THE APPLICATIONS OF DEEP LEARNING IN ROBUST SPEAKER RECOGNITION

ZHILI TAN

PhD

The Hong Kong Polytechnic University

2018

The Hong Kong Polytechnic University
Department of Electronic and Information Engineering

The Applications of Deep Learning in Robust Speaker Recognition

Zhili TAN

A thesis submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

January 2018

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

_____(Signed)

___Zhili TAN___(Name of student)

ABSTRACT

Speaker verification aims to verify whether a test utterance is spoken by a target speaker. Since 2011, the i-vector approach together with probabilistic linear discriminant analysis (PLDA) have dominated this field. Under this framework, each utterance is represented by a low-dimensional i-vector that captures speaker- and channel-dependent characteristics, and the PLDA model aims to separate the speaker variability from channel variability in the i-vector space. On the other hand, in recent years, deep learning has achieved a great success in many areas, including speech recognition, computer vision, speech synthesis and music recognition. This thesis explores the applications of deep learning in speaker verification, especially under the i-vector/PLDA framework.

To address the limitations of hand-crafted acoustic features, this thesis proposes a deep architecture formed by stacking a deep belief network (DBN) on top of a denoising autoencoder (DAE) for noise robust speaker identification. After backpropagation fine-tuning, the network – referred to as denoising autoencoder–deep neural network (DAE–DNN) – outputs the posterior probabilities of speakers and the top hidden layer outputs speaker-dependent bottleneck (BN) features. The autoencoder aims to reconstruct the clean spectra of a noisy test utterance using the spectra of the noisy test utterance and its SNR as input. With this denoising capability, the output from the bottleneck layer can be considered as a low-dimensional representation of the denoised utterances. These frame-based bottleneck features are then used to train an i-vector extractor and a PLDA model for speaker identification. Experimental results based on a noise-contaminated YOHO corpus show that the bottleneck features out-

perform the conventional MFCC under low SNR conditions and that the fusion of the two features leads to further performance gain, suggesting that the two features are complementary to each other.

A limitation of the above network is that the BN feature vectors tend to be very similar across the whole utterance, causing numerical difficulty when training the UBM and the i-vector extractor. This problem, however, can be overcome by training the DAE–DNN to produce senone posteriors instead of speaker posteriors. The resulting DAE–DNN produces not only denoised BN features, but also senone posteriors from which a senone i-vector extractor can be trained and senone i-vectors can be extracted. Because the frame-based BN features are now aligned to senone clusters instead of acoustic clusters, the resulting i-vectors characterize how individual speakers pronounce different phones, which allows more precise comparisons between speakers. Through extensive evaluations on NIST 2012 SRE, this thesis demonstrates that senone i-vectors outperform conventional GMM i-vectors. More interestingly, the BN features are not only phonetically discriminative, results suggest that they also contain sufficient speaker information to produce BN-based senone i-vectors that outperform the conventional senone i-vectors. This thesis also shows that DAE training is more beneficial to BN feature extraction than senone posterior estimation.

Although the denoised BN-based senone i-vectors improve the noise robustness significantly compared to the MFCC–GMM ones, adverse acoustic conditions and duration variability in utterances could still have detrimental effect on PLDA scores. This thesis also proposes and investigates several DNN-based PLDA score compensation, transformation and calibration algorithms for enhancing the noise robustness of i-vector/PLDA systems. Unlike conventional calibration methods where the required score shift is a linear function of SNR or log-duration, the DNN approach learns the complex relationship between the score shifts and the combination of i-vector pairs

and uncalibrated scores. Furthermore, with the flexibility of DNNs, it is possible to explicitly train a DNN to recover the clean scores without having to estimate the score shifts. To alleviate the overfitting problem, multi-task learning is applied to incorporate auxiliary information such as SNRs and speaker ID of training utterances into the DNN. Experiments on NIST 2012 SRE show that score calibration derived from multi-task DNNs can improve the performance of the conventional score-shift approach significantly, especially under noisy conditions.

AUTHOR'S PUBLICATIONS

Journal Papers

1. **Z. L. Tan**, M. W. Mak, Y. K. Zhu, and B. Mak, "Denoised Senone I-Vectors for Robust Speaker Verification", IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 26, no. 4, pp. 820–830, April 2018.

2. **Z. L. Tan**, M. W. Mak, and B. Mak, "DNN-Based Score Calibration with Multi-Task Learning for Noise Robust Speaker Verification," IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 26, no. 4, pp. 700–712, April 2018.

Conference Papers

1. **Z. L. Tan** and M. W. Mak, "I-Vector DNN Scoring and Calibration for Noise Robust Speaker Verification," in Proc. Interspeech, 2017, pp. 1562–1566.

2. **Z. L. Tan**, Y. K. Zhu, M. W. Mak, and B. Mak, "Senone i-vectors for robust speaker verification," in ISCSLP, 2016.

3. **Z. L. Tan** and M. W. Mak, "Bottleneck features from SNR-adaptive denoising deep classifier for speaker identification," in APSIPA ASC, 2015, pp. 1035–1040.

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude and deep regards to my supervisor Dr. Man-Wai Mak, for his guidance, support, advice and encouragement throughout my Ph.D. study period in the Hong Kong polytechnic University. His persistent patience and vast knowledge in many areas touched me deeply, and I have learned a lot from him in both academic and daily life. It is my fortune to have Dr. Mak as my supervisor. Without his assistance and suggestion in academic and writing, it is impossible to complete my Ph.D. study and this dissertation.

I would like to thank Prof. Brian Kan-Wing MAK from the Hong Kong University of Science and Technology, who is our collaborator and coauthor, for his constructive comments and suggestions to improve our papers. My gratitude also goes to his students Dr. Dongpeng Chen, Yingke Zhu and Hengguan Huang, for their efforts and expertise.

I would like to express sincere gratitude to various bodies from the Hong Kong polytechnic University, where I have the opportunity to study with. I would like to thank my seniors and friends Dr. Lawrence Chi-Chung Cheung, Dr. Shibiao Wan, Dr. Wei Rao, Dr. Lingling Cao, Wenqiang Xu, Longxin Li, Zhuhai Ye, Xi Zhang and Jing Wang, and all the professors, lecturers, tutors and classmates that I have met during my study period in the Hong Kong polytechnic University, for their teaching, supports and suggestion. They taught and affected me a lot in academic, career and daily life.

Moreover, it is my pleasure to acknowledge the Research Office of the Hong Kong Polytechnic University and the General Office of the department of Electronic and

Information Engineering for their generous support over the past years. They have created an excellent environment for postgraduate study.

Last but not least, I hope to express my deepest gratitude to my parents, my family and my girlfriend for their endless love and support. Their understanding and care helps me a lot during my Ph.D. study period.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ASR	Automatic speech recognition
BN	Bottleneck
BP	Backpropagation
CC	Common evaluation condition
CD	Contrastive divergence
CMVN	Cepstral mean-variance normalization
CNN	Convolutional neural network
DAC13	2013 Domain Adaptation Challenge
DAE	Denoising autoencoder
DAE–DNN	Denoising autoencoder–deep neural network
DCF	Decision cost function
DCT	Discrete cosine transform
DET	Detection error tradeoff
DFT	Discrete Fourier transform
DNN	Deep neural network
EM	Expectation-maximization
FFT	Fast Fourier transform

fMLLR	Feature-space maximum likelihood linear regression
GMM	Gaussian mixture model
HMM	Hidden Markov model
IDFT	Inverse discrete Fourier transform
JFA	Joint factor analysis
LDA	Linear discriminative analysis
LR	Logistic regression
LVCSR	Large vocabulary continuous speech recognition
MAP	Maximum a posteriori
MFCC	Mel-frequency cepstral coefficient
NIST	National Institute of Standards and Technology
PCA	Principal component analysis
PLDA	Probabilistic linear discriminative analysis
RBM	Restricted Boltzmann machine
RNN	recurrent neural network
SNR	Signal-to-niose ratio
SRE	Speaker recognition evaluation
SVM	Support vector machine
UBM	Universal background model
VAD	Voice activity detection
WCCN	Within-class covariance normalization

LIST OF NOTATIONS

$\boldsymbol{s}(n)$	Waveform signal
$\boldsymbol{o}$	Acoustic feature vector
F	Dimension of acoustic feature vectors
$\boldsymbol{\lambda}$	Weights of a GMM
$\boldsymbol{\mu}$	Mean vector of a GMM
$\boldsymbol{\Sigma}$	Covariance matrix of a GMM
$\boldsymbol{\Sigma}^{(b)}$	Covariance matrix of a UBM
γ	Posterior probability
$\boldsymbol{T}$	Total variability matrix of i-vector extraction
$\boldsymbol{w}$	Latent factor of i-vector extraction
$\langle \cdot \vert \cdot \rangle$	Conditional expectation
$\boldsymbol{x}_i^{(j)}$	The i-th i-vector for speaker j
N	0th-order Baum-Welch statistics
$\tilde{\boldsymbol{f}}$	1st-order Baum-Welch statistics
$\boldsymbol{S}$	2nd-order Baum-Welch statistics
$\boldsymbol{V}$	Speaker factor loading matrix of PLDA
$\boldsymbol{z}$	Latent factor of PLDA

