boldsymbol{\phi}^{\text{gate}}(\cdot)$ is modelled as a sigmoid function to perform like "memory gates". The design of gating contributes to preserving effective history information over a long period.

There are a range of extensions on LSTMs, such as bidirectional LSTM (Graves et al., 2013a) and variations on network connectivity. A peephole connection (Gers et al., 2002) is a common setting in the latest LSTMs, which connects the cell unit with different gates to learn a precise timing. For deep LSTMs, high-way connections between cells in adjacent layers (Zhang et al., 2016b) are introduced for effective training.

2.2 Activation Function

This section describes the activation functions, $\boldsymbol{\phi}(\cdot)$, commonly used in neural networks. Intuitively, the activation function on a hidden unit performs like a "switch", which can be turned either "on" (activated) or "off" (deactivated), controlled by its input signal. It is often modelled as a continuous non-linear function to induce distributed representations in the hidden layers that allows the model parameters to be tuned. The non-linearity of the activation function allows DNN models to overcome some trivial degradation. That is, suppose a simple linear function is applied[2]:

$$\boldsymbol{\phi}(\boldsymbol{z}) = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{z} + \boldsymbol{b}. \tag{2.23}$$

This linear setting makes the overall DNN a model consisting of multiple affine transformations. Since combining multiple affine transformations is identical to a single affine transformations, this DNN model degrades to become a simple linear one, which means that the large number of parameters fails to increase the modelling capacity.

Non-linear activation functions can take a range of forms. There are some common desirable properties of an appropriate function form. First, it should be non-linear, as discussed above, to trigger non-trivial models. Second, it should be continuously (sub-)differentiable[3] for direct integration into gradient-based optimisation algorithms. Third, it should be sufficiently smooth to make the gradient stable. Sigmoid, hyperbolic tangent, rectified linear unit, maxout, softmax, Hermite polynomial, and radial basis functions are also discussed in this section.

[2]In this section, $\boldsymbol{z}_t^{(l)}$ is abbreviated as $\boldsymbol{z}$ where the superscript l (layer index) and subscript t (sample index) are omitted, to simplify the notations for discussion.

[3]Some functions may not be differentiable only at some points in its domain. However, on these points, a set of values can be used as gradients, to generalise the concept of the derivative to functions. This is referred to as sub-differentiable, and the picked "gradient" is named as the sub-gradient.

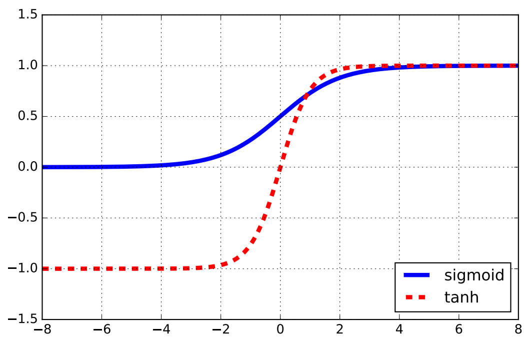


Fig. 2.5 Sigmoid and hyperbolic tangent activation functions.

Sigmoid

The sigmoid activation function is a common choice in most neural network configurations, defined as

$$\phi_i(\boldsymbol{z}) = \mathrm{sig}(z_i) = \frac{1}{1+\exp(-z_i)}. \tag{2.24}$$

Figure 2.5 illustrates the plot of a sigmoid activation function. This function has an "S"-shaped curve, which can be viewed as a soft version of the desired "switch" design: when z_i is very positive, $\phi_i(\boldsymbol{z})$ is close to 1, and when z_i is very negative, $\phi_i(\boldsymbol{z})$ is near 0.

Hyperbolic Tangent

The hyperbolic tangent (tanh) activation function is defined as

$$\phi_i(\boldsymbol{z}) = \tanh(z_i) = \frac{1-\exp(-2z_i)}{1+\exp(-2z_i)}. \tag{2.25}$$

As shown in Figure 2.5, the tanh function also has an "S"-shaped curve, but it works in a different dynamic range, $[-1,1]$, in contrast with the sigmoid function, $[0,1]$. This form of activation function is closely related to the sigmoid function, since

$$\tanh(\boldsymbol{z}_i) = \frac{1-\exp(-2z_i)}{1+\exp(-2z_i)} = \frac{2}{1+\exp(-2z_i)} - 1 = 2\mathrm{sig}(2z_i) - 1. \tag{2.26}$$

Rather than this analytic expression, hard tanh function (Collobert, 2004), defined as

$$\phi_i(\boldsymbol{z}) = \max\{-1, min(1, z_i)\}, \tag{2.27}$$

has also been proposed. This function has a similar shape to tanh but consists only of simple algebraic operations to form a hard "S"-shaped curve, as indicated in its name.

Rectified Linear Unit

The rectified linear unit (ReLU) function (Nair and Hinton, 2010), also known as the ramp function, is defined as

$$\phi_i(\boldsymbol{z}) = \max\{0, z_i\}. \tag{2.28}$$

In its positive half domain, it is identical to a linear function, while it remains at zero in its negative half domain. An advantage of this simple design of ReLU is that its sub-gradient can take a very simple form,

$$\frac{\partial \phi_i}{\partial z_i} = \begin{cases} 0, & z_i \leq 0, \\ 1, & z_i > 0. \end{cases} \tag{2.29}$$

Since there is no complex computation, such as an exponential operation, involved, the network with ReLU units can be fast and efficiently optimised. For very large DNN configurations (Dahl et al., 2013; Glorot et al., 2011; Krizhevsky et al., 2012), ReLU activation functions are often used in DNN models requiring efficient training.

There are a number of variants of the ReLU activation function. One example is the parametric ReLU, which introduces a non-zero slope κ_i for $z_i < 0$:

$$\phi_i(\boldsymbol{z}) = \max\{0, z_i\} + \kappa_i \min\{0, z_i\}. \tag{2.30}$$

The slope κ_i can be either trained as a learnable parameter (He et al., 2015) or tuned in a heuristic fashion (Maas et al., 2013).

Maxout

The maxout activation function (Goodfellow et al., 2013) can be viewed as a generalised version of the ReLU function. Instead of an element-wise function, it divides $\boldsymbol{z}$ into M subsets, $\mathbb{Z}_1, \ldots, \mathbb{Z}_M$, with k elements in each. Maxout is then applied to each group, defined as

$$\phi_i(\boldsymbol{z}) = \max_{z \in \mathbb{Z}_i} \{z\}. \tag{2.31}$$

This form of activation function does not specify the curve shape. Instead, it can approximate an arbitrary convex function using k linear segments. Therefore, the maxout function has the capacity to learn an appropriate activation function itself. This activation function has a similar form to the max pooling (Eq. 2.12). In the pooling operation, the candidates are usually designed as some neighbours that has some explicit meaning, e.g., nearby feature detectors in CNNs. In comparison, the candidates of the maxout function are learned automatically without this kind of pre-defined physical meaning.

Softmax

The softmax function is commonly used in the output layer of neural network for classification tasks with multiple classes:

$$\phi_i(\boldsymbol{z}) = \frac{\exp\left(z_i^{(L)}\right)}{\sum_j \exp\left(z_j^{(L)}\right)}. \tag{2.32}$$

A normalisation term, $\sum_j \exp\left(z_j^{(L)}\right)$, is introduced. It satisfies

$$\phi_i(z) \geq 1 \qquad \forall i, \tag{2.33}$$

$$\sum_i \phi_i(z) = 1. \tag{2.34}$$

Therefore, the softmax function can be interpreted as a discrete distribution (Bishop, 1995; Bridle, 1990).

Hermite Polynomial

The Hermite polynomial activation function is defined as

$$\phi_i(\boldsymbol{z}) = \sum_{r=1}^{R} c_{ir} g_r(z_i) \tag{2.35}$$

where $\boldsymbol{c}_i$ is the parameters associated with this activation function, R is the degree of Hermite polynomial, and $g_r(z_i)$ is the r-th Hermite orthonormal function, which is recursively defined as

$$g_r(z_i) = \kappa_r G_r(z_i)\psi(z_i) \tag{2.36}$$

where

$$\kappa_r = (r!)^{-\frac{1}{2}}\pi^{\frac{1}{4}}2^{-\frac{r-1}{2}}, \tag{2.37}$$

$$\psi(z_i) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{z_i^2}{2}\right), \tag{2.38}$$

$$G_r(z_i) = \begin{cases} 2zG_{r-1}(z_i) - 2(r-1)G_{r-2}(z_i) & r > 1, \\ 2z_i & r = 1, \\ 1 & r = 0. \end{cases} \tag{2.39}$$

Unlike many forms of activation functions, there are parameters $\boldsymbol{c}_i$, in an Hermite polynomial, that can be trained, or manually set. These parameters enrich the expressiveness of the activation function. Siniscalchi et al. (2013) introduced Hermite polynomials as activation functions, and re-estimated such parameters for speaker adaptation in speech recognition tasks.

Radial Basis Function

The radial basis function (RBF) is defined as

$$\phi_i(\boldsymbol{z}) = \exp\left(-\frac{1}{\sigma_i^2}||\boldsymbol{z} - \boldsymbol{c}_i||_2^2\right) \tag{2.40}$$

where σ_i and $\boldsymbol{c}_i$ are the activation function parameters. This activation function defines a desired template $\boldsymbol{c}_i$, and it becomes more active as $\boldsymbol{z}$ approaches the template. Neural networks using this form of activation functions are commonly referred to as RBF networks (Orr et al., 1996).

2.3 Network Training

So far, neural networks have been described as non-linear functions mapping the input vector $\boldsymbol{x}$ to the output vector $\boldsymbol{y}$ (Eq. 2.8). By appropriately designing the output layer, using a softmax function, it is possible to give the network output a probabilistic interpretation (Bishop, 1995; Bridle, 1990). This interpretation enables DNNs to preserve as probabilistic models for classification tasks. This section describes the training and optimisation of parameters in DNN models.

2.3.1 Training Criterion

To perform network training, a training criterion needs to be defined first. The training criterion[4], defined as $\mathcal{L}(\boldsymbol{\theta};\mathbb{D})$, should yield a scalar value that measures how well a model with parameters $\boldsymbol{\theta}$ performs the mapping of feature vector to the class for the training data $\mathbb{D}$. The definition of "well" depends on the task. For example, in speech recognition, the "well" can be defined as the error rate of recognised words in a sentence (Section 3.4).

Cross Entropy Criterion

Cross entropy (CE) is a training criterion widely used in classification tasks. It is defined as

$$\mathcal{L}^{ce}(\boldsymbol{\theta};\mathbb{D}) = -\sum_{t=1}^{|\mathbb{D}|}\sum_{i} P_t^{\text{ref}}(i)\log P(i|\boldsymbol{x}_t). \tag{2.41}$$

where, for the t-th training sample, $P(i|\boldsymbol{x}_t)$ stands for the predicted distribution, and $P_t^{\text{ref}}(i)$ represents the reference distribution. In the context of neural networks, $P(i|\boldsymbol{x}_t)$ is given by the network output, y_{ti}. Usually, a hard target label ω_t is given in one

[4]Rather than training criteria, similar terminologies such as objective, loss, or cost functions are also used by different people. Training criteria can be either maximised or minimised. To maintain consistency in this thesis, a training criterion is defined as a function to minimise.

training sample with no uncertainty. Thus, the reference distribution $P_t^{\text{ref}}(i)$ can be expressed as

$$P_t^{\text{ref}}(i) = \begin{cases} 1, & \mathscr{C}_i = \omega_t, \\ 0, & \mathscr{C}_i \neq \omega_t. \end{cases} \tag{2.42}$$

Therefore, the overall CE criterion can be simplified as

$$\mathcal{L}^{\text{ce}}(\boldsymbol{\theta}; \mathbb{D}) = -\frac{1}{|\mathbb{D}|} \sum_{t=1}^{|\mathbb{D}|} \log P(\omega_t | \boldsymbol{x}_t). \tag{2.43}$$

In this scenario, CE is identical to the negative log likelihood of generating targets given features for the training samples.

2.3.2 Parameter Optimisation

Given training data and an appropriate training criterion, it is necessary to define methods to find parameters that minimises the criterion. Gradient descent, sometimes known as steepest descent, is a simple iterative algorithm for finding the minimum of a function. In general, it updates the parameters iteratively, and the update rule in one iteration is defined as

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \epsilon \left. \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathbb{D}) \right|_{\boldsymbol{\theta} = \boldsymbol{\theta}^{(\tau)}} \tag{2.44}$$

where the hyper-parameter ϵ is referred to as the learning rate, which decides the step size of this update. At iteration $\tau + 1$, the parameters take a step proportional to the negative gradient direction at iteration τ, resulting in a decrease of the training criterion.

Other than gradient-based methods, a broad range of algorithms have been studied to train DNN parameters. Second-order methods, such as Newton and quasi-Newton methods (Bishop, 1995), utilise statistics from second-order derivatives to update the parameters. Such schemes require higher computational complexity in both time and space, but they can yield better local minima with fewer update iterations. In addition, Hessian-free methods (Kingsbury et al., 2012; Martens, 2010) has also been investigated.

Algorithm 1 Stochastic gradient descent.

```
1: initialize $\boldsymbol{\theta}$
2: divide training data $\mathbb{D}$ into $M$ mini-batches $\{\mathbb{M}_j\}_{1\leq j\leq M}$.
3: for $i := 1$ to $I$ do
4:     for $j := 1$ to $M$ do
5:         $\Delta := \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathbb{M}_j)$
6:         $\boldsymbol{\theta} := \boldsymbol{\theta} - \epsilon\Delta$
7:     end for
8: end for
```

This thesis focuses on gradient descent, particularly stochastic gradient descent, which has been widely used in DNN training.

Stochastic Gradient Descent

A simple way to implement the gradient descent is to accumulate partial derivatives computed on the complete training data. In this thesis, this form is referred to as batch-mode gradient descent. At each iteration, it can estimate an accurate gradient, but it requires traversing all training samples. This can be extremely computationally expensive when the dataset is large.

An alternative method, referred to as stochastic gradient descent (SGD), can be used for efficient training. Instead of processing the entire dataset, the gradient in SGD is calculated using a small portion of data. The outline of SGD is illustrated in Algorithm 1. The training data is divided into M mini-batches, and as a result, parameters are updated M times, not once, in each iteration i of SGD.

In contrast with batch-mode gradient descent, stochastic gradient descent updates parameters using "less accurate" gradients, which are estimated on mini-batches. An advantage is that, if the surface of the training criterion is not smooth, error and uncertainty in mini-batch gradients can help to avoid bad local minima. However, in practice, there are issues that need to address to efficiently use the SGD approach. The size of a mini-batch should be selected to balance the gradient accuracy and training efficiency. Furthermore, training data should be randomly shuffled, to make samples in a mini-batch less "biased". For example, in speech recognition, if mini-batches are obtained from unshuffled data, samples in one mini-batch are likely to come from the same speaker or environmental condition, and the update on this mini-batch would degrade its generalisation to arbitrary acoustic conditions.

Learning Rate

The learning rate in SGD determines how much the parameters are changed in one update. If a large learning rate is used, training is likely to fluctuate and skip "good" local minima. If it is too small, training will be very slow to converge, or it will fall into some local minimum in the training criterion error surface. Several empirical and heuristic approaches have been proposed. These approaches adaptively change the learning rate to improve the performance of SGD training. For instance, the NewBob method (Renals et al., 1991) adaptively determines the learning rate according to temporary system performance during training. Decay methods (Bottou, 2010; Xu, 2011) reduce the learning rate gradually after each SGD update. Alternative methods associate an individual learning rate with each parameter and adjust them according to heuristic rules (Duchi et al., 2011; Riedmiller and Braun, 1993).

Momentum

Momentum (Polyak, 1964) is a strategy to accelerate the convergence of optimisation. In gradient-based algorithms such as SGD, the momentum method recursively accumulates a decaying average $\bar{\boldsymbol{\Delta}}$ of past gradients and adds it to the current update. The update rule with momentum can be expressed in a recursion:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \bar{\boldsymbol{\Delta}}^{(\tau+1)}, \tag{2.45}$$

$$\bar{\boldsymbol{\Delta}}^{(\tau+1)} = \kappa \boldsymbol{\Delta}^{(\tau)} - \epsilon \left.\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(\tau)}} \qquad \tau \geq 1, \tag{2.46}$$

$$\bar{\boldsymbol{\Delta}}^{(0)} = \mathbf{0}, \tag{2.47}$$

where a hyper-parameter κ, referred to as the momentum coefficient, determines the impact of past gradients. A higher κ amplifies the influence of past updates compared to the current one. The advantage of momentum is that the inertia on the update direction can be maintained to reduce the risk of oscillation, resulting in a smoother decrease of the training criterion.

In addition to this classic momentum definition, other forms of momentum have also been used in DNN training, such as Nesterov momentum (Nesterov, 1983; Sutskever et al., 2013).

2.3.3 Error Back-propagation Algorithm

To implement the gradient descent, partial derivatives with respect to different parameters need to be calculated. For neural networks, the error back-propagation algorithm (Rumelhart et al., 1988) is an efficient method to calculate parameter gradients. It is based on a concept of signal passing in which "signals" are delivered both forwards and backwards through the network. These signals determine the procedure of gradient calculation.

Consider a feed-forward neural network with L layers. The signal passing starts with the forward step. On training sample $(\boldsymbol{x}_t, \omega_t)$, the forward of the input signal, feature vector $\boldsymbol{x}_t$, yields a series of hidden-layer and output signals, $\boldsymbol{h}_t^{(1)}, \boldsymbol{h}_t^{(2)}, \ldots, \boldsymbol{h}_t^{(L-1)}, \boldsymbol{y}_t$. The backward step is then performed to evaluate "error" signals in different layers. The error signal on the output layer is directly determined by the training criterion. If the CE criterion is used, the gradient of the training criterion with respect to $\boldsymbol{y}_t$ is calculated by

$$\frac{\partial \mathcal{L}}{\partial y_{ti}} = \frac{\partial \mathcal{L}^{ce}}{\partial y_{ti}} = -\frac{\delta(\mathscr{C}_i, \omega_t)}{y_{ti}} \tag{2.48}$$

where $\delta(a,b)$ stands for the Kronecker delta

$$\delta(a,b) = \begin{cases} 0 & a = b, \\ 1 & a \neq b. \end{cases} \tag{2.49}$$

Errors from the output layer, described in $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t}$, are then passed to lower layers. On hidden layer l, error signals from layer $l+1$ are first received. The gradient with respect to activation function outputs can then be calculated using[5]

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t^{(l)}} = \boldsymbol{W}^{(l)} \boldsymbol{D}_t^{(l)} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t^{(l+1)}}, \quad 1 \leq l < L \tag{2.50}$$

where $\boldsymbol{D}_t^{(l)}$ is a matrix representing the gradient of activation function

$$d_{tij}^{(l)} = \frac{\partial h_{ti}^{(l)}}{\partial z_{tj}^{(l)}} = \phi_i' \left(z_{tj}^{(l)} \right). \tag{2.51}$$

This derivative depends on the choice of activation function. For simple forms such as the sigmoid and tanh functions, $\boldsymbol{D}_t^{(l)}$ is a diagonal matrix. The backward step determines the error for each unit in the network, and by using Eq. 2.50, the gradient with respect to network parameters can be written as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}} = \sum_t \boldsymbol{D}_t^{(l)} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t^{(l+1)}} \boldsymbol{h}_t^{\mathrm{T}}, \tag{2.52}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(l)}} = \sum_t \boldsymbol{D}_t^{(l)} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t^{(l+1)}}. \tag{2.53}$$

As shown in Eq. 2.52 and 2.53, the gradient calculation can be performed in a recursive way, according to the network topology. The error back-propagation algorithm reveals a simple and efficient way to calculate parameter gradients.

2.3.4 Parameter Initialisation

Similar to many learning algorithms, parameters in DNN models should be randomly initialised to break modelling symmetries. However, neural networks, particularly deep ones, introduce complex, multi-layer models. A simple procedure using random initialisation and gradient descent optimisation on DNNs is likely to converge to poor local minima (Glorot and Bengio, 2010). For effective training, parameter initialisation requires further steps for DNN models.

[5] For a unified form, the network output $\boldsymbol{y}_t$ is denoted as $\boldsymbol{h}_t^{(L)}$.

Pre-training is one common DNN initialisation strategy[6]. This strategy divides training into two phases, pre-training and fine-tuning phases. The pre-training phase aims at simple but improvable representations in hidden layers, to overcome poor local minima. The fine-tuning phase is the regular training phase, which fully updates parameters to converge. Pre-training can be performed in either generative ways, such as stacking restricted Boltzmann machine (Hinton et al., 2006), or discriminative ways, such as discriminative layer-wise pre-training (Bengio et al., 2007).

Random Initialisation

Usually, the random parameter initialisation is performed by generating independent samples from some distribution, such as a uniform distribution. For DNNs, the interval of sampled parameters should be carefully controlled. If the interval is too narrow, parameters are initialised too close to zero. It causes the activation function outputs to stay around the linear range, which cannot induce effective non-linearity; if too wide, the outputs of activation functions are likely to be out of the dynamic range. This makes parameter optimisation difficult. Therefore, the interval of sampled parameters should be selected to ensure that activation functions in all hidden layers are initialised to sensibly perform in the dynamic range. Based on this concept, Xavier's initialisation (Glorot and Bengio, 2010) samples matrix parameters of a sigmoid DNN from a uniform distribution and keeps the bias vector at zero:

$$w_{ij}^{(l)} \sim \mathcal{U}\left(-\frac{4\sqrt{6}}{\sqrt{N_{\text{in}}^{(l)}+N_{\text{out}}^{(l)}}}, \frac{4\sqrt{6}}{\sqrt{N_{\text{in}}^{(l)}+N_{\text{out}}^{(l)}}}\right), \qquad \forall i,j \tag{2.54}$$

$$b_k^{(l)} = 0, \qquad \forall k \tag{2.55}$$

where $\mathcal{U}(\cdot,\cdot)$ stands for a uniform distribution, $N_{\text{in}}^{(l)}$ and $N_{\text{out}}^{(l)}$ are, respectively, the input and output dimensions of layer l. Activation function inputs are initialised to evenly cover the interval, $[-6,6]$, which prevents outputs from getting too close to either 0 or 1.

[6] At the time of writing, the latest DNN systems on large datasets no longer require pre-training, such as deep ReLU networks (Glorot et al., 2011). However, pre-training plays a useful role with smaller datasets and is related to experimental settings in this thesis.

Generative Pre-training

In the literature, generative pre-training, particularly stacking restricted Boltzmann machines (Hinton et al., 2006) (RBMs), was one of the earliest strategies proposed to initialise sensible DNN parameters. An RBM describes an undirected probabilistic graphical model, consisting of a set of unobserved variables $\boldsymbol{u}$ and a set of observed variables $\boldsymbol{v}$. Connections are only introduced between observed and unobserved variables. It specifies an energy function[7] for any configuration of $\boldsymbol{u}$ and $\boldsymbol{v}$, defined as

$$E(\boldsymbol{v},\boldsymbol{h}) = -\boldsymbol{c}_{\text{rbm}}^{\text{T}}\boldsymbol{v} - \boldsymbol{b}_{\text{rbm}}^{\text{T}}\boldsymbol{u} - \boldsymbol{v}^{\text{T}}\boldsymbol{W}_{\text{rbm}}\boldsymbol{u} \tag{2.56}$$

where $\boldsymbol{W}_{\text{rbm}}$, $\boldsymbol{b}_{\text{rbm}}$, and $\boldsymbol{c}_{\text{rbm}}$ are RBM parameters. In terms of the energy concept, the joint probability of observed and unobserved variables can be expressed as

$$P(\boldsymbol{v},\boldsymbol{u}) = \frac{\exp(-E(\boldsymbol{v},\boldsymbol{u}))}{\sum_{\tilde{\boldsymbol{v}},\tilde{\boldsymbol{u}}}\exp(-E(\tilde{\boldsymbol{v}},\tilde{\boldsymbol{u}}))}. \tag{2.57}$$

In practice, RBMs can be efficiently trained by minimising log likelihood using the contrastive divergence algorithm (Hinton, 2002).

Generative pre-training is performed by stacking RBMs. From lower to upper, any two adjacent layers $\boldsymbol{h}^{(l)}$ and $\boldsymbol{h}^{(l+1)}$, prior to the output layer, are trained as an RBM. By rewriting Eq. 2.57, the conditional probability of $\boldsymbol{u}$ given $\boldsymbol{v}$ can yield an interesting form,

$$P(u_i = 1|\boldsymbol{v}) = \textbf{sig}\left(\boldsymbol{w}_{\text{rbm},i}^{\text{T}}\boldsymbol{v} + b_{\text{rbm},i}\right). \tag{2.58}$$

RBM parameters $\boldsymbol{W}_{\text{rbm}}$ and $\boldsymbol{b}_{\text{rbm}}$ can then be used to initialise the transformation matrix $\boldsymbol{W}^{(l)}$ and bias vector $\boldsymbol{b}^{(l)}$ on layer l. Finally, a randomly initialised output layer is added to the top of stacked RBMs. An advantage of generative pre-training is that it is performed in an unsupervised way, requiring no labels for training data. Especially for resource-limited tasks, this scheme allows unlabelled data to be utilised for parameter initialisation.

[7]A simple configuration of RBM is presented, where $\boldsymbol{u}$ and $\boldsymbol{v}$ are defined as binary variables.

Algorithm 2 Layer-wise discriminative pre-training framework.

```
1: $\boldsymbol{\theta} := \emptyset$
2: for $l := 1$ to $L$ do
3:     initialize $\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}$
4:     $\boldsymbol{\theta} := \boldsymbol{\theta} \cup \{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\}$
5:     initialize temporary $\boldsymbol{W}^{\text{last}}, \boldsymbol{b}^{\text{last}}$
6:     update $\boldsymbol{\theta} \cup \{\boldsymbol{W}^{\text{last}}, \boldsymbol{b}^{\text{last}}\}$ for one iteration
7: end for
```

Discriminative Pre-training

The training criteria of generative pre-training and the original task are usually different. To overcome the criterion inconsistency, discriminative pre-training (Bengio et al., 2007) initialises DNNs using the same training criterion as the original task. The basic idea is to construct DNNs in a greedy way: a network with fewer layers is sensibly trained at first, and new layers are added to the top of this shallow network. This strategy designs a "curriculum" in DNN training: primitive representations are learned in lower layers, and high-level representations can then be derived from them.

A layer-wise discriminative pre-training framework (Seide et al., 2011a) is illustrated in Algorithm 2. For each iteration, a new layer l with associated parameters $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ is added to the network configuration. A temporary last layer with parameters $\boldsymbol{W}^{\text{last}}$ and $\boldsymbol{b}^{\text{last}}$ is also introduced. This temporary DNN is then updated for one iteration. Usually, a relatively large learning rate is used in this pre-training phase, which drives parameters close to a good local minimum.

Discriminative pre-training can also be performed using related tasks rather than the original one. In speech recognition, Zhang and Woodland (2015a) initialised DNNs on an easier context-independent phoneme task for more difficult context-dependent ones. Autoencoder (Vincent et al., 2008), which yields a DNN to predict an input feature itself, is another initialisation strategy that has been shown to yield initial high-level representations.

2.4 Regularisation

One crutial concern in machine learning is how well a model yields to work well on unseen data rather than just the training data. Regularisation methods are commonly used to improve generalisation and reduce over-fitting. Regularisation is the strategy that helps to improve the generalisation. Many forms of regularisation, such as L^2 regularisation (Bishop, 1995), can be described in a framework that explicitly adds a regularisation term $\mathcal{R}(\boldsymbol{\theta};\mathbb{D})$ to the overall training criterion $\mathcal{F}(\boldsymbol{\theta};\mathbb{D})$,

$$\mathcal{F}(\boldsymbol{\theta};\mathbb{D}) = \mathcal{L}(\boldsymbol{\theta};\mathbb{D}) + \eta\mathcal{R}(\boldsymbol{\theta};\mathbb{D}) \tag{2.59}$$

where η is a non-negative hyper-parameter that determines the impact of regularisation on model training. An effective regularisation term imposes a penalty on model complexity to prevent over-fitting. Other approaches, such as dropout (Srivastava et al., 2014) for neural networks, regularise training in implicit ways rather than modifying the training criterion.

Because of the large number of parameters, DNN models usually have sufficient learning capacity to memorise and over-fit to the training data. Therefore, regularisation strategies are generally used in DNN training. This section reviews regularisation approaches for DNN models. It includes parameter norm penalties, multi-task learning (Caruana, 1997), dropout, early stopping, and data augmentation.

Parameter Norm Penalty

Parameter norm penalties (Bishop, 1995; Tibshirani, 1996) are a common form of regularisation for machine learning algorithms. The regularisation term is defined as

$$\mathcal{R}(\boldsymbol{\theta};\mathbb{D}) = \frac{1}{p}||\boldsymbol{\theta}||_p^p \tag{2.60}$$

where $||\cdot||_p$ stands for the L^p-norm[8],

$$||\boldsymbol{\theta}||_p = \sqrt[p]{\sum_i \theta_i^p}. \tag{2.61}$$

This regularisation encourages a small parameter norm during training. In the context of neural networks, it forces the weights of the multiple affine transformations to "decay" towards zero. Intuitively, this regularisation causes the network to prefer small numbers of active parameters. Large numbers of active parameters will only be allowed if they considerably improve the original training criterion $\mathcal{L}(\boldsymbol{\theta};\mathbb{D})$. It can be viewed as a way to balance active parameters and minimising $\mathcal{L}(\boldsymbol{\theta};\mathbb{D})$.

In practice, the norm degree p is usually set to 2 or 1, referred to as L^2 or L^1 regularisation. The L^2 regularisation (Bishop, 1995; Woodland, 1989), also known as weight decay or Tikhonov regularisation, penalises the sum of the squares of individual parameters,

$$\mathcal{R}(\boldsymbol{\theta};\mathbb{D}) = \frac{1}{2}||\boldsymbol{\theta}||_2^2 = \frac{1}{2}\sum_i \theta_i^2. \tag{2.62}$$

Another common form of norm penalty is L^1 regularisation (Tibshirani, 1996), defined as

$$\mathcal{R}(\boldsymbol{\theta};\mathbb{D}) = \frac{1}{2}||\boldsymbol{\theta}||_1^1 = \frac{1}{2}\sum_i |\theta_i|. \tag{2.63}$$

This penalises the sum of the absolute values of parameters rather than squares. L^1 regularisation helps to induce small numbers of parameters. Compared to L^2 regularisation, it results in a more sparse solution, in which a large number of parameters have optimal values as zeros. This sparse property induced by L^1 regularisation has been widely used as a feature selection mechanism. It yields a subset of effective features to simplify the feature space. On DNNs, L^1 regularisation can contribute to eliminate the influence of useless lower-layer units to improve generalisation for hidden units in upper layers.

[8]To simplify the notations for discussion, a unified form, denoted by θ_i, is used to represent an element of $\boldsymbol{\theta}$.

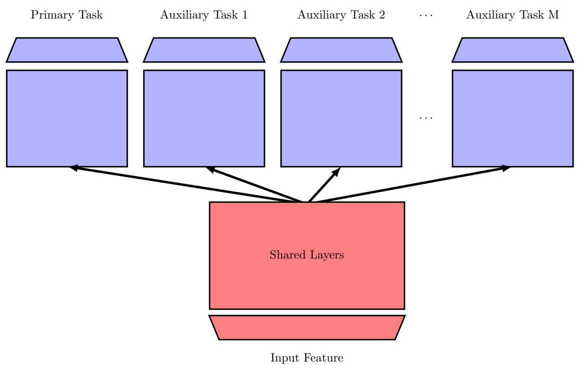


Fig. 2.6 Multi-task learning for DNN.

Multi-task Learning

Multi-task learning (Caruana, 1997) introduces a set of auxiliary tasks along with the primary one for regularisation. The primary and auxiliary tasks are usually related to each other. This approach improves generalisation by using information in the training signals from related tasks as “induction”. Induction is commonly achieved by introducing a set of shared parameters across different tasks. Shared parameters are trained to operate well on multiple tasks, reducing the risk of over-fitting to a specific task.

For DNN models, a common form of multi-task learning is illustrated in Figure 2.6. The whole model is generally divided into two categories of parameters,

1. Shared parameters (marked in red): The lower layers of the neural networks are shared across different tasks. According to studies that examine neural network behaviours (Yosinski et al., 2015; Zeiler and Fergus, 2013), lower layers concentrate on primitive abstraction of raw input features, and such relative raw information is more likely to be shared in different tasks than that in upper layers.

2. Task-specific parameters (marked in blue): The upper layers are introduced separately for each task. This design allows high-level feature abstraction to focus on modelling a specific task.

This framework allows primary and auxiliary tasks to be jointly optimised, regularisation is controlled by the set of shared parameters during training. Several multi-task

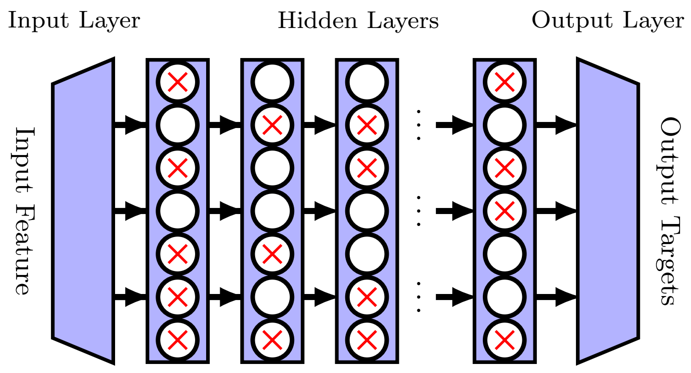


Fig. 2.7 A “thinned” neural network produced by dropout. Crossed units are dropped.

approaches have been proposed for DNN models. An example in speech recognition is multi-lingual neural networks (Heigold et al., 2013) , which extract generalised, cross-language, hidden representations as features for recognition systems. It helps to solve the data scarcity issue and reduce the performance gap between resource-rich and resource-limited languages.

Dropout

Dropout (Srivastava et al., 2014) is a regularisation strategy that efficiently trains a combination of a range of network candidates to reduce over-fitting. Model combination strategy can improve regularisation for learning algorithms. Members of the combination can either be trained separately on different datasets, or using different model configurations. For neural networks, generating a number of sensible models can be extremely hard and expensive, as hyper-parameter tuning on individual DNNs requires many trials and computing resources. Instead of training separate models, dropout addresses the issue of DNN combination in a very simple form. The term “dropout” refers to turning off, dropping, hidden units in DNN training. Figure 2.7 shows an example of performing dropout operations on a feed-forward neural network. A number of units (crossed) are dropped, and thus a “thinned” network architecture is generated. Randomly dropping units can efficiently yield exponentially many configurations by considering each “thinned” configuration as a member of the combination.

To implement dropout, on each hidden layer l, a vector $\boldsymbol{r}^{(l)}$ is introduced to determine the temporary presence of hidden units. It consists of independent Bernoulli random variables,

$$r_i^{(l)} \sim \text{Bernoulli}(\kappa) \tag{2.64}$$

where the hyper-parameter κ controls the chance of a unit to be presented in the network, referred to as present probability. This vector is sampled and then multiplied element-wisely with the output of activation functions. As a result, the output after dropout is then computed via

$$\tilde{\boldsymbol{h}}^{(l)} = \boldsymbol{r}^{(l)} \otimes \boldsymbol{h}^{(l)} \tag{2.65}$$

where the operation $\otimes$ stands for the element-wise product. Notice that $\tilde{\boldsymbol{h}}^{(l)}$, the output of a "thinned" layer, is propagated to the following layer $l+1$,

$$\boldsymbol{z}^{(l+1)} = \boldsymbol{W}^{(l)\mathrm{T}}\tilde{\boldsymbol{h}}^{(l)} + \boldsymbol{b}^{(l)}. \tag{2.66}$$

The uncertainty in the vector $\boldsymbol{r}^{(l)}$ yields a range of "thinned" sub-network configurations.

The training algorithm using dropout regularisation follows the feed-forward topology described in Eq. 2.64 to 2.66. In the propagation phase, on the l-th layer, it starts by drawing a sample of $\boldsymbol{r}^{(l)}$, a range of units are then temporarily dropped out, and the outputs of presented units are propagated to the following layer. In the back-propagation phase, only the parameters associated with presented units are updated accordingly. At test time, it is often not feasible to explicitly generate and combine all network configurations. Instead, an approximate averaging method can be applied, where an overall network is used without dropout operations. Activation function outputs are given as the expectation,

$$\tilde{\boldsymbol{h}}_{\text{test}}^{(l)} = \mathcal{E}_{\boldsymbol{r}^{(l)}}\left(\boldsymbol{r}^{(l)} \otimes \boldsymbol{h}^{(l)}\right) = \kappa\boldsymbol{h}^{(l)}. \tag{2.67}$$

The output of any hidden unit is scaled by the factor κ, the present probability. This scaling approach can be viewed as a combination of 2^{Ln} neural networks with shared parameters, where n is the total number of hidden units in one layer. In practice, this efficient averaging method can improve generalisation and avoid sampling infeasible number of networks.

Algorithm 3 Early Stopping.

```
1: initialize $\boldsymbol{\theta}$
2: $e^{\text{old}} := +\infty$
3: $e^{\text{new}} :=$ ValidateSetError($\boldsymbol{\theta}$)
4: while $e^{\text{old}} > e^{\text{new}}$ do
5:     $e^{\text{old}} := e^{\text{new}}$
6:     update $\boldsymbol{\theta}$ via back-propagation for one iteration
7:     $e^{\text{new}} :=$ ValidateSetError($\boldsymbol{\theta}$)
8: end while
```

Early Stopping

When training a large model, particularly a DNN, the training criterion on the training set often decreases consistently, but on some “held-out” cross validation (CV) data (not used for training), the criterion increases at later iterations. This phenomenon indicates over-fitting on training data. A model with lower validation set error, hopefully with lower generalisation error, can be obtained using fewer iterations of parameter updates. This strategy is referred to as early stopping.

This early stopping strategy is widely used in training DNN models. The full dataset is randomly split into two sets, a training set and a validation set. Usually, the validation set contains a small portion, such as 5% to 10%, of the full data. Note that there is no overlap between the validation and training sets. A basic framework is illustrated in Algorithm 3. The model parameters are only updated on the training set. This strategy tracks the training criterion, i.e. error, on the validation set. Once the validation error begins to increase, the training procedure terminates. Early stopping is a very simple form of regularisation, Unlike parameter norm penalties or dropout, it requires little modification to the underlying training algorithm.

There are a number of variations to implement early stopping. An example is the widely used NewBob training scheduler (Renals et al., 1991), which dynamically decreases the learning rate when the validation error rises. It helps to adapt and find an appropriate learning rate for a specific task.