$\boldsymbol{\epsilon}$	Residual noise of PLDA
$\boldsymbol{\Sigma}^{(p)}$	Covariance matrix of residual noises in PLDA
$\boldsymbol{Q}$	Total covariance matrix of i-vectors
$\boldsymbol{P}$	Across-speaker covariance matrix of i-vectors
S	PLDA score
E	Loss function in backpropagation
$w_{ij}^{(n)}$	The weight connecting the i-th node in layer $n-1$, and the j-th node in layer n
$y_i^{(n)}$	The weighted sum for the i-th node in layer n
t_i	The target output of the i-th node in the output layer
η	Learning rate in backpropagation
v_i	The i-th visible node in RBM
h_i	The i-th hidden node in RBM

Chapter 1

INTRODUCTION

1.1 Automatic Speaker Recognition

Similar to fingerprint recognition and face recognition, automatic speaker recognition is a very important pathway to biometric authentication. Speaker recognition is based on the fact that speech production organs are speaker-dependent, so that we are able to recognize the identity of a speaker purely from his/her speech. It is worth investigating because it can be used in various applications, such as authentication (e.g. telephone bank services), surveillance (e.g. conference recording) and forensics (e.g. phone fraud). Depending on the purpose, there are two types of speaker recognition: speaker identification and speaker verification.

Speaker identification is to determine whether an unknown speaker matches one of the known speakers, which means that it is a one-to-many mapping (see Fig. 1.1). Specifically, given an utterance from an unknown speaker, the posterior probabilities of known speakers in a database are computed, and the known identity is the one whose posterior probability is the highest.

Speaker verification is to determine whether an unknown speaker matches a specific speaker, which means that it is a one-to-one mapping (see Fig. 1.2). Specifically, the likelihood that the unknown speech is spoken by the claimed speaker is compared with the likelihood that it is spoken by somebody else. Then, the system accepts the speaker if the former likelihood is significantly larger than the latter.

Speaker verification can be further divided into text-dependent and text-independent.

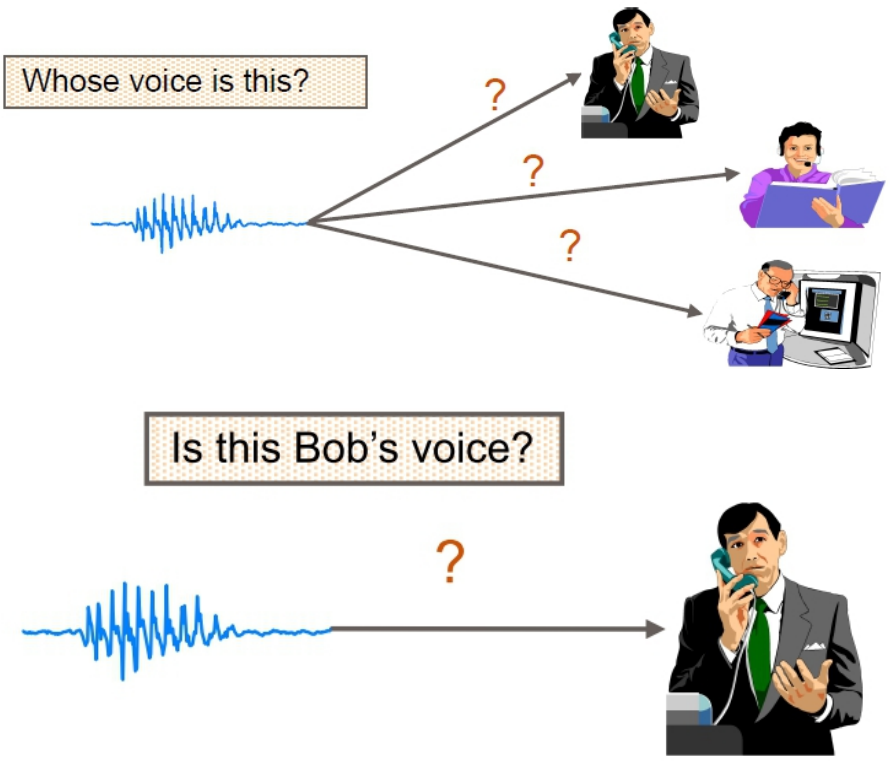

Figure 1.2: Speaker verification [1].

The former requires the speakers to speak exactly the same words and sentences in both enrollment and verification phases, where the latter does not have such restriction. Systems need to consider the phonetic variabilities for the text-independent case, which bring a great challenge especially for short utterances.

Automatic speaker recognition under controlled environments is easy. But under uncontrolled environments, errors are still very high because of different types of variability in speech signals. For example, the environmental noise, channel variabilities, Lombard effect [2], emotions and accents all bring difficulties to speaker verification.

1.2 Thesis Organization

This thesis is organized as follows:

In Chapter 2, we introduce the history of speaker recognition, including the GMM methods, GMM supervectors, and factor analysis. We also give a literature review

on robust speaker recognition.

In Chapter 3, we review the basics of deep learning techniques, focussing on the feedforward fully-connected neural networks. The model structure, backpropagation (BP) fine-tuning and Restricted Boltzmann Machine (RBM) will be discussed.

In Chapter 4, we propose to extract speaker-discriminative bottleneck features from denoising DNNs for robust speaker identification.

In Chapter 5, we propose a noise-robust phonetically discriminative features, namely senone i-vectors, that can be obtained from a denoising autoencoder DNN (DAE–DNN) and a factor analysis model based on the bottleneck features extracted from the DAE–DNN.

In Chapter 6, we propose using multi-task DNNs to calibrate PLDA scores in order to compensate for the score shifts under noisy environments.

In Chapter 7, we draw the conclusions and briefly outline the future research direction in this area.

Chapter 2

SPEAKER RECOGNITION SYSTEMS

This chapter presents the theoretical backgrounds of speaker recognition systems, including feature extraction, speaker modeling and speaker classification. It also provides a literature review on robust speaker recognition.

2.1 Mel-frequency Cepstral Coefficients

Given an utterance, voice activity detection (VAD) is applied to detect the regions of the speech signals that contain speech. Then, to recognize the identities of speakers, all speaker recognition systems require to perform two consecutive tasks: feature extraction and pattern classification. For the former, Mel-frequency cepstral coefficients (MFCCs) [3] are by far the most common *handcrafted* features. MFCCs are considered as handcrafted features because their extraction process incorporates the frequency sensitivity of human ears through a mel-scale filter-bank shown in Fig. 2.1. Another reason is that the extraction process uses discrete cosine transform (DCT) to de-correlate the features so that the resulting MFCC vectors can be modeled by a Gaussian mixture model (GMM) with diagonal covariance matrices, which simplifies computation significantly.

2.2 GMM-based Speaker Models

Because an utterance can be represented as a series of acoustic features such as MFCCs, Reynolds *et al.* [4] proposed to use a Gaussian mixture model (GMM) to model these features in 1995. A GMM is a linear combination of Gaussian probabil-

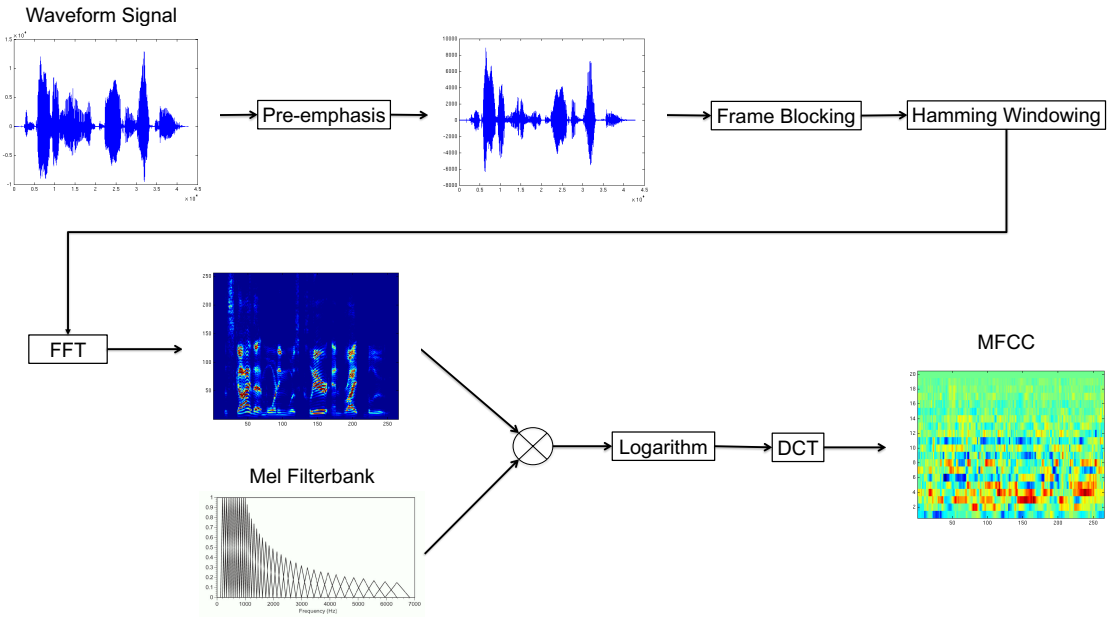

Figure 2.1: Procedure of MFCC computation.