Data Augmentation

For large models such as DNNs, collecting more training data is a direct way to improve generalisation. However, in practice, To collect more training data can be expensive and impracticable. Alternatively, appropriate fake samples can be generated instead of collecting real ones. Data augmentation introduces fake data to the training set to regularise network training.

For machine learning tasks, particularly classification ones, data augmentation can be simply performed via creating new samples from a real sample $(\boldsymbol{x}, y)$, through domain-specific transformations on features $\boldsymbol{x}$ and keeping the target y fixed. In computer vision, such transformations includes cropping, brightness changing, rotation, and scaling on image data. Also, in speech recognition, data augmentation on audio features, such as vocal tract length perturbation, stochastic feature mapping (Cui et al., 2015b), and tempo/speed perturbing (Ko et al., 2015), can help to improve model generalisation.

2.5 Visualisation

Multi-layer transformations and non-linear activation functions in DNN models contribute to modelling complex data. However, there is no explicit meaning for how they process, manipulate, and transform raw features into useful high-level abstractions. A DNN remains a "black box", and the lack of interpretation restricts the potential for further network improvement and post-modification. Research on network visualisation analyses qualitative comparisons of representations learned in different layers. The representations are inverted and visualised in the input space, which can illustrate the intuition of corresponding activation functions.

Maximising Activation

Maximising activation (Erhan et al., 2009) aims at constructing input features to maximise the activation function output of a particular unit. Formally, the optimal input features is given by

$$\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x}} h_i^{(l)}, \qquad \text{subject to } ||\boldsymbol{x}||_2^2 = \kappa \tag{2.68}$$

where *kappa* is a positive hyper-parameter. The maximisation is restricted with a bounded norm of input features, $||\boldsymbol{x}||_2^2$, which prevents trivial solutions. The optimisation can be simply performed via gradient descent on input features

$$\frac{\partial h_i^{(l)}}{\partial \boldsymbol{x}} = -\boldsymbol{W}^{(1)} \frac{\partial h_i^{(l)}}{\partial \boldsymbol{h}^{(1)}}, \tag{2.69}$$

while network parameters remain fixed. In general, maximising an activation function in higher layers is not a convex problem, so it requires a careful initialisation of input features. Another limitation of this approach is that it cannot give information about the unit invariance, i.e. the tolerance on input feature variations, because only a static, reconstructed input feature representation is presented. Ngiam et al. (2010) analysed the Hessian matrix, rather than gradients, of an activation function numerically computed around the optimal solution to give insight into invariance.

Network visualisation is usually conducted on vision tasks, in which input features are images. For this reason, most visualisation approaches are especially designed for CNNs, which is a typical network architecture in computer vision. Deconvnet (Simonyan et al., 2013; Zeiler and Fergus, 2014) introduces an inverted path on each CNN layer to reconstruct images, which oppositely maps hidden units to input pixels.

2.6 Summary

This chapter reviews the fundamentals of neural network and deep learning. It begins with basic network architectures, and three typical forms of DNN are presented: feed-forward neural networks, convolutional neural networks, and recurrent neural networks. The activation functions are then reviewed, which is a key technique in neural networks to trigger meaningful non-linear representations. They include sigmoid, tanh, ReLU, maxout, softmax, Hermite polynomial, and RBF functions. The following section describes the training and optimisation of DNN parameters. Training criteria are described first, which specify the overall goal of training. The basic cross-entropy criterion is presented as an example. Parameter optimisation focuses on practical techniques, including stochastic gradient descent, learning rate adjustment, and momentum. It is followed by the error back-propagation algorithm: according to the chain rule, gradient calculation can be performed in a simple but efficient way. The highly complex model topology in neural networks makes training difficult and cause it ot fall into poor local minima. Therefore, parameters should be appropriately initialised to alleviate such issues. The so-called "pre-training" schemes, in both generative and discriminative fashions, are reviewed. Another crucial issue in training is over-fitting. When a large number of parameters are introduced, DNN models are likely to over-fit to training data. Regularisation is a commonly used to reduce the risk of over-fitting, and improve generalisation. Several regularisation strategies are presented, such as parameter norm penalties, multi-task learning, dropout, early stopping, and data augmentation. The last section of this chapter reviews the methodology of network visualisation, which aims at analysing network behaviour, according to the visualisation and interpretation of hidden-layer representations.

Chapter 3

Speech Recognition and Deep Learning

The objective of automatic speech recognition is to generate the correct word sequence, or transcription, given a speech waveform. In this standard processing pipeline, the raw speech waveform is first processed to extract a sequence of acoustic features $\boldsymbol{x}_{1:T}$,

$$\boldsymbol{x}_{1:T} = \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T \tag{3.1}$$

where T is the length of sequence and $\boldsymbol{x}_t$ represents the feature vector at time t. The length of the sequence can vary from utterance to utterance. The ASR system yields the most likely decoding hypothesis $\hat{\boldsymbol{\omega}}$, according to Bayes' decision rule

$$\hat{\boldsymbol{\omega}} = \arg\max_{\boldsymbol{\omega}} P(\boldsymbol{\omega}|\boldsymbol{x}_{1:T}) \tag{3.2}$$

where $P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T})$ stands for the conditional probability of a hypothesis $\boldsymbol{\omega}$ given the features $\boldsymbol{x}_{1:T}$. This hypothesis can be expressed as a sequence of length M,

$$\boldsymbol{\omega}_{1:M} = \omega_1, \omega_2, \ldots, \omega_M \tag{3.3}$$

where $\omega_m \in \mathbb{V}$, representing an element, such as word, character or phoneme, depending on the task. $\mathbb{V}$ is referred to as the vocabulary, consisting of all possible element candidates that can be recognised.

This chapter reviews principle techniques in automatic speech recognition and deep learning methods for such sequence-to-sequence tasks. Both generative and discriminative methods to model $P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T})$ are described. Other key concepts in ASR are presented as well, including acoustic feature processing, speaker adaptation, and training criteria for speech recognition.

3.1 Acoustic Feature

The raw form of speech data is a continuous speech waveform. To effectively perform speech recognition, a speech waveform is processed and converted into a sequence of time-discrete parametric feature vectors $\boldsymbol{x}_{1:T}$, referred to as acoustic features. Acoustic features are designed to be compact and contain effective information for speech recognition. This section describes the basic mechanisms in acoustic feature extraction and processing.

3.1.1 Feature Extraction

Speech waveform is often considered to be quasi-stationary. It can thus be split into a sequence of discrete segments (usually overlapping), referred to as frames. This process is conduct using at a $10-15$ms frame rate and a $25-30$ms window size. Frames can be then further enhanced by a series of processes, such as pre-emphasis and windowing (Rabiner and Gold, 1975). By applying a discrete Fourier transform (DFT) on each frame, the frame representation in time domain is converted into a frequency-domain power spectrum.

Filter Bank

Filter bank analysis can be applied to the spectrum. The spectrum given by the DFT is evenly distributed in the frequency domain. However, frequencies across the audio spectrum are resolved a non-linear fashion by the human ear. Filter bank analysis can remove this kind of mismatch. The feature vectors after filter bank analysis are usually warped by $\log(\cdot)$ to rescale the dynamic range. The feature extracted by this process is referred to as the filter bank feature.

Cepstral Features

The representation obtain from filter bank analysis can be further processed to obtain cepstral features. There are two types of cepstral features widely used in speech recognition, mel-frequency cepstral and perceptual linear predictive coefficients.

Mel-frequency cepstral coefficients (Davis and Mermelstein, 1980) (MFCCs) use filters equally spaced on the Mel-scale to obtain non-linear resolution

$$\mathrm{Mel}(f) = 2595\log_{10}\left(1+\frac{f}{700}\right) \tag{3.4}$$

where f stands for a preceived frequency. It is equivalent to applying a set of triangular filters on the power spectrum. The discrete cosine transform (Chen et al., 1977) is then performed to yield cepstral coefficients, referred to as MFCC features.

Perceptual linear predictive (Hermansky, 1990) (PLP) coefficients are another type of cepstral features. It warps the power spectrum to the Bark scale,

$$\mathrm{Bark}(f) = 6\log\left(\left(\frac{f}{600}+1\right)^{0.5}+\frac{f}{600}\right). \tag{3.5}$$

The outputs are then processed by a non-linear transform based on equal-loudness and the intensity-loudness power law. Linear prediction (Atal and Hanauer, 1971) is finally performed to obtain the cepstral coefficients, referred to as PLP features.

Raw Waveform

A complete machine learning approach would operate directly onto raw waveforms with few manual designs. In speech recognition, raw features are waveform representations in time domain. Recent DNN-based methods (Sainath et al., 2015; Tüske et al., 2014) introduce special front-end modules to use raw waveforms as features that have achieved comparable state-of-the-art systems.

3.1.2 Feature Post-processing

The extracted feature can be further improved by a range of post-processing methods. Two common approaches are discussed here: dynamic features; and feature normalisation. Dynamic features enrich feathers with context information. Feature normalisation contributes to a robust and compact feature representation.

Dynamic Feature

Acoustic features are extracted in the frame level, which focus more on static information within the time window. To capture sequential properties, dynamical information in successive frames, such as time derivatives

$$\Delta^n \boldsymbol{x}_t = \frac{\sum_{i=1}^{n} (\boldsymbol{x}_{t+n} - \boldsymbol{x}_{t-n})}{2\sum_{j=1}^{n} j^2}, \tag{3.6}$$

can be appended as features, referred to as dynamic features. Commonly, the first-, second-, and third-order dynamic features are used to emphasise the correlation in successive frames (Furui, 1986).

Dynamic features are inconsistent with some statistical models that assumes that features are element-wisely independent. Linear projection methods, such as heteroscedastic linear discriminant analysis (Kumar and Andreou, 1998) (HLDA), are commonly used to resolve this issue, which project features to another space that minimises the correlation of different feature dimensions.

Feature Normalisation

Feature normalisation aims to remove the irrelevant information from the features. Additionally, it can be used to standardise the range of the features, which is practically important for DNN models. Acoustic features may include a range of irrelevant factors, such as accent, gender, environment noise and channel. Normalisation can reduce the impact of such irrelevant factors to features.

Traditional normalisation techniques include cepstral mean normalisation (Atal, 1974) (CMN), cepstral variance normalisation (Viikki and Laurila, 1998) (CVN) and vocal tract length normalisation (Lee and Rose, 1996).

3.2 Generative Model

Generative models are an important category of machine learning models, which are intuitively designed to randomly generate feature samples, given some class label. In the context of sequential data such as speech, a generative model specifies the joint probability distribution, $P(\boldsymbol{\omega}_{1:M}, \boldsymbol{x}_{1:T})$, over acoustic feature and word sequences. This joint distribution is then used to obtain the conditional distribution of label sequence, $P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T})$. Using Bayes' rule, the decision formula in Eq. 3.2 can be rewritten as

$$\begin{aligned}\hat{\boldsymbol{\omega}} &= \arg\max_{\boldsymbol{\omega}} \frac{p(\boldsymbol{\omega}, \boldsymbol{x}_{1:T})}{p(\boldsymbol{x}_{1:T})} \\ &= \arg\max_{\boldsymbol{\omega}} p(\boldsymbol{\omega}, \boldsymbol{x}_{1:T}). \qquad (3.7)\end{aligned}$$

The probability density function (PDF) of the feature sequence, $p(\boldsymbol{x}_{1:T})$, can be omitted, as $\boldsymbol{x}_{1:T}$ is independent of $\boldsymbol{\omega}$. The joint distribution is usually factorised into two components,

$$p(\boldsymbol{\omega}_{1:M}, \boldsymbol{x}_{1:T}) = p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M})P(\boldsymbol{\omega}_{1:M}), \qquad (3.8)$$

the likelihood $p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M})$, referred to as the acoustic model, and the prior $P(\boldsymbol{\omega}_{1:M})$, referred to as the language model. In general, these two models are separately trained. It can be difficult to model $p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M})$ directly, as the sequence lengths, M and T, are different. To address this issue, a sequence of discrete latent variables $\boldsymbol{\psi}_{1:T}$, referred to as the alignment, are introduced to handle the mapping between the sequences. Therefore,

$$p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M}) = \sum_{\boldsymbol{\psi}_{1:T} \in \boldsymbol{\Psi}^{T}_{\boldsymbol{\omega}_{1:M}}} p(\boldsymbol{x}_{1:T}|\boldsymbol{\psi}_{1:T})P(\boldsymbol{\psi}_{1:T}|\boldsymbol{\omega}_{1:M}) \qquad (3.9)$$

where $\boldsymbol{\Psi}^{T}_{\boldsymbol{\omega}_{1:M}}$ represents all valid alignments of length T for the M-length word sequence $\boldsymbol{\omega}_{1:M}$. A standard generative framework for speech recognition is illustrated in Figure 3.1. It consists of five principal components: front-end processing, acoustic model, language model, lexicon and decoder. First, the front-end processing extracts acoustic features from the waveforms. The decoder then uses the acoustic model, language model and lexicon to find the most likely decoding hypothesis. This section discusses

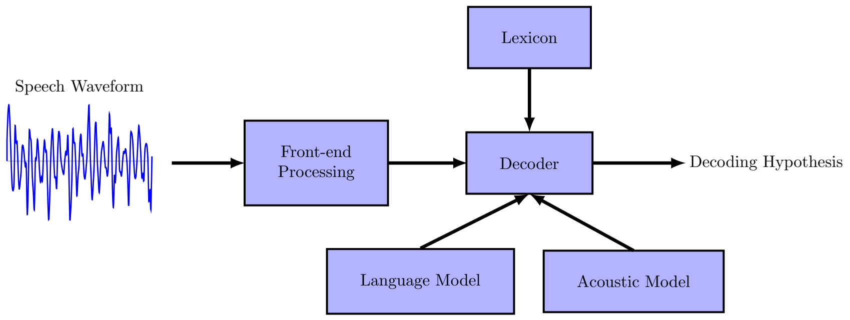


Fig. 3.1 Generative model for speech recognition.

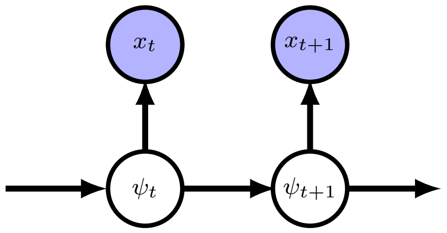


Fig. 3.2 Probabilistic graphical model for HMM. Unobserved variables are marked in white, and observed variables are in blue.

these components. The discussion focuses on the acoustic model $p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M})$, which is a key component and the main focus of this thesis. It also covers a brief description of the language model and decoding methods.

3.2.1 Hidden Markov Model

Hidden Markov Models (HMMs) (Rabiner, 1989) are generative models that have been used widely for acoustic models in speech recognition. The probabilistic graphical model for HMM is shown in Figure 3.2. Alignments are modelled as unobserved variables, referred to as hidden states, to generate feature vectors. At time t, the hidden state ψ_t (white circle) is unobserved, but the feature vector (blue circle) $\boldsymbol{x}_t$, which depends on the state, is observed. Each state ψ_t is associated with a PDF, $p(\boldsymbol{x}|\psi_t)$, to generate feature vector, referred to as the state emitting probability. The state at time $t+1$ is only dependent on that at time t, and the state transition is governed by $P(\psi_{t+1}|\psi_t)$, referred to as the state transition probability. This form of generative model makes two assumptions:

1. **conditional independence**: the probability of generating feature vector $\boldsymbol{x}_t$ only depends on the current state ψ_t;

2. **first-order Markov assumption**: the probability of state transition to state ψ_{t+1} is only dependent on the current state ψ_t.

In terms of probability distribution, these assumptions can be expressed as

$$P(\psi_{t+1}|\boldsymbol{\psi}_{1:t}, \boldsymbol{\omega}_{1:M}) \simeq P(\psi_{t+1}|\psi_t), \tag{3.10}$$

$$p(\boldsymbol{x}_t|\boldsymbol{x}_{1:t-1}, \boldsymbol{\psi}_{1:t}) \simeq p(\boldsymbol{x}_t|\psi_t). \tag{3.11}$$

Using these Markovian approximations, the acoustic model $p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M})$ can be rewritten as

$$p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M}) \simeq \sum_{\boldsymbol{\psi}_{1:T} \in \boldsymbol{\Psi}^T_{\boldsymbol{\omega}_{1:M}}} \left(\prod_{t=1}^{T} p(\boldsymbol{x}_t|\psi_t) P(\psi_t|\psi_{t-1}) \right). \tag{3.12}$$

In speech recognition, a single HMM is used to model each basic acoustic unit. Using the lexicon, HMMs can be composed together to represent words and sentences. Usually, each basic-unit HMM has a fixed number of hidden states, including an initial and an accepting non-emitting states (i.e. states that cannot generate feature vectors). The topology of a standard left-to-right[1] HMM with five states (three emitting and two non-emitting states) is illustrated in Figure 3.3. Two types of parameters are associated with the HMM model:

- **state transition probability:** $\{a_{ij}\}$
  The state transition probability a_{ij} is defined as

$$a_{ij} = P(\psi_{t+1} = j|\psi_t = i), \quad \sum_{j=1}^{N} a_{ij} = 1 \tag{3.13}$$

  where N is the total number of hidden states. Left-to-right HMMs restrict state transition to self loops or the next state; thus, many of a_{ij} are zeros.

[1]Here the term, left-to-right, means that the state transition cannot jump from a latter state to a previous one.

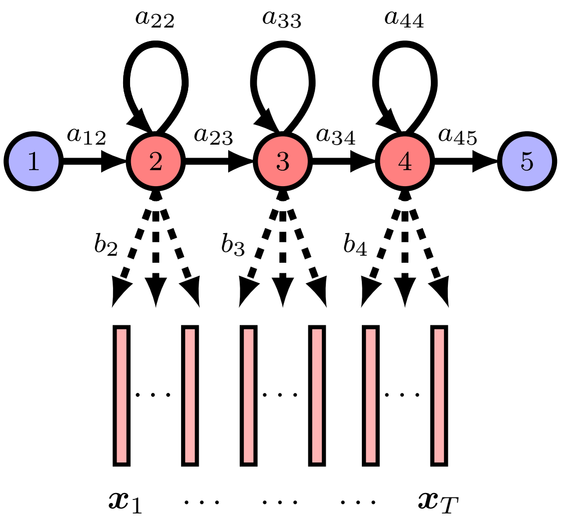


Fig. 3.3 A left-to-right HMM with five states. Emitting states are red circles, and non-emitting states are blue circles.

- **state emitting probability:** $\{b_j(\cdot)\}$

  This type of parameters consists of a series of PDFs, and each PDF defines the density of generating a feature vector on some state j,

$$b_j(\boldsymbol{x}) = p(\boldsymbol{x}|\psi = j). \tag{3.14}$$

  State emitting probabilities are at the heart of HMM models. For many years, the state emitting PDFs were modelled as Gaussian mixture models (GMMs),

$$b_j(\boldsymbol{x}) = \sum_{k=1}^{K} c_k^{(j)} \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}_k^{(j)}, \boldsymbol{\Sigma}_k^{(j)}\right) \tag{3.15}$$

  where K is the total number of Gaussian components; $c_k^{(j)}$, $\boldsymbol{\mu}_k^{(j)}$ and $\boldsymbol{\Sigma}_k^{(j)}$ are the GMM parameters, and $\mathcal{N}(\cdot;\cdot,\cdot)$ stands for the multivariate Gaussian PDF

$$\mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}_k^{(j)}, \boldsymbol{\Sigma}_k^{(j)}\right) = \frac{1}{\sqrt{(2\pi)^d \left|\boldsymbol{\Sigma}_k^{(j)}\right|}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{\mu}_k^{(j)}\right)^T \left(\boldsymbol{\Sigma}_k^{(j)}\right)^{-1} \left(\boldsymbol{x} - \boldsymbol{\mu}_k^{(j)}\right)\right\} \tag{3.16}$$

where d is the dimension of feature vector. To make Eq. 3.15 a valid PDF, it also requires

$$c_k^{(j)} \geq 0 \qquad \forall j, k; \tag{3.17}$$

$$\sum_{k=1}^{K} c_k^{(j)} = 1 \qquad \forall j. \tag{3.18}$$

HMMs with GMM emitting PDFs will be to as GMM-HMMs. A wide range of research has been conduct in this GMM framework (Rabiner, 1989). An alternative approach is to approximate the PDFs using neural networks, which is known as DNN-HMM hybrid systems. Recently, DNN-based systems have outperformed traditional GMM-HMMs in a variety of tasks (Dahl et al., 2012). Section 3.2.2 discusses the DNN-HMM methodology. For more details regarding GMM-HMMs, please refer to Rabiner (1989); Young et al. (2015).

Likelihood Calculation

The likelihood calculation is a basic problem to address for generative models. In statistics, a likelihood function is the probability assumed for those observed data given the parameter values. By substituting Eq. 3.13 and 3.14, the likelihood $p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M})$ in HMM is expressed as

$$p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M}) = \sum_{\phi_{1:T} \in \boldsymbol{\Psi}^T_{\boldsymbol{\omega}_{1:M}}} \left(\prod_{t=1}^{T} a_{\psi_t \psi_{t+1}} b_{\psi_t}(\boldsymbol{x}_t) \right). \tag{3.19}$$

Performing the calculation of the likelihood, the outer summation may take as many as $\mathcal{O}(N^T)$ steps. Thus, even for a small number of states and short sequences, this summation cannot be computed in practice. However, the Markovian approximations in HMM enable an efficient way to calculate the likelihood. The forward-backward algorithm (Baum et al., 1970) is a dynamic programming scheme that can break down the likelihood calculation for the complete sequence into a collection of sub-sequence calculations.

The forward probability, $\text{fwd}(t,j)$, is defined as the likelihood of a t-length sequence that stays at state j at time t. For emitting states, $2 \leq j \leq N-1$, the forward probability can be recursively computed via

$$\begin{aligned}\text{fwd}(t,j) &= p(\boldsymbol{x}_{1:t}, \psi_t = j) \\ &= \left(\sum_{i=2}^{N-1} \text{fwd}(t-1,i)a_{ij}\right) b_j(\boldsymbol{x}_t).\end{aligned} \tag{3.20}$$

The boundary conditions of this programming are given as

$$\text{fwd}(t,j) = \begin{cases} 1 & j=1 \quad t=0, \\ a_{1j}b_j(\boldsymbol{x}_t) & 2<j\leq N \quad t=1, \\ \sum_{i=2}^{N-1} \text{fwd}(T,i)a_{iN} & j=N \quad t=T. \end{cases} \tag{3.21}$$

As a consequence, the likelihood of the whole sequence is given by the forward probability of the non-emitting state N at time T

$$p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M}) = \text{fwd}(T,N). \tag{3.22}$$

Alternatively, the decomposition can begin with the end of the sequence. The backward probability, $\text{bwd}(t,j)$, is defined as the conditional probability of a partial sequence from time $t+1$ to the end, given $\psi_t = j$. Similarly, the calculation of $\text{bwd}(t,j)$ can be performed recursively,

$$\begin{aligned}\text{bwd}(t,j) &= p(\boldsymbol{x}_{t+1:T}|\psi_t = j) \\ &= \left(\sum_{i=2}^{N-2} a_{ji}\text{bwd}(t+1,i)\right) b_j(\boldsymbol{x}_t).\end{aligned} \tag{3.23}$$

And, the boundary conditions are

$$\text{bwd}(t,j) = \begin{cases} a_{jN} & 2\leq j<N \quad t=T, \\ \sum_{i=2}^{N-1} a_{1i}\text{bwd}(2,i) & j=1 \quad t=1. \end{cases} \tag{3.24}$$

Therefore, the likelihood $p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M})$ can be accumulated as the backward probability of the non-emitting state 1 at time 1,

$$p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega}_{1:M}) = \mathrm{bwd}(1,1). \tag{3.25}$$

Using either forward or backward probability, the likelihood calculation takes only $\mathcal{O}(NT)$ steps, which is efficiently performed in polynomial times.

Acoustic Units

HMMs have been introduced to model the basic acoustic units for speech recognition. Acoustic units can be designed at various levels. For simple tasks such as digit recognition, where the vocabulary size is relatively small, units can be specified at the word level, i.e. to train an HMM for each possible word. However, as the vocabulary increases, it is not feasible to robustly estimate word-level HMMs. For example, the typical vocabulary size in English varies between 40k and 100k. Because of the data sparsity, building an HMM for each possible word is not practical. Alternatively, the lexicon can be utilised to break down words into sub-word units, such as phonemes or graphemes. A typical English phoneme set contains 40 to 60 phones only, which is much smaller than the vocabulary size. Given phone-level HMMs, word or even sentence HMM models can be built via concatenating related phone HMMs according to the lexicon.

Two forms of phone-level acoustic units are used in speech recognition, context-independent and context-dependent phones. Context-independent (CI) phones, also known as monophones, use the original linguistic phonemes specified in the lexicon as acoustic units. The limitation of monophones is that context information from adjacent phones is not taken into account. The co-articulation phenomenon (Lee, 1988) states that the acoustic property of a particular phone can be considerately influenced by the preceding or following phones. Context-dependent (CD) phones, also known as triphones, are used to address this issue. A triphone is composed of one central phone and two context phones. For instance, /l-i+t/ stands for a triphone where the central phone is /i/, the preceding phone is /l/, and the following phone is /t/.

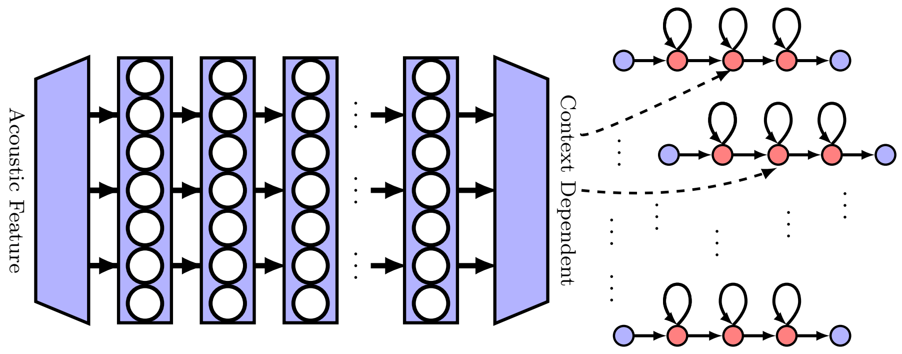


Fig. 3.4 DNN-HMM hybrid model.

Context information is explicitly introduced, which helps to alleviate the impact of co-articulation.

Data sparsity is still an issue in triphone HMMs. The total number of triphones is the cube of the number of monophones. In English, it can come up to 10^5. Additionally, many triphones may not exist in the training data. To mitigate this sparsity issue, clustering techniques are used. Clustering can either be performed on triphones (Hwang and Huang, 1993), or triphone HMM states (Young, 1993; Young et al., 1994). Triphone-state clustering is the most popular way in modern ASR systems and is commonly generated by decision trees (Young et al., 1994) with a range of phonetic questions. As a result, state emitting PDFs are tied and shared across acoustically similar HMM states.

3.2.2 Integrating Deep Learning

At the time of writing, neural network models have been used to significantly improve the performance of state-of-the-art ASR systems. Compared with conventional GMMs, DNNs are able to learn complicated, non-linear functions, which can better handle complex acoustic features. This section discusses the integration of deep learning into HMM models. Two forms of DNN integration, hybrid and tandem, and related extensions are presented.

Hybrid Systems

The DNN-HMM hybrid system (Bourlard and Morgan, 1994; Dahl et al., 2012) replaces the state emitting PDFs, $p(\boldsymbol{x}_t|\psi_t)$, by a deep neural network. DNNs are often trained in a discriminative manner, such as modelling target posterior $P(\psi_t|\boldsymbol{x}_t)$. This cannot directly represent a likelihood function $p(\boldsymbol{x}_t|\psi_t)$. By Bayes' rule, it can be converted to form a "pseudo-likelihood" equation,

$$p(\boldsymbol{x}_t|\psi_t) = \frac{P(\psi_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t)}{P(\psi_t)} \propto \frac{P(\psi_t|\boldsymbol{x}_t)}{P(\psi_t)} \tag{3.26}$$

where $P(\psi_t)$ is the prior probability distribution for the state generated as ψ_t, and $p(\boldsymbol{x}_t)$ can be omitted, as the feature vector $\boldsymbol{x}_t$ is independent of the target ψ_t. The state prior distribution $P(\psi_t)$ can be simply calculated from for frame state alignments of the training data. Figure 3.4 illustrates the DNN-HMM hybrid model. The DNN outputs are specified as context-dependent triphone states. The input feature consists of several successive frames, not a single speech frame, to reinforce context information.

Hybrid systems have been extensively used in state-of-the-art ASR systems (Dahl et al., 2012). Several approaches have been investigated to improve its performance. These include large-scale training (Kingsbury et al., 2012), discriminative training (Section 3.4), and speaker adaptation (Section 3.5).

Tandem System

In contrast to the hybrid system, the DNN-HMM tandem system (Grezl et al., 2007; Hermansky et al., 2000) is built on the GMM-HMM framework. The neural network for tandem system is used to extract features, rather than discriminative models. As shown in Figure 3.5, a bottleneck layer, consisting of much fewer units than other layers, is designed prior to the output layer. The output of this bottleneck layer, referred to as bottleneck features, is then concatenated with raw acoustic features, such as PLP, to train the GMM-HMM model. Bottleneck features are a compact, low-dimensional representation for raw acoustic features. They are discriminatingly trained to distinguish between phone states.

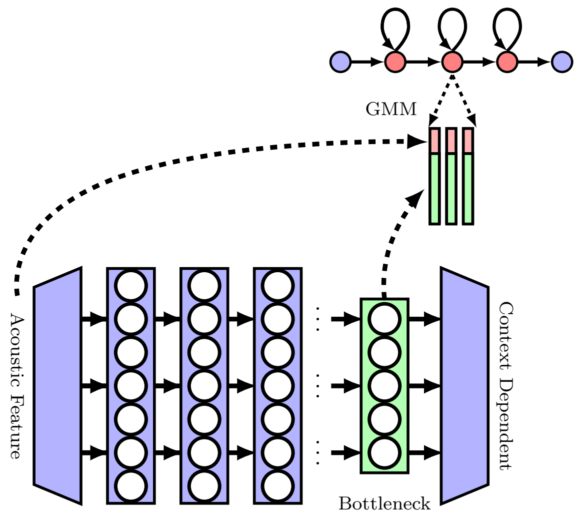


Fig. 3.5 DNN-HMM tandem model.

System Combination

System combination is a strategy that combines multiple systems to yield better performance than that of the constituent system alone. In speech recognition, joint decoding (Wang et al., 2015b) is a popular method to combine multiple acoustic models. It combines the log-likelihood from different systems via linear interpolation.

An example of joint decoding is shown in Figure 3.6. Hybrid and tandem systems are be combined via

$$\log p^{\mathrm{j}}(\boldsymbol{x}_t|\psi t) \propto \kappa^{\mathrm{hyb}} \log p^{\mathrm{hyb}}(\boldsymbol{x}_t|\psi t) + \kappa^{\mathrm{tan}} \log p^{\mathrm{tan}}(\boldsymbol{x}_t|\psi t) \tag{3.27}$$

where κ^{hyb} and κ^{tan} stand for the interpolation weights of hybrid and tandem systems respectively. The interpolation weights can be manually tuned. The combination result $\log p^{\mathrm{j}}(\boldsymbol{x}_t|\psi_t)$ is finally used as the combined score for acoustic model in decoding.

3.2.3 Language Modelling

Language models (LMs) play an important role in generative models for speech recognition. An LM specifies the prior probability of the word sequence, $P(\boldsymbol{\omega})$. Using the chain rule, $P(\boldsymbol{\omega}_{1:M})$ can be factorised as the product of conditional probabilities of

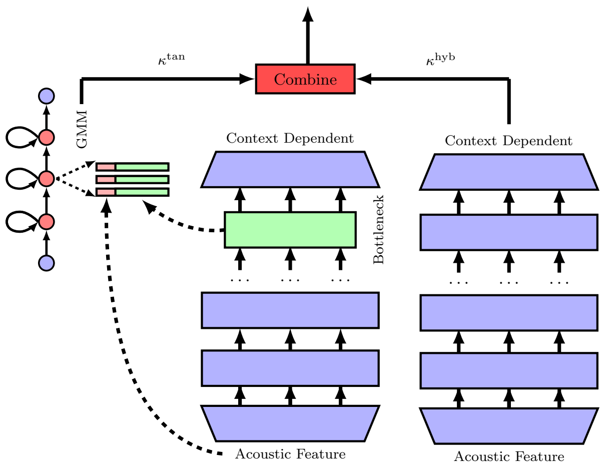


Fig. 3.6 Joint decoding for DNN-HMM hybrid and tandem systems.

each word given the word history,

$$P(\boldsymbol{\omega}_{1:M}) = \prod_{m=2}^{M} P(\omega_m|\boldsymbol{\omega}_{1:m-1}). \tag{3.28}$$

There are two special word symbol introduced: the sentence start symbol $\langle \text{s} \rangle$,

$$P(\omega_1) = P(\langle \text{s} \rangle) = 1, \tag{3.29}$$

and the sentence end symbol $\langle /\text{s} \rangle$. Language models are trained on data with text only to estimate the conditional distribution $P(\omega_m|\boldsymbol{\omega}_{1:m-1})$. To directly model $P(\omega_m|\boldsymbol{\omega}_{1:m-1})$ can be impractical due to data sparsity. The word history $\boldsymbol{\omega}_{1:m-1}$ needs to be approximated to address this issue. There are two popular strategies of approximation,

- **Markovian:** The full word history is approximated by a fixed-length history of $n-1$ words

$$P(\omega_m|\boldsymbol{\omega}_{1:m-1}) \simeq P(\omega_m|\boldsymbol{\omega}_{m-n+1:m-1}). \tag{3.30}$$

  This simplifies the language model to an $(n-1)$th-order Markov chain. This approximation can be implemented via traditional n-gram and feed-forward neural network LMs (Bengio et al., 2003).

- **Non-Markovian:** A fixed-length history window may not capture enough context information. An alternative method is to introduce a fixed-length history vector to represent the complete word history

$$P(\omega_m|\boldsymbol{\omega}_{1:m-1}) \simeq P(\omega_m|\boldsymbol{v}_{m-1}) \tag{3.31}$$

  where $\boldsymbol{v}_{m-1}$ is a fixed-length, continuous vector which can explicitly memorise all useful past information. Recurrent neural network LM (Mikolov et al., 2010) is based on this concept. The history vector is specified by the recurrent units. In addition, gated RNNs such as LSTMs have also been used for language models (Sundermeyer et al., 2012, 2015; Sutskever et al., 2014).

N-gram Language Model

N-gram LMs directly estimate $P(\omega_m|\boldsymbol{\omega}_{m-n+1:m-1})$ on the maximum likelihood criterion,

$$P(\omega_m|\boldsymbol{\omega}_{m-n+1:m-1}) = \frac{\#(\boldsymbol{\omega}_{m-n+1:m})}{\#(\boldsymbol{\omega}_{m-n+1:m-1})} \tag{3.32}$$

where $\#(\cdot)$ stands for the total number of an n-gram in the training corpus. The term n is referred to as the order of the n-gram LM. In ASR systems, bi-gram (2-gram), tri-gram (3-gram) and 4-gram LMs are often used. The simple expression in Eq. 3.32 makes it possible to efficiently train n-gram LMs on very large corpora.

There are several extensions to improve n-gram LMs. Smoothing techniques adjusts the distribution for non-zero and robust probability for all n-grams, such as Katz smoothing (Katz, 1987), absolute discounting (Ney et al., 1994) and Kneser-Ney smoothing (Kneser and Ney, 1995). LM adaptation (Bellegarda, 2004; Gildea and Hofmann, 1999) combines multiple LMs to resolve the issue of text mismatches in different topics.

3.2.4 Decoding

Decoding is a key module in speech recognition system. Given a sequence of acoustic features $\boldsymbol{x}_{1:T}$, the decoder searches for the "optimal" word sequence $\hat{\omega}$ using the acoustic model, language model and lexicon,

$$\begin{aligned}\hat{\boldsymbol{\omega}} &= \arg\max_{\boldsymbol{\omega}} p(\boldsymbol{x}_{1:T}|\boldsymbol{\omega})P(\boldsymbol{\omega}) \\ &= \arg\max_{\boldsymbol{\omega}} P(\boldsymbol{\omega}) \sum_{\boldsymbol{\psi}_{1:T}} p(\boldsymbol{x}_{1:T}|\boldsymbol{\psi}_{1:T})P(\boldsymbol{\psi}_{1:T}|\boldsymbol{\omega}). \end{aligned} \tag{3.33}$$

A summation over exponentially many possible state sequences $\boldsymbol{\psi}_{1:T}$ is required, which is not feasible in practice. The sum in Eq. 3.33 can be approximated by a $\max(\cdot)$ operator. Instead of the optimal word sequence, the approximated decoding algorithm searches for the word sequence corresponding to the optimal state sequence

$$\hat{\boldsymbol{\omega}} \simeq \arg\max_{\boldsymbol{\omega}} \left(P(\boldsymbol{\omega}) \max_{\boldsymbol{\psi}_{1:T}} p(\boldsymbol{x}_{1:T}|\boldsymbol{\psi}_{1:T})P(\boldsymbol{\psi}_{1:T}|\boldsymbol{\omega}) \right). \tag{3.34}$$

This search process can be viewed as finding the best path though a directed graph, referred to as decoding network, constructed from the acoustic model, language model and lexicon. It can be performed in either a breadth-first or depth-first fashion (Aubert, 2002). The Viterbi algorithm (Forney, 1973) is is a dynamic programming algorithm based on a breadth-first concept, which have been extensively used in ASR systems. In detail, given the acoustic feature sequence $\boldsymbol{x}_{1:T}$, the search process is based on computing the following term[2]

$$\text{lik}(t,j) = \max_{\boldsymbol{\psi}_{1:t-1}} p(\boldsymbol{x}_{1:t}, \boldsymbol{\psi}_{1:t-1}, \psi_t = j), \tag{3.35}$$

which represents the maximum likelihood of the partial "best" state sequence that stays on state j at time t. According to the Markov property, this term can be recursively accumulated by

$$\text{lik}(t,j) = \max_{i} \left\{ \text{lik}(t-1,i) a_{ij} b_j(\boldsymbol{x}_t) \right\}, \tag{3.36}$$

[2]To simplify the discussion, a simple form without the language model is discussed here. For more details regarding the decoding algorithm, please refer to Gales and Young (2008).

with boundary conditions as follows

$$\mathrm{lik}(1,1) = 1, \tag{3.37}$$

$$\mathrm{lik}(2,j) = a_{1j} b_j(\boldsymbol{x}_1) \quad \forall j. \tag{3.38}$$

The optimal state sequence, $\hat{\boldsymbol{\psi}}_{1:T}$, can thus be retrieved in a recursion

$$\hat{\psi}_t = \arg\max_j \left\{ \mathrm{lik}(t,j) a_{j\hat{\psi}_{t+1}} \right\} \quad 1 \le t \le T-1, \tag{3.39}$$

with the boundary condition at time T

$$\hat{\psi}_T = \arg\max_i \left\{ \mathrm{lik}(T,i) a_{iN} \right\}. \tag{3.40}$$

It takes $\mathcal{O}(N^2T)$ steps to complete the searching algorithm. In practice, the implementation of the Viterbi algorithm can be very complex due to the HMM topology, language model constraints, context-dependent acoustic units (Gales and Young, 2008). This algorithm can either be implemented on a dynamic decoding network (Odell et al., 1994; Ortmanns et al., 1997) (i.e. constructing the network while decoding) or on a static network such as a weighted finite-state transducer (Mohri et al., 2002; Povey et al., 2011). The Viterbi algorithm is based on Markov properties. When non-Markovian modules such as RNN LMs are used, it is no longer possible to simply perform dynamic programming for decoding. Approximations for the RNN LMs (Liu et al., 2014, 2015) have been investigated to use non-Markovian LMs in the Viterbi decoding framework.