ity density functions (PDFs). Denote $\mathcal{O}_i = \{\boldsymbol{o}_{i,1}, \ldots, \boldsymbol{o}_{i,T_i}\}$ as a set of F-dimensional acoustic vectors of the i-th utterance from speaker j,[1] and C as the number of mixtures in a GMM whose parameters are given by $\Lambda^{(j)} = \left\{\boldsymbol{\lambda}^{(j)}, \boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right\}$. Then, the likelihood of the t-th frame $\boldsymbol{o}_{i,t}$ is

$$p(\boldsymbol{o}_{i,t}) = \sum_{c=1}^{C} \lambda_c^{(j)} \mathcal{N}(\boldsymbol{o}_{i,t} | \boldsymbol{\mu}_c^{(j)}, \boldsymbol{\Sigma}_c^{(j)}),$$

where $\lambda_c^{(j)}$ and $\mathcal{N}(\boldsymbol{o}_{i,t} | \boldsymbol{\mu}_c^{(j)}, \boldsymbol{\Sigma}_c^{(j)})$ are respectively the weight and the Gaussian density of the c-th mixture with mean vector $\boldsymbol{\mu}_c^{(j)} \in \Re^F$ and covariance matrix $\boldsymbol{\Sigma}_c^{(j)} \in \Re^{F \times F}$.

To train the GMM for speaker j, we used the dataset $\{\mathcal{O}_1, \ldots, \mathcal{O}_i, \ldots, \mathcal{O}_I\}$ from speaker j to estimate the parameters $\left\{\boldsymbol{\lambda}^{(j)}, \boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right\}$ using the expectation-maximization (EM) algorithm [5]. Specifically, the update formulas for the GMM parameters are

[1] To avoid cluttering of symbols in equations, we drop the index j in the acoustic vectors $\boldsymbol{o}$.

given by

$$\lambda_c^{(j)} = \frac{\sum_i \sum_t \gamma_c^{(j)}(\boldsymbol{o}_{i,t})}{\sum_i \sum_t 1},$$

$$\boldsymbol{\mu}_c^{(j)} = \frac{\sum_i \sum_t \gamma_c^{(j)}(\boldsymbol{o}_{i,t}) \boldsymbol{o}_{i,t}}{\sum_i \sum_t \gamma_c^{(j)}(\boldsymbol{o}_{i,t})},$$

$$\boldsymbol{\Sigma}_c^{(j)} = \frac{\sum_i \sum_t \gamma_c^{(j)}(\boldsymbol{o}_{i,t})(\boldsymbol{o}_{i,t} - \boldsymbol{\mu}_c^{(j)})(\boldsymbol{o}_{i,t} - \boldsymbol{\mu}_c^{(j)})^\intercal}{\sum_i \sum_t \gamma_c^{(j)}(\boldsymbol{o}_{i,t})},$$

where $\gamma_c^{(j)}(\boldsymbol{o}_{i,t})$ is the posterior probability that $\boldsymbol{o}_{i,t}$ is generated by the c-th mixture of the GMM $\boldsymbol{\Lambda}^{(j)} = \left\{\boldsymbol{\lambda}^{(j)}, \boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right\}$:

$$\gamma_c^{(j)}(\boldsymbol{o}_{i,t}) = \frac{\lambda_c^{(j)} \mathcal{N}(\boldsymbol{o}_{i,t} | \boldsymbol{\mu}_c^{(j)}, \boldsymbol{\Sigma}_c^{(j)})}{\sum_{k=1}^{C} \lambda_k^{(j)} \mathcal{N}(\boldsymbol{o}_{i,t} | \boldsymbol{\mu}_k^{(j)}, \boldsymbol{\Sigma}_k^{(j)})}.$$

However, training a speaker-specific GMM needs a large amount of data from the corresponding target speaker. In 2000, Reynolds *et al.* [6] proposed to train a universal background model (UBM) $\Lambda^{(b)} = \left\{\boldsymbol{\lambda}^{(b)}, \boldsymbol{\mu}^{(b)}, \boldsymbol{\Sigma}^{(b)}\right\}$ by using the speech of many speakers, followed by performing maximum *a posteriori* (MAP) adaptation using speaker-dependent data. Specifically, the mean vectors of the UBM are adapted as follows [7]:

$$\boldsymbol{\mu}_c^{(j)} = \frac{\sum_t \gamma_c(\boldsymbol{o}_t) \boldsymbol{o}_t}{\sum_t \gamma_c(\boldsymbol{o}_t) + r} + \frac{r \boldsymbol{\mu}_c^{(b)}}{\sum_t \gamma_c(\boldsymbol{o}_t) + r},$$

where r is called the relevance factor. The method avoids over-fitting the speaker-dependent GMM, especially when the amount of speech from the target speaker is very limited.

Given the acoustic vectors $\mathcal{O}^{(t)}$ from a test speaker and a claimed identity j, the verification score is a log-likelihood ratio :

$$S_{\text{LR}}(\mathcal{O}^{(t)} | \Lambda^{(j)}, \Lambda^{(b)}) = \log p(\mathcal{O}^{(t)} | \Lambda^{(j)}) - \log p(\mathcal{O}^{(t)} | \Lambda^{(b)}), \tag{2.1}$$

where $\log p(\mathcal{O}^{(t)}|\Lambda^{(j)})$ is the log-likelihood of $\mathcal{O}^{(t)}$ given the speaker model $\Lambda^{(j)}$, which is given by

$$\log p(\mathcal{O}^{(t)}|\Lambda^{(j)}) = \sum_{\mathbf{o}\in\mathcal{O}^{(t)}} \log \sum_{c=1}^{C} \lambda_c^{(b)} \mathcal{N}(\mathbf{o}|\boldsymbol{\mu}_c^{(j)}, \boldsymbol{\Sigma}_c^{(b)}). \tag{2.2}$$

Note that only the mean vectors in the speaker model are speaker-dependent, i.e., $\Lambda^{(j)} = \{\lambda_c^{(b)}, \boldsymbol{\mu}_c^{(j)}, \boldsymbol{\Sigma}_c^{(b)}\}_{c=1}^{C}$. This is because only the means are adapted in practice.

2.3 GMM-Supervectors

The lack of discriminative training in GMM–UBM limits its performance. However, most of the discriminative classifiers require the input patterns to have a fixed-dimension. Therefore, how to represent variable-length utterances by fixed-length feature vectors is always an issue in speaker verification.

In 2006, Campbell *et al.* [8] proposed to use a fixed-dimensional vector, namely supervector, to represent a single utterance. A supervector is formed by stacking the mean vectors of a MAP-adapted GMM. After scaling by the mixture coefficients and covariance matrices of the UBM, a GMM-supervector is produced. Then, the speaker-dependent supervector is presented to a speaker-dependent support vector machine (SVM) for classification and scoring.

2.4 Factor Analysis, I-Vectors and PLDA

2.4.1 I-Vectors

Although the supervector system in Section 2.3 can represent variable-length utterances in a fixed-dimensional space, the dimension is too large, which limits the choice of classifiers. For example, if the acoustic features have 60 dimensions and the GMM have 1024 mixtures, the dimension of the supervector is $60 \times 1024 = 61,440$.

In 2011, Dehak *et al.* [9] conjectures that the GMM-supervectors vary in a subspace of the supervector space. To find such subspace, they consider the utterance-

dependent GMM-supervectors to be generated by a factor analysis model whose factor loading matrix defines the subspace. Instead of actually finding the utterance-dependent GMM-supervectors and then estimate the loading matrix, they aligned the speech frames of many utterances to a universal background model (a speaker-independent GMM) to determine the sufficient statistics of the utterances, which in turn determine the maximum-likelihood estimate of the loading matrix. As these utterances are spoken by many speakers under different channels and acoustic environments, the loading matrix define not only speaker variability but also non-speaker variabilities such as channel variability. They refer to the subspace defined by the loading matrix as the total variability (TV) space and the factors as the total factors. For each utterance, the posterior mean of the factors is called the i-vector. Similar to the GMM-supervectors, the i-vectors are of fixed dimension regardless of the utterances' duration. However, unlike GMM-supervectors, the dimension of i-vectors are significantly smaller, e.g., 500 versus 61,400.

The GMM-supervector representing the i-th utterance is assumed to be generated by a factor analysis model [10]:

$$\boldsymbol{\mu}_i = \boldsymbol{\mu}^{(b)} + \boldsymbol{T}\boldsymbol{w}_i, \tag{2.3}$$

where $\boldsymbol{\mu}^{(b)}$ is the supervector formed by stacking the mean vectors of a universal background model (UBM), $\boldsymbol{T}$ is a $CF \times D$ low-rank total variability matrix (T-matrix) modeling the speaker and channel subspaces, and $\boldsymbol{w}_i$ is a D-dimensional latent factor whose prior follows a standard Gaussian distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. While $\boldsymbol{\mu}_i$ and $\boldsymbol{w}_i$ are utterance-dependent, $\boldsymbol{\mu}^{(b)}$ and $\boldsymbol{T}$ are shared across all speakers and utterances.

Eq. 2.3 is a generative model in that given the $\boldsymbol{w}_i$ of a speaker, his/her supervector $\boldsymbol{\mu}_i$ can be generated. Of course, the model is not perfect and there will be discrepancy (error) between the truth value of $\boldsymbol{\mu}_i$ and the generated one. In factor analysis, we typically assume that the discrepancy follows a Gaussian distribution with zero mean

and covariance $\boldsymbol{\Sigma}$. As the dimension of $\boldsymbol{\mu}_i$ is very high, $\boldsymbol{\Sigma}$ is assumed to be diagonal. In most practical implementation of i-vector extraction, $\boldsymbol{\Sigma}$ is approximated by the covariance matrices in the UBM, i.e.,

$$\boldsymbol{\Sigma} \approx \text{diag}\{\boldsymbol{\Sigma}^{(b)}\} = \text{diag}\{\boldsymbol{\Sigma}_1^{(b)}, \ldots, \boldsymbol{\Sigma}_C^{(b)}\},$$

where $\boldsymbol{\Sigma}_c^{(b)}$ is the c-th covariance matrix of the UBM, which is typically diagonal.

Given N training utterances, the T-matrix can be estimated by the following EM algorithm [10, 11]:

• E-step:

$$\langle \boldsymbol{w}_i | \mathcal{O}_i \rangle = \boldsymbol{L}_i^{-1} \sum_{c=1}^{C} \boldsymbol{T}_c^{\intercal} (\boldsymbol{\Sigma}_c^{(b)})^{-1} \tilde{\boldsymbol{f}}_{ic}, \tag{2.4a}$$

$$\langle \boldsymbol{w}_i \boldsymbol{w}_i^{\intercal} | \mathcal{O}_i \rangle = \boldsymbol{L}_i^{-1} + \langle \boldsymbol{w}_i | \mathcal{O}_i \rangle \langle \boldsymbol{w}_i | \mathcal{O}_i \rangle^{\intercal}, \tag{2.4b}$$

$$\boldsymbol{L}_i = \boldsymbol{I} + \sum_{c=1}^{C} N_{ic} \boldsymbol{T}_c^{\intercal} (\boldsymbol{\Sigma}_c^{(b)})^{-1} \boldsymbol{T}_c, \tag{2.4c}$$

where $i = 1, \ldots, N$.

• M-step:

$$\boldsymbol{T}_c = \left[\sum\nolimits_i \tilde{\boldsymbol{f}}_{ic} \langle \boldsymbol{w}_i | \mathcal{O}_i \rangle^{\intercal} \right] \left[\sum\nolimits_i N_{ic} \langle \boldsymbol{w}_i \boldsymbol{w}_i^{\intercal} | \mathcal{O}_i \rangle \right]^{-1}. \tag{2.5}$$

In Eq. 2.4 and Eq. 2.5, $\langle \cdot | \cdot \rangle$ denotes conditional expectation; i indexes the set of training utterances; N is the number of training utterances; $\boldsymbol{T}_c$ is the c-th partition of $\boldsymbol{T}$; $\boldsymbol{\Sigma}_c^{(b)}$ is the c-th covariance matrix of the UBM; N_{ic} and $\tilde{\boldsymbol{f}}_{ic}$ are the 0th- and 1st-order Baum-Welch statistics respectively:

$$\begin{aligned} N_{ic} &= \sum\nolimits_t \gamma_c(\boldsymbol{o}_{i,t}), \\ \tilde{\boldsymbol{f}}_{ic} &= \sum\nolimits_t \gamma_c(\boldsymbol{o}_{i,t})(\boldsymbol{o}_{i,t} - \boldsymbol{\mu}_c^{(b)}). \end{aligned} \tag{2.6}$$

Given the t-th frame of the i-th utterance, $\boldsymbol{o}_{i,t}$ is the MFCC vector of the t-th frame and $\gamma_c(\boldsymbol{o}_{i,t})$ in Eq. 2.6 is the posterior of the c-th mixture component in the UBM:

$$\gamma_c(\boldsymbol{o}_{i,t}) = \frac{\lambda_c^{(b)} \mathcal{N}(\boldsymbol{o}_{i,t}|\boldsymbol{\mu}_c^{(b)}, \boldsymbol{\Sigma}_c^{(b)})}{\sum_{j=1}^{C} \lambda_j^{(b)} \mathcal{N}(\boldsymbol{o}_{i,t}|\boldsymbol{\mu}_j^{(b)}, \boldsymbol{\Sigma}_j^{(b)})}, \tag{2.7}$$

where $\left\{\lambda_j^{(b)}, \boldsymbol{\mu}_j^{(b)}, \boldsymbol{\Sigma}_j^{(b)}\right\}_{j=1}^{C}$ are UBM parameters.

Once the T-matrix has been estimated, the i-vector $\boldsymbol{x}_i = \langle \boldsymbol{w}_i | \mathcal{O}_i \rangle$ representing the i-th utterance can be computed according to Eq. 2.4a.

Note that the acoustic vectors $\boldsymbol{o}_{i,t}$'s are not limited to MFCCs. Instead, they can be bottleneck (BN) vectors extracted from a DNN. However, caution should be taken when BN vectors are used. If the DNN is trained to produce speaker posteriors, the BN vectors from the same utterance will be very similar because they come from the same speaker. In other words, for the entire utterance, the frame-based activations at the bottleneck layer are very similar so that the DNN can give a large posterior probabilities in the output node corresponding to the speaker and small probabilities in the rest. The similarity in the BN vectors causes them to align to the same (potentially small) group of Gaussians in the BN-based UBM. This property leads to sparsity in the zeroth- and first-order statistics in Eq. 2.6, which in turns causes numerical difficulty when computing the matrix inverses in Eq. 2.4 and Eq. 2.5. Another issue is that the BN vectors tend to form isolated islands in the BN space (otherwise they cannot differentiate speakers). The small within-speaker variances essentially reduce the effective number of vectors for training the BN-based UBM, which again causes numerical problems. Both of these drawbacks motivate us to use senone posteriors instead of speaker posteriors, as detailed in Chapter 5.

2.4.2 Within-Class Covariance Normalization

To normalize the within-speaker covariance of i-vectors, within-class covariance normalization (WCCN) [89] is applied. First, the within-speaker scatter matrix of i-vectors, $\boldsymbol{S}_w$, is estimated:

$$\boldsymbol{S}_w = \sum_{j=1}^{J} \sum_{i=1}^{N_j} (\boldsymbol{x}_i^{(j)} - \boldsymbol{\mu}^{(j)})(\boldsymbol{x}_i^{(j)} - \boldsymbol{\mu}^{(j)})^\intercal,$$

where J is the number of speakers, N_j is the number of i-vectors from speaker j, $\boldsymbol{x}_i^{(j)}$ is the i-th i-vector from speaker j, and $\boldsymbol{\mu}^{(j)}$ is the mean of the i-vectors from speaker j. Then, through the Cholesky decomposition

$$\boldsymbol{B}\boldsymbol{B}^\intercal = \left(\frac{1}{J}\boldsymbol{S}_w\right)^{-1},$$

we compute the WCCN transform matrix $\boldsymbol{B}$.

2.4.3 Probabilistic Linear Discriminative Analysis

As no labels are used in the EM algorithm in Section 2.4.1, the unsupervised training of T-matrix causes the i-vectors to capture both the speaker- and channel-dependent characteristics of the utterances. Inspired by the work of Prince [12], in 2010, Kenny [13] proposed a Bayesian factor analysis model known as heavy-tailed probabilistic linear discriminant analysis (HT-PLDA) that can separate the speaker and channel variabilities in the i-vector space. The basic idea is that the prior of i-vectors follows a Student's t-distribution. Subsequent to Kenny's pioneer work, Garcia-Romero and Espy-Wilson [14] found that after applying length normalization to the i-vectors, the heavy-tailed distribution can be replaced by a simple Gaussian distribution. This finding leads to the so called Gaussian PLDA (G-PLDA). By leveraging the speaker labels of the training i-vectors, a speaker loading matrix and a channel loading matrix

can be jointly trained.