The hypothesis $\hat{\boldsymbol{\omega}}$ given by the Viterbi algorithm is the most probable word sequence with minimum error rate at the sentence level. However, the recognition performance are usually evaluated at the word level (Section 3.6). The output of Viterbi decoding is sub-optimal for word error rate. The Mimimum Bayes' risk decoding, such as confusion network decoding (Evermann and Woodland, 2000; Mangu et al., 2000), is designed to address this mismatch.

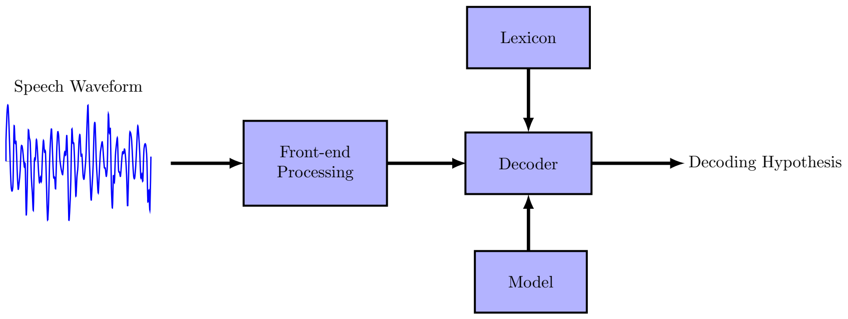


Fig. 3.7 Discriminative model for speech recognition.

3.2.5 Lexicon

The lexicon, also known as the dictionary, is used in modern ASR systems to map words, or characters, into sub-word units. This mapping allows acoustic-model parameters to be robustly estimated, and unseen words in the training data to be modelled. To build the lexicon, a standard approach is to map words into phones. The word punctuation can be generated manually by experts or automatically by grapheme-to-phoneme algorithms (Bisani and Ney, 2008). For low-resource languages, it may be impractical to manually generate a phonetic lexicon. An alternative approach is to build a graphemic lexicon (Gales et al., 2015b; Kanthak and Ney, 2002; Killer et al., 2003), where the "pronunciation" for a word is defined by the letters forming that word. The graphemic lexicon enables ASR systems to be build with no phonetic information provided.

3.3 Discriminative Model

Discriminative models in speech recognition, sometimes known as end-to-end models, directly examine the posterior of word sequence $P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T})$, which is closely related to Bayes' decision rule (Eq. 3.2). The framework of discriminative model for ASR is illustrated in Figure 3.7. In contrast to the generative framework, separate acoustic and language models are not introduced. Instead, a single model is used to perform decoding. This section discusses discriminative models using deep learning. Particularly, recurrent neural networks are utilised to model such sequential data. The discussion includes connectionist temporal classification, encode-decode models, and attention-based models.

3.3.1 Connectionist Temporal Classification

One challenge in tasks such as speech recognition is the unsegmented labelling nature, that is, the length of the feature sequence $\boldsymbol{x}_{1:T}$ and that of the word sequence $\boldsymbol{\omega}_{1:M}$ can be different. Similar to the design in HMMs, the connectionist temporal classification (Graves et al., 2006) (CTC) model introduces a special type of alignment sequence to deal with the unsegmented issue. In the CTC framework, each alignment sequence $\boldsymbol{\psi}_{1:T}$ uniquely identifies a word sequence $\boldsymbol{\omega}_{1:M}$,

$$P(\boldsymbol{\omega}_{1:M}|\boldsymbol{\psi}_{1:T}) = 1, \quad \forall \boldsymbol{\psi}_{1:T} \in \boldsymbol{\Psi}^{T}_{\boldsymbol{\omega}_{1:M}} \tag{3.41}$$

where $\boldsymbol{\Psi}^{T}_{\boldsymbol{\omega}_{1:M}}$ consists of all possible alignment sequences that are of length T and identifies $\boldsymbol{\omega}_{1:M}$. In this way, the posterior of word sequence $P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T})$ can be rewritten as

$$\begin{aligned} P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T}) &= \sum_{\boldsymbol{\psi}_{1:T} \in \boldsymbol{\Psi}^{T}_{\boldsymbol{\omega}_{1:M}}} P(\boldsymbol{\omega}_{1:M}|\boldsymbol{\psi}_{1:T}) P(\boldsymbol{\psi}_{1:T}|\boldsymbol{x}_{1:T}) \\ &= \sum_{\boldsymbol{\psi}_{1:T} \in \boldsymbol{\Psi}^{T}_{\boldsymbol{\omega}_{1:M}}} P(\boldsymbol{\psi}_{1:T}|\boldsymbol{x}_{1:T}). \end{aligned} \tag{3.42}$$

In a similar fashion as the HMM, the CTC model converts the original unsegmented labelling task to a segmented one. In practice, the label set may include a set of characters, graphemes or phonemes. A special label ε is also added to the label set and referred to as a blank label. The CTC blank label functions similarly to the silence unit in HMMs that can separate successive words in speech.

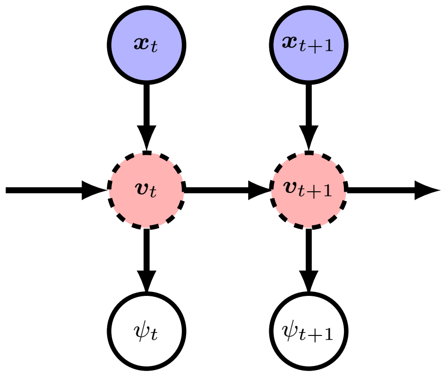


Fig. 3.8 Connectionist temporal classification.

Figure 3.8 illustrates the framework for CTC models[3]. CTC approximates and factorises the conditional probability of alignment sequence, $P(\boldsymbol{\psi}_{1:T}|\boldsymbol{x}_{1:T})$, as

$$\begin{aligned} P(\boldsymbol{\psi}_{1:T}|\boldsymbol{x}_{1:T}) &\simeq \prod_{t=1}^{T} P(\psi_t|\boldsymbol{x}_{1:t}) \\ &\simeq \prod_{t=1}^{T} P(\psi_t|\boldsymbol{x}_t, \boldsymbol{v}_{t-1}) \\ &\simeq \prod_{t=1}^{T} P(\psi_t|\boldsymbol{v}_t) \end{aligned} \tag{3.43}$$

where $\boldsymbol{v}_t$ stands for a time-dependent and fixed-length vector, encoding information in $\boldsymbol{x}_{1:t}$. A recurrent neural network, referred to as the CTC network, is used to do the sequence-to-sequence labelling. At time utterance t, it depicts the distribution $P(\psi|\boldsymbol{v}_t)$, and $\boldsymbol{v}_t$ is the output of recurrent units at time t. By substituting Eq. 3.43 to Eq. 3.42,

$$P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T}) \simeq \sum_{\boldsymbol{\psi}_{1:T} \in \boldsymbol{\Psi}^{T}_{\boldsymbol{\omega}_{1:M}}} \prod_{t=1}^{T} P(\psi_t|\boldsymbol{v}_t). \tag{3.44}$$

[3] Note that this diagram does not illustrate the exact probabilistic graphical model for the CTC. The symbols $\boldsymbol{v}_t$ and $\boldsymbol{v}_{t+1}$ in dotted circles are deterministic, not random variables. This thesis introduces this type of diagram to emphasis the relationship across observed variables (in blue circles), unobserved variables (in white circles), and DNN hidden units (in dotted circles). Figure 3.9 and 3.10 in the following discussion are designed using the same concept.

This form allows the conditional probability $P(\boldsymbol{\omega}_{1:M}|\boldsymbol{\psi}_{1:T})$ to be efficiently calculated via dynamic programming, which is similar to the forward-backward algorithm (Section 3.2.1).

Decoding in CTC models can be performed using a best-path approximation, which is similar to the Viterbi algorithm (Section 3.2.4). Given the feature sequence $\boldsymbol{x}_{1:T}$, the most likely word sequence is given by the most probable label sequence

$$\hat{\boldsymbol{\omega}} = \mathcal{B}(\hat{\boldsymbol{\psi}}_{1:T}), \quad \hat{\boldsymbol{\psi}}_{1:T} = \underset{\boldsymbol{\psi}_{1:T}}{\arg\max}\, P(\boldsymbol{\psi}_{1:T}|\boldsymbol{x}_{1:T}) \tag{3.45}$$

where $\mathcal{B}(\cdot)$ maps an alignment sequence to the corresponding word sequence. The best-path approximation cannot guarantee to find the most probable labelling. Alternatively, Graves et al. (2006) discussed the prefix search decoding method, which performs decoding with a prefix tree to efficiently accumulate the statistics of label prefixes. Given enough time, prefix search decoding can find the most probable labelling.

3.3.2 Encoder-Decoder Model

The CTC model still relies on an alignment sequence, $\boldsymbol{\psi}_{1:T}$, to deal with the length mismatch between $\boldsymbol{\omega}_{1:M}$ and $\boldsymbol{x}_{1:T}$. Encoder-decoder models (Cho et al., 2014; Sutskever et al., 2014) are a type of recurrent neural network model, which can directly compute the posterior of word sequence $P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T})$ without assuming alignments. Instead, an encoding mechanism is introduced to project the feature sequence $\boldsymbol{x}_{1:T}$ to a fixed-length vector representation $\boldsymbol{c}$. This vector $\boldsymbol{c}$ is referred to as the context vector. It is assumed to contain sufficient information to guide the generation of the word sequence $\boldsymbol{\omega}_{1:M}$.

The topology of encoder-decoder RNN is illustrated in Figure 3.9. It explicitly consists of two modules: the encoder; and the decoder. The encoder module processes the input features, and the output of the encoder recurrent units at time T, $\boldsymbol{v}_T$, is used as the context vector

$$\boldsymbol{c} = \boldsymbol{v}_T. \tag{3.46}$$

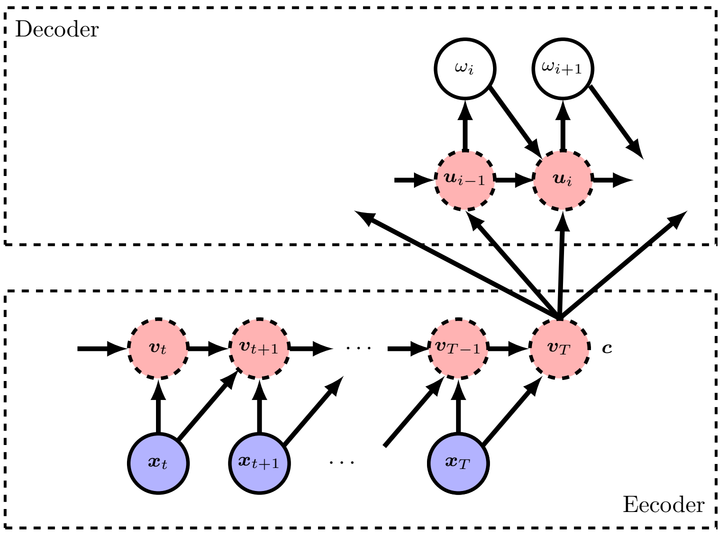


Fig. 3.9 Encoder-decoder RNN.

This context vector $\boldsymbol{c}$ is then used by the decoder module to guide the word sequence generation. Formally, the conditional probability $P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T})$ is factorised as

$$\begin{aligned} P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T}) &= \prod_{i=1}^{M} P(\omega_i|\boldsymbol{\omega}_{1:i-1}, \boldsymbol{x}_{1:T}) \\ &\simeq \prod_{i=1}^{M} P(\omega_i|\omega_{i-1}, \boldsymbol{u}_{i-2}, \boldsymbol{c}) \end{aligned} \tag{3.47}$$

where $\boldsymbol{u}_i$ stands for the output of decoder recurrent units.

The encoder-decoder model can be implemented in a range of configurations. For example, the encoder component can either be a RNN or a feed-forward network to learn a compact context vector (Lu et al., 2015). In speech, the feature sequence is normally much longer than the word sequence, i.e. $T \gg M$. Down-sampling on the feature sequences can be applied to mitigate the issue of long-time dependency for the encoder RNN.

3.3.3 Attention-based Model

The encoder-decoder model uses a static context vector to represent the whole feature sequence. This context vector is shared across time when generating the word sequence. This design restricts the modelling flexibility with temporary information. In speech

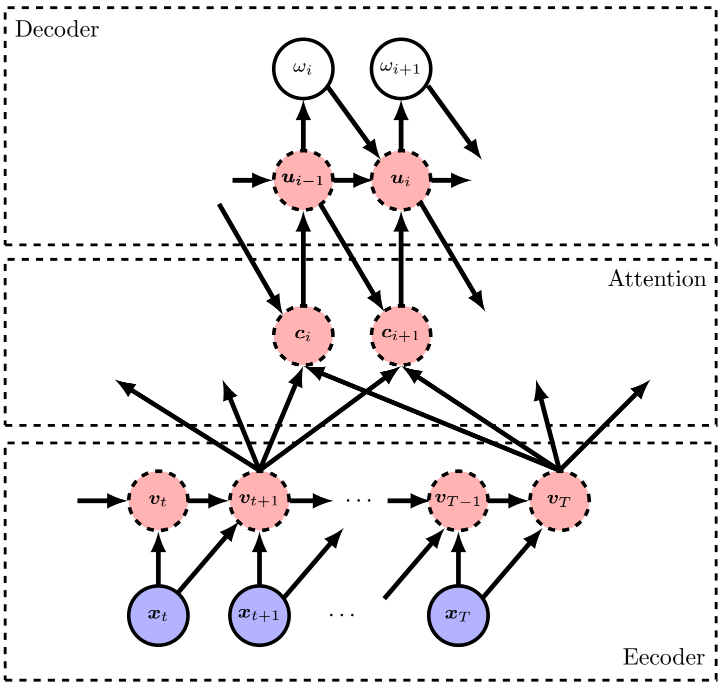


Fig. 3.10 Encoder-decoder RNN with attention model.

recognition, to generate a particular pronunciation is related to frames in a short period of time and not the complete waveform. The attention-based model (Chan et al., 2015; Chorowski et al., 2015) extends the encoder-decoder model with an attention layer to achieve a, time-variant, dynamic context vector. As shown in Figure 3.10, an attention layer is introduced at an intermediate stage between decoder and encoder, yielding a dynamic context vector $\boldsymbol{c}_i$. This context vector is defined as a linear interpolation of the outputs of encoder recurrent units $\boldsymbol{v}_{1:T}$

$$\boldsymbol{c}_i = \sum_t \alpha_{it} \boldsymbol{v}_t \tag{3.48}$$

where interpolation weights, $\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{iT}$, are dynamically determined via

$$\alpha_{it} = \frac{\exp\left(s\left(\boldsymbol{v}_t, \boldsymbol{u}_{i-2}\right)\right)}{\sum_{k=1}^{T} \exp\left(s\left(\boldsymbol{v}_k, \boldsymbol{u}_{i-2}\right)\right)}. \tag{3.49}$$

The function $s\left(\boldsymbol{v}_t, \boldsymbol{u}_{i-2}\right)$ is referred to as attention score, measuring how well position t in input matches position $i-1$ in output. The conditional probability of word sequence can then be factored as

$$
\begin{aligned}
P(\boldsymbol{\omega}_{1:M}|\boldsymbol{x}_{1:T}) &\simeq \prod_{i=1}^{M} P(\omega_i|\omega_{i-1},\boldsymbol{u}_{i-2},\boldsymbol{c}_i) \\
&\simeq \prod_{i=1}^{M} P(\omega_i|\boldsymbol{u}_{i-1}).
\end{aligned} \tag{3.50}
$$

The attention mechanism makes context vector $\boldsymbol{c}_t$ dynamically capture specific frames when generating different words.

3.4 Training Criteria for Speech Recognition

As discussed in Section 2.3.1, a training criterion measures how well a model with parameters $\boldsymbol{\theta}$ performs mapping features to expected targets for the training data $\mathbb{D}$. The definition of "well" depends on the task. This section describes training criteria for speech recognition. In ASR, the training data consists of acoustic feature utterances with corresponding transcription references

$$
\mathbb{D} = \left\{(\boldsymbol{x}^{(1)}_{1:T_1},\boldsymbol{\omega}^{(1)}_{\text{ref}}),(\boldsymbol{x}^{(2)}_{1:T_2},\boldsymbol{\omega}^{(2)}_{\text{ref}}),\ldots,(\boldsymbol{x}^{(U)}_{1:T_U},\boldsymbol{\omega}^{(U)}_{\text{ref}})\right\} \tag{3.51}
$$

where $(\boldsymbol{x}^{(u)}_{1:T_u},\boldsymbol{\omega}^{(u)}_{\text{ref}})$ stands for one training instance. Discussion in this section is split into two parts. The maximum likelihood estimation for generative model is presented at first. The second part describes discriminative training criteria for both generative and discriminative models.

3.4.1 Maximum Likelihood Estimation

For generative models, such as HMMs, the maximum likelihood (ML) training criterion is based on the concept of maximising the likelihood of generating data (i.e. feature sequence) given the class label (i.e. reference transcription). It can be expressed as

$$
\mathcal{L}^{\text{ml}}(\boldsymbol{\theta};\mathbb{D}) = -\frac{1}{U}\sum_{u=1}^{U}\log p(\boldsymbol{x}^{(u)}_{1:T_u}|\boldsymbol{\omega}^{(u)}_{\text{ref}}). \tag{3.52}
$$

For HMM models, the likelihood term $p(\boldsymbol{x}_{1:T_u}^{(u)}|\boldsymbol{\omega}_{\text{ref}}^{(u)})$ can be calculated via the forward-backward algorithm described in Section 3.2.1. The appropriateness of ML estimation needs to specify a number of requirements, in particular, training data sufficiency and model-correctness. In general, neither of them can be satisfied for speech data (Brown, 1987). Alternative methods, such as discriminative training criteria, have been proposed to overcome such mismatches.

3.4.2 Discriminative Training Criteria

Discriminative training criteria are designed to optimise the model in a fashion related to the decision rule. This objective can be expressed in terms of the posterior of word sequence $P(\boldsymbol{\omega}_{\text{ref}}^{(u)}|\boldsymbol{x}_{1:T_u}^{(u)})$. For discriminative models, this conditional probability is explicitly modelled. However, generative models define the likelihood $P(\boldsymbol{x}_{1:T_u}^{(u)}|\boldsymbol{\omega}_{\text{ref}}^{(u)})$ and the prior distribution $P(\boldsymbol{\omega}_{\text{ref}}^{(u)})$, instead of the posterior $P(\boldsymbol{\omega}_{\text{ref}}^{(u)}|\boldsymbol{x}_{1:T_u}^{(u)})$. To train generative models on discriminative criteria, a common strategy is to convert the posterior in terms of prior and likelihood according to Bayes' rule,

$$P(\boldsymbol{\omega}_{\text{ref}}^{(u)}|\boldsymbol{x}_{1:T_u}^{(u)}) = \frac{p(\boldsymbol{x}_{1:T_u}^{(u)}|\boldsymbol{\omega}_{\text{ref}}^{(u)})P(\boldsymbol{\omega}_{\text{ref}}^{(u)})}{\sum\limits_{\boldsymbol{\omega}\in\boldsymbol{\Omega}} p(\boldsymbol{x}_{1:T_u}^{(u)}|\boldsymbol{\omega})P(\boldsymbol{\omega})} \tag{3.53}$$

where $\boldsymbol{\Omega}$ denotes the hypothesis space containing all possible word sequences. In practice, it is infeasible to explore the complete hypothesis space $\boldsymbol{\Omega}$ to compute the exact denominator of Eq. 3.53. The denominator calculation is usually approximated by a smaller candidate set of possible word sequences, such as n-best lists, or decoding lattices, generated by a sensible recognition system. Alternatively, the lattice-free approach (Povey et al., 2016) can be applied to some discriminative criteria, such as the maximum mutual information criterion, which can avoid the requirement of decoding lattices. The term $P(\boldsymbol{\omega})$ is usually specified by a separate language model, which is not trained in conjunction with this generative model.

This section discusses three forms of discriminative training criteria: maximum mutual information, minimum classification error, and minimum Bayes' risk.

Maximum Mutual Information

Maximum mutual information (Bahl et al., 1986; Povey, 2004) (MMI) is a discriminative training criterion closely related to the classification performance. The aim of MMI is to minimise[4] the negative mutual information between the word sequence $\boldsymbol{\omega}$ and the information extracted by a recogniser from the the associated feature sequence[5] $\boldsymbol{x}_{:}$, $\mathcal{I}(\boldsymbol{\omega}, \boldsymbol{x}_{:})$. Because the joint distribution of the word-sequences and observations is unknown, it is approximated by the empirical distributions over the training data. This can be expressed as (Brown, 1987)

$$\mathcal{I}(\boldsymbol{\omega}, \boldsymbol{x}_{:}) \simeq -\frac{1}{U}\sum_{u=1}^{U} \log \frac{p(\boldsymbol{\omega}_{\text{ref}}^{(u)}, \boldsymbol{x}_{1:T_u}^{(u)})}{P(\boldsymbol{\omega}_{\text{ref}}^{(u)})p(\boldsymbol{x}_{1:T_u}^{(u)})}. \tag{3.54}$$

Since the language model $P(\boldsymbol{\omega}_{\text{ref}}^{(u)})$ is fixed, this is equivalent to minimise the negative average log-posterior probability of the correct word sequence. The MMI criterion can be expressed as

$$\mathcal{L}^{\text{mmi}}(\boldsymbol{\theta}, \mathbb{D}) = -\frac{1}{U}\sum_{u=1}^{U} \log P(\boldsymbol{\omega}_{\text{ref}}^{(u)} | \boldsymbol{x}_{1:T_u}^{(u)}). \tag{3.55}$$

When this form of training criterion is used with discriminative models, it is also known as the conditional maximum likelihood criterion.

Minimum Classification Error

Minimum classification error (Juang et al., 1997) (MCE) aims at minimising the difference between the log-likelihood of the reference $\boldsymbol{\omega}_{\text{ref}}^{(u)}$ and that of other competing decoding hypothesis $\boldsymbol{\omega}$

$$\mathcal{L}^{\text{mce}}(\boldsymbol{\theta}, \mathbb{D}) = \frac{1}{U}\sum_{u=1}^{U} \left(1 + \left(\frac{P(\boldsymbol{\omega}_{\text{ref}}^{(u)} | \boldsymbol{x}_{1:T_u}^{(u)})}{\sum\limits_{\boldsymbol{\omega} \neq \boldsymbol{\omega}_{\text{ref}}^{(u)}} P(\boldsymbol{\omega} | \boldsymbol{x}_{1:T_u}^{(u)})} \right)^{\xi} \right)^{-1} \tag{3.56}$$

[4]Notice that this criterion is defined as a negative MMI to keep a consistent "minimisation" form for training criterion.

[5]The notation $\boldsymbol{x}_{:}$ is defined as a feature sequence of arbitrary length.

where ξ is a hyper-parameter for smoothness. There are two major differences between the MMI and MCE criteria. One is that the denominator term of MCE excludes the reference sequence $\boldsymbol{\omega}_{\text{ref}}^{(u)}$. The other is that the MCE criterion smooths the posterior via a sigmoid function. When $\xi = 1$, the MCE criterion is given as

$$\mathcal{L}^{\text{mce}}(\boldsymbol{\theta}, \mathbb{D}) = 1 - \frac{1}{U}\sum_{u=1}^{U} P(\boldsymbol{\omega}_{\text{ref}}^{(u)} | \boldsymbol{x}_{1:T_u}^{(u)}). \tag{3.57}$$

This is a special case of minimum Bayes' risk criteria discussed as follows.

Minimum Bayes' Risk

Minimum Bayes' risk (Byrne, 2006; Kaiser et al., 2000) (MBR) aims at minimising the expectation of a particular form of loss during recognition,

$$\mathcal{L}^{\text{mbr}}(\boldsymbol{\theta}; \mathbb{D}) = -\frac{1}{U}\sum_{u=1}^{U}\sum_{\boldsymbol{\omega}\in\boldsymbol{\Omega}} P(\boldsymbol{\omega}|\boldsymbol{x}_{1:T_u}^{(u)})\mathcal{D}(\boldsymbol{\omega}, \boldsymbol{\omega}_{\text{ref}}^{(u)}) \tag{3.58}$$

where $\mathcal{D}(\boldsymbol{\omega}, \boldsymbol{\omega}_{\text{ref}}^{(u)})$ defines an appropriate loss function measuring the mismatch between $\boldsymbol{\omega}$ and $\boldsymbol{\omega}_{\text{ref}}$. The loss can be defined in a range of levels,

- **Sentence** The sentence loss, also known as "0/1 loss", is defined as

$$\mathcal{D}(\boldsymbol{\omega}, \boldsymbol{\omega}_{\text{ref}}^{(u)}) = \begin{cases} 0 & \text{if } \boldsymbol{\omega} = \boldsymbol{\omega}_{\text{ref}}^{(u)}, \\ 1 & \text{otherwise.} \end{cases} \tag{3.59}$$

  It gives the same form as the MCE criterion with $\xi = 1$.

- **Word** The word-level loss is directly related to the exception of word error rate, which is the ASR performance metric. This loss is commonly defined as word-level Levenshtein distance between $\boldsymbol{\omega}$ and $\boldsymbol{\omega}_{\text{ref}}$. The MBR using such loss is referred to as the minimum word error (MWE) criterion (Mangu et al., 2000).

- **Phone/State** The phone-level loss is defined as the phone Levenshtein distance between the two word sequences. The MBR using phone-level losses, known as minimum phone error (MPE) criterion (Povey, 2004), has been broadly used in speech recognition. Rather than phone-level loss, state-level loss have also been investigated, which yields state MBR (sMBR) criterion (Su et al., 2013; Zheng and Stolcke, 2005).

- **Frame** The frame-level loss is defined as the Hamming distance, which computes the number of frames with incorrect phone labels. The MBR with this frame loss is referred to as the minimum phone frame error (MPFE) criterion (Zheng and Stolcke, 2005).

3.5 Adaptation

A fundamental assumption in machine learning algorithms is that the training and test data have the same distribution; otherwise, their mismatch is likely to degrade the performance of related systems. In speech recognition, unseen speakers or environment conditions often exist, which may be poorly presented in the training data. One solution to address the data-mismatch issue is adaptation. Adaptation allows a small amount of data from an unseen speaker to be used to transform a model to more closely match that speaker. It can be used either in the training phase to induce a more robust model, or in the evaluation phase to reduce the recognition errors. This section describes the adaptation methods for neural networks. The methodology of adaptation can be roughly categorised into two groups: feature-based adaptation and model-based adaptation. The feature-based approaches only depend on the acoustic features and aim at compensating the features to a more compact representation. Alternatively, the model-based approaches change the model parameters to achieve the compensation.

This thesis presents adaptation schemes in the context of speaker adaptation. Speaker adaptation in speech recognition aims at adapting original models to handle acoustic variations, such as accent and gender, across different speakers. The unadapted neural network is referred to as a speaker-independent (SI) model. The adapted model is referred to as a speaker-dependent (SD) one.

3.5.1 Conservative Training

A simple strategy of adaptation is to re-estimate all parameters on the adaptation data starting from a SI model. However, a direct parameter re-estimation with no constraints would over-fit the limited adaptation data, resulting in a performance degradation. To prevent this over-fitting issue, conservative training (Yu et al., 2013) have been proposed, which modifies training criteria for adaptation. Regularisation is introduced to adaptation criteria, restricting the adapted SD model not to be "far away" from the SI one.

Conservative training requires no modification to the network topology, and can be applied to most DNN configurations. Three conservative training criteria are discussed: L^2 regularisation, sensitive parameter selection and KL-divergence regularisation.

L^2 Regularization

An adaptation criterion was formed of an L^2 regularisation term (Li and Bilmes, 2006; Liao, 2013) to the overall training criterion. This can be expressed as

$$\mathcal{F}(\boldsymbol{\theta},\mathbb{D}) = \mathcal{L}(\theta,\mathbb{D}) + \eta||\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathrm{SI}}||_2^2 \tag{3.60}$$

where $\boldsymbol{\theta}^{\mathrm{SI}}$ stands for the parameters in the SI model. The introduced L^2 penalty term decays the parameters of the adapted model towards the original SI model. In contrast with the discussion in Section 2.4, this L^2 regularisation term for adaptation is based on a different concept, which well-tuned parameters $\boldsymbol{\theta}^{\mathrm{SI}}$ are treated as a prior, not zeros.

Sensitive Parameter Selection

Instead of re-estimating the complete parameter set, the re-estimation can be performed on a small set of parameters, which are sensitive to acoustic variations. Sensitive parameter selection (Stadermann and Rigoll, 2005) chooses hidden units in the SI DNN with higher activation variance over the adaptation data. These hidden units are expected to significantly influence the output; thus, associated parameters are then chosen to be re-estimated. Define the index set of selected parameters as $\mathbb{S}(\theta)$, the adaptation criterion can be expressed as

$$\begin{aligned}&\mathcal{F}(\boldsymbol{\theta},\mathbb{D}) = \mathcal{L}(\boldsymbol{\theta},\mathbb{D}),\\ &\text{subject to} \qquad\qquad \theta_i = \theta_i^{\text{SI}} \quad \forall i \notin \mathbb{S}(\theta).\end{aligned} \tag{3.61}$$

This strategy keeps insensitive parameters unchanged to ensure that the SD model is not driven far away from the SI model.

KL-divergence Regularization

Adaptation can also be based on the KL-divergence regularisation (Yu et al., 2013), that is,

$$\mathcal{F}(\boldsymbol{\theta},\mathbb{D}) = \mathcal{L}(\boldsymbol{\theta},\mathbb{D}) + \eta \frac{1}{|\mathbb{D}|} \sum_{t=1}^{|\mathbb{D}|} \sum_{y} P^{SI}(y|\boldsymbol{x}_t) \log P(y|\boldsymbol{x}_t) \tag{3.62}$$

where $P^{SI}(y|\boldsymbol{x}_t)$ is the target distribution from the SI model. It encourages the KL-divergence between the SI and SD target distributions to be small.

3.5.2 Feature-based Adaptation

Feature-based adaptation transforms the input features feeding to the network. There are two basic strategies: feature normalisation; and feature augmentation. Feature normalisation aims to remove speaker-dependent variations. Thus, normalised features should be more compact and insensitive to acoustic variations. Feature augmentation introduces axillary features to the input features, which informs neural networks about speaker-dependent variations. These axillary features can be derived from the acoustic features, such as i-vectors (Dehak et al., 2011; Glembek et al., 2011), or from external information, such as environment-related indicators (Feng et al., 2015).

This section presents the feature normalisation and two types of axillary features, i-vectors (Saon et al., 2013) and speaker codes (Abdel-Hamid and Jiang, 2013a).

Feature Normalisation

Feature normalisation is designed to remove speaker variations, such as accent and environment noise, contained in the acoustic features. CMLLR normalisation (Gales, 1998), which was originally proposed for GMM-HMMs, can be applied to normalise acoustic features for DNN models (Rath et al., 2013; Seide et al., 2011a; Yoshioka et al., 2014). In addition, denoising autoencoders (Feng et al., 2014; Ishii et al., 2013) have also been used to improve the input feature representation.

I-vectors

I-vectors (Dehak et al., 2011; Glembek et al., 2011), i.e. information vectors, are a low-dimensional fixed-length vector representation of speaker space spanning the dimensions of highest speech variability. It was initially proposed for speaker verification, but it has recently been used for DNN adaptation in speech recognition (Saon et al., 2013; Senior and Lopez-Moreno, 2014b). The i-vector representation contains relevant information about the identity of speakers, which can inform the DNN training about corresponding acoustic variations and distortions. The details of i-vector estimation and extraction are presented in Appendix A.

A range of i-vector variations have also been examined for DNN adaptation as well. Informative prior (Karanasou et al., 2015) smooths i-vector representation to handle highly distorted acoustic conditions. In addition, i-vector factorisation (Karanasou et al., 2014) enriches its representation with multiple acoustic factors.

Speaker Codes

Speaker codes (Abdel-Hamid and Jiang, 2013a,b; Xue et al., 2014) are an alternative type of axillary feature for DNN adaptation. In contrast to most axillary features, they are trained jointly with the DNN parameters. In contrast with i-vectors, which are extracted by an independent model, speaker codes are learned in conjunction with the DNN model. This design avoids the potential modelling mismatch caused by separate models.

The network configuration with speaker code is illustrated in Figure 3.11. To reinforce their importance, speaker codes are introduced as the input signal to multiple layers, often the bottom of the network. In these layers, the activation function inputs are

$$\boldsymbol{z}_t^{(ls)} = \boldsymbol{W}^{(l)\mathrm{T}}\boldsymbol{h}_t^{(l-1,s)} + \boldsymbol{A}^{(l)\mathrm{T}}\boldsymbol{c}^{(s)} + \boldsymbol{b}^{(l)} \tag{3.63}$$

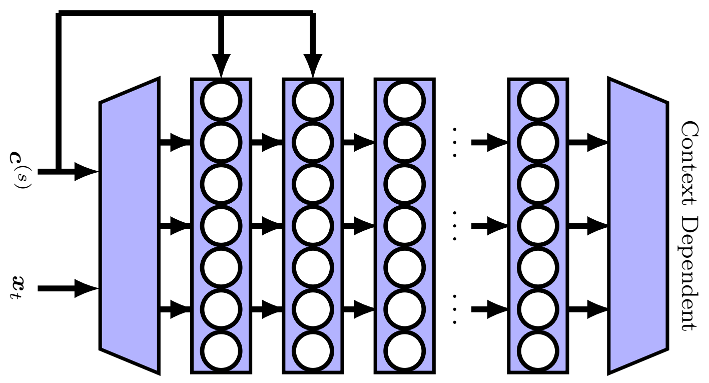


Fig. 3.11 DNN with speaker codes. Speaker codes $\boldsymbol{c}^{(s)}$ are introduced to several bottom layers to emphasise its importance.

where $\boldsymbol{c}^{(s)}$ stands for the speaker code for speaker s, and $\boldsymbol{W}^{(l)}$, $\boldsymbol{A}^{(l)}$, $\boldsymbol{b}^{(l)}$ are parameters associated with this layer. The network training can follow the error back-propagation algorithm, and speaker codes $\boldsymbol{c}^{(s)}$ are jointly updated for all the speakers. In the adaptation phase, the speaker code is firstly learned on the adaptation data via back-propagation. It is then used as input to perform decoding on test data.

3.5.3 Model-based Adaptation

Model-based adaptation re-estimates the model parameters on the adaptation data. For neural networks, it would be impractical to re-estimate all the parameters due to the size of the model. In contrast to conservative training, model-based schemes for DNNs perform adaptation on specific network components, such as hidden-layer transformations and activation functions, not the whole network. Three types of model-based adaptation are discussed here: linear hidden layer; transformation interpolation; and parametrised activation functions.

This thesis will describe model-based adaptation in terms of the canonical model and speaker-dependent transforms. The model parameters $\boldsymbol{\theta}$ are split into two categories: the canonical model $\mathcal{M}$ is referred to as the shared SI parameters, and the SD transform $\boldsymbol{\Lambda}^{(s)}$ include the SD parameters for a particular speaker s.

Linear Hidden Layer

This category of model-based adaptation introduces a speaker-dependent linear hidden layer to the network topology. It can be introduced prior to the input layer (Abrash et al., 1995; Li and Sim, 2010; Neto et al., 1995; Seide et al., 2011a; Trmal et al., 2010; Xiao et al., 2012), to the hidden layers (Gemello et al., 2007; Yao et al., 2012) or prior to the softmax output layer (Li and Sim, 2010; Yao et al., 2012). This adaptation strategy can be expressed as

$$\boldsymbol{z}_t^{(ls)} = \boldsymbol{A}^{(ls)\mathrm{T}}\mathbf{W}^{(l)\mathrm{T}}\boldsymbol{h}_t^{(l-1,s)} + \mathbf{b}^{(ls)} \tag{3.64}$$

where $\boldsymbol{A}^{(ls)}$ and $\mathbf{b}^{(ls)}$ are speaker-dependent parameters on layer l. Normally, $\boldsymbol{A}^{(ls)}$ is a large matrix that may contain several million parameters. Using the full matrix is impractical. This matrix can be restricted in different ways, such as block-diagonal (Seide et al., 2011a) or diagonal (Yao et al., 2012). For DNNs with SD linear hidden layers, the canonical model includes the parameters of SI neural network, while the SD transforms includes $\boldsymbol{A}^{(ls)}$ and $\mathbf{b}^{(ls)}$ for each speaker.

An alternative way to model $\boldsymbol{A}^{(ls)}$ is to used the subspace method (Dupont and Cheboub, 2000), which only introduces a small number of SD parameters. In detail, Principal component analysis (PCA) is first performed on the transformation matrix $\boldsymbol{W}^{(l)}$ to obtain a set of K eigenvector matrix,

$$\mathbb{E}^{(l)} = \{\boldsymbol{E}_1^{(l)}, \boldsymbol{E}_2^{(l)}, \ldots, \boldsymbol{E}_K^{(l)}\}. \tag{3.65}$$

Adaptation to speaker s is then performed as a linear combination of the retained eigenvectors

$$\boldsymbol{z}_t^{(ls)} = \left(\sum_{k=1}^{K} \lambda_k^{(s)} \boldsymbol{E}_k^{(l)}\right)^{\mathrm{T}} \boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}^{(l)} \tag{3.66}$$

where $\boldsymbol{\lambda}^{(s)}$ represents the SD parameters for speaker s. Because $\boldsymbol{\lambda}^{(s)}$ only consists of a limited number of interpolation weights, it can be robustly trained even with very limited adaptation data.