The G-PLDA algorithm can be applied to i-vectors to represent the speaker-dependent characteristics in a lower dimensional subspace. Consider a data set comprising J speakers and N_j i-vectors from the j-th speaker, $\mathcal{X} = \left\{ \boldsymbol{x}_i^{(j)}; i = 1, \ldots, N_j; j = 1, \ldots, J \right\}$. Similar to Eq. 2.3, the M-dimensional speaker representation $\boldsymbol{z}^{(j)}$ is the latent factor of the FA model [10]:

$$\boldsymbol{x}_i^{(j)} = \boldsymbol{m} + \boldsymbol{V}\boldsymbol{z}^{(j)} + \boldsymbol{\epsilon}_i^{(j)},$$

where $\boldsymbol{x}_i^{(j)} \in \Re^D$, $\boldsymbol{m} \in \Re^D$ and $\boldsymbol{V} \in \Re^{D \times M}$ are the i-th i-vector from speaker j, the global mean of all i-vectors and the speaker factor loading matrix, respectively. The residual noise $\boldsymbol{\epsilon}_i^{(j)} \in \Re^D$ is assumed to follow a normal distribution with zero mean and covariance matrix $\boldsymbol{\Sigma}^{(p)}$. The parameters $\left\{ \boldsymbol{m}, \boldsymbol{V}, \boldsymbol{\Sigma}^{(p)} \right\}$ can be estimated by the EM algorithm similarly to Section 2.4.1, but with speaker labels and $C = 1$.

The PLDA score of i-vector pair ($\boldsymbol{x}_{tgt}$, $\boldsymbol{x}_{tst}$) can be expressed in terms of the log-likelihood ratio $\mathrm{LLR}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst})$:

$$\begin{aligned}
&S_{\mathrm{PLDA}}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}) \\
&= \mathrm{LLR}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}) \\
&= \log \left(\frac{p(\text{i-vector pair}|\text{same speaker})}{p(\text{i-vector pair}|\text{different speaker})} \right) \\
&= \log \mathcal{N} \left(\begin{bmatrix} \boldsymbol{x}_{tgt} \\ \boldsymbol{x}_{tst} \end{bmatrix}; \begin{bmatrix} \boldsymbol{m} \\ \boldsymbol{m} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{tot} & \boldsymbol{\Sigma}_{ac} \\ \boldsymbol{\Sigma}_{ac} & \boldsymbol{\Sigma}_{tot} \end{bmatrix} \right) - \log \mathcal{N} \left(\begin{bmatrix} \boldsymbol{x}_{tgt} \\ \boldsymbol{x}_{tst} \end{bmatrix}; \begin{bmatrix} \boldsymbol{m} \\ \boldsymbol{m} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{tot} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Sigma}_{tot} \end{bmatrix} \right) \\
&= \frac{1}{2} \left[\boldsymbol{x}_{tgt}^{\intercal} \boldsymbol{Q} \boldsymbol{x}_{tgt} + \boldsymbol{x}_{tst}^{\intercal} \boldsymbol{Q} \boldsymbol{x}_{tst} + 2 \boldsymbol{x}_{tgt}^{\intercal} \boldsymbol{P} \boldsymbol{x}_{tst} \right] + \text{const},
\end{aligned} \tag{2.8}$$

where $\boldsymbol{\Sigma}_{tot} = \boldsymbol{V}\boldsymbol{V}^{\intercal} + \boldsymbol{\Sigma}^{(p)}$ and $\boldsymbol{\Sigma}_{ac} = \boldsymbol{V}\boldsymbol{V}^{\intercal}$, and $\boldsymbol{Q} = \boldsymbol{\Sigma}_{tot}^{-1} - (\boldsymbol{\Sigma}_{tot} - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac})^{-1}$ and $\boldsymbol{P} = \boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac}(\boldsymbol{\Sigma}_{tot} - \boldsymbol{\Sigma}_{ac}\boldsymbol{\Sigma}_{tot}^{-1}\boldsymbol{\Sigma}_{ac})^{-1}$ are matrices derived from the total covariances and across-speaker covariances of i-vectors [14].

2.5 Performance Evaluation Metrics

There are two types of error rates to measure the performance of a speaker verification system:

1. False Rejection Rate (FRR) ($P_{Miss|Target}$), also known as the miss probability, represents the chance of falsely rejecting the true speakers:

$$P_{Miss|Target} = \frac{N_{Miss}}{N_{Target}},$$

   where N_{Miss} is the number of false rejections given a decision threshold, and N_{Target} is the total number of trials with the test-speakers being the target-speakers. FRR increases with the decision threshold increases.

2. False Acceptance Rate (FAR) ($P_{FalseAlarm|Nontarget}$), also known as the false alarm probability, represents the chance of falsely accepting the imposters as true speakers:

$$P_{Miss|Target} = \frac{N_{Miss}}{N_{Target}},$$

   where $N_{FalseAlarm|Nontarget}$ is the number of false acceptances given a decision threshold, and N_{Target} is the total number of trials with the test-speakers being non-target speakers. FAR increases with the decision threshold decreases.

To have a balance between FRR and FAR, equal error rate (EER) and decision cost function (DCF) are used to measure the the performance of a speaker verification system. EER is the error rate when the decision threshold makes FAR and FRR equal. DCF is a linear combination of FRR and FAR, and minimum DCF (minDCF) is the minimum value of DCF when the decision threshold can be varied. The actual DCF (actDCF) is the DCF value at a fixed threshold.

2.6 Robust Speaker Recognition

2.6.1 Robust Acoustic Feature Extraction

The I-vector/PLDA system can improve the accuracy of automatic speaker recognition. However, under noisy environments, the performance degrades rapidly due to the variability in speech signals, such as the environmental noise and Lombard effect.

Fig. 2.2 is the t-SNE [15] plot of MFCCs from a 1-minute utterance under different SNR conditions. We can observe that at low SNR (e.g. 0dB), the MFCCs form very dense clusters. This means that the traditional hand-crafted acoustic features suffer from severe distortion under noisy conditions. Therefore, the use of speech enhancement techniques to improve speaker recognition performance has drawn the attention of the speaker recognition community. For example, Hasen and Hansen [16] proposed to enhance and normalize acoustic features by feature-domain factor analysis. Denoising autoencoders (DAE) [17] have been applied to restore speech either in the spectral domain [18] or in the i-vector space [19–21]. Note that unlike conventional speech enhancement, the goal of speech enhancement for speaker recognition is to robustify the acoustic feature vectors instead of reconstructing the clean waveforms.

Another approach to improving the robustness of i-vector systems is to directly reduce the distortion at the spectral level. For example, Xing and Hansen [22] reduced the frequency-shift distortion due to modulation/demodulation carrier mismatch for speaker recognition. Human tend to change their vocal effort under noisy environments (a phenomenon known as the Lombard effect), causing acoustic mismatch between normal speech and shouted speech. Saedi *et al.* [23] addressed this problem by compressing/expanding the power spectra in autocorrelation-based linear prediction features. Both [22] and [23] demonstrate that reducing spectral distortion can make the i-vectors more resilient to background noise.

It is believed that better and possibly more robust features can be extracted from DNNs. For example, bottleneck features were extracted from DNNs in [24, 25]. The

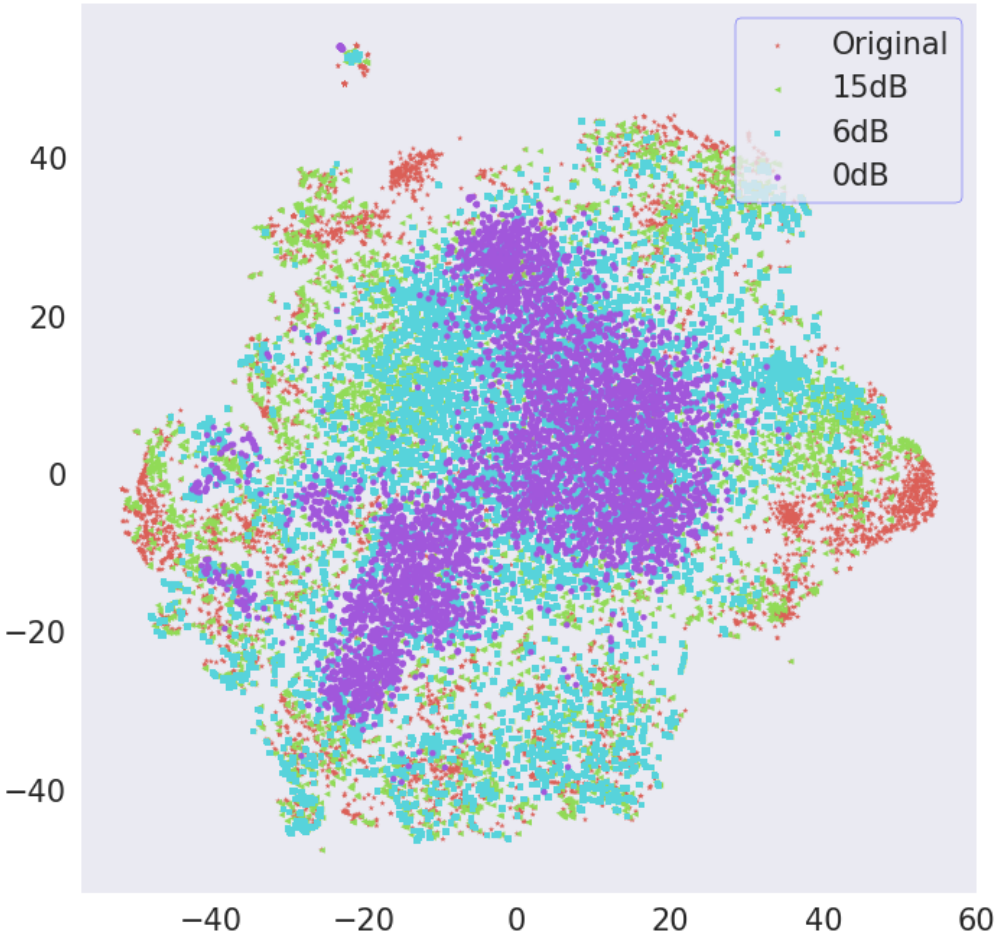

Figure 2.2: The t-SNE plot of MFCCs from a utterance under different SNR conditions.

bottleneck features can replace the standard mel-frequency cepstral coefficients [26]. A similar idea has also been applied to i-vector based DNN adaptation for robust speaker recognition [27]. Richardson *et al.* [28] demonstrated that GMM i-vectors based on the phonetically discriminative BN features outperforms the ones based on MFCC significantly on the 2013 Domain Adaptation Challenge (DAC13). Sarker *et al.* [29] showed that the phonetically discriminative BN features are complementary to the short-term cepstral features, and therefore improve the performance significantly on NIST 2008 and 2010 SRE by both score domain and feature domain fusion. These works show that the phonetically discriminative BN features still retain speaker-specific information, possibly taking the benefits of the contextual input window of DNNs.

2.6.2 Robust I-Vector Extraction

Another promising approach is to integrate DNNs into the i-vector framework. Campell [30] used DBNs pre-trained by contrastive divergence [31] to generate the posteriors of the mixtures of a universal background model (UBM). The posteriors are then used for computing the sufficient statistics of vector-based speaker recognition systems. Lei *et al.* [32, 33] replaced the posteriors of UBM's mixture components in the i-vector extractor by the posteriors of senones. In this approach, acoustic frames are aligned to senones by a DNN so that speakers can be compared based on the same set of sub-phonetic units [34].

To raise the efficiency of this framework, efforts have been made to improve the combination of i-vector and PLDA. For example, Cumani and Laface [35] proposed nonlinearly transforming the i-vectors to make them more suitable for PLDA modeling.

2.6.3 Robust PLDA Modeling

Fig. 2.3 is the t-SNE plot of i-vectors under different SNR conditions. We can observe that at low SNR (e.g. 0dB), two highly dense clusters are form. This means that the conventional i-vectors suffer from severe distortion under noisy conditions. This i-vector clustering phenomenon suggests that conventional PLDA modeling, which assumes that i-vectors follow a single Gaussian distribution, will have difficulty in modeling the multi-modal distribution of the i-vectors.

In light of the limitation of the conventional PLDA models, attempts have been made to improve noise robustness in PLDA models. For example, Hasan *et al.* [36] and Garcia-Romero *et al.* [37] trained a PLDA model by pooling speech from multiple conditions, and Li and Mak [38, 39] modeled the noise-level variability in utterances by introducing an SNR factor and an SNR subspace into the PLDA model. In [40,41], Mak *et al.* advocated that utterances of different SNR levels will not only cause the i-vectors to fall on different regions of the i-vector spaces but also change the orientation of the speaker subspace. A mixture PLDA model with mixture alignments determined by the SNR level of utterances was then derived to model the SNR-dependent i-vectors.

Besides, a number of score calibration methods [42–44] have been proposed to compensate for the detrimental effect on the PLDA scores. While many of these methods can compensate for the duration mismatch only, there are techniques also take the SNR mismatch into account [45–48]. All of these methods compensate for the detrimental effect by modeling it as a shift in the PLDA scores. The goal is to estimate the appropriate shift from some meta data (e.g., duration and SNR [45,46]) or from the i-vectors [47] to counteract the effect.

In [42, 45, 46], the score shift was deterministic and was assumed to be linearly related to the SNR of utterances and/or to the logarithm of utterance duration. In [48], the shift was stochastic and was assumed to follow a Gaussian distribution

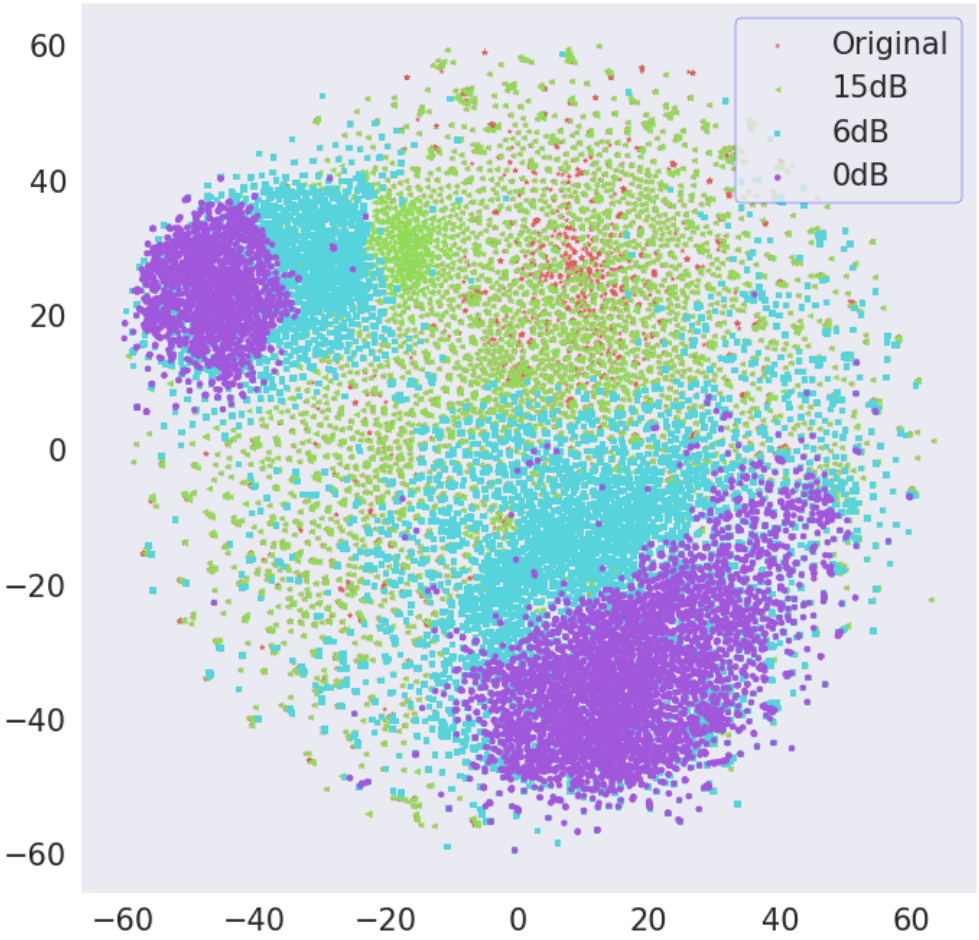

Figure 2.3: The t-SNE plot of i-vectors under different SNR conditions.

with mean and variance dependent on the speech quality. Given an observed noisy score, a Bayesian network was used to infer the posterior distribution of the target and non-target hypotheses, from which a calibrated likelihood-ratio is computed. On the other hand, the score shift in [47,49] was assumed to be simple functions (bilinear transformation and cosine distance) of the two quality vectors derived from the i-vectors involved in the scoring. While promising results have been achieved, the relationship between score shift and SNR and log-duration may not be linear, and the bilinear transformation and cosine distance scores may not accurately reflect the true relationship between the score shift and i-vector quality.