Transformation Interpolation

Transformation interpolation (Delcroix et al., 2015; Tan et al., 2015b) is similar to the subspace method. However, rather than using sets of eigenvectors, transformation interpolation applies sets of "free" matrices without the orthogonal requirement. This strategy can be expressed as

$$\boldsymbol{z}_t^{(ls)} = \left(\sum_{k=1}^{K} \lambda_k^{(s)} \boldsymbol{W}_k^{(l)} \right)^{\mathrm{T}} \boldsymbol{h}_t^{(l-1,s)} + \boldsymbol{b}^{(l)} \tag{3.67}$$

where $\boldsymbol{\lambda}^{(s)}$ stands for SD parameters, and $\boldsymbol{W}_1^{(l)}, \boldsymbol{W}_2^{(l)} \ldots, \boldsymbol{W}_K^{(l)}$ are K parallel weight matrices. In network training, the parallel transformations and interpolation weights $\boldsymbol{\lambda}^{(s)}$ for each speaker are jointly learned via back-propagation. In adaptation, the transformations are kept fixed, while $\boldsymbol{\lambda}^{(s)}$ is tuned for each speaker. The limited number of interpolation weights enable adaptation to be performed in a rapid and robust way. The canonical model of this method includes the multiple transformation matrices and bias vectors, and the SD transforms are the interpolation weights for all speakers. Delcroix et al. (2016a) has examined transformation interpolation to adapt CNNs. Model combination with i-vector method has also been investigated (Delcroix et al., 2016b) for transformation interpolation schemes.

Transformation interpolation is similar to the multi-basis adaptive neural network that will be discussed in Chapter 4. Rather than sets of parallel weight matrices, multi-basis adaptive neural networks use parallel sub-network to handle different aspects of the data.

Parametrised Activation Function

The previous two schemes deal network adaptation with affine transformations in DNN models. Instead, parametrised activation function (Siniscalchi et al., 2012; Swietojanski and Renais, 2016; Swietojanski and Renals, 2014; Zhang and Woodland, 2015b) generalises the form of activation function to compensate speaker variations. Figure 3.12 illustrates the general framework of parametrised activation function. It introduces three types of SD parameters. The outputs of adapted activation functions

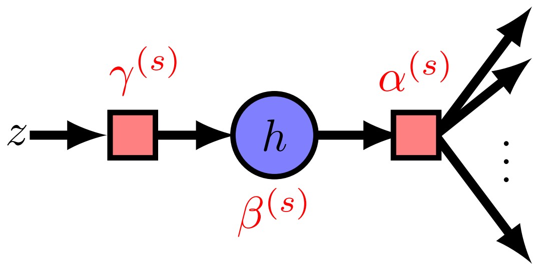


Fig. 3.12 Parametrised activation function.

can be expressed as

$$\boldsymbol{h}_t^{(ls)} = \boldsymbol{\alpha}^{(ls)} \otimes \boldsymbol{\phi}\left(\boldsymbol{\gamma}^{(ls)} \otimes \boldsymbol{h}_t^{(l-1,s)}; \boldsymbol{\beta}^{(ls)}\right) \tag{3.68}$$

where $\otimes$ stands for element-wise multiplication, $\boldsymbol{\alpha}^{(ls)}$ is applied to the activation outputs, $\boldsymbol{\gamma}^{(ls)}$ is applied to the input of activation function, and $\boldsymbol{\beta}^{(ls)}$ are the parameters associated with activation function. For example, the intrinsic parameters in Hermitian polynomial can be used for adaptation (Siniscalchi et al., 2013). For output scaling factors $\boldsymbol{\alpha}^{(ls)}$, the learning hidden unit contributions (Swietojanski and Renals, 2014) (LHUC) method wraps the raw factors using sigmoid function with amplitude 2,

$$\tilde{\boldsymbol{\alpha}}^{(ls)} = \frac{2}{1 + \mathbf{exp}\left(-\boldsymbol{\alpha}^{(ls)}\right)}, \tag{3.69}$$

to restrict the output to be in the range $(0, 2)$. This warping approach was reported to yield a robust representation, especially for adaptation in noisy acoustic conditions.

One limitation in parametrised activation function is that the number of parameters to adapt is equal to the number of hidden units. It is difficult to robustly estimate a large number of parameters when there are few limited adaptation data.

3.6 Performance Evaluation

Performance evaluation in speech recognition examines the quality of recognised transcriptions compared with the corresponding transcriptions. The word error rate (WER) is a metric widely used to evaluate the performance of ASR systems. To calculate the WER, recognised transcriptions are first aligned against the reference transcriptions. This is usually performed via a dynamic programming algorithm that minimises the Levenshtein distance between the two sequences. Given the alignment, the number of substitution (Sub), deletion (Del) and insertion (Ins) errors are respectively counted by comparing the words in the recognised and reference transcriptions. The WER is then calculated via

$$\text{WER} = \frac{\text{Sub} + \text{Del} + \text{Ins}}{\text{Tot}} \times 100\% \tag{3.70}$$

where Tot is the total number of words in the reference. Word error rate is quoted as percentages. Rather than WER, error rate can be evaluated in a different level for specific tasks. For phone recognition tasks, phone error rate is commonly used. For languages such as Chinese, character error rate are used to remove mismatches in word segmentation. Another popular metric is the sentence error rate, which reports the rate of recognised sentences that are fully correct in the test data.

3.7 Summary

This chapter has described principal concepts in speech recognition and related deep learning approaches for ASR. It started with the description of acoustic feature, including filter banks, PLPs as well as MFCCs, and post-processing methods, such as dynamic feature and feature normalisation. Generative and discriminative models for speech recognition were then discussed, respectively. The description of generative model focused on hidden Markov models and the integration of DNNs to HMMs. It also covered language models, decoding approaches and lexicon. The discussion of discriminative models included three approaches of deep learning: connectionist temporal classification, encoder-decoder models, and attention-based models. Training criteria for sequential tasks were then presented. The discussion included a range of discriminative training criteria, such as maximum mutual information, minimum classification error and minimum Bayes' risk. Next, the adaptation schemes for neural networks were discussed, including conservative training, feature-based and model-based adaptation. The last section presented the evaluation metrics for ASR tasks.

Chapter 4

Multi-basis Adaptive Neural Network

In standard DNN configurations, it remains as an open problem how to rapidly adapt network parameters with limited, unsupervised, data. Existing approaches have been proposed to re-estimate network parameters of affine transformations (Abrash et al., 1995; Li and Sim, 2010; Neto et al., 1995; Seide et al., 2011a; Trmal et al., 2010; Xiao et al., 2012) and activation functions (Siniscalchi et al., 2012; Swietojanski and Renais, 2016; Swietojanski and Renals, 2014; Zhang and Woodland, 2015b) associated with the DNN model. Due to the large number of parameters to adapt, it is often impractical to perform DNN adaptation rapidly with limited adaptation data using these schemes. Bayesian adaptation techniques, such as maximum a posteriori (Gauvain and Lee, 1994; Shinoda and Lee, 2001), are commonly used to address data scarcity issues in rapid adaptation. For DNNs, Bayesian methods have been applied to adapt the hidden-layer parameters (Huang et al., 2014, 2015, 2016). As the unstructured nature of DNN, i.e. network parameters do not have physical meanings, adaptation are often performed on some part of the network, which may not be directly related to the speaker or noise variations. Appropriate structured designs to the network, with explicit meanings, are believed to handle adaptation more effectively.

In the following three chapters, three types of structured deep neural networks are presented. The first type, proposed in this chapter, is referred to as multi-basis adaptive neural networks (MBANNs) (Wu and Gales, 2015). The MBANN is designed

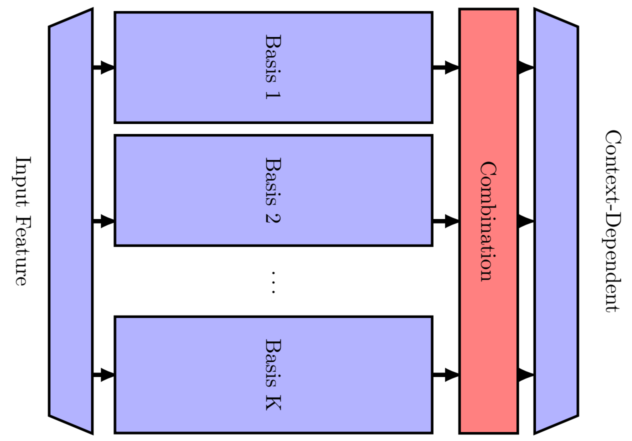


Fig. 4.1 Multi-basis adaptive neural network.

to yield a structured network, allowing the network adaptation to be performed robustly and rapidly. In the topology of MBANN, a set of parallel sub-networks are introduced to collaboratively model different aspects of the training data. The outputs of those sub-networks are then combined together in a combination structure. This combination structure is modelled as an adaptable component. By carefully choosing the combination scheme, the speaker-dependent transform can be re-estimated in a neat and compact fashion. This design makes the adaptation on an MBANN only require to estimate a small number of parameters for one speaker, allowing it to be adapted rapidly.

The discussion in this chapter includes: the network topology; parameter training; and adaptation schemes for multi-basis adaptive neural networks. Also, several extended MBANN models are presented, including combination schemes with i-vector representation, target-dependent interpolation, and inter-basis connectivity.

4.1 Network Topology

A multi-basis adaptive neural network is illustrated in Figure 4.1. A set of distinct sub-networks are introduced to the network topology, referred to as bases. There is a common input layer and a common output layer. Optionally, common hidden layers can be introduced before propagating to the bases or after their output combination. Each basis k is composed of several hidden layers, and the output and input of basis hidden layers are recursively defined as

$$\boldsymbol{h}_t^{(l,k)} = \boldsymbol{\phi}(\boldsymbol{z}_t^{(l,k)}), \tag{4.1}$$

$$\boldsymbol{z}_t^{(l,k)} = \boldsymbol{W}^{(l,k)\mathrm{T}}\boldsymbol{h}_t^{(l-1,k)} + \boldsymbol{b}^{(l,k)}. \tag{4.2}$$

The hidden units between successive layers in one basis are fully connected, while there is no connection between units from different bases. With this restricted configuration, a basis should be able to model a specific aspect of the training data. The outputs of the bases are then merged together at the combination stage.

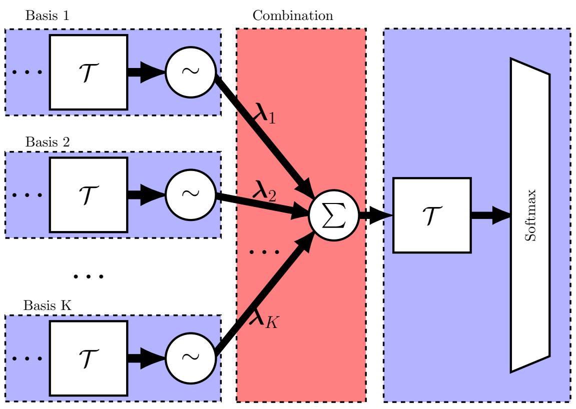


Fig. 4.2 Combining bases of MBANN via linear interpolation.

The combination stage is a core structure in the MBANN framework, as it handles the network adaptation. In this thsis, the linear interpolation scheme is investigated to combine the bases, as shown in Figure 4.2. In this configuration, the outputs of multiple bases are linearly combined using the interpolation weights $\boldsymbol{\lambda}_{\mathrm{mb}}$, referred to as basis weight vector,

$$\boldsymbol{\lambda}_{\mathrm{mb}} = \left[\lambda_{\mathrm{mb},1}, \lambda_{\mathrm{mb},2}, \ldots, \lambda_{\mathrm{mb},K}\right]^{\mathrm{T}} \tag{4.3}$$

where K is the total number of bases. A significant advantage of interpolation is that only a small number of additional parameters, i.e. interpolation weights, are introduced. For the discussion of the framework, the combination on the last hidden layer is presented, and other topologies are possible. The output after combination is expressed as

$$\bar{\boldsymbol{h}}_t^{(L-1)} = \sum_{k=1}^{K} \lambda_{\mathrm{mb},k} \boldsymbol{h}_t^{(L-1,k)}. \tag{4.4}$$

This combined result is then propagated to subsequent common layers. The structured topology of MBANN imposes a regularisation on the activation functions in the bases prior to the combination. That is, the activation functions, which are interpolated, are ordered together and restricted to perform similar behaviours. This mitigates the issue of arbitrary-ordered activation functions, which has the potential to improve the network regularisation.

To perform adaptation on an MBANN, the basis weight vector must be estimated. By finding an appropriate $\boldsymbol{\lambda}_{\mathrm{mb}}^{(s)}$ for each speaker s, the MBANN can be efficiently adapted to any speaker. The details of parameter training and adaptation for MBANNs are presented below.

4.2 Parameter Training

The parameter optimisation of the MBANN model can be performed using an adaptive training framework, i.e. the canonical model and speaker-dependent parameters are jointly optimised. For multi-basis adaptive neural networks, the canonical model $\mathcal{M}$ is defined as

$$\mathcal{M} = \left\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(K)}, \boldsymbol{\theta}^{\mathrm{share}}\right\} \tag{4.5}$$

where $\boldsymbol{\theta}^{(k)}$ represents the parameters in basis k, and $\boldsymbol{\theta}^{\mathrm{share}}$ denotes the parameters of the shared layers. The speaker-dependent transform $\boldsymbol{\Lambda}^{(s)}$ for speaker s is defined as

$$\boldsymbol{\Lambda}^{(s)} = \{\boldsymbol{\lambda}_{\mathrm{mb}}^{(s)}\}. \tag{4.6}$$

Defining $\boldsymbol{\theta}$ as the complete parameter set for MBANN, yields

$$\boldsymbol{\theta} = \mathcal{M} \cup \{\boldsymbol{\Lambda}^{(s)}\}_{1 \le s \le S}. \tag{4.7}$$

Given the training data $\mathbb{D}$ and training criterion, $\mathcal{F}(\boldsymbol{\theta},\mathbb{D})$, the canonical model $\mathcal{M}$ and SD parameters for all training speakers $\{\boldsymbol{\Lambda}^{(s)}\}_{1\leq s\leq S}$ are jointly optimised in the training phase.

Parameter Initialisation

A crucial concern in training the MBANN model is the parameter initialisation. To achieve a sensible initial performance, one option to initialise the bases is to use a sensible speaker-independent DNN system. That is, the hidden layers from this SI system are duplicated to build the multiple bases. To break the symmetry among the bases, the speaker-dependent parameters $\Lambda^{(s)}$ should not be initialised identically. They can be initialised by a range of methods, such as random values, prior knowledge like gender information, or automatic approaches such as i-vector clustering. To ensure that the initial performance of MBANN is the same as the SI system, the sum of initial $\lambda^{(s)}_{\text{mb},1},\ldots,\lambda^{(s)}_{\text{mb},K}$ should be equal to one,

$$\sum_{k=1}^{K}\lambda^{(s)}_{\text{mb},k}=1. \tag{4.8}$$

This constraint is only used in the parameter initialisation. It is not introduced in the parameter training, in order to reinforce the effect of speaker-dependent parameters to the maximal extent.

Interleaved Training

The canonical model and SD transforms are updated iteratively in MBANN training. The interleaved parameter training is described in Algorithm 4. The canonical model $\mathcal{M}$ and speaker-dependent parameters $\{\boldsymbol{\Lambda}^{(s)}\}_{1\leq s\leq S}$ are interleavingly updated till convergence or the maximal iterations are reached. In each iteration, parameters of the canonical model and the basis weight vectors for all training speakers are updated using stochastic gradient descent. The gradient calculation can be performed via error back-propagation. In addition to the gradients used in standard DNN training, it

Algorithm 4 Interleaved parameter training for MBANN.

```
1: initialize $\mathcal{M}$ from the SI DNN model
2: initialize $\{\boldsymbol{\Lambda}^{(s)}\}_{1\leq s\leq S}$
3: while not convergence do
4:    update $\mathcal{M}$
5:    for $s := 1$ to $S$ do
6:       update $\boldsymbol{\Lambda}^{(s)}$
7:    end for
8: end while
```

requires the following two gradients

$$\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}_t^{(L-1,k)}} = \lambda_{\text{mb},k}^{(s)} \frac{\partial \mathcal{F}}{\partial \bar{\boldsymbol{h}}_t^{(L-1)}}, \tag{4.9}$$

$$\frac{\partial \mathcal{F}}{\partial \lambda_{\text{mb},k}^{(s)}} = \boldsymbol{h}_t^{(L-1,k)\text{T}} \frac{\partial \mathcal{F}}{\partial \bar{\boldsymbol{h}}_t^{(L-1)}}. \tag{4.10}$$

4.3 Adaptation

After the training phase, the SD transforms belonging to training speakers are wiped out, and only the canonical model $\mathcal{M}$ is used for adaptation. By keeping $\mathcal{M}$ to be fixed, the SD basis weight vector $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ for a test speaker can be optimised using Eq. 4.10. The estimated SD transform is then combined with $\mathcal{M}$ to decode testing utterances.

The estimation of $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ requires the corresponding reference target labels aligned with the feature vectors. In speech recognition, adaptation can either be performed in the supervised or unsupervised fashion. In supervised adaptation, reference transcriptions are available for the testing data; thus, speaker-dependent parameters can be estimated using the true labels. In unsupervised adaptation, there is no reference transcription available. In this case, adaptation can be performed using decoding hypotheses from an SI system utilised as the reference. The potential errors in the hypotheses can significantly influence the performance of adaptation. To alleviate the influence of decoding errors, the cross-entropy criterion is normally used as the adaptation criterion. If the combination is designed prior to the output layer of MBANN, the optimisation of $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ is a convex problem. This convex property brings two advantages. First, the performance of adaptation is insensitive to the initialisation of SD transform. Second, the optimisation can be performed even more rapidly by second-order methods while ensuring a sensible performance. The proof of convexity is discussed in Appendix B.

4.4 Combining I-vector Representation

To handle acoustic distortions and variations, i-vectors are often added to the DNN input as speaker-informed features. The i-vector system handles adaptation in a different way to the multi-basis adaptive neural networks, and the combination of the two techniques may further improve the adaptation performance. In this section, two i-vector combination schemes for MBANNs (Karanasou et al., 2017; Wu et al., 2016a) are presented: the MBANN with i-vector input features and the predictive SD transform using i-vectors.

4.4.1 MBANN with I-vector Input Features

As presented in Section 3.5, the i-vector adaptation approach explicitly informs the neural network about acoustic identifiers along with the acoustic features. Because the DNN representation in higher layers is less sensitive to input variations (Goodfellow et al., 2016), i-vectors are often introduced at the input stage. In comparison, the multi-basis adaptive neural network delays the adaptation phase to a latter stage where activation functions in a higher layer are interpolated. The two forms of adaptation are very different and may be complementary, which warrants further investigation.

The first combination scheme is to use i-vectors as input features, as shown in Figure 4.3. The acoustic feature vector $\boldsymbol{x}_t$, such as PLP or filter bank, is concatenated with the associated i-vector $\boldsymbol{\lambda}_{\text{iv}}^{(s)}$ to form the input feature propagated to each of the multiple bases. In training, the layers of the bases are informed with acoustic characteristics; thus, the hidden-layer representation should be reinforced to capture the expected aspect of the data, e.g., noise or accent types for each basis.

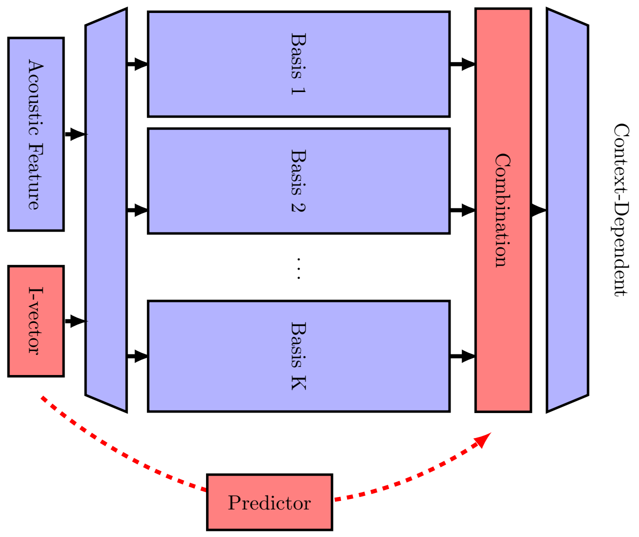


Fig. 4.3 Combining MBANN with I-vectors. Adaptable modules are coloured in red.

4.4.2 Predictive Speaker-dependent Transform Using I-vectors

In the MBANN framework, the estimation of a speaker-dependent transform $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ is required to adapt to an unknown speaker. As discussed in Section 4.3, decoding hypotheses from an initial speaker-independent system is required for unsupervised adaptation. However, this two-pass decoding scheme is not efficient enough in stringent real-time systems. Additionally, the performance may be poor under highly-mismatched conditions with hypotheses of poor quality.

The fast predictive estimation module of the SD transform is proposed to address these issues. I-vectors are used to directly estimate the multi-basis SD transform. In this scheme, a predictor, as illustrated in the dashed-line part of Fig. 4.3, is discriminatingly trained to establish a mapping from the i-vector $\boldsymbol{\lambda}_{\text{iv}}^{(s)}$ to the basis interpolation weights $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$,

$$\hat{\boldsymbol{\lambda}}_{\text{mb}}^{(s)} = \boldsymbol{g}_{\text{pred}}(\boldsymbol{\lambda}_{\text{iv}}^{(s)}) \tag{4.11}$$

where $\boldsymbol{g}_{\text{pred}}(\cdot)$ represents the predictive model. The adaptation performance of MBANN is then undertaken by the precision of the prediction mappings, which is irrelevant to the quality of decoding hypothesis. Besides, the predictive procedure avoids the first decoding pass, thus allowing adaptation to be performed efficiently.

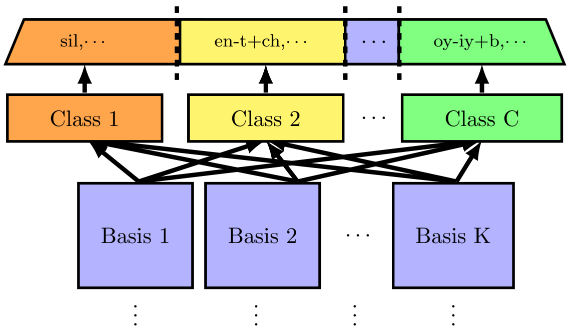


Fig. 4.4 MBANN with target-dependent interpolation.

The mismatch between the distribution of predicted interpolation weights and that of the original weights is likely to degrade the performance. To reduce the degradation caused by this sort of difference, an interleaving mode is utilised to update the MBANN and predictor jointly. In each iteration, the predictor is trained on the estimated SD transforms $\{\boldsymbol{\Lambda}_{\text{mb}}^{(s)}\}_s^{\text{esti}}$ for the training speakers from the current MBANN system, and the re-estimated SD transforms $\{\boldsymbol{\Lambda}_{\text{mb}}^{(s)}\}_s^{\text{pred}}$ given by this trained predictor is then used to update the neural network for the next iteration. The conjugate pair of neural network and predictor of each iteration is then used in evaluation.

4.5 Target-dependent Interpolation

In speech recognition, the network output consists of a set of phonetic units, such as context-dependent triphone states. These targets have a range of characteristics depending on phone context; thus, in adaptation, modelling them separately has the potential to improve the adaptation performance. For multi-basis adaptive neural networks, if the combination module is introduced just prior to the output layer, the separate adaptation depending on the target can be performed. This scheme will be referred to as the target-dependent interpolation in the MBANN framework.

Since a large number of targets are often used, it is not practical to estimate the interpolation weight per target. Instead, the targets can be appropriately merged into several classes. These classes can be defined using prior knowledge, such as silence/non-silence, consonant/vowel classes, or they can be automatically determined via some statistical methods. For example, column vector in the matrix of the last-layer transformation, i.e. $\boldsymbol{W}^{(L)}$, can be viewed as representations of each target. By performing k-means clustering on these column vectors, targets with similar characters can be grouped together.

Suppose that there are C target classes in total, and an interpolation weight $\boldsymbol{\lambda}_{\mathrm{mb}}^{(s,i)}$ is introduced for each class i, the speaker-dependent transform for this MBANN configuration is modelled as

$$\boldsymbol{\Lambda}^{(s)} = \left\{ \boldsymbol{\lambda}_{\mathrm{mb}}^{(s,1)}, \boldsymbol{\lambda}_{\mathrm{mb}}^{(s,2)}, \ldots, \boldsymbol{\lambda}_{\mathrm{mb}}^{(s,C)} \right\}. \tag{4.12}$$

Using this form of SD transform, the output of the adapted MBANN can be expressed as

$$P(\psi = i|\boldsymbol{x}_t) = \frac{\exp\left(\sum_k \lambda_{\mathrm{mb},k}^{(s,c(i))} z_{ti}^{(L,k)}\right)}{\sum_j \exp\left(\sum_k \lambda_{\mathrm{mb},k}^{(s,c(j))} z_{tj}^{(L,k)}\right)} \tag{4.13}$$

where $c(i)$ maps target i to its corresponding target class, and $\boldsymbol{z}_t^{(L,k)}$ is defined as

$$\boldsymbol{z}_t^{(L,k)} = \boldsymbol{W}^{(L)\mathrm{T}} \boldsymbol{h}_t^{(L-1,k)} + \boldsymbol{b}^{(L)}. \tag{4.14}$$

To perform adaptation using MBANNs with target-dependent interpolation weights, the concept of multiple interpolations can either be used in both training and adaptation, or in adaptation only. The gradient $\frac{\partial F}{\partial \lambda_{\mathrm{mb},k}^{(s,i)}}$ is calculated using

$$\frac{\partial \mathcal{F}}{\partial \lambda_{\mathrm{mb},k}^{(s,i)}} = \sum_{j:c(j)=i} z_{tj}^{(L,k)} \frac{\partial \mathcal{F}}{\partial \bar{z}_{tj}^{(L)}} \tag{4.15}$$

where $\bar{z}_{tj}^{(L)}$ is defined as

$$\bar{z}_{tj}^{(L)} = \sum_k \lambda_{\mathrm{mb},k}^{(s,c(j))} z_{tj}^{(L,k)}. \tag{4.16}$$

Appendix B shows that the optimisation of target-dependent interpolations is still a convex adaptation problem if the CE training is used.

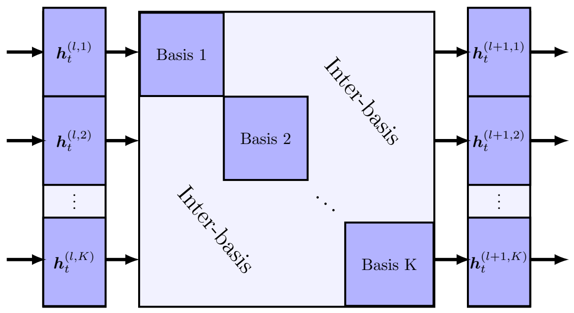


Fig. 4.5 MBANN with inter-basis connectivity.

4.6 Inter-basis Connectivity

In the proposed MBANN framework, the hidden units are fully connected within one basis, while there is no connection between units of different bases. This design, separating connections, can encourage a diverse representation among the bases. This section presents a generalised form of multi-basis adaptive neural network, which enables some level of inter-basis connections. Rather than a hard split, MBANNs may benefit from weak inter-basis connections. Figure 4.5 illustrates the hidden-layer configuration of MBANN with inter-basis connectivity. Defining $\tilde{\boldsymbol{h}}^{(l)}$ as the concatenation of the outputs the l-th layers from all bases,

$$\tilde{\boldsymbol{h}}_t^{(l)} = [\boldsymbol{h}_t^{(l,1)\mathrm{T}}, \boldsymbol{h}_t^{(l,2)\mathrm{T}}, \ldots, \boldsymbol{h}_t^{(l,K)\mathrm{T}}]^{\mathrm{T}}. \tag{4.17}$$

The hidden-layer output can then be expressed as

$$\tilde{\boldsymbol{h}}_t^{(l+1)} = \boldsymbol{\phi}\left(\tilde{\boldsymbol{W}}^{(l)\mathrm{T}} \tilde{\boldsymbol{h}}_t^{(l)} + \tilde{\boldsymbol{b}}^{(l)}\right) \tag{4.18}$$

where $\tilde{\boldsymbol{b}}^{(l)}$ concatenates the bias vectors $\{\boldsymbol{b}^{(l,k)}\}_k$, and $\tilde{\boldsymbol{W}}^{(l)}$ consists of the matrix parameters. If the inter-basis connectivity is turned off, $\tilde{\boldsymbol{W}}^{(l)}$ is a block-diagonal matrix, and the diagonal blocks are the matrix parameters for different bases $\{\boldsymbol{W}^{(l,k)}\}_k$. The introduction of inter-basis connections can enable the matrix parameters out of diagonal blocks.

To train an MBANN with the inter-basis connectivity, the restriction on connections between bases distinguishes it from a fully connected network, as different bases should be forced to capture different aspects of the training data. In this way, the network training is performed with aggressive L^2 regularisation on inter-basis parameters to limit inter-basis connections. The overall training criterion is expressed as

$$\mathcal{F}(\boldsymbol{\theta};\mathbb{D}) = \mathcal{L}(\boldsymbol{\theta};\mathbb{D}) + \eta\mathcal{R}(\boldsymbol{\theta};\mathbb{D}) \tag{4.19}$$

where the regularisation term $\mathcal{R}(\boldsymbol{\theta};\mathbb{D})$ penalises large parameters for inter-basis connections,

$$\mathcal{R}(\boldsymbol{\theta};\mathbb{D}) = \sum_{l} \sum_{(i,j)\in\mathbb{I}^{(l)}} \tilde{w}_{ij}^{(l)2} \tag{4.20}$$

where $\mathbb{I}^{(l)}$ stands for the index set of inter-basis parameters. Inter-basis connections are penalised to be small, which can help to reinforce the importance of parameters within each basis.

4.7 Preliminary Experiments

This section reports the preliminary experiments for multi-basis adaptive neural networks on the AURORA 4 task. AURORA 4 (Pearce and Picone, 2002) is a medium vocabulary task based on the 15-hour Wall Street Journal data (Paul and Baker, 1992). The original AURORA 4 dataset provides training sets in two conditions: clean and multi-style. The clean data is identical to the Wall Street Journal dataset, consisting of 7185 utterances from 83 speakers and totalling 15 hours of speech. The multi-style data were obtained by artificially corrupting the clean data, in which half were recorded using the primary Sennheiser microphones, and the other half were recorded by a number of secondary microphones. Six different types of noises were added to this training set with the signal-to-noise ratio (SNR) ranging from 10 to 20 dB. To evaluate the performance of MBANN in multiple acoustic conditions, the multi-style data was used as the training set in the experiments. The evaluation dataset of AURORA-4 was based on the development set of 1992 November NIST evaluation (Paul and Baker, 1992), consisting of 330 utterances from 8 speakers. It was corrupted by 6 types of noise under two microphone conditions where the SNR ranged 5 to 15 dB to form 14

	A	B	C	D
Type	clean	noise	channel	noise+channel
Total (hrs)	0.7	4.0	0.7	4.0
#Uttr	330	1980	330	1980
AvgUttr (secs)	7.3			

Table 4.1 AURORA 4: Summary of evaluation sets. It includes the type of acoustic distortion, total hours, number of utterances (#Uttr) and average utterance duration (AvgUttr). "Noise" represents additive noise, and "channel" is channel distortion.

test sets. They were split into 4 sets: A, B, C and D . Table 4.1 summarises the four sets used in the evaluation.

In this thesis, unless otherwise stated, the ASR system was build using the DNN-HMM hybrid framework. The relevant GMMs, DNNs and the proposed models were implemented and trained on an extended version of the HTK Toolkit 3.5 (Young et al., 2015).

4.7.1 Experimental Setup

The 13-dimensional PLP coefficients with their delta (Δ) and delta-delta ($\Delta\Delta$) dynamic features, processed by the utterance-level CMN and corpus-level CMN, were used as features to train a GMM-HMM system (with about 3k tied triphone states). This GMM-HMM system was further extended to include $\Delta\Delta\Delta$ using HLDA and discriminatively trained on the MPE criterion. This GMM-HMM MPE system was used to generate state alignments for DNN-HMM hybrid systems. Instead of using the PLP features[1], the 72-dimensional filter bank feature with the first- and second- order dynamic features, processed by utterance-level CMN and corpus-level CVN, was used to train the baseline DNN system. For the DNN baseline, five hidden layers were introduced, and the neural network configuration was $648 \times 1000 \times 1000 \times 1000 \times 1000 \times 1000 \times 3\text{k}$, with a context window of 9 frames as the input feature. The parameters of this DNN were initialised using the layer-by-layer mono-phone discriminative pre-training (Zhang and Woodland, 2015a) and further fine-tuned by back-propagation on the CE criterion. In

[1]On this task, the DNN with the filter bank feature yielded a better performance comparing to that with PLP or MFCC.

both training stages, 650 utterances belonging to 8 speakers were used as the cross validation set. This CE DNN was subsequently used to generate the lattices of the training set and further tuned for four iterations on the MPE criterion to obtain the baseline MPE DNN system. To fairly compare MBANNs and DNN baselines with a comparable number of parameters, a larger DNN configuration, denoted as "DNN (large)" in the following discussion, was trained. It included five hidden layers and 2000 units in each layer. The CE and MPE DNN systems of this large configuration were trained in similar settings as the systems of the small configuration. In evaluation, decoding was performed with the standard WSJ bi-gram language model.

Multi-basis adaptive neural networks were trained using similar settings as the baseline DNN system. As discussed in Section 4.2, the MBANN model was initialised by the well-trained DNN baseline and updated in the interleaved fashion. To prevent over-fitting caused by the additional tuning epochs, a lower learning rate was used to optimise the MBANN. If there is no additional descriptions, the adaptation of MBANNs was performed in an utterance-level unsupervised fashion. That is, the speaker-dependent parameters, i.e. basis weight vectors $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$, were estimated per test utterance, and the supervision was obtained from the decoding hypothesis of the SI DNN baseline. Detailed discussions are presented below.

4.7.2 Results and Discussion

This section discusses the performance of several models and configurations. The investigation on MBANNs included different basis settings, the target-dependent interpolation, and the inter-basis connectivity.

Comparison of Different Basis Configurations

To achieve a sensible initial performance, the MBANN was initialised with the SI CE DNN baseline. In parameter training, the bases were modelled to represent different i-vector clusters. The utterance i-vectors of the training data were clustered into 2, 4 and 6 clusters by k-means. A 1-of-K vector (a vector with one element containing a 1 and all other elements as 0) was specified to each utterance as its initial SD basis weight vector, representing its cluster index. The MBANNs were trained using the

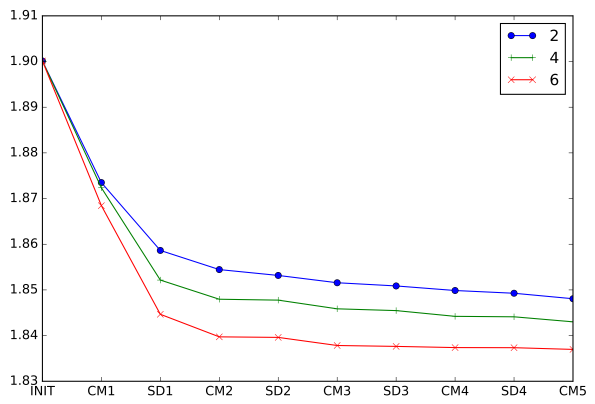


Fig. 4.6 AURORA 4: Learning curves of MBANNs with 2, 4 and 6 bases. "CM" is an update of the canonical model; "SD" is an update of the speaker-dependent parameters.

System	#Bases	A	B	C	D	Avg
DNN	–	4.2	8.4	9.1	19.7	13.0
DNN (large)		4.2	8.2	8.2	19.3	12.7
MBANN	2	4.0	8.1	9.4	18.7	**12.4**
	4	4.0	8.0	9.4	18.7	**12.4**
	6	4.0	7.9	9.4	18.8	**12.4**
MBANN (oracle)	2	3.9	7.7	8.1	18.4	12.1
	4	3.8	7.5	7.9	17.8	11.7
	6	3.7	7.4	7.9	17.7	11.6

Table 4.2 AURORA 4: Recognition performance (WER %) of CE MBANN. The "MBANN" block represent adaptations performed on decoding hypotheses of the DNN baseline system. The "MBANN (Oracle)" block represent adaptations performed on reference transcriptions. Adaptation was performed at the utterance level.

interleaved method (Algorithm 4) for five iterations on the CE criterion. The learning curves of MBANNs with 2, 4 and 6 bases are illustrated in Figure 4.6. By performing the interleaved training scheme, a lower cross entropy on the training set could be obtained. As much fewer parameters are contained in the speaker-dependent transform (i.e. interpolation weights), the decrease of cross-entropy value in the update phase of SD parameters is smaller than that of the canonical model.

The recognition performance of CE MBANN systems is summarised in Table 4.2. In the block "MBANN", the supervision for estimating $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ was obtained from the decoding hypotheses of the CE SI DNN. The MBANNs with 2, 4 and 6 bases achieved similar performance, i.e. about 5% relative error reduction compared with the SI DNN

System	A	B	C	D	Avg
DNN	3.8	7.7	8.5	19.0	12.3
DNN (large)	3.9	7.7	8.0	18.5	**12.1**
MBANN	3.8	7.9	8.1	18.4	**12.1**

Table 4.3 AURORA 4: Recognition performance (WER %) of MPE MBANN with 2 bases. Adaptation was performed at the utterance level.

baseline, reducing the WER from 13.0% to 12.4%. As indicated in the results of Sets B and D, increasing the number of bases can slightly improve the adaptation performance on low-WER scenarios (Set B), while the performance in high-WER scenarios (Set D) actually decreases. The "DNN (large)" system has a comparable number of parameters to the 4-bases MBANN. Nevertheless, the corresponding MBANN still yielded a lower-error performance. This indicates the effectiveness of structured design of MBANN. To illustrate the performance of MBANN to the maximal extent, the oracle adaptation, which was performed using the reference transcriptions, is also reported in the block "MBANN (oracle)". Better performance was obtained with more bases in the oracle experiments.

The MBANN with 2 bases were further tuned on the MPE criterion for two interleaving iterations[2]. To perform adaptation on MPE MBANNs, the decoding hypotheses of the MPE SI DNN were used as supervision. The performance of MPE systems are reported in Table 4.3. In contrast to the SI MPE DNN baseline, The adapted MPE MBANN reduced the WER from 12.3% to 12.1%. This performance is similar to the DNN large system, which contained much more parameters than the 2-basis MBANN.