2.6.4 Robust Back-ends

Another promising approach is to replace the PLDA back-end by DNNs. For instance, Ghahabi and Hernando [50] trained one DNN for each target speaker to discriminate his/her i-vectors from those of the other speakers. Each DNN receives i-vectors as input and produces the posterior probabilities of the target and non-target classes as output. Given a test i-vector, the log-posterior ratio can then be obtained from the network outputs. In [51], the whole i-vector extraction cum PLDA scoring pipeline is replaced by RNNs. Specifically, long short-term memory RNNs were collaboratively trained for speech and speaker recognition tasks, and the contextual information obtained from the speech recognition RNN was found to be assistive to the speaker recognition RNN.

Observing that adverse acoustic conditions and duration variability in utterances could have detrimental effect on PLDA scores, researchers explored the potential of other back-ends to replace the PLDA models, e.g., support vector machines (SVMs) [52] or even end-to-end learning [53].

Chapter 3

BASICS OF DEEP LEARNING

In recent years, deep learning has achieved a great success in many areas, including speech recognition [54], computer vision [55], speech synthesis [56,57] and music recognition [58]. Because of the great success of deep neural networks (DNNs) [59], convolutional neural networks (CNNs) [60] and recurrent neural networks (RNNs) [61–63] in automatic speech recognition (ASR), the application of deep learning [64] to speaker verification has been under the spotlight recently. In many of these studies, DNNs are used as classifiers. This is achieved by adding a softmax layer on top of the hidden layers of a deep feedforward network. Deep learning is powerful in that the resulting deep networks have strong ability to disentangle the variation in the input patterns, and therefore greatly improve the performance in many classification problems. The posterior probabilities generated by the softmax layer can replace the ones generated by other generative models, e.g. Gaussian mixture models (GMM) in speaker recognition [30], hidden Markov models (HMM) in large vocabulary continuous speech recognition (LVCSR) [54], and the posterior of Gaussian mixtures in i-vector based speaker verification [32].

In this section, we will introduce the basics of the feedforward fully-connected neural networks, including the model structure, backpropagation (BP) fine-tuning, and RBM pre-training.

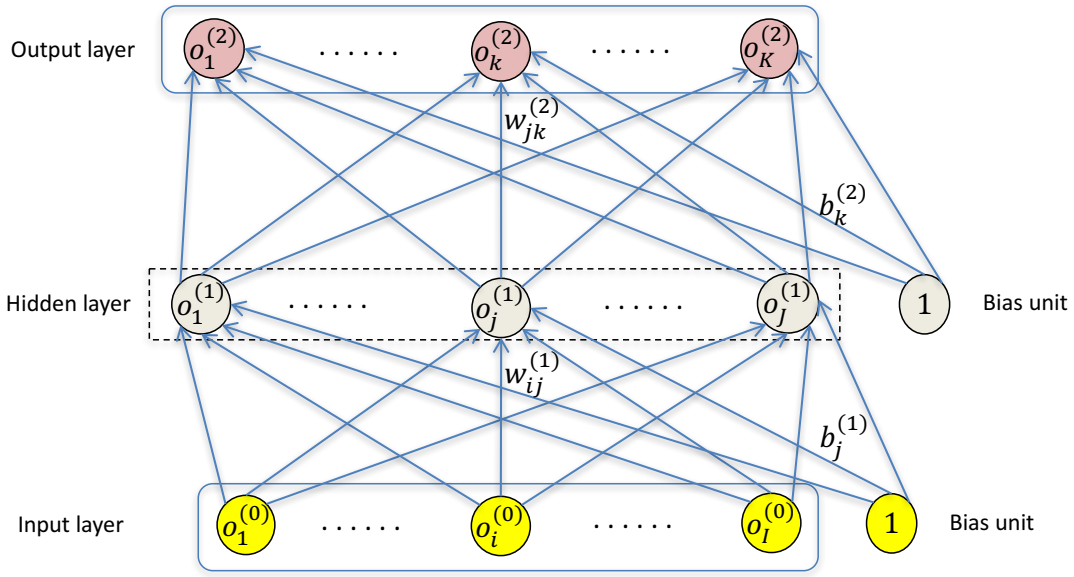

Figure 3.1: The structure of a neural network with one hidden layer.

3.1 Deep Neural Networks

3.1.1 Structure of DNNs

A feedforward neural network comprises a collection of connection nodes (called neurons) arranged in a layer-wise structure. The outputs of the nodes in the upper layers depend on the outputs of the nodes in its lower layers. The lowest layer is the input of the network and is called the input layer, and the top layer produces the network outputs and is called the output layer. Any layers in between the input and output layers are called the hidden layers. Fig. 3.1 shows a feedforward neural network consisting of an input layer with I neurons $\left\{o_1^{(0)}, \ldots, o_I^{(0)}\right\}$, a hidden layer with J neurons $\left\{o_1^{(1)}, \ldots, o_J^{(1)}\right\}$ and an output layer with K neurons $\left\{o_1^{(2)}, \ldots, o_K^{(2)}\right\}$. Generally, the weight $w_{ij}^{(n)}$ denotes the connection strength between the i-th node in layer $n-1$, and the j-th node in layer n. The output of a node at layer n is non-linearly related to all of the output at layer $n-1$. Specifically, for the j-th node in the hidden layer in

Fig. 3.1, we have

$$o_j^{(1)} = f(y_j^{(1)}),$$

where $f(y)$ is the activation function, which usually is the sigmoid function:

$$f(y) = \frac{1}{1+e^{-y}}. \tag{3.1}$$

In Eq. 3.1, $y_j^{(1)}$ is the linear weighted sum of $\left\{o_1^{(0)}, \ldots, o_I^{(0)}\right\}$:

$$y_j^{(1)} = \sum_{i=1}^{I} w_{ij}^{(1)} o_i^{(0)} + b_j^{(1)}, \tag{3.2}$$

where $b_j^{(1)}$ is the bias term for node j.

The weighted sums in the output layer, $\left\{y_1^{(2)}, \ldots, y_K^{(2)}\right\}$, have a definition similar to Eq. 3.2. However, for classification tasks, the last layer uses the softmax function as the activation function. More specifically, the output of node k at the output layer is

$$o_k^{(2)} = f(y_k^{(2)}) = \frac{\exp(y_k^{(2)})}{\sum_{k'=1}^{K} \exp(y_{k'}^{(2)})}, \qquad k = 1, \ldots, K.$$

For regression tasks, the outputs are linear, i.e.,

$$f(y_k^{(2)}) = y_k^{(2)}, \qquad k = 1, \ldots, K.$$

Once the neural network has been well-trained, we can extract the outputs from the last layer by presenting vectors to its input.

3.1.2 Training by Backpropagation

The training of a neural network is to estimate its weights $\boldsymbol{W}$,[1] and it can be achieved by using the backpropagation (BP) algorithm [65]. To define the distance between the actual output of the model and the target output of the training data, we need to choose a loss function which is to be minimized during the training process [66]. For regression tasks, the loss function is usually the mean squared error (MSE):

$$E = \frac{1}{2}\sum_{k}(o_k^{(2)} - t_k)^2,$$

where t_k is the target output of the k-th node in the output layer.

As BP is a gradient descent algorithm, we update the weights by calculating the gradient of the loss function. For example, to update the weights in the last layer, the partial derivative of the loss function with respect to the weights are computed:

$$\begin{aligned}\frac{\partial E}{\partial w_{jk}^{(2)}} &= \frac{\partial E}{\partial o_k^{(2)}}\frac{\partial o_k^{(2)}}{\partial w_{jk}^{(2)}} \\ &= \frac{\partial E}{\partial o_k^{(2)}}\frac{\partial o_k^{(2)}}{\partial y_k^{(2)}}\frac{\partial y_k^{(2)}}{\partial w_{jk}^{(2)}}.\end{aligned}$$

When MSE is defined as the loss function, and the output nodes are linear, i.e. $o_k^{(2)} = f(y_k^{(2)}) = y_k^{(2)}$, we have

$$\frac{\partial E}{\partial o_k^{(2)}} = o_k^{(2)} - t_k \qquad \text{and} \qquad \frac{\partial o_k^{(2)}}{\partial y_k^{(2)}} = 1.$$

Because $y_k^{(2)} = \sum_{j=1}^{J} w_{jk} o_j^{(1)} + b_k^{(2)}$, we have

$$\frac{\partial y_k^{(2)}}{\partial w_{jk}^{(2)}} = o_j^{(1)}.$$

[1] Using Fig. 3.1 as an example, $\boldsymbol{W} = \left\{w_{ij}^{(1)}, w_{jk}^{(2)}; i = 1, \ldots, I, j = 1, \ldots, J \text{ and } k = 1, \ldots, K\right\}$.

Therefore, the weight update formula is:

$$\begin{aligned} w_{jk}^{(2)} &\leftarrow w_{jk}^{(2)} - \eta \frac{\partial E}{\partial w_{jk}^{(2)}} \\ \Longrightarrow w_{jk}^{(2)} &\leftarrow w_{jk}^{(2)} - \eta (o_k^{(2)} - t_k) o_j^{(1)}, \end{aligned}$$

where η is the learning rate, which is a hyperparameter.