Target-dependent Interpolation Weights

The MBANN with 2 bases was selected to evaluate the target-dependent interpolation scheme[3]. As discussed in Section 4.5, based on the meanings of DNN targets, there are

[2] In network training, the SD transforms of MPE MBANNs were optimised on the MPE criterion. In adaptation, the SD transforms were optimised on the CE criterion. This is a common configuration for adaptation, which can alleviate the impact of errors contained in decoding hypothesis.

[3]For the target-dependent interpolation scheme, parameter training with multiple target classes did not yield significant gains. This thesis reports a simple configuration that trained the MBANN with one class (default), and in adaptation, multiple target classes were used.

Clustering	A	B	C	D	Avg
–	4.0	8.1	9.4	18.7	12.4
sil/nonsil	4.0	8.0	8.7	18.8	12.4
k-means	4.0	8.0	8.7	18.8	12.4

Table 4.4 AURORA 4: Comparison of different clustering settings for target-dependent interpolation on the 2-basis CE MBANN. Silence/non-silence (sil/nonsil) and k-means (2 clusters) classes are compared. Adaptation was performed at the utterance level.

System	#Cls	A	B	C	D	Avg
DNN	–	4.2	8.4	9.1	19.7	13.0
MBANN	1	4.0	8.1	9.4	18.7	**12.4**
	2	4.0	8.0	8.7	18.8	**12.4**
	3	4.1	8.3	8.7	19.4	12.8
MBANN (oracle)	1	3.9	7.7	8.1	18.4	12.1
	2	3.8	7.7	8.4	18.3	12.0
	3	3.8	7.5	8.0	18.1	11.8

Table 4.5 AURORA 4: Comparison of 1,2 and 3 k-means target classes for target-dependent interpolation scheme on the 2-basis CE MBANN. “Oracle” systems stand for performing adaptation on reference transcriptions. Adaptation was performed at the utterance level.

several ways to specify the target classes using prior knowledge or automatic clustering. The DNN targets in this task were modelled as context-dependent triphone states. Two types of target classes were investigated: silence/non-silence classes, and k-means clustering classes. The silence/non-silence method split the targets into two classes: states belonging to silence, and those belonging to triphones. This configuration is based on the fact that silence frames are noticeably different to the non-silence ones. To separately adapt silence/non-silence targets is likely to improve the adaptation performance. The second type of target classes was obtained by the k-means clustering on the column vectors of the last-layer transformation matrix of MBANN (i.e. $\boldsymbol{W}^{(L)}$). The adaptation performance of MBANN target-dependent interpolations using the silence/non-silence and the k-means (2 clusters) methods are compared in Table 4.4. On the 2-class settings, the k-means results were similar result to the silence/non-silence classes, which mostly consisted of silence states and a range of states belonging to voiceless consonants. The overall performance on both target-class settings yielded little

System	#Cls	A	B	C	D	Avg
DNN	–	4.2	8.4	9.1	19.7	13.0
MBANN	1	4.0	7.7	8.5	18.6	12.2
	2	4.0	7.6	8.1	18.4	**12.0**
	3	3.9	7.8	8.1	18.5	12.1
	4	4.0	7.7	8.1	18.5	12.1

Table 4.6 AUROAR 4: Comparison of k-means target classes for target-dependent interpolation scheme on the 2-basis CE MBANN. Adaptation was performed at the speaker level.

benefit. However, on Set C, which is corrupted by channel distortion, the adaptation significantly reduced the WER from 9.4% to 8.7%. This can be explained by the fact that the channel distortion caused by microphones is likely to distort the representation of low-energy acoustic units, such as silence and voiceless consonants. The performance comparison of the k-means clustering with 1, 2 and 3 classes are illustrated in Table 4.5. The adaptation of MBANN turned to be more unstable when more interpolation classes were introduced. So far, the adaptation was performed in the utterance level. The potential issue of data scarcity can cause that different target classes are not sufficiently represented in the adaptation data. Therefore, the speaker-level adaptation was conduct to verify this assumption. That is, the SD transform was estimated per test speaker, not per utterance. Table 4.6 summarises the speaker-level adaptation using different number of k-means target classes. It shows that, with sufficient adaptation data, the adaptation performance can be further improved with multiple interpolation classes.

Inter-basis Connectivity

Next, the MBANN model with inter-basis connectivity (Section 4.6) was evaluated. By enabling the inter-basis connections, this 2-basis MBANN and the "DNN (large)" system have a comparable number of parameters. Therefore, experiments of inter-basis connectivity were conduct using the MBANN configuration with 2 bases. The impact of different regularisation penalties, ranged from 0 to infinity, of inter-basis connections is summarised in Table 4.7. A higher penalty will drive inter-basis weights closer to

System	η	A	B	C	D	Avg
DNN	–	4.2	8.4	9.1	19.7	13.0
DNN (large)		4.2	8.2	8.2	19.3	12.7
MBANN	0	4.0	8.0	8.3	18.8	12.4
	10^{-1}	4.0	8.0	8.4	18.7	**12.3**
	1	4.0	8.0	8.3	18.7	**12.3**
	10^{2}	4.0	8.0	8.4	18.8	12.4
	$\infty(\simeq 10^{5})$	4.0	8.1	9.4	18.7	12.4

Table 4.7 AUROAR 4: Comparison of different regularisation penalties of inter-basis connections on 2-basis CE MBANN.

zero. The "∞" row represents the default MBANN setting that turned off inter-basis connections, and its regularisation penalty η is approximately equivalent to 10^5, the reciprocal of learning rate. By introducing the inter-basis connectivity, it yielded little performance gains in contrast to the original MBANN model. This indicates that the inter-basis connections are less important than those in each basis.

4.8 Summary

In this chapter, multi-basis adaptive neural networks were proposed. Conventional model-based adaptation schemes (Section 3.5) for DNNs usually involve a large number of parameters being adapted, which makes effective adaptation impractical when there are limited adaptation data. The MBANN model aims at introducing structures to the network topology, allowing network adaptation to be performed robustly and rapidly. A set of parallel sub-networks, i.e. bases, are introduced. Weights are restricted to connect within a single basis, and different bases share no connectivity. The outputs among different bases are subsequently combined via speaker-dependent interpolation.

This chapter also discussed several extensions to the basic MBANN model. To combine i-vector representation, two combination schemes were presented. The first scheme appends i-vectors to the DNN input features. In this configuration, the bases are explicitly informed about acoustic attributes, and the robustness to acoustic variations can be reinforced. The second scheme uses i-vectors to directly predict the speaker-dependent transform for MBANN. This avoids the requirement for decoding hypotheses in adaptation, which helps to reduce the computational cost, as well as improve the robustness to hypothesis errors. The target-dependent interpolation was discussed, which introduces multiple sets of interpolation weights to separately adapt different DNN targets. Lastly, the inter-basis connectivity generalises the MBANN framework with parameters between different bases.

Chapter 5

Stimulated Deep Neural Network

In the previous chapter, multi-basis adaptive neural networks were presented. A set of bases, i.e. parallel sub-network structures, with restricted connections are modelled, and the restricted connectivity allows different aspects of data to be modelled separately. This design can be viewed as a "hard" version to group the hidden units.

Alternatively, the concept of hidden-unit grouping can be performed in a "soft" way. In this chapter, stimulated deep neural networks (Ragni et al., 2017; Tan et al., 2015a; Wu et al., 2016b) are proposed[1]. This type of structured neural network encourages activation function outputs in regions of the network to be related, aiding the interpretability and visualisation of network parameters. In standard neural network training, hidden units can take an arbitrary ordering; thus, it is difficult to relate parameters to each other. The lack of interpretability can cause problems for network regularisation and speaker adaptation. The design of stimulated DNN first resolves the issue of arbitrary ordering of hidden units. In the network topology, the units of each hidden layer are reorganised to form a grid. Activation functions with similar behaviours are then learned to group together in this grid space. This objective is achieved by introducing a special form of regularisation term to the overall training criterion, referred to as activation regularisation. The activation regularisation is designed to encourage the outputs of activation functions to satisfy a target pattern. By defining appropriate target patterns, different visualising, partitioning or grouping

[1]The term "stimulated" follows the naming fashion in stimulated learning (Tan et al., 2015a) which performs stimulation on the outputs of activation functions to induce interpretation.

concepts can be imposed on the network. The stimulated DNN prevents the arbitrary-ordering issue in standard DNN models. This kind of manipulation is believed to reduce over-fitting and improve the model generalisation. In addition, based on the similarity between activation functions in this spatial ordering, smoothness techniques can be used on stimulated DNNs to regularise a range of DNN adaptation methods.

In literature, a range of approaches have been proposed to interpret DNN parameters. These schemes mainly focus on analysing a well-trained neural network instead of inducing useful interpretations in parameter training. For instance, Garson's algorithm (Nguyen et al., 2015) was used to inspect feature importance in DNN models. In the area of computer vision, weight analysis of neural networks has been examined to interpret neural networks. In Mahendran and Vedaldi (2015); Nguyen et al. (2015); Simonyan et al. (2013), the input feature was optimised to maximise the output of a given hidden activation in the network. The visualisation of the feature implies the function of that activation. However, stimulated DNNs achieve the network interpretation in a different fashion, which are induced in the training procedure. The strategy of inducing desired network interpretations offers a flexible tool to analyse DNN models. Instead of being deciphered from complex "black boxes", DNN parameters can be regularised to present pre-defined concepts.

This chapter discusses the network topology, the design of activation regularisation, and adaptation methods for stimulated deep neural networks.

5.1 Network Topology

As discussed above, one crucial issue in standard neural network training is that the hidden-layer units can take an arbitrary order. This lack of an ordering constraint leads to two problems: first, a direct visualisation of a hidden layer gives no insights or interpretations; second, manipulation of hidden units in groups rather than individual elements is challenging. To mitigate the issue of arbitrary ordering, stimulated neural network training introduces spatial-ordering constraints on hidden units, allowing hidden units to be related to each other. For the discussion of the proposed stimulated

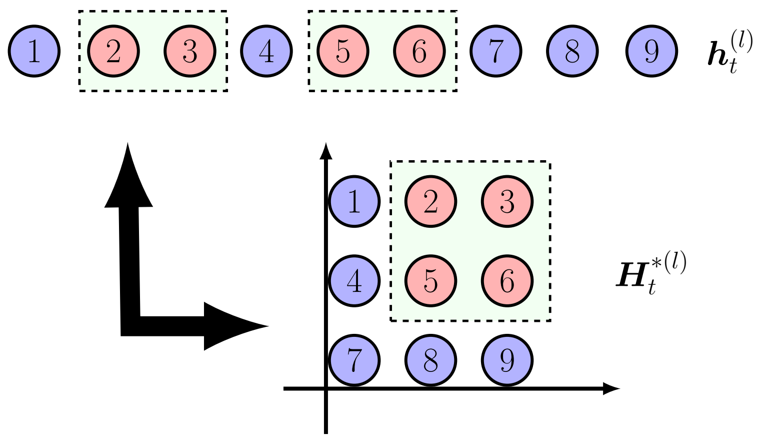


Fig. 5.1 Reorganise units to form a grid in one hidden layer. Non-contiguous elements (in dotted boxes) can form a contiguous region in the grid representation.

DNN, this chapter uses a feed-forward network architecture as an example. However, the concept of stimulated DNN can be applied to more complex architectures.

In each hidden layer of a stimulated DNN, hidden units are reorganised to form a grid, referred to as the activation grid. For instance, a hidden layer with 1024 units can form a 32×32, 2-dimensional, grid. Given a network with parameters $\boldsymbol{\theta}$ and an input feature vector $\boldsymbol{x}_t$, the reorganisation operation $\mathcal{G}(\boldsymbol{x}_t, \boldsymbol{\theta})$ generates the grid representation $\boldsymbol{H}_t^{*(l)}$ for activation function outputs $\boldsymbol{h}_t^{(l)}$ of each hidden layer[2]

$$\mathcal{G}(\boldsymbol{x}_t, \boldsymbol{\theta}) = \left(\boldsymbol{H}_t^{*(1)}, \boldsymbol{H}_t^{*(2)}, \cdots, \boldsymbol{H}_t^{*(L-1)}\right). \tag{5.1}$$

The exact form of the representation $\boldsymbol{H}_t^{*(l)}$ depends on the dimensionality of grid representation. If a 1D grid is introduced, $\boldsymbol{H}_t^{*(l)}$ is a vector; in a 2D configuration, $\boldsymbol{H}_t^{*(l)}$ is a matrix. For example, on a layer with n^2 hidden units, The 2D representation of $\boldsymbol{H}_t^{*(l)}$ can be expressed as

$$\boldsymbol{H}_t^{*(l)} = \begin{bmatrix} h_{t,1}^{(l)} & \cdots & h_{t,n}^{(l)} \\ \vdots & \ddots & \vdots \\ h_{t,n^2-n+1}^{(l)} & \cdots & h_{t,n^2}^{(l)} \end{bmatrix}. \tag{5.2}$$

[2]The superscript “*” is introduced to disambiguate vector and grid representations. In this chapter, variables in the grid representation are marked with superscript “*”, and variables in the vector representation are not. For example, h^*_{tij} is the activation function output of unit (i,j) in the grid, while h_{ti} is the activation function output of the i-th hidden unit in the vector representation.

For higher dimensional configurations, $\boldsymbol{H}_t^{*(l)}$ can be expressed as a higher-order tensor. This chapter concentrates on the 2D situation as an example for the discussion. Figure 5.1 shows an example of unit reorganisation on a hidden layer with 9 units. Non-contiguous nodes (in dotted boxes) can form a contiguous region in the corresponding grid representation. This grid representation can be viewed as a Cartesian coordinate system. The hidden unit located at (i,j) in the grid is located at a point in this space, denoted as $\boldsymbol{s}_{ij}$. The network topology of stimulated DNN defines a spatial order for each hidden layer, on which the grouping or interpretable regions can be defined using Euclidean metrics.

5.2 Activation Regularisation

The network topology of stimulated DNNs enables activation function outputs to be related to each other, by introducing regularisation over the activation function outputs. This section presents the general framework for activation regularisation. Activation regularisation encourages the outputs of activation functions to comply with some reference, i.e. target pattern. By defining an appropriate target pattern, desired attributes, grouping or partitions can be imposed to influence the behaviour of activation function. The imposed learning concept in target patterns can prevent the arbitrary ordering of hidden units, which has the potential to improve network regularisation. To implement this approached, a regularisation term $\mathcal{R}(\boldsymbol{\theta};\mathbb{D})$ is added to the training criterion $\mathcal{F}(\boldsymbol{\theta};\mathbb{D})$

$$\mathcal{F}(\boldsymbol{\theta};\mathbb{D}) = \mathcal{L}(\boldsymbol{\theta},\mathbb{D}) + \eta\mathcal{R}(\boldsymbol{\theta};\mathbb{D}) \tag{5.3}$$

where $\mathcal{L}(\boldsymbol{\theta};\mathbb{D})$ is the standard training criterion, and the hyper-parameter η determines the contribution of the activation regularisation term $\mathcal{R}(\boldsymbol{\theta},\mathbb{D})$. The complete framework for activation regularisation can be described in three discrete stages:

1. a transformation $\mathcal{T}(\cdot)$, referred to as activation transformation, is applied to the grid outputs $\boldsymbol{H}_t^{*(l)}$, which yields the transformed grid representation $\tilde{\boldsymbol{H}}_t^{*(l)}$.

2. a target pattern $\boldsymbol{G}_t^{(l)}$ is specified. Several concepts can be embedded in the target pattern, for example, interpretation or smoothness.

3. a regularisation function $\mathcal{R}(\boldsymbol{\theta};\mathbb{D})$ is applied to minimise the difference between the transformed grid representation $\tilde{\boldsymbol{H}}_t^{*(l)}$ and the target pattern $\boldsymbol{G}_t^{(l)}$.

In this section, each of these stages is described in detail. In training, it requires to calculate the gradient $\partial \frac{\mathcal{R}}{\partial h_{tij}^{*(l)}}$. Using the chain rule, it can be factorised into two components,

$$\frac{\partial \mathcal{R}}{\partial h_{tij}^{*(l)}} = \left\langle \frac{\partial \tilde{\boldsymbol{H}}_t^{*(l)}}{\partial h_{tij}^{*(l)}}, \frac{\partial \mathcal{R}}{\partial \tilde{\boldsymbol{H}}_t^{*(l)}} \right\rangle_F \tag{5.4}$$

where $\langle \cdot, \cdot \rangle_F$ is the matrix Frobenius inner product,

$$\langle \boldsymbol{A}, \boldsymbol{B} \rangle_F = \sum_{i,j} a_{ij} b_{ij}. \tag{5.5}$$

The calculation of the gradients $\frac{\partial \tilde{\boldsymbol{H}}_t^{*(l)}}{\partial h_{tij}^{*(l)}}$ and $\frac{\partial \mathcal{R}}{\partial \tilde{\boldsymbol{H}}_t^{*(l)}}$ is also be discussed below.

5.2.1 Activation Transformation

Given the activation grid, a transform can be applied to the outputs of the activation functions. This activation transformation can be used to yield a specific interpretation of the hidden-layer outputs. For instance, the outputs of activation functions can be normalised and transformed to form an activation "distribution", or a high-pass filtered activation "image". The transformed activation function outputs are defined as $\tilde{\boldsymbol{H}}_t^{*(l)}$, given by a transform $\mathcal{T}(\cdot)$ applied to $\boldsymbol{H}_t^{*(l)}$

$$\tilde{\boldsymbol{H}}_t^{*(l)} = \mathcal{T}(\boldsymbol{H}_t^{*(l)}). \tag{5.6}$$

There are multiple possible transforms $\mathcal{T}(\cdot)$. The most trivial option is the identity transform, which yields the original activation function output

$$\tilde{\boldsymbol{H}}_t^{*(l)} = \boldsymbol{H}_t^{*(l)}. \tag{5.7}$$

Three types of transforms are investigated in this thesis: the normalised activation, the probability mass function, and the high-pass filtering.

Normalised Activation

Using an identity transform may cause a problem in determining the importance of each activation function. Some of the activation function outputs only stay close to the extreme ends of the range of activation function. At the same time, the contribution that they make to the next layer depends on the parameters of the following layer. Therefore, both the output range of an activation function and its associated parameters to the next layer should be considered when evaluating the impact of a particular activation function. The normalised activation (Tan et al., 2015a) is proposed to address these problems. It is defined as

$$\tilde{h}^{*(l)}_{tij} = h^{*(l)}_{tij}\,\xi^{(l)}_{ij} \tag{5.8}$$

where the term $\xi^{(l)}_{ij}$ reflects the impact that the activation function has on the following layer $l+1$. This is expressed as

$$\xi^{(l)}_{ij} = \sqrt{\sum_k w^{(l+1)2}_{k,o(i,j)}}. \tag{5.9}$$

where $o(i,j)$ represents the original node index in $\boldsymbol{h}^{(l)}_t$ of the (i,j)-th grid unit. This provides a method to consider both aspects of the problem: the empirical range of the activation function, and the influence of the next-layer parameters. The gradient with respect to raw activation function outputs is given by

$$\frac{\partial \tilde{h}^{*(l)}_{tmn}}{h^{*(l)}_{tij}} = \begin{cases} \xi^{(l)}_{ij} & m=i \quad n=j, \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

This normalised activation can be integrated with other transformations, such as the probability mass function and the high-pass filtering presented below.

Probability Mass Function

The grid can also be treated as a discrete probability space. The activation function outputs can be transformed to yield a probability mass function (PMF). This probability mass function is defined as follows

$$\tilde{h}_{tij}^{*(l)} = \frac{h_{tij}^{*(l)}}{\sum_{u,v} h_{tuv}^{*(l)}} \tag{5.11}$$

where activation function outputs are normalised by their sum. There are some constraints that need to be satisfied for this PMF transform. To ensure that $\tilde{\boldsymbol{H}}_t^{*(l)}$ is a valid distribution, $\tilde{h}_{tij}^{*(l)}$ should be non-negative. This restricts the potential choices of activation function. Simple methods such as an $\exp(\cdot)$ wrapping can be utilised for an arbitrary function, but this may disable the effect of the negative range in the activation function. The gradient with respect to raw activation function outputs can be calculated via

$$\frac{\partial \tilde{h}_{tmn}^{*(l)}}{h_{tij}^{*(l)}} = \begin{cases} \dfrac{1}{\sum_{u,v} h_{tuv}^{*(l)}} - \dfrac{h_{tmn}^{*(l)}}{(\sum_{u,v} h_{tuv}^{*(l)})^2} & m = i \quad n = j, \\ -\dfrac{h_{tmn}^{*(l)}}{(\sum_{u,v} h_{tuv}^{*(l)})^2} & \text{otherwise.} \end{cases} \tag{5.12}$$

High-pass Filtering

The high-pass filtering transform is design to induce smoothness over the activation function outputs. It includes information about nearby units via a convolution operation,

$$\tilde{\boldsymbol{H}}_t^{*(l)} = \boldsymbol{H}_t^{*(l)} * \boldsymbol{K} \tag{5.13}$$

where $\boldsymbol{K}$ is a filter. The filter can take a range of forms. For example, a Gaussian high-pass filter, used in Wu et al. (2016b), assigns the impact of other nodes according to the distance; a simple 3×3 kernel, used in Xiong et al. (2016), only introduces adjacent nodes.

5.2.2 Target Pattern

Activation regularisation encourages the transformed activation function output $\tilde{\boldsymbol{H}}_t^{*(l)}$ to satisfy a target pattern $\boldsymbol{G}_t^{(l)}$. Two types of target pattern are a time-variant pattern defines target pattern $\boldsymbol{G}_t^{(l)}$ that depends on the data; and a time-invariant pattern that has a static pattern $\boldsymbol{G}^{(l)}$.

Time-variant Pattern

The activation grid can be split into a set of spatial regions. The meanings of regions can take a variety of concepts, such as phonemes, noise types or speaker variations. In this way, different grid regions can model and respond to different concepts in the data. The time-variant pattern is designed on the regions. On different types of data, a time-variant pattern can encourage the activation function outputs in the corresponding regions. This design aids the network interpretability, that is, a particular unit can be interpreted according to its location in the grid.

For example, phone-dependent patterns encourage the regions to model different phones. In each hidden layer, a set of phoneme (or grapheme) dependent target patterns is defined by the targets of training data. A point in this grid space is associated with each phone $/p/$, denoted as $\boldsymbol{s}_{/p/}$. These phoneme positions can be determined using a range of methods, such as t-SNE (Maaten and Hinton, 2008) using the acoustic feature means of the phones. It is then possible to apply a transform to target patterns in a similar fashion to its activation function output transform

$$g_{tij}^{(l)} = \frac{\exp\left(-\frac{1}{2\sigma^2}||\boldsymbol{s}_{ij} - \hat{\boldsymbol{s}}_{/p_t/}||_2^2\right)}{\sum_{m,n}\exp\left(-\frac{1}{2\sigma^2}||\boldsymbol{s}_{mn} - \hat{\boldsymbol{s}}_{/p_t/}||_2^2\right)} \tag{5.14}$$

where $\hat{\boldsymbol{s}}_{p_t}$ is the position in the grid space of the “correct” phoneme at time t, and the sharpness factor σ controls the sharpness of the surface of target pattern. For each phoneme, a Gaussian contour is defined at its nearby region in the grid. It encourages nodes to correspond to the same phoneme to be grouped in the same region. An example of phone-dependent target pattern is shown in Figure 5.2. The target pattern induces hidden units a deterministic ordering where activation functions with similar

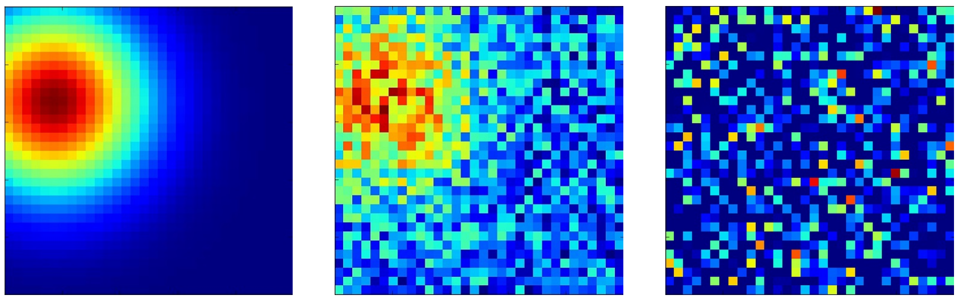

(a) Target pattern (b) Stimulated (c) Unstimulated

Fig. 5.2 Phone-dependent target pattern. It includes an example of target pattern and the corresponding activation function outputs yielded by stimulated and unstimulated DNNs. The models were trained on the Wall Street Journal data used for preliminary experiments in this section.

behaviours were grouped in nearby regions. This form of target pattern prevents the arbitrary ordering, which has the potential to improve the regularisation.

Time-invariant Pattern

Time variant patterns require time-dependent "labels" to be derived from the training data. Alternatively, time invariant patterns can be used to specify general, desirable, attributes of the network activation pattern. It can be expressed as

$$\boldsymbol{G}_t^{(l)} = \boldsymbol{G}^{(l)} \qquad \forall t. \tag{5.15}$$

This time-invariant pattern can induce the activation grid with a global concept.

5.2.3 Regularisation Function

Finally, a regularisation function $\mathcal{R}(\boldsymbol{\theta};\mathbb{D})$ is required to relate the transformed activation output $\tilde{\boldsymbol{H}}_t^{*(l)}$ and target pattern $\boldsymbol{G}_t^{(l)}$, which is usually applied in a layer-by-layer fashion,

$$\mathcal{R}(\boldsymbol{\theta};\mathbb{D}) = \frac{1}{|\mathbb{D}|}\sum_{t=1}^{|\mathbb{D}|}\sum_{l}\mathcal{D}(\tilde{\boldsymbol{H}}_t^{*(l)},\boldsymbol{G}_t^{(l)}) \tag{5.16}$$

where $\mathcal{D}(\tilde{\boldsymbol{H}}_t^{*(l)},\boldsymbol{G}_t^{(l)})$ measures the mismatch between the activation output and the target pattern for layer l. There are a range of approaches to defining the function

$\mathcal{D}(\cdot,\cdot)$. Three are investigated in this thesis: the mean squared error, the KL-divergence, and the cosine similarity.

Mean Squared Error

A simple way to measures the difference is the mean squared error (MSE) method, defined as

$$\mathcal{D}(\tilde{\boldsymbol{H}}_t^{*(l)}, \boldsymbol{G}_t^{(l)}) = ||\tilde{\boldsymbol{H}}_t^{*(l)} - \boldsymbol{G}_t^{(l)}||_F^2 \tag{5.17}$$

where $||\cdot||_F$ stands for the Frobenius norm of a matrix

$$||\boldsymbol{M}||_F = \sqrt{\sum_{i,j} m_{ij}^2}. \tag{5.18}$$

The gradient $\frac{\partial \mathcal{D}}{\partial \tilde{\boldsymbol{H}}_t^{(l)}}$ can be calculated by

$$\frac{\partial \mathcal{D}}{\partial \tilde{\boldsymbol{H}}_t^{(l)}} = 2\left(\tilde{\boldsymbol{H}}_t^{*(l)} - \boldsymbol{G}_t^{(l)}\right). \tag{5.19}$$

This method minimises the element-wise squared error between $\tilde{\boldsymbol{H}}_t^{(l)}$ and $\boldsymbol{G}_t^{(l)}$. For example, using the raw activation function output as the transformed one and a time-invariant target pattern $\boldsymbol{G}_t^{(l)} = \boldsymbol{0}$ yields

$$\mathcal{D}(\tilde{\boldsymbol{H}}_t^{*(l)}, \boldsymbol{G}_t^{(l)}) = ||\boldsymbol{H}_t^{*(l)}||_F^2. \tag{5.20}$$

This is similar to the standard L^2 regularisation, but it is applied to activation function outputs rather than the model weights. Alternatively, using a high-pass filtering and a zero time-invariant target pattern, $\boldsymbol{G}_t^{(l)} = \boldsymbol{0}$, yields

$$\mathcal{D}(\tilde{\boldsymbol{H}}_t^{*(l)}, \boldsymbol{G}_t^{(l)}) = ||\boldsymbol{H}_t^{*(l)} * \boldsymbol{K}||_F^2. \tag{5.21}$$

The output of an activation function is encouraged to be “smooth” to its nearby ones; thus, a smooth surface is formed in a local region of the grid.

The mean squared error regularisation works well when the elements in $\tilde{\boldsymbol{H}}_t^{*(l)}$ and $\boldsymbol{G}_t^{(l)}$ are in a similar range. However, this may require careful selection of the target patterns depending on the activation function. For activation functions with fixed ranges, such as sigmoid or tanh, $\boldsymbol{G}_t^{(l)}$ can be easily rescaled to an appropriate range. However, for activation functions such as ReLU, in which the range is not restricted, the rescaling on $\boldsymbol{G}_t^{(l)}$ tends to require empirical tuning.

KL-divergence

One way to address the dynamic range issue between $\tilde{\boldsymbol{H}}_t^{*(l)}$ and $\boldsymbol{G}_t^{(l)}$ is to combine the PMF transformation with distribution distances such as the KL-divergence method. The difference $\mathcal{D}(\tilde{\boldsymbol{H}}_t^{*(l)}, \boldsymbol{G}_t^{(l)})$ is the KL-divergence of the two distributions, the target pattern $\boldsymbol{G}_t$ and the activation distribution $\tilde{\boldsymbol{H}}_t^{*(l)}$,

$$\mathcal{D}(\tilde{\boldsymbol{H}}_t^{*(l)}, \boldsymbol{G}_t^{(l)}) = \sum_{i,j} g_{tij}^{(l)} \log\left(\frac{g_{tij}^{(l)}}{\tilde{h}_{tij}^{*(l)}}\right). \tag{5.22}$$

For example, by using the phoneme-dependent target pattern (Eq. 5.14) and the probability mass function (Eq. 5.11), the KL-divergence can spur different regions in the grid to correspond to different phonemes. The gradient $\frac{\partial \mathcal{D}}{\partial \tilde{\boldsymbol{H}}_t^{(l)}}$ can be calculated by

$$\frac{\partial \mathcal{D}}{\partial \tilde{h}_{tij}^{*(l)}} = \frac{g_{tij}^{(l)}}{\tilde{h}_{tij}^{*(l)}}. \tag{5.23}$$

The KL-divergence regularisation requires $\tilde{\boldsymbol{H}}^{*(l)}$ to be positive to yield a valid distribution. This limits the choices of activation function. There are several approaches to convert specific activation functions to be positive. For example, by using tanh+1, instead of tanh, in Eq. 5.11, the KL-divergence regularisation can manipulate the hyperbolic tangent function with a similar pattern as the sigmoid function. However, these methods require a pre-defined lower bound on the activation function, which cannot be applied in all cases.

Cosine Similarity

An alternative approach, the negative cosine similarity, can be used,

$$\begin{aligned}\mathcal{D}(\tilde{\boldsymbol{H}}_t^{*(l)}, \boldsymbol{G}_t^{(l)}) &= -\cos\left(\mathbf{vec}\left(\tilde{\boldsymbol{H}}_t^{*(l)}\right), \mathbf{vec}\left(\boldsymbol{G}_t^{(l)}\right)\right) \\ &= -\frac{\sum_{i,j} \tilde{h}_{tij}^{*(l)} g_{tij}^{(l)}}{\left\|\tilde{\boldsymbol{H}}_t^{*(l)}\right\|_F \left\|\boldsymbol{G}_t^{(l)}\right\|_F}\end{aligned} \tag{5.24}$$

where $\mathbf{vec}(\cdot)$ converts a matrix to a vector representation. This regularisation defines the similarity between two vectors in an inner-product space. It measures the difference as the angle between the transformed activation output vector $\mathbf{vec}\left(\tilde{\boldsymbol{H}}_t^{*(l)}\right)$ and the target pattern vector $\mathbf{vec}\left(\boldsymbol{G}_t^{(l)}\right)$. This supports all forms of activation function. The gradient $\frac{\partial \mathcal{D}}{\partial \tilde{h}_{tij}^{*(l)}}$ can be calculated by

$$\frac{\partial \mathcal{D}}{\partial \tilde{h}_{tij}^{*(l)}} = \frac{2\tilde{h}_{tij}^{*(l)} \sum_{m,n} \tilde{h}_{tmn}^{*(l)} g_{tmn}^{(l)}}{\left\|\tilde{\boldsymbol{H}}_t^{*(l)}\right\|_F^3 \left\|\boldsymbol{G}_t^{(l)}\right\|_F} - \frac{g_{tij}^{*(l)}}{\left\|\tilde{\boldsymbol{H}}_t^{*(l)}\right\|_F \left\|\boldsymbol{G}_t^{(l)}\right\|_F}. \tag{5.25}$$

5.3 Smoothness Method for Adaptation

In section 3.5, model-based adaptation methods for neural network models were discussed. To perform effective network adaptation, parameters of specific DNN components are re-estimated to compensate the model to work appropriately on the adaptation data. Because the parameters are not interpretable, many of model-based adaptation methods require a large number of parameters to be adapted. This causes issues in robust and rapid adaptation.

In the grid representation, the activation function outputs of a stimulated DNN can yield a smoothed surface. This smoothness property provides the opportunity for regularising the adaptation schemes. This section proposes the smoothness method for adapting stimulated DNNs. The LHUC adaptation scheme (Swietojanski and Renals, 2014) is used as an example for the discussion.

In the LHUC scheme, a speaker-dependent scaling factor $\alpha_i^{(ls)}$ is introduced independently to every activation function of each hidden layer,

$$h_i^{(ls)} = \alpha_i^{(ls)} h_i^{(l)} \tag{5.26}$$

where s stands for the speaker index. Scaling factors are introduced per activation; thus, the number of independent parameters to adapt is equal to the number of DNN hidden units. The lack of interpretable meanings in standard-trained DNN causes that scaling factors are modelled as independent components instead of groups based on functional similarities. However, in stimulated DNNs, $\tilde{\boldsymbol{H}}^{*(ls)}$ is regularised to behave as a smooth surface. That is, nearby activation functions in the spatial ordering are likely to perform analogously. Based on this property, the LHUC adaptation with smoothness regularisation can be performed, which aims to smooth the adapted activation outputs by spatial neighbours. This regularisation is achieved by a special adaptation regularisation term. Defining $\mathbf{\Lambda}^{(s)}$ as the speaker-dependent transform for LHUC, which consists of all scaling factors, gives

$$\mathbf{\Lambda}^{(s)} = \{\alpha_i^{(ls)}\}_{i,l}. \tag{5.27}$$

The adaptation regularisation term can be expressed as

$$\mathcal{R}_L(\mathbf{\Lambda}^{(s)};\mathbb{D}) = \frac{1}{2T}\sum_{t=1}^{|\mathbb{D}|}\sum_{l}\sum_{i,j}\sum_{m,n}\left(q_{mn}^{ij}\left(\tilde{h}_{tij}^{*(ls)} - \tilde{h}_{tmn}^{*(ls)}\right)^2\right) \tag{5.28}$$

where the constant q_{mn}^{ij} determines the importance of grid unit (m,n) to (i,j),

$$q_{mn}^{ij} = \frac{1}{Q_{ij}}\exp\left(-\frac{1}{2\sigma_L^2}||\boldsymbol{s}_{ij} - \boldsymbol{s}_{mn}||_2^2\right), \tag{5.29}$$

where the hyper-parameter σ_L specifies the distance-decay factor, and Q_{ij} is a normalisation term,

$$Q_{ij} = \sum_{\tilde{i},\tilde{j}}\exp\left(-\frac{1}{2\sigma_L^2}||\mathbf{s}_{ij} - \mathbf{s}_{\tilde{i}\tilde{j}}||_2^2\right). \tag{5.30}$$

	Train	H1-Dev	H1-Eval
Total (hrs)	15.2	0.8	0.9
#Uttr	7185	310	316
AvgUttr (secs)	7.6	9.4	10.1

Table 5.1 WSJ-SI84: Summary of training and evaluation sets. It includes the total hours, number of utterances (#Uttr) and average utterance duration (AvgUttr).

Thus, the overall adaptation criterion $\mathcal{F}(\mathbf{\Lambda}^{(s)})$ can be expressed as

$$\mathcal{F}(\mathbf{\Lambda}^{(s)};\mathbb{D}) = \mathcal{L}(\mathbf{\Lambda}^{(s)};\mathbb{D}) + \eta_L R_L(\mathbf{\Lambda}^{(s)};\mathbb{D}) \tag{5.31}$$

where the hyper-parameter η_L penalises the importance of the regularisation term. $\mathcal{L}(\mathbf{\Lambda}^{(s)};\mathbb{D})$ is a standard training criterion for adaptation.

The regularisation for adaptation is based on the smoothness property of activation functions in stimulated DNNs. For the LHUC scheme, the information from a unit's spatial neighbours are utilised to robustly smooth the scaling factors, helping to regularise the adaptation even when there is insufficient adaptation data.

5.4 Preliminary Experiments

This section reports the preliminary experiments for stimulated deep neural networks on the Wall Street Journal task (WSJ-SI84). The Wall Street Journal (Paul and Baker, 1992) is a medium-vocabulary continuous speech recognition task. It is identical to the clean set of the AURORA 4 task used in Section 4.7. The experiments in this section were conduct on the clean data, not the multi-style data, for the purpose of extracting phone-dependent target patterns from "pure" features without acoustic distortions. The 1994 H1-Dev and H1-Eval testsets (Woodland et al., 1995) were used for evaluation. A brief summary of the training and evaluation sets is presented in Table 5.1, including the total hours, number of utterances and average utterance duration.

5.4.1 Experimental Setup

Using a similar configuration as Section 4.7.1, a GMM-HMM system was trained to obtain the state alignments of the training data for DNN-HMM hybrid system. For the DNN baseline, the 468-dimensional input feature to the neural network was formed by the 52-dimensional PLP+Δ+$\Delta\Delta$+$\Delta\Delta\Delta$ in a context window of 9 frames. Three activation functions were used for three distinct DNNs: sigmoid, tanh and ReLU. For each form of activation function, the respective DNN consisted of five hidden layers with 1024 nodes in each layer. The DNN parameters were initialised using the mono-phone discriminative pre-training (Zhang and Woodland, 2015a) and further fine-tuned by back-propagation on the CE criterion. L^2 regularisation was used during the training phase for the baseline DNN systems as well as the stimulated DNN systems. Also, a DNN system with dropout (Srivastava et al., 2014) was trained and the present probability (Eq. 2.64) was set to 0.8. Relevant hyper-parameters were tuned on the H1-Dev testset. In evaluation, decoding was performed with the WSJ tri-gram language model. Detailed information can be found in Woodland et al. (1995).

The stimulated DNNs were trained using similar configurations as the baseline DNN. The network consisted of five hidden layers with 1024 units on each layer, forming a 32×32 grid. Activation regularisation was perform on all hidden layers, and the investigation on stimulated DNNs included three types of activation regularisation:

1. **KL**: The KL system used the KL-divergence regularisation (Eq. 5.22) with the activation PMF (Eq. 5.11) and the phoneme-dependent target pattern (Eq. 5.14).

2. **Cos**: The Cos system used the cosine similarity regularisation (Eq. 5.24) with the normalised activation (Eq. 5.8) and the phoneme-dependent target pattern (Eq. 5.14).

3. **Smooth**: The smooth system used the mean-squared-error regularisation (Eq. 5.17) with the high-pass filtering activation transformation (Eq. 5.13) and the zero target pattern (Eq. 5.21). A 3×3 kernel was used as the high-pass filter, in which the central tap was 1 and others were -0.125. In this way, the activation function outputs were smoothed with their adjacent ones in the grid; and a smooth surface was formed over the activation grid.

For the KL and Cos systems where time-variant target patterns were required, 46 English phonemes were used to define the time-variant target patterns, and 2D positions of the phonemes were estimated via the t-SNE method (Maaten and Hinton, 2008) over the average of frames of different phonemes. They were then scaled to fit in a

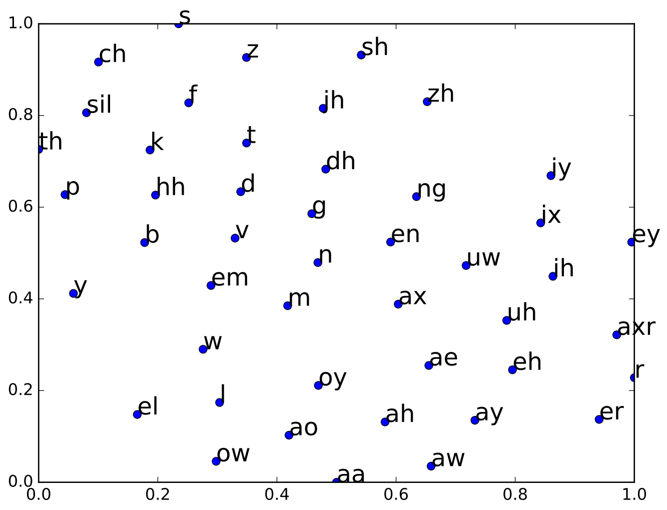


Fig. 5.3 WSJ-SI84: 2D mapping of English phonemes via t-SNE.

unit square $[0, 1] \times [0, 1]$. Figure 5.3 illustrates the phoneme positions. For the Cos and Smooth systems, the sigmoid, ReLU and tanh DNNs were investigated. For the KL system, only the sigmoid DNN was investigated, due to the positive constraint of activation function required by the KL-divergence regularisation function.

5.4.2 Results and Discussion

This section discusses the performance of several forms of stimulated DNNs. The KL activation regularisation is investigated first. Three types of activation regularisation are then discussed.

KL Activation Regularisation

The first experiment investigated the impact of the normalised activation that is defined in Eq. 5.8. As discussed in Section 5.2.1, the normalised activation (Eq. 5.8) can be combined with other activation transformations. The combination of the normalised

Regularisation	NormAct	WER (%)
L^2	–	10.0
KL	✗	9.9
	✓	9.7

Table 5.2 WSJ-SI84: Recognition performance (WER %) of stimulated DNNs using the KL regularisation with and without normalised activation (NormAct) on H1-Dev.

σ^2 \ η	0.1	0.2	0.3	0.5
0.05	10.0	9.9	–	–
0.1	9.9	9.7	10.1	10.6
0.2	9.9	9.8	–	–

Table 5.3 WSJ-SI84: Recognition performance (WER %) of sigmoid stimulated DNNs using the KL regularisation on H1-Dev. Different settings on the regularisation penalty η and the sharpness factor σ^2 are compared.

activation and the PMF activation transformation (Eq. 5.11) was examined on the KL system, as shown in Table 5.2. By combining the normalised activation, the KL system could further reduce the word error rate. Similar results were also found in the Cos and Smooth systems.

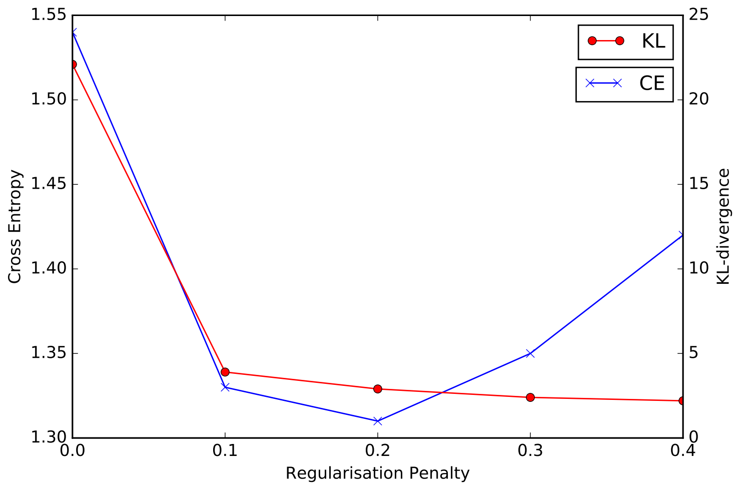


Fig. 5.4 WSJ-SI84: Cross-entropy and KL-divergence values of the CV set on different regularisation penalties.

Regularisation	H1-Dev	H1-Eval
L^2	10.0	10.2
Dropout	10.0	**10.0**
KL	**9.7**	**10.0**

Table 5.4 WSJ-SI84: Recognition performance (WER %) of stimulated DNNs using the KL regularisation.

Table 5.3 compares the impact of different configurations of regularisation penalty, η, and sharpness factor (Eq. 5.14), σ, on the H1-Dev evaluation set. The sharpness factor determines how many activation functions in the grid were encouraged to activate by the regularisation. The best performance was achieved by setting η to 0.2 and σ^2 to 0.1. For $\sigma^2 = 0.1$, it encouraged approximately 322 activation functions in each layer to model a specific type of phonemes. Figure 5.4 illustrates the cross entropy and KL-divergence values of the cross validation set on different η settings by fixing σ^2 as 0.1. As discussed in Section 2.4, the performance on the CV set can be viewed as an indicator of the generalisation error. The minimal cross-entropy value was achieved when η were in the range between 0.1 and 0.2, which is consistent with the optimal decoding performance on H1-Dev.

Table 5.4 summarises the recognition performance of the KL system on both test sets. The KL system outperformed the default DNN system with the L^2 regularisation, reducing the relative WER up to 3%. On the H1-Dev, it outperformed the dropout regularisation; however, it yielded a similar result as the dropout on H1-Eval.

Comparison of Activation Regularisations

Next, different types of activation regularisation on stimulated DNNs were investigated. Figure 5.5 compares the cross-entropy values of stimulated DNNs (sigmoid) using KL, Cos and Smooth activation regularisations on the training and CV sets. By using the activation regularisations, the generalisation error (i.e. the CV-set cross entropy value) was reduced. This indicates that activation regularisation can improve the regularisation in DNN training. Figure 5.6 shows the outputs of the third-hidden-layer activation grid of raw, KL, cos and smooth sigmoid DNNs on an “ay” frame sample. As expected, both the KL and Cos systems yielded the target patterns: the activation

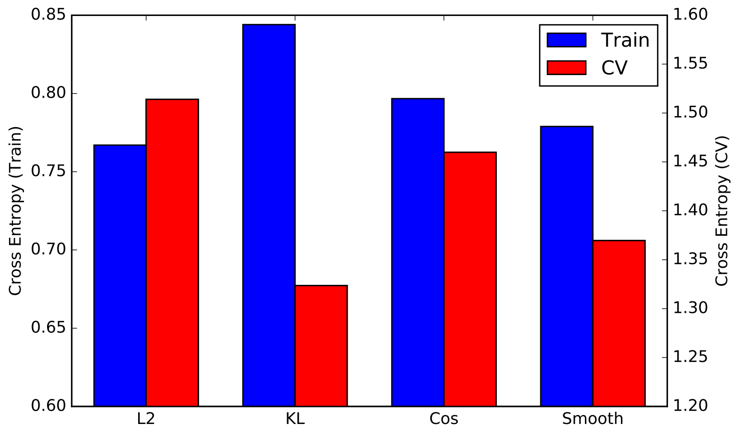


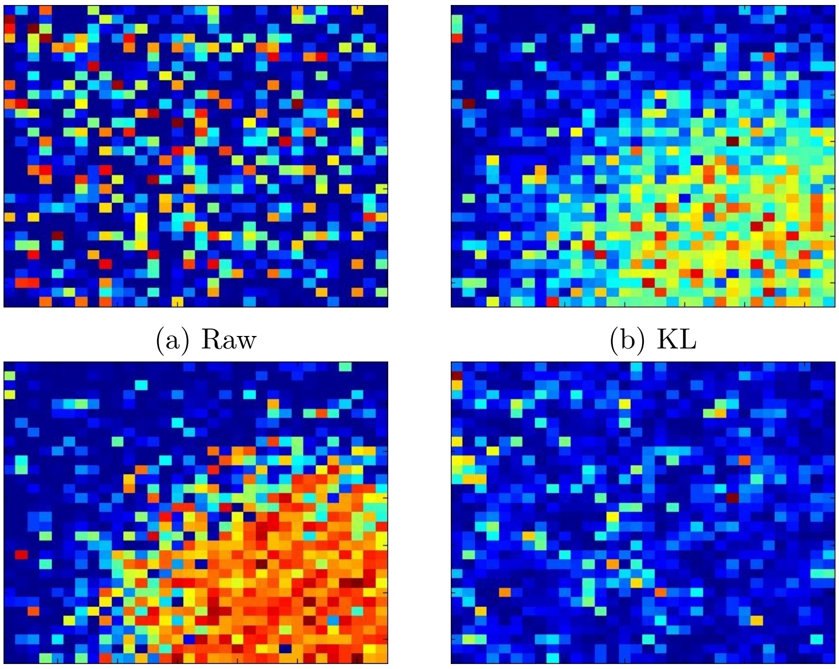
Fig. 5.5 WSJ-SI84: CE values of training and CV sets using different activation regularisations on sigmoid stimulated DNNs.

Regularisation	Sigmoid	ReLU	Tanh
L2	10.0	10.9	10.5
KL	9.7	–	–
Cos	9.7	10.5	10.5
Smooth	9.8	10.8	10.4

Table 5.5 WSJ-SI84: Recognition performance (WER %) of of stimulated DNNs using different activation regularisations on H1-Dev.

functions around the phoneme "ay" location echoed higher values than other regions. The Smooth system, which does not have a specific target, just yielded a smoothed pattern. The presented grid outputs matched with the corresponding target patterns, indicating the effectiveness of activation regularisation to induce the behaviour of activation function.

Tables 5.5 and 5.6 summarises the decoding performance of different stimulated DNNs on the H1-Dev and H1-Eval testsets. The Cos and Smooth regularisation on sigmoid, ReLU and tanh DNNs yielded similar performance. On this relatively small task, small consistent gains could be obtained.


(a) Raw (b) KL

(c) Cos (d) Smooth

Fig. 5.6 WSJ-SI84: Comparison of activation grid outputs of raw, KL, Cos and Smooth systems on an "ay" frame.

5.5 Summary

Stimulated deep neural networks were presented in this chapter. This type of structured neural network relates activation functions in regions of the network to aid interpretation and visualisation. In the network topology of stimulated DNN, hidden units are reorganised to form a grid, and activation functions with similar behaviours are then grouped together in this grid space. This goal is obtained by introducing a special form of regularisation, which is the activation regularisation. The activation regularisation is designed to encourage the outputs of activation functions to satisfy a target pattern. By defining appropriate target patterns, different learning, partitioning or grouping concepts can be imposed and shaped on the network. This design prevents hidden units from an arbitrary order, which has the potential to improve network regularisation. Also, based on the restricted ordering of hidden units, smoothness techniques can be used to improve the adaptation schemes on stimulated DNNs. This chapter used the

Regularisation	Sigmoid	ReLU	Tanh
L2	10.2	10.9	11.2
KL	10.0	–	–
Cos	10.1	10.8	11.0
Smooth	10.1	10.8	10.9

Table 5.6 WSJ-SI84: Recognition performance (WER %) of stimulated DNNs using different activation regularisations on H1-Eval.

LHUC adaptation approach as an example to explain how the smoothness method can be performed.

Chapter 6

Deep Activation Mixture Model

In the previous chapter, stimulated deep neural networks were presented. This type of structured DNN imposes the hidden-layer representation to some desired target pattern using the activation regularisation. The concept of target pattern can be viewed as a "reference model", which roughly controls a general behaviour for activation functions. Activation regularisation is an implicit stimulating mechanism that induces activation functions to learn separate parts of the target pattern by themselves.

This chapter proposes deep activation mixture models (DAMMs) (Wu and Gales, 2017). Inspired by stimulated DNNs, the DAMM is also designed to relate the hidden units in regions of the network. The activation functions in a DAMM are modelled as the sum of a mixture and residual models. The mixture model expands an activation contour that roughly describes a general behaviour for activation functions. Rather than being implemented as a regularisation term in stimulated DNNs, the induced behaviour of activation functions in DAMMs are controlled by distinct network structures, i.e. mixture models. The residual model specifies a fluctuation term for each activation function on this contour. Consequently, the resultant activation functions stay on a smooth contour controlled by the mixture model, which triggers activation functions of nearby hidden units to be similar. The introduction of mixture model in DAMM can be viewed as an informed regularisation that controls a dynamic prior pattern for the activation functions. The activation functions in the DAMM are related and controlled, which has the potential to improve network regularisation. In addition, the highly restricted nature of the mixture model allows it to be robustly re-estimated. This design

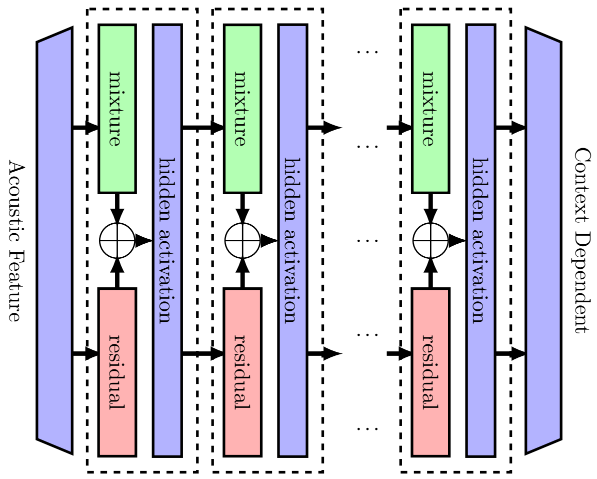


Fig. 6.1 Deep activation mixture model.

enables a novel approach to network adaptation, even when there is limited adaptation data. In contrast to mixture density networks (Bishop, 1994; Richmond, 2006; Variani et al., 2015; Zen and Senior, 2014; Zhang et al., 2016a), deep mixture activation models utilise the contour of mixture-model distributions, instead of estimating density functions in a "deep" configuration.

The discussion in this chapter includes the network topology, parameter training and adaptation methods for deep activation mixture models.

6.1 Network Topology

The deep activation mixture model extends activation functions in one hidden layer with a shared component to depict their relations. The DAMM introduces structures onto the activation functions, and the activation functions are formed as the combination of two separate components: mixture model; and residual model. The mixture model forms a smooth contour that models the general behaviours of activation functions. The residual model specifies a fluctuation around the contour for each activation function. This scheme can be applied to different network architectures. In the discussion of the proposed model, this chapter takes the feed-forward neural network as an example.

The topology of deep activation mixture model is illustrated in Figure 6.1. In each hidden layer of the DAMM, the hidden units are reorganised to form an activation

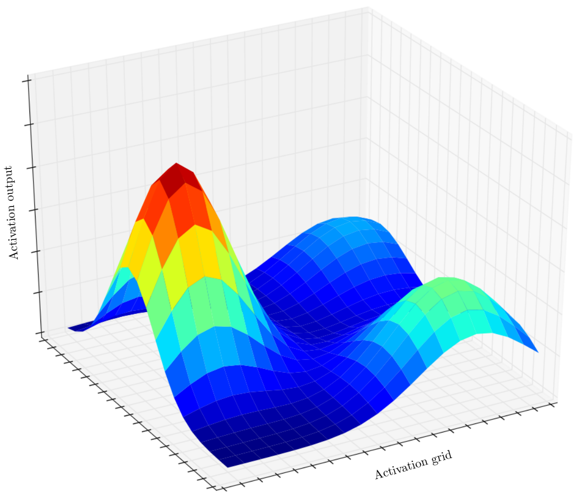


Fig. 6.2 The effect of mixture model in DAMM.

grid, which is the same as that in stimulated DNNs (Section 5.1). In the l-th hidden layer, the output of the activation activations $\boldsymbol{h}_t^{(l)}$ is defined as the sum of a mixture model $\boldsymbol{h}_t^{(\mathrm{mix},l)}$ and a residual model $\boldsymbol{h}_t^{(\mathrm{res},l)}$.

$$\boldsymbol{h}_t^{(l)} = \boldsymbol{h}_t^{(\mathrm{mix},l)} + \boldsymbol{h}_t^{(\mathrm{res},l)} \qquad 1 \leq h < L. \tag{6.1}$$

The effect of mixture model is illustrated in Figure 6.2. It can be viewed as an informed regularisation on activation functions. The contour, dynamically generated by the mixture model, forces the outputs of activation functions to stay around it. By turning off the mixture model, the DAMM with the residual model only will degrade to a standard DNN. The outputs of activation functions of a standard DNN can be viewed as fluctuations over a static plane that depends on the choice of activation function. For example, if tanh is used, the plane is located at zero; if sigmoid is used, the plane is located at 0.5. L^2 regularisation is usually used in network training and its effect can be viewed as encouraging the activation function outputs to stay around the static plane. Though regularisation can often be achieved, there is no feasible meaning of the static plane. Instead, in DAMM, the use of mixture model extends the static plane to a dynamic surface that can inform activation functions about rough outputs. This informed design has the potential to improve the network regularisation. The mixture and residual models are defined below.

- **Mixture model:** The rough outputs of activation functions is governed by the mixture model. It is modelled as the contour of the PDF of a Gaussian mixture distribution (Figure 6.2),

$$h_{ti}^{(\mathrm{mix},l)} = \zeta_t^{(l)} \sum_{k=1}^{K} \pi_{tk}^{(l)} \mathcal{N}\left(\boldsymbol{s}_i; \boldsymbol{\mu}_k^{(l)}, \boldsymbol{\Sigma}_k^{(l)}\right) \tag{6.2}$$

  where K stands for the number of Gaussian components. $\zeta_t^{(l)}$ is a scaling factor to specify the importance of the mixture model

$$\zeta_t^{(l)} = \mathrm{sig}\left(\boldsymbol{g}^{(l)\mathrm{T}} \boldsymbol{h}_t^{(l-1)} + r^{(l)}\right). \tag{6.3}$$

  where $\boldsymbol{g}^{(l)}$ and $r^{(l)}$ are the associated parameters. It is modelled by a sigmoid function to perform in the range $(0,1)$. This scaling factor is introduced to dynamically scale the output of mixture model in an appropriate range.

  The mixing weights $\boldsymbol{\pi}_t^{(l)}$ for different mixture components are modelled as a softmax function,

$$\pi_{tk}^{(l)} = \frac{\exp\left(\boldsymbol{a}_k^{(l)\mathrm{T}} \boldsymbol{h}_t^{(l-1)} + c_k^{(l)}\right)}{\sum_{\tilde{k}} \exp\left(\boldsymbol{a}_{\tilde{k}}^{(l)\mathrm{T}} \boldsymbol{h}_t^{(l-1)} + c_{\tilde{k}}^{(l)}\right)} \tag{6.4}$$

  where $\boldsymbol{A}^{(l)}$ and $\boldsymbol{c}^{(l)}$ are its associated parameters. To make $\boldsymbol{\pi}_t^{(l)}$ a valid distribution, the softmax function is used, by which the sum-to-one and positive constraints on the mixture weights are satisfied. The Gaussian mean vectors $\{\boldsymbol{\mu}_k^{(l)}\}$ and covariance matrices $\{\boldsymbol{\Sigma}_k^{(l)}\}$ can be introduced with desired interpolations. For example, by setting $\{\boldsymbol{\mu}_k^{(l)}\}$ as the 2D projection of phonemes, the regions in the DAMM activation grid can be induced with phone meanings.

- **Residual model:** The mixture model introduces minimal numbers of parameters, enabling smoothness and regularisation for DAMM. The residual model is used to model the precise behaviour for each activation function. It can be viewed as fluctuations over the contour, which can either be positive or negative. Therefore, the activation function associated with the residual model is a hyperbolic tangent function,

$$\boldsymbol{h}^{(\mathrm{res},l)} = \mathbf{tanh}\left(\boldsymbol{W}^{(l)\mathrm{T}}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right) \tag{6.5}$$

where $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are parameters. It describes precise variations for different locations over the contour, enriching the expressiveness of every activation function.

The total number of Gaussian components is usually smaller than that of units in one hidden layer. This mixture model is highly restricted, due to fewer parameters associated with it. This compact property allows the mixture model to be robustly re-estimated, which is suitable for network post-modifications, such as speaker adaptation (Section 6.3).

6.2 Parameter Training

To train a deep activation mixture model, two sets of parameters need to be optimised: the parameter set of the mixture model $\boldsymbol{\theta}^{\mathrm{mix}}$ and that of the residual model $\boldsymbol{\theta}^{\mathrm{res}}$. They are, respectively, defined as

$$\boldsymbol{\theta}^{\mathrm{mix}} = \left\{\boldsymbol{g}^{(l)}, r^{(l)}, \boldsymbol{A}^{(l)}, \boldsymbol{c}^{(l)}\right\}_{1\leq l<L,} \tag{6.6}$$

$$\boldsymbol{\theta}^{\mathrm{res}} = \left\{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\right\}_{1\leq l\leq L.} \tag{6.7}$$

Note that, mean vectors $\{\boldsymbol{\mu}_k^{(l)}\}$ and covariance matrices $\{\boldsymbol{\Sigma}_k^{(l)}\}$ of the Gaussian components are fixed during the training phase of DAMM. This configuration forces the mixture model to form a significant contour. According to experiments in joint training with mean and covariance (Section 6.4.2), the optimisation is expected to deactivate the effect of the mixture model. This phenomenon is harmful to the overall model. However in Section 6.3, these parameters are used as the speaker-dependent transform to adapt a well-trained DAMM.

The residual model contains much more parameters than the mixture model. If a simple joint training scheme is performed, the effect of the mixture model can be easily

Algorithm 5 Isolating mode of parameter training for DAMM.

```
1: for $l := 1$ to $L$ do
2:     initialise $\boldsymbol{\theta}^{(\text{res},l)} = \mathbf{0},\ \boldsymbol{\theta}^{(\text{mix},l)}$
3:     finetune $\boldsymbol{\theta}^{\text{mix}}$
4:     update $\boldsymbol{\theta}^{(\text{res},l)}$
5: end for
6: finetune $\boldsymbol{\theta}^{\text{res}}$
```

absorbed by that of the residual model. It causes the mixture model unable to generate a sensible activation contour to regularise the activation function outputs. To emphasise the informed regularisation, the mixture model should be trained to its maximal extent. Parameter training for DAMM should be organised appropriately for these concerns, in addition to optimising the primary training criterion. The outline of this training mode is described in Algorithm 5. The DAMM is constructed a layer-wise manner. During the construction phase (Line 1–5), the l-th iteration first initialises and adds parameters for the mixture and residual models for the l-th layer, denoted as $\boldsymbol{\theta}^{(\text{mix},l)}$ and $\boldsymbol{\theta}^{(\text{res},l)}$, respectively. The parameters of the mixture model are randomly initialised to break the modelling symmetry. In contrast, the parameters of the residual model are initialised as zeros, and this zero initialisation acts as an implicit regularisation to encourage small parameters. The update of the mixture model is performed till convergence (referred to as finetune). The residual model is fully optimised at last (Line 6). Intuitively, this isolating mode specifies a “curriculum” to train different network components in a pre-defined order. The complete network is constructed greedily from shallow to deep, which is similar to layer-wise DNN pre-training. In each layer, the mixture model is introduced at first, and optimised till convergence, prior to the introduction of the residual one. This design ensures that the mixture model is trained to its maximal extent.

Parameter training for DAMM is again designed in the stochastic gradient descent and error back-propagation. In this paper, the overall training criterion is defined as

$$\mathcal{F}(\boldsymbol{\theta}^{\text{mix}},\boldsymbol{\theta}^{\text{res}};\mathbb{D}) = \mathcal{L}(\boldsymbol{\theta}^{\text{mix}},\boldsymbol{\theta}^{\text{res}};\mathbb{D}) + \eta\mathcal{R}(\boldsymbol{\theta}^{\text{res}};\mathbb{D}) \tag{6.8}$$

where the term $\mathcal{R}(\boldsymbol{\theta}^{\text{res}},\mathbb{D})$ stands for an L^2 regularisation term with hyper-parameter η. For DAMMs, the regularisation is only used on parameters of the residual model $\boldsymbol{\theta}^{\text{res}}$,

$$\mathcal{R}(\boldsymbol{\theta}^{\text{res}};\mathbb{D}) = \frac{1}{2}\sum_{l}\sum_{i}\left(b_i^{(l)2} + \sum_{j} w_{ij}^{(l)2}\right). \tag{6.9}$$

L^2 regularisation helps to penalise large parameters. Compared with standard L^2 regularisation that restricts activation functions on a flat plane, the underlying “plane” of DAMM activation functions is not fixed, that is, the residual model is regularised on the dynamic mixture-model contour. This design can be viewed as an extension to the L^2 regularisation. It enriches the flexibility of L^2 regularisation, which can further improve the parameter regularisation.

The gradients required for training can be calculated by error back-propagation. For residual-model parameters, related gradients $\frac{\partial \mathcal{F}}{\partial \boldsymbol{W}^{(l)}}$ and $\frac{\partial \mathcal{F}}{\partial \boldsymbol{b}^{(l)}}$ can be recursively computed using the following term (using Eq. 2.52 and 2.53),

$$\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}_t^{(\text{res},l)}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}_t^{(l)}}. \tag{6.10}$$

For parameters in the mixture model, the calculation of $\frac{\partial \mathcal{F}}{\boldsymbol{g}^{(l)}}$ and $\frac{\partial \mathcal{F}}{r^{(l)}}$ can be performed using

$$\frac{\partial \mathcal{F}}{\partial \zeta_t^{(l)}} = \sum_{k}\pi_{tk}^{(l)}\sum_{i}\mathcal{N}\left(\boldsymbol{s}_i;\boldsymbol{\mu}_k^{(l)},\boldsymbol{\Sigma}_k^{(l)}\right)\frac{\partial \mathcal{F}}{\partial h_{ti}^{(l)}}. \tag{6.11}$$

To calculate $\frac{\partial \mathcal{F}}{\partial \boldsymbol{A}^{(l)}}$ and $\frac{\partial \mathcal{F}}{\boldsymbol{c}^{(l)}}$, it requires

$$\frac{\partial \mathcal{F}}{\partial \pi_{tk}^{(l)}} = \zeta_t^{(l)}\sum_{i}\mathcal{N}\left(\boldsymbol{s}_i;\boldsymbol{\mu}_k^{(l)},\boldsymbol{\Sigma}_k^{(l)}\right)\frac{\partial \mathcal{F}}{\partial h_{ti}^{(l)}}. \tag{6.12}$$

These gradients can then be integrated with stochastic gradient descent to perform the network training.

6.3 Adaptation

This section discusses the adaptation on a well-trained DAMM. In standard DNN configurations, since there is no explicit meanings of activation functions, independent, not tied, parameters are often introduced for each activation functions to handle the adaptation. In comparison, the DAMM uses mixture models to form the rough outputs of activation functions. The adaptation of mixture models can affect all the activation functions in a "tied" fashion. In the thesis, the adaptation of mixture model is performed on Gaussian components. In the adaptation phase, the change of the contour should effectively adapt the DAMM to an unseen speaker. The outputs of adapted activation functions can be expressed as

$$\boldsymbol{h}_t^{(ls)} = \boldsymbol{h}_t^{(\text{mix},ls)} + \boldsymbol{h}_t^{(\text{res},l)} \tag{6.13}$$

where s stands for the speaker index. The mixture model $\boldsymbol{h}_{\text{mix}}^{(ls)}$ is adapted to speaker s by mean vectors and covariance matrices of Gaussian components

$$h_{ti}^{(\text{mix},ls)} = g^{(l)} \sum_{k=1}^{K} c_k^{(l)} \mathcal{N}\left(\boldsymbol{s}_i; \boldsymbol{\mu}_k^{(ls)}, \boldsymbol{\Sigma}_k^{(ls)}\right). \tag{6.14}$$

In the context to adapt a DAMM, the canonical model includes parameters of affine transformations in mixture and residual models for all layers (optimised in training)

$$\mathcal{M} = \boldsymbol{\theta}^{\text{res}} \cup \boldsymbol{\theta}^{\text{mix}}. \tag{6.15}$$

The speaker-dependent transform $\Lambda^{(s)}$ consists of mean vectors and covariance matrices of all mixture components for all layers

$$\Lambda^{(s)} = \left\{\boldsymbol{\mu}_k^{(ls)}, \boldsymbol{\Sigma}_k^{(ls)}\right\}_{1\le l\le L, 1\le k\le K.} \tag{6.16}$$

The re-estimation of $\boldsymbol{\mu}_k^{(ls)}$ and $\boldsymbol{\Sigma}_k^{(ls)}$ changes the contour of the Gaussian mixture model, which affects all activation functions to some level in this layer.

To perform effective adaptation, the mean vector and covariance matrix of any Gaussian component are parametrised as follows. Mean vector $\boldsymbol{\mu}_k^{(ls)}$ can be used as SD parameters directly. However, to make $\boldsymbol{\Sigma}_k^{(ls)}$ a valid covariance matrix, it should satisfy the positive-definite property. This requirement can be satisfied by constrained optimization methods. Alternatively, the 2D grid configuration presented in this chapter allows a simple optimisation scheme. In this 2D configuration, $\boldsymbol{\Sigma}_k^{(ls)}$ stands for a 2×2 covariance matrix of a bivariate Gaussian PDF, thus can be factorised as

$$\boldsymbol{\Sigma}_k^{(ls)} = \begin{vmatrix} \sigma_{k1}^{(ls)2} & \rho_k^{(ls)}\sigma_{k1}^{(ls)}\sigma_{k2}^{(ls)} \\ \rho_k^{(ls)}\sigma_{k1}^{(ls)}\sigma_{k2}^{(ls)} & \sigma_{k2}^{(ls)2} \end{vmatrix} \tag{6.17}$$

where $\boldsymbol{\sigma}_k^{(ls)}$ represents the unit variance vector that should be positive and $\rho_k^{(ls)}$ is the correlation coefficient that should lay in the range $[-1,1]$. They can be parametrised as

$$\boldsymbol{\sigma}_k^{(ls)} = \exp\left(\tilde{\boldsymbol{\sigma}}_k^{(ls)}\right), \tag{6.18}$$

$$\rho_k^{(ls)} = \tanh\left(\tilde{\rho}_k^{(ls)}\right) \tag{6.19}$$

to comply with the mathematical constraints. $\tilde{\boldsymbol{\sigma}}_k^{(l)}$ and $\tilde{\rho}_k^{(l)}$ are then used as parameters instead of the raw unit variance and correlation coefficient. By using the matrix form in Eq. 6.17, the positive-definite property of $\boldsymbol{\Sigma}_k^{(ls)}$ can inherently be satisfied, requiring no additional constraints during optimisation.

Given adaptation data and criterion $\mathcal{F}(\Lambda^{(s)};\mathbb{D})$, the speaker-dependent transform can be re-estimated by stochastic gradient descent. Define a vector $\boldsymbol{\lambda}_k^{(ls)}$ consisting of the five adaptable parameters (mean, unit variance and correlation coefficient) of the k-th Gaussian mixture component in the l-th layer,

$$\boldsymbol{\lambda}_k^{(ls)} = \left[\mu_{k1}^{(ls)}, \mu_{k2}^{(ls)}, \sigma_{k1}^{(ls)}, \sigma_{k2}^{(ls)}, \rho_k^{(ls)}\right]^T. \tag{6.20}$$

The gradients are calculated by

$$\frac{\partial \mathcal{F}}{\partial \boldsymbol{\lambda}_k^{(ls)}} = \sum_t \zeta_t^{(l)} \pi_{tk}^{(l)} \sum_i \frac{\partial \tilde{\mathcal{N}}_i}{\partial \boldsymbol{\lambda}_k^{(ls)}} \mathcal{N}\left(\boldsymbol{s}_i; \boldsymbol{\mu}_k^{(ls)}, \boldsymbol{\Sigma}_k^{(ls)}\right) \frac{\partial \mathcal{F}}{\partial h_{ti}^{(ls)}} \tag{6.21}$$

where $\frac{\partial \tilde{\mathcal{N}}_i}{\partial \boldsymbol{\lambda}_k^{(ls)}}$ represents an expression with respect to mean, unit variance and correlation coefficient[1]

$$\frac{\partial \tilde{\mathcal{N}}_i}{\partial \boldsymbol{\mu}_k} = (\boldsymbol{\Sigma})^{-1}(\boldsymbol{s}_i - \boldsymbol{\mu}_k), \tag{6.22}$$

$$\frac{\partial \tilde{\mathcal{N}}_i}{\partial \sigma_{k1}} = \frac{1}{\sigma_{k1}} + \frac{(s_{i1}-\mu_{k1})(s_{i2}\rho\sigma_{k1} + \sigma_{k2}\mu_{k1} - \sigma_{k2}s_{i1} - \rho_k\mu_{k2}\sigma_{k1})}{(\rho^2-1)\sigma_{k1}^3\sigma_{k2}}, \tag{6.23}$$

$$\frac{\partial \tilde{\mathcal{N}}_i}{\partial \sigma_{k2}} = \frac{1}{\sigma_{k2}} + \frac{(s_{i2}-\mu_{k2})(s_{i1}\rho\sigma_{k2} + \sigma_{k1}\mu_{k2} - \sigma_{k2}s_{i2} - \rho_k\mu_{k1}\sigma_{k2})}{(\rho^2-1)\sigma_{k2}^3\sigma_{k1}}, \tag{6.24}$$

$$\begin{aligned}\frac{\partial \tilde{\mathcal{N}}_i}{\partial \rho_k} =& \frac{(s_{i2}\rho_k\sigma_{k1} - s_{i1}\sigma_{k2} - \rho_k\mu_{k2}\sigma_{k1} + \mu_{k1}\sigma_{k2})(s_{i1}\rho_k\sigma_{k2} - s_{i2}\sigma_{k1} - \rho_k\mu_{k1}\sigma_{k2} + \mu_{k2}\sigma_{k1})}{(\rho_k^2-1)^2\sigma_{k1}^2\sigma_{k2}^2} \\ &+ \frac{\rho_k}{1-\rho_k^2}\end{aligned} \tag{6.25}$$

In the adaptation, these parameters can be partially re-estimated, e.g. to only re-estimate mean vectors for a compact SD transform.

6.4 Preliminary Experiments

This section describes the preliminary experiments for deep activation mixture models on the Wall Street Journal task that has been used in Section 5.4.

6.4.1 Experimental Setup

The baseline was identical to that in Section 5.4.1. The sigmoid and tanh DNN systems were used as baseline DNN systems. For DAMMs, the network consisted of five hidden layers with 1024 hidden units on each layer, which formed a 32×32 grid. On each hidden layer, the mixture model included 48 Gaussian components, and each component was interpreted as a English phoneme. The mean vectors of the Gaussian

[1]To simply the expressions, the superscript "(ls)" is omitted in Eq. 6.22 $\sim$ 6.25.

System	Comp-updt	Dev03	Eval03
DNN (tanh)	–	10.5	11.2
DNN (sigmoid)	–	10.0	10.2
DAMM	✗	**9.8**	**10.1**
	✓	10.1	10.2

Table 6.1 WSJ-SI84: Recognition performance (WER %) of SI DAMMs with and without updating the Gaussian component parameters (Comp-updt) in training.

components were given by the 2D projection described in Section 5.4.1. Every $\rho_k^{(l)}$ was set to 0 and $\boldsymbol{\sigma}_k^{(l)}$ was empirically set to $\left[\sqrt{0.1}, \sqrt{0.1}\right]$, i.e. setting the unit variance to 0.1. This model configuration has a comparable number of parameters as the baseline DNN system. The cross-entropy DAMM model was initialised and well-tuned in the isolating training mode as shown in Algorithm 5. On each layer, the mixture model was fully optimised prior to the introduction of the residual model, and the residual model was updated for three iterations. The penalty of residual-model L^2 regularisation η was set to 10^{-4}.

6.4.2 Results and Discussion

This section discusses the results of different configurations and setups for deep activation mixture models.

As discussed in Section 6.2, to induce a significant contour, the mean vectors and covariance matrices of the Gaussian components in all layers should be fixed in training. To explain the reasons for this design, the first experiment compared the DAMMs trained with and without the parameters of the Gaussian components. The recognition performance of the SI DAMM systems is shown in Table 6.1. As discussed in Section 6.1, the DNN system using tanh activation functions can be viewed as an extreme case of DAMM where the mixture model is turned off; thus, it can be viewed as the DNN baseline. Both DAMMs outperformed the tanh DNN; and the DAMM with fixed Gaussian components in training slightly outperformed the sigmoid DNN. This shows that the informed regularisation controlled by the mixture model contributes to a better network regularisation. By updating the parameters of the Gaussian components in training, the performance of the DAMM degraded on both

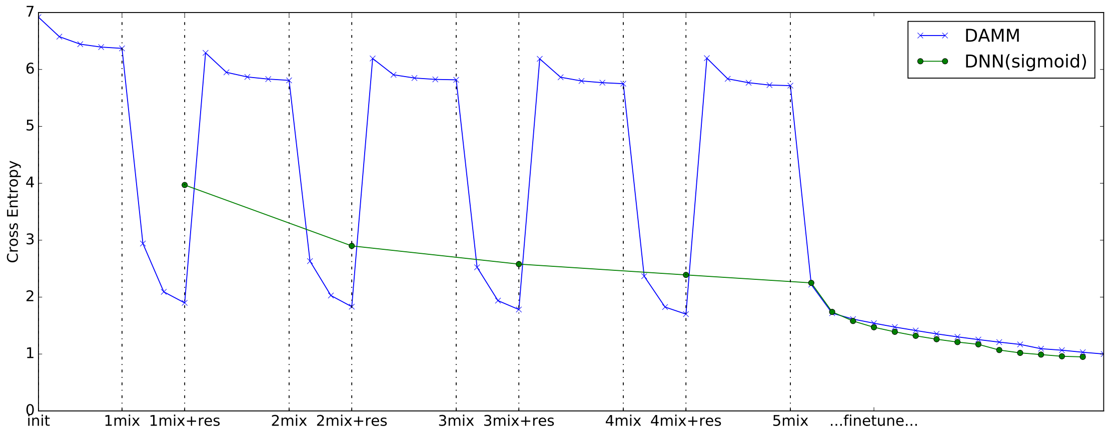
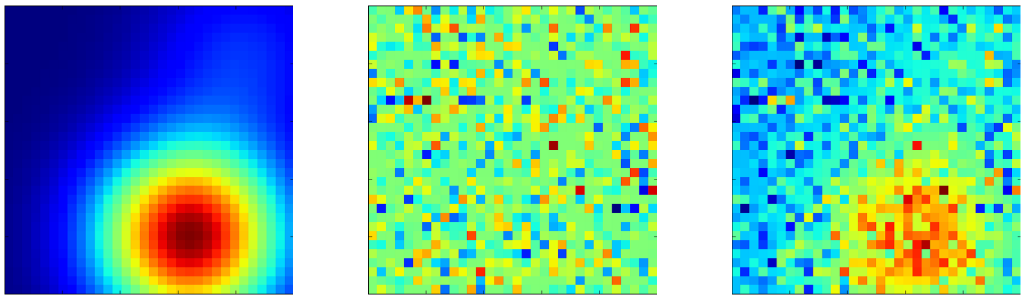


Fig. 6.3 WSJ-SI84: Learning curves of the DAMM and sigmoid DNN.



(a) Mixture (b) Residual (c) Mixture+residual

Fig. 6.4 WSJ-SI84: First-hidden-layer outputs of mixture and residual models of DAMM on one training frame.

test sets. According to the analysis, many of the updated mean vectors $\{\boldsymbol{\mu}_k^{(l)}\}$ were tuned to move far away from the unit square $[0,1]\times[0,1]$. Thus, the contours generated by these Gaussian components have little contribution to the model. This can explain the gains achieved by disabling the update of Gaussian components in training. The DAMM system without training the Gaussian components was further investigated in the following experiments.

The learning curves of the DAMM and the sigmoid DNN are shown in Figure 6.3. On the layer construction of the DAMM, the mixture model was tuned to the maximal extent before enabling the residual model. Because of the highly restricted nature, the mixture model cannot achieve good performance. However, it learned the rough behaviour for activation functions. Figure 6.4 illustrates the first-layer activation function outputs of the mixture and residual models on one training frame. The

System	Dev03	Eval03
DNN (sigmoid)	10.0	10.2
DAMM	9.8	10.1
+adapt	**9.6**	**9.9**

Table 6.2 WSJ-SI84: Recognition performance (WER %) of SD DAMM. Adaptation is performed at the utterance level on Gaussian mean vectors and covariance matrices of all hidden layers.

mixture model in Figure 6.4a constructed an activation contour, and the residual model in Figure 6.4b added a small variation to each activation function, which was expected in the network training.

Lastly, the adaptation of DAMM was investigated. The decoding hypotheses of the SI DAMM were used as the supervision for adaptation. To adapt the DAMM, the mean vectors and covariance matrices of the Gaussian components in all hidden layers were tuned on the supervision. To examine rapid adaptation on the DAMM, adaptation was performed at the utterance level. Table 6.2 reports the adaptation performance of the SD DAMM. By performing the adaptation on DAMM, small consistent gains could be obtained. The relative WER reduction is up to 4%, compared to the performance of the sigmoid DNN baseline.

6.5 Summary

This chapter proposed deep activation mixture models. This type of structured neural network uses a mixture model and a residual model to jointly form activation functions. The mixture model defines a smooth activation contour, and the residual model describes fluctuations around this contour. The effect of mixture model can be viewed as an informed regularisation that has the potential to improve the network regularisation. Also, it allows novel adaptation schemes on this form of structured DNN. The discussion started with the network topology of DAMM. To address the unbalance numbers of parameters in mixture and residual models, the isolating mode for parameter training was presented. Lastly, the adaptation scheme on DAMM was discussed.

Chapter 7

Experiments

This chapter presents the evaluation of the three forms of structured deep neural networks: the multi-basis adaptive neural network in Chapter 4; the stimulated deep neural network in Chapter 5; and the deep activation mixture model in Chapter 6. The proposed models were evaluated on two large vocabulary continuous speech recognition tasks tasks: the Babel languages; and the broadcast news English.

7.1 Babel Languages

This section reports the experiments on conversational telephone speech (CTS) tasks from the IARPA Babel program (Harper, 2013). Experiments were conducted on seven development languages and one surprise language[1] from the IARPA Babel program in the option period 3. Table 7.1 provides basic information about each language. One challenge of this data is that, for each language, the phonetic lexicon is not supplied. To address it, for each languages, an automatic, unicode based, graphemic dictionary (Section 3.2.5) generation was applied (Gales et al., 2015a). The Babel languages are provided with a wide range of language attributes, with all the data collected and annotated in a consistent fashion. Therefore, "pure" graphemes were appended with position information and language dependent attributes. A full language pack

[1]Pashto IARPA-babel104b-v0.4bY, Guarani IARPA-babel305b-v1.0a, Igbo IARPA-babel306b-v2.0c, Amharic IARPA-babel307b-v1.0b, Mongolian IARPA-babel401b-v2.0b, Javanese IARPA-babel402b-v1.0b, Dholuo IARPA-babel403b-v1.0b, Georgian IARPA-babel404b-v1.0a

Language	Family	System	Script	Graphemes
Pashto	Indo-European	Abjad	Arabic	47
Guarani	Tupian	Alphabet	Latin	71†
Igbo	Niger-Congo	Alphabet	Latin	52†
Amharic	Afro-Asiatic	Abugida	Ethiopic	247
Mongolian	Mongolic	Alphabet	Cyrillic	66†
Javanese	Austronesian	Alphabet	Latin	52†
Dholuo	Nilo-Saharan	Alphabet	Latin	52†
Georgian	Kartvelian	Alphabet	Mkhedruli	33

Table 7.1 Babel: Summary of used languages. Scripts marked with † utilise capital letter in the graphemic dictionary.

(FLP) was used for each language. This consists of 40-hour training data and 10-hour development data (Dev). The development data was used for evaluation.

For the Babel project, the performance of the system was evaluated in two ways: the word error rate, for recognition performance; and the maximum term-weighted value (MTWV), for keyword-spotting performance. In keyword spotting, the ASR system is used to generate decoding lattices, and the keyword query is searched in all possible paths in lattices. Given the keyword list $\boldsymbol{Q}$, a metric, named as term-weighted value (Fiscus et al., 2007), is defined as

$$\mathrm{TWV}(\xi;\boldsymbol{Q}) = 1 - \frac{1}{|\boldsymbol{Q}|}\sum_{\boldsymbol{\omega}\in\boldsymbol{Q}}\Big(P^{\mathrm{ms}}(\boldsymbol{\omega};\xi) + 999.9P^{\mathrm{fa}}(\boldsymbol{\omega};\xi)\Big) \tag{7.1}$$

where $P^{\mathrm{ms}}(\boldsymbol{\omega};\xi)$ and $P^{\mathrm{fa}}(\boldsymbol{\omega};\xi)$ are, respectively, the rates of miss and false alarm errors at detection threshold ξ, and defined as

$$P^{\mathrm{ms}}(\boldsymbol{\omega};\xi) = 1 - \frac{\#\mathrm{cor}(\boldsymbol{\omega};\xi)}{\#\mathrm{ref}(\boldsymbol{\omega})}, \qquad P^{\mathrm{fa}}(\boldsymbol{\omega};\xi) = \frac{\#\mathrm{incor}(\boldsymbol{\omega};\xi)}{\#\mathrm{trail}(\boldsymbol{\omega})}$$

where $\#\mathrm{cor}(\boldsymbol{\omega};\xi)$ is the number of correctly hypothesised occurrence of keyword $\boldsymbol{\omega}$ at the threshold ξ; $\#\mathrm{ref}(\boldsymbol{\omega})$ is the number of reference occurrence of keyword $\boldsymbol{\omega}$; $\#\mathrm{incor}(\boldsymbol{\omega};\xi)$ is the number of incorrectly hypothesised occurrence of keyword $\boldsymbol{\omega}$ at the threshold ξ; and, $\#\mathrm{trail}(\boldsymbol{\omega})$ is the number of trials for keyword $\boldsymbol{\omega}$. The higher the term-weighted value is, the better the keyword-spotting performance is. To avoid the

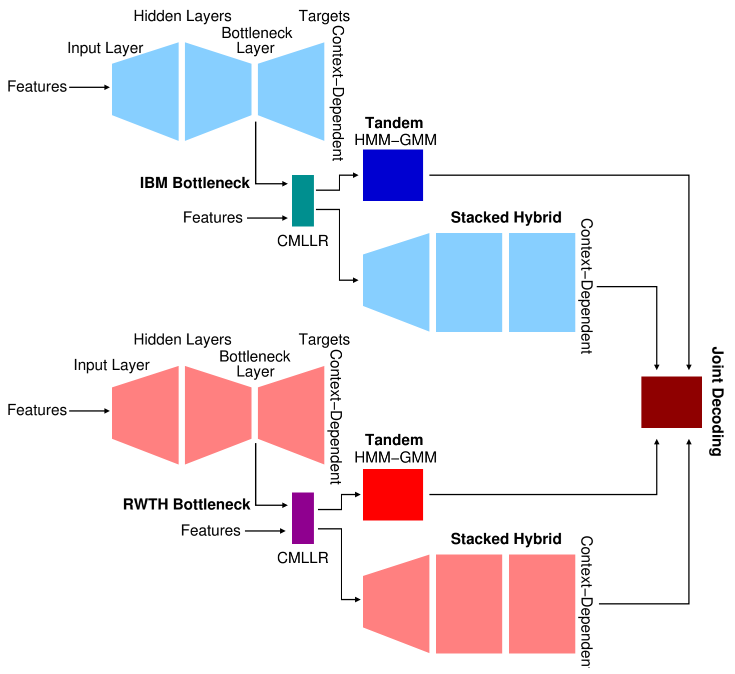


Fig. 7.1 4-way joint decoding for babel languages.

impact of threshold selection, the maximum term-weighted value is used as the metric for keyword spotting.

7.1.1 Experimental Setup

For all the Babel language experiments, two language models were used: the n-gram LM and the RNN LM trained using the CUED RNN LM toolkit (Chen et al., 2016). These were both trained on acoustic data transcripts containing approximately 500k words. Additional n-gram LMs were trained on data collected by Columbia University from the web (Mendels et al., 2015). These web LMs were then interpolated with the FLP LMs by optimising interpolation weights on the development data. Acoustic models were speaker adaptively trained Tandem and stacked Hybrid systems that shared the same set of features. The DNN input features were formed as concatenating PLP, pitch, probability of voicing and multi-language bottleneck features (Tuske et al., 2014) provided by IBM and RWTH Aachen in a temporal context window of 9 frames (Cui et al., 2015a). The multi-language bottleneck features were trained on FLP data of 24 Babel languages and CTS data of 4 additional languages, English, Spanish, Arabic and Mandarin, released by LDC. IBM features are language independent, whereas

RWTH Aachen additionally fine-tuned their bottleneck feature extractors to each target language. Thus, a total of 4 acoustic models were built for each language. To improve the performance, system combination (Section 3.2.2) was used, and the four acoustic models were combined via joint decoding (Wang et al., 2015a), as illustrated in Figure 7.1.

For each language, the DNN configuration consisted of 5 hidden layers with 1024 units in each layer. Stacked Hybrids were trained using mono-phone discriminative pre-training initialisation (Zhang and Woodland, 2015a) and followed by CE training and MPE training. For structured deep neural networks, the stimulated deep neural network was investigated in this task. It was used to replace the DNN model of the hybrid system. The activation regularisation was performed on all hidden layers. Similar to Section 5.4, the KL, Cos and Smooth systems were investigated. In the KL and Cos systems, the target patterns were defined on the graphemes, instead of the phonemes.

Keyword search was performed using the joint decoding lattices, and about 2k keywords were available for each language (Cui et al., 2014).

7.1.2 Results and Discussion

This section discusses the performance of several models and configurations. To find the most effective form of activation regularisation for Babel languages, the three forms of activation regularisation were investigated first in Javanese, which was the development language during the Babel evaluation. The experiments were then conduct on all Babel languages. To further improve the performance on the most challenge languages (i.e. Pashto, Igbo, Mongolian and Javanese), larger model configurations for stimulated deep neural networks were investigated.

Comparison of Activation Regularisations in Javanese

To find the most effective activation regularisation for Babel languages, the first experiment compared the performance of KL, Cos and Smooth systems. They were evaluated in Javanese, which was picked as the development language in the evaluation, using a simplified system configuration. That is, a single DNN using the RWTH multi-

Regularisation	Sigmoid	ReLU	Tanh
L^2	58.2	59.2	58.5
KL	**57.2**	–	–
Cos	57.9	59.0	57.9
Smooth	57.9	58.9	58.0

Table 7.2 Babel: Recognition performance (WER %) of CE stimulated DNNs using different forms of activation regularisation in Javanese.

Regularisation	CE	MPE
L^2	58.2	56.5
KL	**57.2**	**55.8**
Cos	57.9	56.2
Smooth	57.9	56.3

Table 7.3 Babel: Recognition performance (WER %) to compare CE and MPE sigmoid stimulated DNNs using different forms of activation regularisation in Javanese.

language bottleneck features was investigated, and decoding was performed using a tri-gram language model. Three types of activation function were investigated: sigmoid; tanh; and ReLU. The Cos and Smooth regularisations were used for stimulated DNNs using sigmoid, ReLU and tanh activation functions. The KL regularisation was only used in the stimulated DNN with sigmoid functions, due to the positive constraint on it.

The recognition performance of the CE systems is summarised in Table 7.2. On different activation function settings, the Cos and Smooth systems outperformed their corresponding baselines. The sigmoid function yielded better performance than other activation functions, and the best performance was achieved by the KL system. Sigmoid CE systems were further tuned using the MPE criterion. Table 7.3 reports the recognition performance of the MPE systems. The MPE training on different systems yielded lower-error performance than their corresponding CE baselines. All the KL, Cos and Smooth DNNs outperformed the baseline MPE DNN. The best performance was achieved by the KL system, reducing the TER from 56.5% to 55.8%. The KL regularisation was further investigated in the following experiments.

Language	Stimulated	WER (%)	MTWV
Pashto	✗	44.6	0.4644
	✓	**44.4**	**0.4672**
Guarani	✗	45.2	0.5800
	✓	**44.9**	**0.5869**
Igbo	✗	55.3	0.3974
	✓	**55.1**	**0.3986**
Amharic	✗	41.1	0.6402
	✓	**40.8**	**0.6521**
Mongolian	✗	47.8	0.5316
	✓	**47.6**	**0.5431**
Javanese	✗	50.9	0.4924
	✓	**50.7**	**0.4993**
Dholuo	✗	38.5	0.6434
	✓	**38.3**	**0.6451**
Georgian	✗	39.4	0.7179
	✓	**38.9**	**0.7265**

Table 7.4 Babel: Recognition and Keyword-spotting performance (WER % and MTWV) of joint decoding systems, with and without stimulated DNNs, in all languages. Stimulated DNNs were trained using the KL activation regularisation.

Experiments on All Languages

The second experiment contrasted the impact of stimulated deep neural networks on all languages in more advanced configuration combining 4 acoustic models and interpolated FLP and web data LMs in a single joint decoding run. For these results both Tandem and Hybrid systems were combined using joint decoding with stimulated DNN only being applied to the hybrid systems. Stimulated DNNs were training using the KL regularisation. The results in Table 7.4 show that recognition gains are seen even after system combination for all languages. Similarly, gains can be seen in keyword-spotting performance for all languages. Because being examined in the joint-decoding system, the gains achieved by stimulated DNNs are relatively small. However, stimulated DNNs still yielded the complementarity that was not achieved by other system candidates.

Language	Grid Size	WER %	MTWV
Pashto	32×32	44.4	0.4672
	45×45	**43.8**	**0.4750**
Igbo	32×32	55.1	0.3986
	45×45	54.7	**0.4026**
	55×55	**54.6**	0.4024
Mongolian	32×32	47.6	0.5431
	45×45	**46.8**	**0.5559**
Javanese	32×32	50.7	0.4993
	45×45	**50.5**	**0.5001**

Table 7.5 Babel: Performance (WER % and MTWV) of joint decoding with stimulated DNNs of different grid sizes in the four most challenging languages, Pashto, Igbo, Mongolian and Javanese. Stimulated DNNs were trained using the KL activation regularisation.

Impact of Grid Size on Most Challenging Languages

Experiments have so far examined a 32×32 grid. To improve the performance of most challenging languages, Pashto, Igbo, Mongolian and Javanese, in the evaluation, the third experiment assessed whether activation regularisation scaled with increasing the grid size. A larger 45×45 grid was examined for all languages plus an even larger 55×55 grid for the most challenging language. The use of a larger 45×45 grid in Table 7.5 shows recognition and keyword-spotting gains for all languages. Further increase in the grid size, 55×55, for the most challenging language Igbo, yielded little benefit. The results in Tables 7.4 and 7.5 show the advantages of stimulated DNN, which results in ASR and KWS gains in all examined languages.

7.2 Broadcast News English

This section describes the experiments on the US English broadcast news (BN) transcription task. The training dataset of this task includes the 1996 (Graff, 1997) & 1997 (Pallett et al., 1998) Hub-4 English Broadcast News Speech dataset (LDC97S44, LDC98S71). It consists of 288 shows from approximately 8k speakers. The speakers are distributed in a very unbalanced way with a few dominant speakers and many speakers with limited data. Another difficulty of the data is that they included seven

	BN			YTB	
	Train	Dev03	Eval03	GDev	GEval
Total(hrs)	144.2	2.7	2.6	7.4	7.0
#Uttr	44667	918	816	4288	3624
AvgUttr(secs)	11.6	10.7	10.9	6.2	6.9

Table 7.6 Broadcast News: Summary of training and evaluation sets, including total hours, number of utterances and average utterance duration.

so called "focus conditions", corresponding to a mix of noise conditions and speech style. Working with a corpus with different noise conditions and speech styles and an unbalanced speaker distribution enabled us to investigate the effectiveness of proposed models in a real world scenario.

For evaluation, the BN testsets Dev03 and Eval03 from the 2003 DARPA RT03 Evaluation were used. For evaluating the performance in highly-mismatched conditions, two Youtube (YTB) general testsets, GDev and GEval, were also used. They are both randomly selected from Youtube channels. These data are provided by Google as part of the EPSRC Project EP/I006583/1 ("Generative Kernels and Score Spaces for Classfication of Speech") within the Global Uncertainties Programme (Gales and et. al, 2013). Due to the variety of topics and speaking styles, transcribing the YTB data is a particularly difficult task. This is a particularly challenging scenario and an opportunity to investigate the sensitivity of different methods when the acoustic condition is far away from the modelled training space.

The training utterances were manually segmented. The BN and YTB testsets were automatically segmented and clustered using the RT04 Cambridge segmentation system (Tranter et al., 2004). A brief summary, including the total hours, number of utterances and average utterance duration, of training and evaluation sets is presented in Table 7.6. Decoding in this task was performed with the RT04 tri-gram language model, and detailed settings of this language model were described in Tranter et al. (2004).

7.2.1 Experimental Setup

Using a similar procedure as described in Section 4.7.1, a GMM-HMM model, with about 6k tied triphone states, was trained to provide the state alignments for the DNN models. For the DNN baseline, the 468-dimensional input feature to the neural network was formed by 12 PLPs, zeroth cepstrum, delta, delta-delta, delta-delta-delta coefficients with a context window of 9 frames. Features were processed by global CMN and CVN. The neural network consisted of five hidden layers with 1000 nodes for each layer[2]. The sigmoid activation function was used. DNN parameters were initialised in a discriminative layer-by-layer pre-training fashion, followed by fine-tuning of the full network using the cross-entropy criterion. This CE system was then further discriminatively trained using the MPE criterion. In training the CE and MPE models, 28 shows with about 600 speakers were randomly selected and used as the cross validation set. The hyper-parameters such as learning rate were tuned on the development set Dev03.

An i-vector system was build as a standard adaptation scheme in comparison with the proposed adaptation models. For estimating i-vectors, an SI UBM GMM model with 2048 mixture components was trained on the full training corpus. Each component corresponded to a 39-dimensional feature vector consisting of 12 PLPs appended with zeroth cepstrum, delta and the delta-delta coefficients. SD models were trained on all utterances of each speaker, from which the speaker-level i-vectors were extracted. To achieve robustness, informative priors (Karanasou et al., 2015) were introduced and they were estimated on a randomly selected subset of the training data (around 10% of the training data). Concerning the test i-vectors and aiming at a rapid adaptation framework, each test utterance was treated as a separate entity (i.e. speaker) and utterance-level i-vectors were extracted. The dimension of i-vector was set to 30. Once estimated, the 30-dimensional i-vector was concatenated with the 468-dimensional acoustic features to form the 498-dimensional input features for the i-vector DNN system. Before the concatenation, the i-vectors were normalised so that they have zero mean and unit variance on the training data, as normally used

[2]Stimulated DNNs requires the number of nodes in each layer to be a square number, for making the activation grid a complete square. A baseline system using 1024 nodes, to form a 32×32 grid, was also trained and there is no performance difference in contrast with that using 1000 nodes.

for neural network training. An improvement is observed in each testset by using the priors. The extracted i-vectors were also used in multi-basis adaptive neural networks with i-vector representation.

7.2.2 Results and Discussion

This section discusses the performance of several models and configurations. The investigation on structured deep neural networks includes all the three proposed models in this thesis: multi-basis adaptive neural networks, stimulated deep neural networks and deep activation mixture models. Also, the adaptation on these models was investigated. If there is no additional descriptions, adaptation was performed in an utterance-level unsupervised fashion with the supervision provided by the corresponding SI system.

Multi-basis Adaptive Neural Network

For multi-basis adaptive neural networks, basis parameters were initialised by the CE SI DNN model to achieve a sensible initial performance, as discussed in Section 4.2. The network was then optimised using the CE criterion in the interleaving mode described in Algorithm 4. This CE system was further interleavingly tuned using the MPE criterion for four iterations. In the case of the MPE system, the bases were initialised using the bases of the multi-basis CE system. In this task, the number of bases was set to two, and the basis weight vector $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ for each training speaker were initialised by setting one weight to 1 and the other to 0 according to its gender type. $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ was optimised for each speaker in training. To evaluate the effectiveness of multi-basis systems in rapid scenarios, $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ was estimated for each test utterance in the unsupervised fashion.

To combine i-vector representation with multi-basis models (Section 4.4), the i-vectors extracted in Section 7.2.1 were again used. The first combination scheme appended i-vectors with acoustic features to form the DNN input, and a multi-basis system with i-vector features was trained. The second combination scheme, predictive multi-basis transform, used i-vectors to directly predict multi-basis interpolation weights. A support-vector regression model with linear kernels was trained to estimate $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$

System	BN		YTB	
	Dev03	Eval03	GDev	GEval
DNN	12.5	10.9	58.5	62.1
+ivec	**11.1**	9.9	57.0	**60.2**
MBANN	11.9	10.3	56.9	61.2
+ivec	**11.1**	**9.8**	**56.6**	60.5

Table 7.7 Broadcast News: Recognition performance (WER %) of CE MBANN with and without i-vector input feature. Adaptation was performed in the utterance level.

Trn	BN		YTB	
	Dev03	Eval03	Gdev	Geval
5.4	5.9	5.8	10.0	9.4

Table 7.8 Broadcast News: Average i-vector distance of training and evaluation datasets.

for each speaker from the corresponding training i-vectors from the original multi-basis system. The open source toolkit SVMLight (Joachims, 2002) was used for the estimation of the prediction model[3]. The initial DNN and predictor were then interleavingly updated to adjust the system to the predictive $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ space (Section 4.4.2).

Table 7.7 presents the results of the use of i-vector input in the MBANN. Such combination (row "MBANN+ivec") slightly outperformed both the primary i-vector (row "DNN+ivec") and MBANN systems in the case of matched acoustic conditions (columns "BN"), indicating a complementarity of the two approaches. The results are not as consistent as far as mismatched YTB data are concerned. In this case, i-vectors achieved the best performance for GEval, while the combination scheme improved GDev.

The MBANN system with i-vector input features did not consistently outperform the i-vector DNN baseline on YTBGdev and YTBGeval testsets. To explain this case, a comparison of the average euclidean distance between the test i-vectors and the mean of training ones on different evaluation sets is shown in Table 7.8. The BN testsets gave a similar average i-vector distance as the training set which pointed out their consistency spanning in the acoustic space. The BN test sets present a similar average i-vector distance as the training set which indicates a similar span of the BN test and

[3]http://svmlight.joachims.org

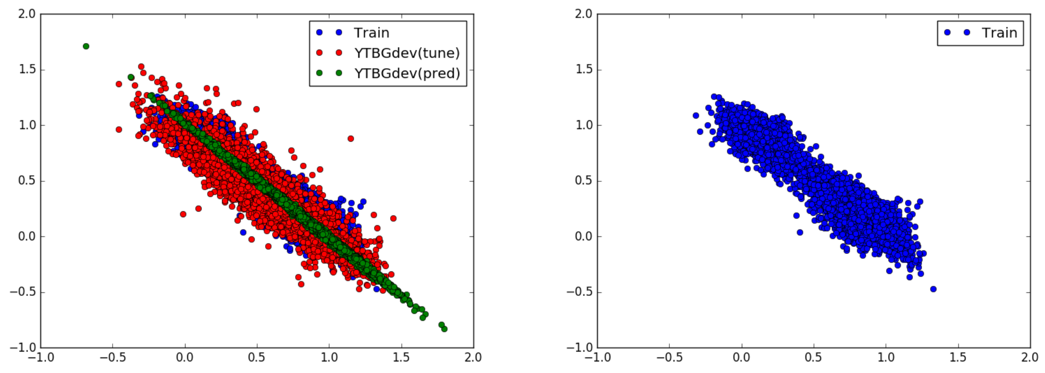
System	BN		YTB	
	Dev03	Eval03	GDev	GEval
DNN	12.5	10.8	58.5	62.1
MBANN	**11.9**	**10.3**	56.9	61.2
+pred	12.1	10.4	56.7	60.8
+pred-updt	12.0	**10.3**	**56.1**	**60.5**

Table 7.9 Broadcast News: Recognition performance (WER %) of CE MBANN with i-vector predictive model. Adaptation was performed in the utterance level.

training speaker spaces. The longer distances observed for the YTB i-vectors indicate the presence of i-vector estimations which are not or are not sufficiently represented by the training speaker space. This may explain why the i-vectors do not improve the performance in the case of mismatched acoustic conditions. In addition, the mismatched i-vector inputs seem to incorrectly compensate the hidden representations among the bases of the multi-basis DNN system and degrade the performance of the combined system.

In Table 7.9, the second combination scheme is examined where the i-vectors were used as fast predictors of the MBANN interpolation weights. This is indicated by the suffix “+pred” in the naming conventions. Moreover, the MBANN with i-vector predictor system was updated in the mode described in Section 4.4.2 for two iterations to obtain the refined predictive systems (noted with the suffix “+pred-updt”). For the BN test sets, the performance of the predictive system is similar to that of the default MBANN. However, for the mismatched YTB sets, the performance gains by the predictive model became more consistent. Thus, using the i-vector predictor in these cases achieves the best results and the desired rapid adaptation.

The improvement observed by using the i-vector predictor in the MBANN system is investigated in Figure 7.2. This figure compares the distribution of the MBANN weights of the training set and of GDev set. The training set (blue dots) weights are repeated in the right figure (Figure 7.2b) for a cleaner representation, as they are mostly covered by the test weights in the left figure. Concerning Figure 7.2a, the green dots present the MBANN weights using an i-vector predictor, while the red dots present the MBANN weights extracted after alignment with the hypothesis extracted



(a) YTBGdev and Training weights (b) Training weights only

Fig. 7.2 Broadcast News: Comparison of MBANN interpolation weights of the training speakers and YTBGdev test utterances.

from a first pass decoding. It can be seen that the predicted test estimations and the training ones are distributed in a linear space, presenting higher consistency than the initial MBANN system (red dots), which may explain the better performance of the predictive approach.

System	BN		YTB	
	Dev03	Eval03	GDev	GEval
DNN	11.2	10.2	55.5	59.2
+ivec	10.8	9.3	57.0	59.6
MBANN	10.7	9.5	55.4	60.3
+pred	11.2	9.7	**54.0**	**58.6**
+ivec	**10.3**	9.0	55.1	59.4
+ivec+pred	10.4	**8.9**	55.2	59.1

Table 7.10 Broadcast News: Recognition performance (WER%) of MPE MBANN. Adaptation was performed in the utterance level.

The performance of the MPE models is summarised in Table 7.10. The multi-basis system combined with i-vector input still gave a lower WER under matched acoustic conditions (“BN” columns) and the MBANN system with i-vector predictor still achieved the best performance for Youtube test sets.

System	η	WER (%)
DNN	0	12.5
Stimulated (KL)	0.05	**11.9**
	0.1	12.1
	0.15	12.5
	0.2	12.6

Table 7.11 Broadcast News: Recognition performance (WER %) of CE stimulated DNNs with different regularisation penalties on Dev03. The KL regularisation was used as the activation regularisation.

Regularisation	Dev03	Eval03
L^2	12.5	10.8
KL	**11.9**	**10.3**
Cos	12.3	10.6
Smooth	12.2	10.7

Table 7.12 Broadcast News: Recognition performance (WER %) of CE stimulated DNNs using different activation regularisations.

Stimulated Deep Neural Network

For stimulated deep neural networks, the sigmoid activation function was examined only, as it has shown better performance than tanh and ReLU in the experiments in Section 5.4 and 7.1. The configuration of stimulated DNN consisted of 5 hidden layers, and each layer formed a default 32×32 grid. The activation regularisation was perform on all hidden layers. The investigation on stimulated deep neural networks included three types of activation regularisation: KL, Cos and Smooth, as described in Section 5.4.1. For the KL and Cos systems, the sharpness factor σ (defined in Eq. 5.14) was empirically set to 0.1.

Table 7.11 reports the impact of the regularisation penalty (defined in Eq. 5.3) on CE KL systems. All the stimulated DNNs penalized from 0.05 to 0.2 outperformed the DNN baseline. The best system was achieved by that with 0.05, decreasing the word error rate by 0.6% in absolute value. Table 7.12 summarises the performance of different CE systems, and all the systems using activation regularisation outperformed the DNN baseline. The KL system again achieved the best performance.

System	η_L	WER (%)
LHUC	–	12.4
regLHUC	0	11.6
	0.05	11.5
	0.1	**11.4**
	0.15	11.5
	0.2	11.5

Table 7.13 Broadcast News: Recognition performance (WER %) of regularised LHUC adaptation on the CE stimulated DNN using different smoothness penalty on Dev03. The KL regularisation was used as the activation regularisation. Adaptation was performed in the utterance level.

System	Dev03	Eval03
DNN	12.5	10.8
+LHUC	12.4	10.6
+regLHUC	12.8	10.8
Stimulated DNN	11.9	10.3
+LHUC	11.6	10.0
+regLHUC	**11.4**	**9.9**

Table 7.14 Broadcast News: Recognition performance (WER %) of LHUC adaptation on the CE stimulated DNN using different smoothness penalty. The KL regularisation was used as the activation regularisation. Adaptation was performed in the utterance level.

The KL system ($\eta = 0.05$) was subsequently used to investigate the smoothness technique to regularise the LHUC adaptation approach (+regLHUC), discussed in Section 5.3. The distance decay σ_L^2, defined in Eq. 5.29, was fixed as 0.01 according to empirical results. The impact of smoothness penalty η_L (defined in Eq. 5.31) for adaptation is compared in Table 7.13. The best adaptation performance was obtained when η_L was set to 0.1, which outperformed the original LHUC method by 0.2% absolutely on WER. Then, η_L was fixed as 0.1 and a summary of rapid adaptation on the CE systems is given in Table 7.14, including the performance on the unseen testset Eval03. The regularised LHUC method on top of the stimulated DNN acquired improvement compared with the default LHUC on both testsets. The regularised LHUC was also tested on the DNN baseline. However, since the arbitrary neighbors were unable to provide useful information, no enhancement was achieved on top of the unstimulated system.

This CE stimulated DNN was then used to train the MPE stimulated system. Table 7.15 reports the comparison of the performance of different MPE systems. Similar to the CE ones, the MPE stimulated DNN outperformed the unstimulated

System	Dev03	Eval03
DNN	11.4	10.1
+LHUC	11.2	9.8
Stimulated DNN	11.2	9.8
+LHUC	10.9	9.5
+regLHUC	**10.6**	**9.4**

Table 7.15 Broadcast News: Recognition performance (WER %) of LHUC adaptation on the MPE stimulated DNN using different smoothness penalty. The KL regularisation was used as the activation regularisation. Adaptation was performed in the utterance level.

MPE baseline. The regularised LHUC on the stimulated system achieved the best performance as well, reducing the WER up to 5% relatively compared with the SI MPE stimulated system.

Deep Activation Mixture Model

For the deep activation mixture model, the network configuration and training followed a similar procedure as described Section 6.4.1. The recognition performance of CE SI systems is summarised in Table 7.16. As discussed in Section 6.1, by disabling the mixture model, the DAMM degrades to a tanh DNN. The DAMM outperformed the tanh DNN baseline, yielding up to 4% relative WER reduction. In addition, the DAMM yielded a slightly better performance than the sigmoid DNN system. The SD performance of the adapted CE DAMM is given in Table 7.17, comparing the impacts of adapting the Gaussian mean vector, unit variance vector and correlation coefficient. The change of unit variance applied homologous effects to activations located on nearby contour lines, while the move of mean vector applied opposite effects to the activations on the same contour line, which could not correspond to the similarity of activations

System	Dev03	Eval03
DNN (tanh)	12.8	11.0
DNN (sigmoid)	12.5	10.8
DAMM	**12.3**	**10.6**

Table 7.16 Broadcast News: Recognition performance (WER %) of SI CE DAMM.

System	Adapt			Dev03	Eval03
	Mean	Variance	Correlation		
SI	✗	✗	✗	12.3	10.6
SD	✓	✗	✗	12.2	10.6
	✗	✓	✗	12.1	10.5
	✗	✓	✓	12.1	10.5
	✓	✓	✗	12.1	**10.4**
	✓	✓	✓	**12.0**	**10.4**

Table 7.17 Broadcast News: Recognition performance (WER %) of SD CE DAMM. Adaptation was performed in the utterance level.

in the contour. Thus, the adaptation on the covariance matrix yielded a more effective impact than the mean vector. The relative WER reduction is up to 3%.

The recognition performance of MPE systems is compared in Table 7.18. The MPE DAMM yielded a similar performance as the sigmoid MPE DNN baseline. The adaptation on all the mean, variance and correlation coefficient achieved further performance gains than the SI MPE DAMM. The adaptation performance of the MPE DAMM obtained up to 3% relative WER reduction, which is similar to that of the CE DAMM.

Comparison of Structured Deep Neural Networks

To compare the performance of the three proposed models, a summary of the MPE systems is shown in Table 7.19. All the three types of structured deep neural networks outperformed the SI MPE DNN baseline. Also, by further performing the associated adaptation schemes on the SI stimulated DNN and DAMM, consistent adaptation gains can be achieved.

System	Adapt			Dev03	Eval03
	Mean	Variance	Correlation		
DNN (sigmoid)	–	–	–	11.4	10.1
DAMM	✗	✗	✗	11.4	10.0
	✓	✓	✓	**11.1**	**9.8**

Table 7.18 Broadcast News: Recognition performance (WER %) of MPE DAMM. Adaptation was performed in the utterance level.

System	Dev03	Eval03
DNN	11.4	10.1
MBANN	10.7	9.5
Stimulated DNN (KL)	11.2	9.8
+regLHUC	**10.6**	**9.4**
DAMM	11.4	10.0
+adapt	11.1	9.8

Table 7.19 Broadcast News: Recognition performance (WER %) of the MPE multi-basis adaptive neural network, stimulated deep neural network and deep activation mixture model. Adaptation was performed in the utterance level.

Chapter 8

Conclusion

This thesis investigated structured deep neural networks for automatic speech recognition. Three forms of structured deep neural networks were proposed: multi-basis adaptive neural network, stimulated deep neural network and deep activation mixture model. These structured DNNs explicitly introduce special structures to the network topology, making specific aspects of the data modelled explicitly.

Standard DNN models are commonly treated as "black boxes", in which parameters are difficult to be interpreted and grouped. This makes regularisation and adaptation challenging. The major contribution of this thesis is that the proposed structured DNNs induce and impose interpretation on the introduced network structures. These structured designs can improve regularisation and adaptation for DNN models. For regularisation, parameters can be separately regularised according to their meanings instead of a universal and indiscriminate regularisation to all parameters such as the L^2 regularisation. For adaptation, parameters can be adapted in groups or partially adapted according to their functions. This can help achieve robust adaptation when limited adaptation data are provided. A brief review of the thesis and future work are presented as follows.

8.1 Review of Work

Multi-basis adaptive neural networks were proposed in Chapter 4. This form of structured DNN introduces a set of parallel sub-networks, i.e. bases, with restricted connectivity, while basis and different bases share no connectivity. The outputs among different bases are combined via linear interpolation. To perform adaptation on an MBANN, only a compact set of parameters, i.e. interpolation weights, needs to be estimated. Therefore, rapid adaptation scenarios with limited data can be resolved within this framework. Several extensions to this basic MBANN model were also investigated. To combine i-vector representation, two combination schemes were presented. The first scheme appends i-vectors as DNN input features. In this configuration, the bases are explicitly informed with acoustic attributes, and the robustness to acoustic variations can be reinforced. The second scheme uses i-vectors to directly predict the speaker-dependent transform for MBANN. It avoids the requirement of decoding hypotheses in adaptation, which helps to reduce the computational cost as well as improve the robustness to hypothesis errors. The target-dependent interpolation introduces multiple sets of interpolation weights to separately adapt different DNN targets. The inter-basis connectivity generalises the MBANN framework with parameters between different bases.

Stimulated deep neural networks were proposed in Chapter 5. This form of structured neural network relates activation functions in regions of the network to aid interpretation and visualisation. In the network topology, hidden units are reorganised to form a grid, and activation functions with similar behaviours can then be grouped together in this grid space. This goal is obtained by introducing a special form of regularisation, which is the activation regularisation. The activation regularisation is designed to encourage the outputs of activation functions to satisfy a target pattern. By defining appropriate target patterns, different learning, participating or grouping concepts can be imposed and shaped on the network. This design prevents hidden units from an arbitrary order, which has the potential to improve network regularisation. Also, based on the restricted ordering of hidden units, smoothness techniques can be used to improve the adaptation schemes on stimulated DNNs. The LHUC adaptation approach was discussed as an example to explain how the smoothness method can be performed. In contrast with multi-basis adaptive neural networks, a “hard” version to partition the hidden units, stimulated deep neural networks perform the hidden-unit partitioning in a "soft" fashion. Activation functions with similar behaviours are induced to group together.

Deep activation mixture models were proposed in Chapter 6. Similar to stimulated DNNs, this form of structured DNN encourages activation function outputs to achieve a smooth surface. The output of one hidden layer is explicitly modelled as the sum of a mixture and residual models. The mixture model forms an activation contour, and the residual model depicts fluctuations around this contour. This type of structured neural network uses a mixture model and a residual model to jointly form activation functions. The mixture model defines a smooth activation contour, and the residual model describes fluctuations around this contour. This configuration in DAMM can improve network regularisation and also allows novel adaptation schemes. The mixture model of DAMM is highly restricted due to the nature of a few parameters. An important advantage of this restricted configuration is that the mixture model can be used for rapid adaptation. Its compact parameters can be robustly re-estimated on limited adaptation data to boost the resultant activation functions. Compared to stimulated DNNs, deep activation mixture models handle the concept of "target pattern" for activation function outputs by specific network structures, rather than a regularisation term in network training.

8.2 Future Work

There are many points discussed in this thesis that are worth further investigation. A number of suggestions for these future work are given as follows.

- In this thesis, the structured DNNs are discussed using the feed-forward neural network architecture. These concepts can be extended to more complex architectures, such as RNNs and CNNs.
- The DNN-HMM hybrid ASR framework was examined in the experiments. Instead, one natural extension to this is to combine structured DNNs with discriminative models, such as encoder-decoder RNNs.

- The discussions in Chapters 4, 5 and 6 pay much attention on the adaptability of structured DNNs. The objective of network adaptation is achieved by designing special structures on the network topology that enable it to be robustly adapted. This concept of structure designing can be further investigated to contribute to the area of DNN adaptation.
- The structured DNNs are applied to speech recognition for the discussion in this thesis. However, these models can be used in other tasks as well. For example, the target patterns of stimulated DNN can be designed using expertise other than acoustic knowledge. To use stimulated deep neural networks for language modelling, part-of-speech target patterns have the potential to improve the regularisation of neural-network-based language models.

Appendix A

I-vector Estimation

I-vectors are a low-dimensional fixed-length representation of speaker space spanning the dimensions of highest variability, and they are a convenient method for unsupervised adaptation of DNNs. Following Karanasou et al. (2014), the i-vector approach is presented as a type of model-based CAT estimation (Gales, 2000) where the HMM model is replaced by a GMM model, meaning that no transcriptions of the data are required. This equivalence allows extensions developed for CAT to be applied directly, such as the factorised approach with an explicit orthogonality constraint in Karanasou et al. (2014). These approaches are closely related to the joint factor analysis (Kenny et al., 2007) and i-vectors (Dehak et al., 2010) traditionally used in speaker verification.

In the i-vector extraction, the intrinsic phoneme variability is represented by a canonical model $\mathcal{M}$, which here is a GMM universal background model with M mixture components (Reynolds et al., 2000). It is defined by a mean supervector of component means $\boldsymbol{\mu}_0^{(m)}$, diagonal component covariance matrices $\boldsymbol{\Sigma}^{(m)}$ and mixture coefficients $\omega^{(m)}$. The input acoustic feature vectors $\boldsymbol{x}_t \in \mathbb{R}^D$ are seen as samples generated by the model $\mathcal{M}$.

Usually, an i-vector is extrated per speaker, estimated on all the data of the particular speaker and being constant across all utterances of the speaker. The same procedure is followed if each utterance is considered to correspond to a different speaker and the i-vector extraction can be performed at the utterance level. Each speaker is represented by a point in the "speaker eigenspace" spanned by the i-vectors. There is a linear dependence between the speaker-adapted means (i.e. speaker-dependent supervector) and the canonical means, which for a particular Gaussian component $m \in M$ is given by

$$\boldsymbol{\mu}^{(sm)} = \boldsymbol{\mu}_0^{(m)} + \boldsymbol{M}^{(m)} \boldsymbol{\lambda}_{\text{iv}}^{(s)} \tag{A.1}$$

where $\boldsymbol{\mu}^{(sm)}$ is the m-th component of speaker-dependent supervector, $\boldsymbol{M}^{(m)}$ is the factor submatrix for component m of size $D \times P$, representing P bases spanning the subspaces with the highest variability in the mean supervector space, and $\boldsymbol{\lambda}_{\text{iv}}^{(s)}$ is a vector of size P representing the i-vector of speaker s.

To extract the initial speaker i-vectors, an SD model using all the data of each speaker is trained and used to extract a mean supervector of dimension $MD \times 1$. Principal component analysis is then applied to these supervectors to obtain the speaker i-vectors that span the P-space. The i-vectors are the first "eigenvoices" that capture most of the variation in the data. Next, maximum-likelihood estimation of the model parameters and of the i-vectors is iteratively performed. The auxiliary function to be maximised is

$$Q(\mathcal{M}, \boldsymbol{\lambda}_{\text{iv}}^{(s)}; \hat{\mathcal{M}}, \hat{\boldsymbol{\lambda}}_{\text{iv}}^{(s)}) = -\frac{1}{2} \sum_{s,t,m} \gamma_t^{(m)}(s) (\boldsymbol{x}_t - \boldsymbol{\mu}^{(sm)})^T \boldsymbol{\Sigma}^{(m)-1} (\boldsymbol{x}_t - \boldsymbol{\mu}^{(sm)}) \tag{A.2}$$

where $\mathcal{M}$ is the canonical model to be estimated and $\hat{\mathcal{M}}$ is the "old" model. $\boldsymbol{\lambda}_{\text{iv}}^{(s)}$ are the i-vectors to be estimated and $\hat{\boldsymbol{\lambda}}_{\text{iv}}^{(s)}$ the "old" i-vectors. $\gamma_t^{(m)}(s)$ is the posterior probability of Gaussian component m at time t determined using the canonical model parameters $\hat{\mathcal{M}}$ and the speaker i-vectors $\hat{\boldsymbol{\lambda}}_{\text{iv}}^{(s)}$.

The training procedure uses the expectation-maximisation algorithm to estimate the parameters. This is the typical CAT model-based training procedure (Gales, 2000) or the ML Eigen-decomposition (Kuhn et al., 1998). By differentiating Eq. A.2 with respect to the i-vector of a particular speaker and equating to zero, the i-vector for speaker s may be shown to be

$$\boldsymbol{\lambda}_{\text{iv}}^{(s)} = \boldsymbol{G}_{\lambda_{\text{iv}}}^{(s)-1} \boldsymbol{k}_{\lambda_{\text{iv}}}^{(s)} \tag{A.3}$$

where $\boldsymbol{G}_{\lambda_{\text{iv}}}^{(s)}$ and $\boldsymbol{k}_{\lambda_{\text{iv}}}^{(s)}$ are given by

$$\boldsymbol{G}_{\lambda_{\text{iv}}}^{(s)} = \sum_{m,t} \gamma_t^{(m)}(s) \boldsymbol{M}^{(m)T} \boldsymbol{\Sigma}^{(m)-1} \boldsymbol{M}^{(m)}, \tag{A.4}$$

$$\boldsymbol{k}_{\lambda_{\text{iv}}}^{(s)} = \sum_{m} \boldsymbol{M}^{(m)T} \boldsymbol{\Sigma}^{(m)-1} \sum_{t} \gamma_t^{(m)}(s)(\boldsymbol{x}_t - \boldsymbol{\mu}_0^{(m)}). \tag{A.5}$$

To estimate the factor matrix $\boldsymbol{M}^{(m)}$, it suffices to differentiate Equation A.2 with respect to $\boldsymbol{M}^{(m)}$ and equate to zero. Doing so, the sufficient statistics are collected,

$$\boldsymbol{G}_{\texttt{M}}^{(m)} = \sum_{s,t} \gamma_t^{(m)}(s) \boldsymbol{\lambda}_{\text{iv}}^{(s)} \boldsymbol{\lambda}_{\text{iv}}^{(s)T}, \tag{A.6}$$

$$\boldsymbol{K}_{\texttt{M}}^{(m)} = \sum_{s,t} \gamma_t^{(m)}(s)(\boldsymbol{x}_t - \boldsymbol{\mu}_0^{(m)}) \boldsymbol{\lambda}_{\text{iv}}^{(s)T}. \tag{A.7}$$

The factor matrix $\boldsymbol{M}^{(m)}$ is estimated as

$$\boldsymbol{M}^{(m)} = \boldsymbol{K}_{\texttt{M}}^{(m)} \boldsymbol{G}_{\texttt{M}}^{(m)-1}. \tag{A.8}$$

Appendix B

Convex Optimisation of MBANN Speaker-dependent Transform

This appendix discusses the convexity in optimising the MBANN speaker-dependent transform, i.e. the basis weight vector $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$, if the interpolation is introduced prior to the output layer and the cross-entropy training criterion is used.

B.1 Convexity in Basic MBANN

In the adaptation, MBANN parameters except $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ are fixed. Thus the outputs of different bases are unchanged. Define matrix $\boldsymbol{B}_t$, consisting of all the basis outputs,

$$\boldsymbol{B}_t = \left[\boldsymbol{h}_t^{(L-1,1)}, \boldsymbol{h}_t^{(L-1,2)}, \ldots, \boldsymbol{h}_t^{(L-1,K)}\right]. \tag{B.1}$$

The input to the softmax function, $\boldsymbol{z}_t^{(Ls)}$ can then be rewritten as

$$\boldsymbol{z}_t^{(Ls)} = \boldsymbol{W}^{(L)\mathrm{T}} \boldsymbol{B}_t \boldsymbol{\lambda}_{\text{mb}}^{(s)} + \boldsymbol{b}^{(L)}. \tag{B.2}$$

Defining the function $\boldsymbol{f}(\boldsymbol{x}_t, j)$, which can be viewed as feature extractors on the raw features $\boldsymbol{x}_t$,

$$\boldsymbol{f}(\boldsymbol{x}_t, j) = \boldsymbol{B}_t^{\mathrm{T}} \boldsymbol{w}_j^{(L)}. \tag{B.3}$$

It is unchanged during $\boldsymbol{\lambda}_{\text{mb}}^{(s)}$ optimisation. Using $\boldsymbol{f}(\boldsymbol{x}_t, j)$, the CE criterion can then be rewritten as

$$\mathcal{L}(\boldsymbol{\lambda}_{\text{mb}}^{(s)};\mathbb{D}) = \sum_t \left\{ \log \sum_j \exp\left(\boldsymbol{\lambda}_{\text{mb}}^{(s)\text{T}} \boldsymbol{f}(\boldsymbol{x}_t, j) + b_j^{(L)}\right) - \boldsymbol{\lambda}_{\text{mb}}^{(s)\text{T}} \boldsymbol{f}(\boldsymbol{x}_t, j) + b_{y_t}^{(L)} \right\}. \tag{B.4}$$

This form is identical to the log-linear model (Nelder and Baker, 1972), which is a convex model. Therefore, the convexity of MBANN interpolation weight optimisation is held.

B.2 Convexity in MBANN with Target-dependent Interpolation Weights

For MBANN with target-dependent interpolation weights, suppose M target classes are introduced. By duplicating $\boldsymbol{B}_t$ defined in Eq. B.1 for M times, a large matrix $\tilde{\boldsymbol{B}}_t$ is formed as

$$\tilde{\boldsymbol{B}}_t = [\boldsymbol{B}_t, \boldsymbol{B}_t, \ldots, \boldsymbol{B}_t]. \tag{B.5}$$

Define the extend basis weight vector $\tilde{\boldsymbol{\lambda}}_{\text{mb}}^{(s)}$ as the concatenation of interpolation weights for all target classes

$$\tilde{\boldsymbol{\lambda}}_{\text{mb}}^{(s)} = \left[\boldsymbol{\lambda}_{\text{mb}}^{(s,1)\text{T}}, \boldsymbol{\lambda}_{\text{mb}}^{(s,2)\text{T}}, \ldots, \boldsymbol{\lambda}_{\text{mb}}^{(s,M)\text{T}}\right]^{\text{T}}. \tag{B.6}$$

It can follow a similar fashion described in the previous section to define the feature extractor $\tilde{\boldsymbol{f}}(\boldsymbol{x}_t, j)$,

$$\tilde{\boldsymbol{f}}(\boldsymbol{x}_t, j) = \tilde{\boldsymbol{B}}_t^{\text{T}} \boldsymbol{w}_j^{(L)}. \tag{B.7}$$

The CE criterion can then be rewritten as

$$\mathcal{L}(\boldsymbol{\lambda}_{\text{mb}}^{(s,1)}, \ldots, \boldsymbol{\lambda}_{\text{mb}}^{(s,M)};\mathbb{D}) = \sum_t \left\{ \log \sum_j \exp\left(\tilde{\boldsymbol{\lambda}}_{\text{mb}}^{(s)\text{T}} \tilde{\boldsymbol{f}}(\boldsymbol{x}_t, j) + b_j^{(L)}\right) - \tilde{\boldsymbol{\lambda}}_{\text{mb}}^{(s)\text{T}} \tilde{\boldsymbol{f}}(\boldsymbol{x}_t, j) + b_{y_t}^{(L)} \right\}. \tag{B.8}$$

It also yields the same form as the log-linear model, which is convex.

References

Ossama Abdel-Hamid and Hui Jiang. Fast speaker adaptation of hybrid NN/HMM model for speech recognition based on discriminative learning of speaker code. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7942–7946. IEEE, 2013a.

Ossama Abdel-Hamid and Hui Jiang. Rapid and effective speaker adaptation of convolutional neural network based models for speech recognition. In *INTERSPEECH*, pages 1248–1252, 2013b.

Victor Abrash, Horacio Franco, Ananth Sankar, and Michael Cohen. Connectionist speaker normalization and adaptation. In *in Eurospeech*, pages 2183–2186, 1995.

Bishnu S Atal. Effectiveness of linear prediction characteristics of the speech wave for automatic speaker identification and verification. *the Journal of the Acoustical Society of America*, 55(6):1304–1312, 1974.

Bishnu S Atal and Suzanne L Hanauer. Speech analysis and synthesis by linear prediction of the speech wave. *The journal of the acoustical society of America*, 50 (2B):637–655, 1971.

Xavier L Aubert. An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech & Language*, 16(1):89–114, 2002.

L Brown Bahl, P de Souza, and R P Mercer. Maximum mutual information estimation of hidden Markov model parameters for speech recognition. Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86., 1986.

James Baker. The DRAGON system–an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):24–29, 1975.

Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

Jerome R Bellegarda. Statistical language model adaptation: review and perspectives. *Speech communication*, 42(1):93–108, 2004.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech communication*, 50(5):434–451, 2008.

Christopher M Bishop. Mixture density networks. 1994.

Christopher M Bishop. *Neural networks for pattern recognition.* Oxford university press, 1995.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Herve A Bourlard and Nelson Morgan. *Connectionist speech recognition: a hybrid approach*, volume 247. Springer, 1994.

John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

Peter F Brown. The acoustic-modeling problem in automatic speech recognition. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1987.

William Byrne. Minimum Bayes risk estimation and decoding in large vocabulary continuous speech recognition. *IEICE TRANSACTIONS on Information and Systems*, 89(3):900–907, 2006.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

William Chan, Navdeep Jaitly, Quoc V Le, and Oriol Vinyals. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015.

Wen-Hsiung Chen, CH Smith, and SC Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE Transactions on communications*, 25(9):1004–1009, 1977.

X. Chen, X. Liu, Y. Qian, M. J. F Gales, and P. C. Woodland. CUED-RNNLM – an open-source toolkit for efficient training and evaluation of recurrent neural network language models. In *ICASSP*, 2016.

David Chiang, Kevin Knight, and Wei Wang. 11,001 new features for statistical machine translation. In *Proceedings of human language technologies: The 2009 annual conference of the north american chapter of the association for computational linguistics*, pages 218–226. Association for Computational Linguistics, 2009.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585, 2015.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Ronan Collobert. Large scale machine learning. 2004.

J. Cui, J. Mamou, B. Kingsbury, and B. Ramabhadran. Automatic keyword selection for keyword search development and tuning. In *ICASSP*, 2014.

Jia Cui, Brian Kingsbury, Bhuvana Ramabhadran, Abhinav Sethy, Kartik Audhkhasi, Xiaodong Cui, Ellen Kislal, Lidia Mangu, Markus Nussbaum-Thom, Michael Picheny, et al. Multilingual representations for low resource speech recognition and keyword search. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 259–266. IEEE, 2015a.

Xiaodong Cui, Vaibhava Goel, and Brian Kingsbury. Data augmentation for deep neural network acoustic modeling. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(9):1469–1477, 2015b.

George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

KH Davis, R Biddulph, and Stephen Balashek. Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, 24(6):637–642, 1952.

Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

Najim Dehak, Patrick J. Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech and Language Processing*, 2010.

Najim Dehak, Patrick J Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798, 2011.

Marc Delcroix, Keisuke Kinoshita, Takaaki Hori, and Tomohiro Nakatani. Context adaptive deep neural networks for fast acoustic model adaptation. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*, pages 4535–4539, 2015.

Marc Delcroix, Keisuke Kinoshita, Atsunori Ogawa, Takuya Yoshioka, Dung T Tran, and Tomohiro Nakatani. Context adaptive neural network for rapid adaptation of deep cnn based acoustic models. In *INTERSPEECH*, pages 1573–1577, 2016a.

Marc Delcroix, Keisuke Kinoshita, Chengzhu Yu, Atsunori Ogawa, Takuya Yoshioka, and Tomohiro Nakatani. Context adaptive deep neural networks for fast acoustic model adaptation in noisy conditions. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5270–5274. IEEE, 2016b.

Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, et al. Recent advances in deep learning for speech research at Microsoft. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8604–8608. IEEE, 2013.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 (Jul):2121–2159, 2011.

Stéphane Dupont and Leila Cheboub. Fast speaker adaptation of artificial neural networks for automatic speech recognition. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 3, pages 1795–1798. IEEE, 2000.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341:3, 2009.

Gunnar Evermann and PC Woodland. Posterior probability decoding, confidence estimation and system combination. In *Proc. Speech Transcription Workshop*, volume 27, page 78. Baltimore, 2000.

Xue Feng, Yaodong Zhang, and James Glass. Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 1759–1763. IEEE, 2014.

Xue Feng, Brigitte Richardson, Scott Amman, and James Glass. On using heterogeneous data for vehicle-based speech recognition: a DNN-based approach. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4385–4389. IEEE, 2015.

J. G. Fiscus et al. Results of the 2006 spoken term detection evaluation. In *Proc. ACM SIGIR Workshop on Searching Spontaneous Conversational Speech*, 2007.

George Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of machine learning research*, 3(Mar):1289–1305, 2003.

G David Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

Sadaoki Furui. Speaker-independent isolated word recognition based on emphasized spectral dynamics. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.*, volume 11, pages 1991–1994. IEEE, 1986.

M. J. F. Gales, K. M. Knill, and A. Ragni. Unicode-based graphemic systems for limited resource languages. In *ICASSP*, 2015a.

Mark Gales and et. al. Generative kernels and score-spaces for classification of speech. http://mi.eng.cam.ac.uk/ mjfg/Kernel/index.html, 2013.

Mark Gales and Steve Young. The application of hidden Markov models in speech recognition. *Foundations and trends in signal processing*, 1(3):195–304, 2008.

Mark JF Gales. Maximum likelihood linear transformations for HMM-based speech recognition. *Computer speech & language*, 12(2):75–98, 1998.

Mark JF Gales. Cluster adaptive training of hidden Markov models. *IEEE transactions on speech and audio processing*, 8(4):417–428, 2000.

Mark JF Gales, Kate M Knill, and Anton Ragni. Unicode-based graphemic systems for limited resource languages. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5186–5190. IEEE, 2015b.

J-L Gauvain and Chin-Hui Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE transactions on speech and audio processing*, 2(2):291–298, 1994.

Roberto Gemello, Franco Mana, Stefano Scanzio, Pietro Laface, and Renato De Mori. Linear hidden transformations for adaptation of hybrid ANN/HMM models. *Speech Communication*, 49(10):827–835, 2007.

Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

Daniel Gildea and Thomas Hofmann. Topic-based language models using em. In *Sixth European Conference on Speech Communication and Technology*, 1999.

Ondřej Glembek, Lukáš Burget, Pavel Matějka, Martin Karafiát, and Patrick Kenny. Simplification and optimization of i-vector extraction. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 4516–4519. IEEE, 2011.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feed-forward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

John J Godfrey, Edward C Holliman, and Jane McDaniel. SWITCHBOARD: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1, pages 517–520. IEEE, 1992.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

David Graff. The 1996 broadcast news speech and language-model corpus. In *Proc. 1997 DARPA Speech Recognition Workshop*, pages 11–14, 1997.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013a.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013b.

Frantisek Grezl, Martin Karafiát, Stanislav Kontár, and Jan Cernockỳ. Probabilistic and bottle-neck features for LVCSR of meetings. In *ICASSP (4)*, pages 757–760, 2007.

Mary Harper. The BABEL program and low resource speech technology. *Proc. of ASRU 2013*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Georg Heigold, Vincent Vanhoucke, Alan Senior, Patrick Nguyen, M Ranzato, Matthieu Devin, and Jeffrey Dean. Multilingual acoustic models using distributed deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8619–8623. IEEE, 2013.

John F Hemdal and George W Hughes. A feature based computer recognition program for the modeling of vowel perception. *Models for the Perception of Speech and Visual Form, Wathen-Dunn, W. Ed. MIT Press, Cambridge, MA*, 1967.

Hynek Hermansky. Perceptual linear predictive (PLP) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

Hynek Hermansky, Daniel PW Ellis, and Sangita Sharma. Tandem connectionist feature extraction for conventional HMM systems. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 3, pages 1635–1638. IEEE, 2000.

Geoffrey Hinton et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6): 82–97, 2012.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Zhen Huang, Jinyu Li, Sabato Marco Siniscalchi, I-Fan Chen, Chao Weng, and Chin-Hui Lee. Feature space maximum a posteriori linear regression for adaptation of deep neural networks. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Zhen Huang, Sabato Marco Siniscalchi, I-Fan Chen, Jiadong Wu, and Chin-Hui Lee. Maximum a posteriori adaptation of network parameters in deep models. *arXiv preprint arXiv:1503.02108*, 2015.

Zhen Huang, Sabato Marco Siniscalchi, I-Fan Chen, and Chin-Hui Lee. Towards a direct bayesian adaptation framework for deep models. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2016 Asia-Pacific*, pages 1–4. IEEE, 2016.

David H Hubel and Torsten N Wiesel. Binocular interaction in striate cortex of kittens reared with artificial squint. *Journal of neurophysiology*, 28(6):1041–1059, 1965.

Mei-Yuh Hwang and Xuedong Huang. Shared-distribution hidden markov models for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 1(4): 414–420, 1993.

Takaaki Ishii, Hiroki Komiyama, Takahiro Shinozaki, Yasuo Horiuchi, and Shingo Kuroiwa. Reverberant speech recognition based on denoising autoencoder. In *Interspeech*, pages 3512–3516, 2013.

Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, 1975.

Frederick Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

Thorsten Joachims. *Learning to classify text using support vector machines: Methods, theory and algorithms.* Kluwer Academic Publishers, 2002.

Biing-Hwang Juang, Wu Hou, and Chin-Hui Lee. Minimum classification error rate methods for speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 5(3):257–265, 1997.

Janez Kaiser, Bogomir Horvat, and Zdravko Kacic. A novel loss function for the overall risk criterion based discriminative training of HMM models. In *Sixth International Conference on Spoken Language Processing*, 2000.

Stephan Kanthak and Hermann Ney. Context-dependent acoustic modeling using graphemes for large vocabulary speech recognition. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 1, pages I–845. IEEE, 2002.

Penny Karanasou, Yongqiang Wang, Mark JF Gales, and Phil C Woodland. Adaptation of deep neural network acoustic models using factorised i-vectors. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Penny Karanasou, Mark Gales, and Philip Woodland. I-vector estimation using informative priors for adaptation of deep neural networks. ISCA, 2015.

Penny Karanasou, Chunyang Wu, Mark Gales, and Philip C Woodland. I-vectors and structured neural networks for rapid adaptation of acoustic models. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 25(4):818–828, 2017.

Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401, 1987.

Patrick Kenny, Gilles Boulianne, Pierre Ouellet, and Pierre Dumouchel. Joint factor analysis versus eigenchannels in speaker recognition. *IEEE Transactions on Audio, Speech & Language Processing*, 15(4):1435–1447, 2007.

Mirjam Killer, Sebastian Stüker, and Tanja Schultz. Grapheme based speech recognition. In *INTERSPEECH*, 2003.

Brian Kingsbury, Tara N Sainath, and Hagen Soltau. Scalable minimum Bayes risk training of deep neural network acoustic models using distributed Hessian-free optimization. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.

Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. In *INTERSPEECH*, pages 3586–3589, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Roland Kuhn, Patrick Nguyen, Jean-Claude Junqua, Lloyd Goldwasser, Nancy Niedzielski, Steven Fincke, Ken Field, and Matteo Contolini. Eigenvoices for speaker adaptation. In *International Conference on Spoken Language Processing*, 1998.

Nagendra Kumar and Andreas G Andreou. Heteroscedastic discriminant analysis and reduced rank HMMs for improved speech recognition. *Speech communication*, 26(4): 283–297, 1998.

Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

Kai-Fu Lee. On large-vocabulary speaker-independent continuous speech recognition. *Speech communication*, 7(4):375–379, 1988.

Li Lee and Richard C Rose. Speaker normalization using efficient frequency warping procedures. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 353–356. IEEE, 1996.

Christopher J Leggetter and Philip C Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech & Language*, 9(2):171–185, 1995.

Bo Li and Khe Chai Sim. Comparison of discriminative input and output transformations for speaker adaptation in the hybrid NN/HMM systems. 2010.

Xiao Li and Jeff Bilmes. Regularized adaptation of discriminative classifiers. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 1, pages I–I. IEEE, 2006.

Hank Liao. Speaker adaptation of context dependent deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7947–7951. IEEE, 2013.

Xunying Liu, Yongqiang Wang, Xie Chen, Mark JF Gales, and Philip C Woodland. Efficient lattice rescoring using recurrent neural network language models. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 4908–4912. IEEE, 2014.

Xunying Liu, Xie Chen, Mark JF Gales, and Philip C Woodland. Paraphrastic recurrent neural network language models. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5406–5410. IEEE, 2015.

Liang Lu, Xingxing Zhang, Kyunghyun Cho, and Steve Renals. A study of the recurrent neural network encoder-decoder for large vocabulary speech recognition. In *INTERSPEECH*, pages 3249–3253, 2015.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5188–5196, 2015.

Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus in speech recognition: word error minimization and other applications of confusion networks. *Computer Speech & Language*, 14(4):373–400, 2000.

James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

G. Mendels, E. Cooper, V. Soto, J. Hirschberg, M. Gales, K. Knill, A. Ragni, and H. Wang. Improving speech recognition and keyword search for low resource languages using web data. In *Interspeech*, 2015.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

John Ashworth Nelder and R Jacob Baker. *Generalized linear models*. Wiley Online Library, 1972.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

Joao Neto, Luís Almeida, Mike Hochberg, Ciro Martins, Luís Nunes, Steve Renals, and Tony Robinson. Speaker-adaptation for hybrid HMM-ANN continuous speech recognition system. 1995.

Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38, 1994.

Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang W Koh, Quoc V Le, and Andrew Y Ng. Tiled convolutional neural networks. In *Advances in neural information processing systems*, pages 1279–1287, 2010.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

JJ Odell, V Valtchev, Philip C Woodland, and Steve J Young. A one pass decoder design for large vocabulary recognition. In *Proceedings of the workshop on Human Language Technology*, pages 405–410. Association for Computational Linguistics, 1994.

Mark JL Orr et al. Introduction to radial basis function networks, 1996.

Stefan Ortmanns, Hermann Ney, and Xavier Aubert. A word graph algorithm for large vocabulary continuous speech recognition. *Computer Speech & Language*, 11 (1):43–72, 1997.

David S. Pallett, Jonathan G. Fiscus, Alvin Martin, and Mark A. Przybocki. 1997 broadcast news benchmark test results: English and non-English. In *Proc. 1998 DARPA Broadcast News Transcription and Understanding Workshop*, pages 5–11, 1998.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

Douglas B Paul and Janet M Baker. The design for the Wall Street Journal-based CSR corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362. Association for Computational Linguistics, 1992.

David Pearce and J Picone. Aurora working group: DSR front end LVCSR evaluation AU/384/02. *Inst. for Signal & Inform. Process., Mississippi State Univ., Tech. Rep*, 2002.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Daniel Povey. *Discriminative training for large vocabulary speech recognition.* PhD thesis, Ph. D. thesis, Cambridge University, 2004.

Daniel Povey and Philip C Woodland. Minimum phone error and i-smoothing for improved discriminative training. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 1, pages I–105. IEEE, 2002.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The Kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for ASR based on lattice-free MMI. In *INTERSPEECH*, pages 2751–2755, 2016.

Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Lawrence R Rabiner and Bernard Gold. Theory and application of digital signal processing. *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975. 777 p.*, 1975.

A Ragni, C Wu, MJF Gales, J Vasilakes, and KM Knill. Stimulated training for automatic speech recognition and keyword search in limited resource conditions. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 4830–4834. IEEE, 2017.

Shakti P Rath, Daniel Povey, Karel Vesely̌, and Jan Cernocky̌. Improved feature processing for deep neural networks. In *Interspeech*, pages 109–113, 2013.

Steve Renals, Nelson Morgan, Herve Bourlard, Michael Cohen, Horacio Franco, Chuck Wooters, and Phil Kohn. Connectionist speech recognition: Status and prospects. Technical report, Technical Report TR-91-070, University of California at Berkeley, 1991.

Douglas A. Reynolds, Thomas F. Quatieri, and Robert B. Dunn. Speaker verification using adapted Gaussian mixture models. *Digital Signal Processing*, 10(1-3):19–41, 2000.

Korin Richmond. A trajectory mixture density network for the acoustic-articulatory inversion mapping. In *Interspeech*, 2006.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations.* MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Tara N Sainath, Ron J Weiss, Andrew Senior, Kevin W Wilson, and Oriol Vinyals. Learning the speech front-end with raw waveform CLDNNs. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

George Saon, Hagen Soltau, David Nahamoo, and Michael Picheny. Speaker adaptation of neural network acoustic models using i-vectors. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 55–59. IEEE, 2013.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 24–29. IEEE, 2011a.

Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011b.

Oliver G Selfridge. Pandemonium: a paradigm for learning in mechanisation of thought processes. 1958.

Andrew Senior and Ignacio Lopez-Moreno. Improving dnn speaker independence with i-vector inputs. In *Proc. of ICASSP*, pages 225–229, 2014a.

Andrew Senior and Ignacio Lopez-Moreno. Improving DNN speaker independence with i-vector inputs. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 225–229. IEEE, 2014b.

Koichi Shinoda and C-H Lee. A structural bayes approach to speaker adaptation. *IEEE Transactions on Speech and Audio Processing*, 9(3):276–287, 2001.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Sabato Marco Siniscalchi, Jinyu Li, and Chin-Hui Lee. Hermitian based hidden activation functions for adaptation of hybrid HMM/ANN models. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

Sabato Marco Siniscalchi, Jinyu Li, and Chin-Hui Lee. Hermitian polynomial for speaker adaptation of connectionist speech recognition systems. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(10):2152–2161, 2013.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

J Stadermann and G Rigoll. Two-stage speaker adaptation of hybrid tied-posterior acoustic models. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 1, pages 977–980. IEEE, 2005.

Hang Su, Gang Li, Dong Yu, and Frank Seide. Error back propagation for sequence training of context-dependent deep networks for conversational speech transcription. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6664–6668. IEEE, 2013.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

Martin Sundermeyer, Hermann Ney, and Ralf Schlüter. From feedforward to recurrent LSTM neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(3):517–529, 2015.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Pawel Swietojanski and Steve Renais. SAT-LHUC: Speaker adaptive training for learning hidden unit contributions. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5010–5014. IEEE, 2016.

Pawel Swietojanski and Steve Renals. Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 171–176. IEEE, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

S. Tan, K. C. Sim, and M. Gales. Improving the interpretability of deep neural networks with stimulated learning. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 617–623, 2015a.

Tian Tan, Yanmin Qian, Maofan Yin, Yimeng Zhuang, and Kai Yu. Cluster adaptive training for deep neural network. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4325–4329. IEEE, 2015b.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

SE Tranter, MJF Gales, R Sinha, S Umesh, and PC Woodland. The development of the Cambridge University RT-04 diarisation system. In *Proc. Fall 2004 Rich Transcription Workshop (RT-04)*, 2004.

Jan Trmal, Jan Zelinka, and Ludek Müller. Adaptation of a feedforward artificial neural network using a linear transform. In *TSD'10*, pages 423–430, 2010.

Zoltán Tüske, Pavel Golik, Ralf Schlüter, and Hermann Ney. Acoustic modeling with deep neural networks using raw time signal for LVCSR. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Zoltan Tuske, David Nolden, Ralf Schluter, and Hermann Ney. Multilingual MRASTA features for low-resource keyword search and speech recognition systems. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 7854–7858. IEEE, 2014.

Ehsan Variani, Erik McDermott, and Georg Heigold. A Gaussian mixture model layer jointly optimized with discriminative features within a deep neural network architecture. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4270–4274. IEEE, 2015.

Olli Viikki and Kari Laurila. Cepstral domain segmental feature vector normalization for noise robust speech recognition. *Speech Communication*, 25(1):133–147, 1998.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

H. Wang, A. Ragni, M. J. F. Gales, K. M. Knill, P. C. Woodland, and C. Zhang. Joint decoding of tandem and hybrid systems for improved keyword spotting on low resource languages. In *Interspeech*, 2015a.

Haipeng Wang, Anton Ragni, Mark JF Gales, Kate M Knill, Philip C Woodland, and Chao Zhang. Joint decoding of tandem and hybrid systems for improved keyword spotting on low resource languages. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015b.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

PC Woodland. Weight limiting, weight quantisation and generalisation in multi-layer perceptrons. In *Artificial Neural Networks, 1989., First IEE International Conference on (Conf. Publ. No. 313)*, pages 297–300. IET, 1989.

Philip C Woodland, Chris J Leggetter, JJ Odell, Valtcho Valtchev, and Steve J Young. The 1994 HTK large vocabulary speech recognition system. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 73–76. IEEE, 1995.

Chunyang Wu and Mark JF Gales. Multi-basis adaptive neural network for rapid adaptation in speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4315–4319. IEEE, 2015.

Chunyang Wu and Mark JF Gales. Deep activation mixture model for speech recognition. *Proc. Interspeech 2017*, pages 1611–1615, 2017.

Chunyang Wu, Penny Karanasou, and Mark JF Gales. Combining i-vector representation and structured neural networks for rapid adaptation. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5000–5004. IEEE, 2016a.

Chunyang Wu, Penny Karanasou, Mark JF Gales, and Khe Chai Sim. Stimulated deep neural network for speech recognition. In *Proc. Interspeech*, pages 400–404, 2016b.

Chunyang Wu, Mark Gales, Anton Ragni, Penny Karanasou, and Khe Chai Sim. Improving interpretation and regularisation in deep learning. *submitted to IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 2017.

Yeming Xiao, Zhen Zhang, Shang Cai, Jielin Pan, and Yonghong Yan. A initial attempt on task-specific adaptation for deep neural network-based large vocabulary continuous speech recognition. In *INTERSPEECH'12*, pages –1–1, 2012.

Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

Wei Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490*, 2011.

Shaofei Xue, Ossama Abdel-Hamid, Hui Jiang, and Lirong Dai. Direct adaptation of hybrid DNN/HMM model for fast speaker adaptation in LVCSR based on speaker code. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6339–6343. IEEE, 2014.

Kaisheng Yao, Dong Yu, Frank Seide, Hang Su, Li Deng, and Yifan Gong. Adaptation of context-dependent deep neural networks for automatic speech recognition. In *SLT*, pages 366–369, 2012.

Takuya Yoshioka, Anton Ragni, and Mark JF Gales. Investigation of unsupervised adaptation of dnn acoustic models with filter bank input. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6344–6348. IEEE, 2014.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying A Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Anton Ragni, Valtcho Valtchev, Phil Woodland, and Chao Zhang. The HTK book (for HTK version 3.5). 2015.

Steve J Young. The use of state tying in continuous speech recognition. In *Proc. of Eurospeech'93*, 1993.

Steve J Young, Julian J Odell, and Philip C Woodland. Tree-based state tying for high accuracy acoustic modelling. In *Proceedings of the workshop on Human Language Technology*, pages 307–312. Association for Computational Linguistics, 1994.

Dong Yu, Kaisheng Yao, Hang Su, Gang Li, and Frank Seide. KL-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7893–7897. IEEE, 2013.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *arXiv preprint arXiv:1311.2901*, 2013.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Heiga Zen and Andrew Senior. Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3844–3848. IEEE, 2014.

C Zhang and PC Woodland. Context independent discriminative pre-training. *unpublished work*, 2015a.

Chao Zhang and Philip C Woodland. Parameterised sigmoid and ReLU hidden activation functions for DNN acoustic modelling. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015b.

Shiliang Zhang, Hui Jiang, and Lirong Dai. Hybrid orthogonal projection and estimation (HOPE): A new framework to learn neural networks. *Journal of Machine Learning Research*, 17(37):1–33, 2016a.

Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yaco, Sanjeev Khudanpur, and James Glass. Highway long short-term memory rnns for distant speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5755–5759. IEEE, 2016b.

Jing Zheng and Andreas Stolcke. Improved discriminative training using phone lattices. In *Ninth European Conference on Speech Communication and Technology*, 2005.

Y-T Zhou, Rama Chellappa, Aseem Vaid, and B Keith Jenkins. Image restoration using a neural network. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1141–1151, 1988.