To update the weights in lower layers, the partial derivatives can be calculated by the chain rule:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}^{(1)}} &= \frac{\partial E}{\partial y_j^{(1)}} \frac{\partial y_j^{(1)}}{\partial w_{ij}^{(1)}} \\ &= \sum_k \frac{\partial E}{\partial o_k^{(2)}} \frac{\partial o_k^{(2)}}{\partial y_k^{(2)}} \frac{\partial y_k^{(2)}}{\partial o_j^{(1)}} \frac{\partial o_j^{(1)}}{\partial y_j^{(1)}} \frac{\partial y_j^{(1)}}{\partial w_{ij}^{(1)}}. \end{aligned}$$

The first layer has the sigmoid function as activation, whose partial derivative is:

$$\frac{\partial f(y)}{\partial y} = f(y) \left(1 - f(y)\right).$$

For classification tasks, the loss function is usually the cross-entropy:

$$E = -\sum_k t_k \log o_k^{(2)}.$$

Its derivative is:

$$\frac{\partial E}{\partial o_k^{(2)}} = -\frac{t_k}{o_k^{(2)}}.$$

The derivative of the softmax function is similar to the one for the sigmoid function:

$$\begin{aligned} \frac{\partial o_k^{(2)}}{\partial y_k^{(2)}} &= o_k^{(2)}(1 - o_k^{(2)}), \\ \frac{\partial o_k^{(2)}}{\partial y_r^{(2)}} &= -o_k^{(2)} o_r^{(2)} \text{ for } r \neq k. \end{aligned}$$

Because of the one-hot encoding in classification tasks, we have $\sum_{r=1}^{K} t_r = 1$. Therefore, we have:

$$\begin{aligned}
\frac{\partial E}{\partial y_k^{(2)}} &= \frac{\partial E}{\partial o_k^{(2)}} \frac{\partial o_k^{(2)}}{\partial y_k^{(2)}} + \sum_{r \neq k} \frac{\partial E}{\partial o_r^{(2)}} \frac{\partial o_r^{(2)}}{\partial y_k^{(2)}} \\
&= -\frac{t_k}{o_k^{(2)}} o_k^{(2)}(1 - o_k^{(2)}) + \sum_{r \neq k} \frac{t_r}{o_r^{(2)}} o_r^{(2)} o_k^{(2)} \\
&= -t_k(1 - o_k^{(2)}) + \sum_{r \neq k} t_r o_k^{(2)} \\
&= t_k - o_k^{(2)}.
\end{aligned}$$

And thus the gradient is:

$$\begin{aligned}
\frac{\partial E}{\partial w_{jk}^{(2)}} &= \frac{\partial E}{\partial y_k^{(2)}} \frac{\partial y_k^{(2)}}{\partial w_{jk}^{(2)}} \\
&= (t_k - o_k^{(2)}) o_j^{(1)}.
\end{aligned}$$

In backpropagation training, we presented a mini-batch of input patterns and update the network parameters to bring the actual outputs closer to the target values. It uses the chain rule to iteratively compute the error gradient of each layer, collects the gradients from top to bottom layers, and updates the weights layer by layer.

3.2 Restricted Boltzmann Machines

Backpropagation (BP) is commonly used for training DNNs. However, BP is a gradient descent algorithm, which could be easily trapped in local minima, especially when the neural network has a deep structure (having many hidden layers). This is because the gradients in the bottom layers are too small. In this case, we can consider the DNN as comprising a number of stacked restricted Boltzmann machines (RBMs), which is trained layer-by-layer via the contrastive divergence algorithm [31]. It is commonly believed that this pre-training step can bring the DNN close to the

global optimal solution, which helps the backpropagation algorithm to find to a better solution.

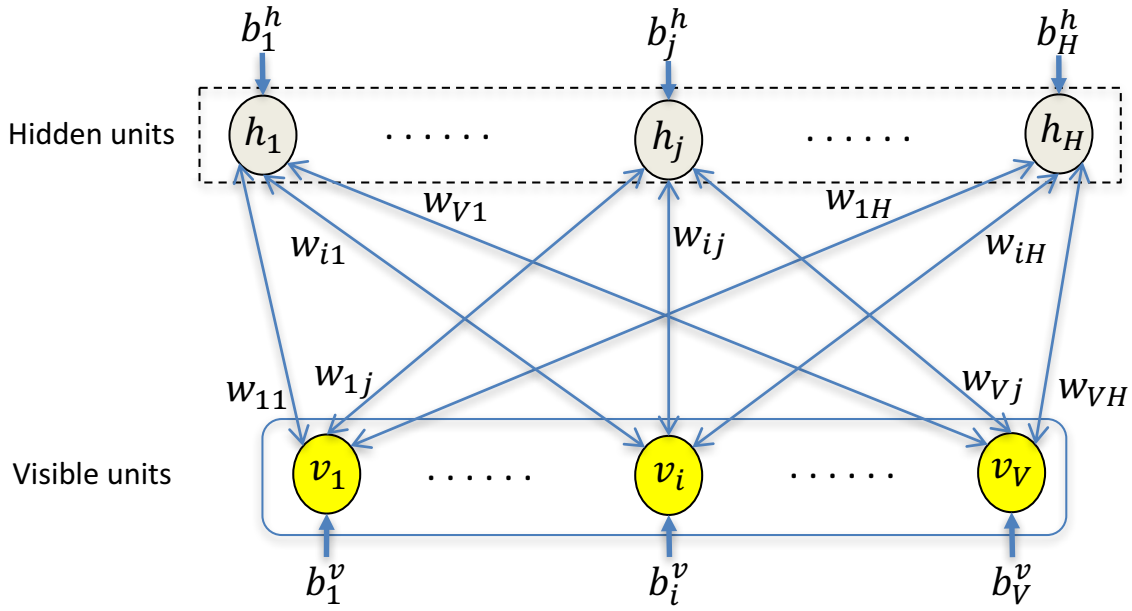

Figure 3.2: The structure of restricted Boltzmann machine.

An RBM is an energy-based model in which nodes within the same layer do not have interaction. Fig. 3.2 shows the architecture of an RBM, which has V visible nodes $\boldsymbol{v} = [v_1, \ldots, v_V]^\intercal$ and H hidden nodes $\boldsymbol{h} = [h_1, \ldots, h_H]^\intercal$. The weight w_{ij} connects v_i and h_j, and b_i^v and b_j^h are the bias of v_i and h_j, respectively. For Bernoulli-Bernoulli RBMs, the visible and hidden nodes can only be in one of two possible stages: 0 or 1. The conditional propability of the states in the hidden nodes is given by

$$p(h_j = 1|\boldsymbol{v}) = s(z_{h_j}),$$

where z_{h_j} is the weighted sum in h_j:

$$z_{h_j} = \sum_{i=1}^{V} w_{ij} v_i + b_j^h,$$

and $s(z_{h_j})$ is the activation function:

$$s(z_{h_j}) = \frac{1}{1 + e^{-z_{h_j}}}.$$

Similarly, the conditional probabilities of visible states are:

$$\begin{aligned} p(v_i = 1|\boldsymbol{h}) &= s(z_{v_i}) \\ &= \frac{1}{1 + e^{-z_{v_i}}} \\ &= \frac{1}{1 + e^{-\sum_{j=1}^{H} w_{ij} h_j + b_i^v}}, \qquad i = 1, \ldots, V. \end{aligned}$$

The energy function of an RBM is defined as:

$$E\left(\boldsymbol{v}, \boldsymbol{h}\right) = -\sum_{i=1}^{V} \sum_{j=1}^{H} v_i h_j w_{ij} - \sum_{i=1}^{V} v_i b_i^v - \sum_{j=1}^{H} j_j b_j^h.$$

By using contrastive divergence [31], we maximize the probability that the network produces the visible vectors, $\boldsymbol{v}$'s:

$$\prod_{\boldsymbol{v} \in \mathcal{V}} p(\boldsymbol{v}) = \prod_{\boldsymbol{v} \in \mathcal{V}} \frac{1}{Z} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})},$$

where $\mathcal{V}$ is the training set.

For real-value inputs, such as MFCCs after feature warping [67], the visible nodes follow Gaussian distributions. For Gaussian-Bernoulli RBMs, the activation function

in the visible layer is linear:

$$s(z) = z = \sum_{i=1}^{V} w_{ij} v_i + b_j^h$$

and the energy function is defined as:

$$E(\boldsymbol{v}, \boldsymbol{h}) = \sum_{i=1}^{V} \frac{(v_i - b_i^v)^2}{2\sigma_i^2} - \sum_{j=1}^{H} b_j^h h_j - \sum_{i=1}^{V} \sum_{j=1}^{H} \frac{v_i}{\sigma_i} h_j w_{ij}, \tag{3.3}$$

where σ_i is the standard deviation of the Gaussian noise for visible unit i.

When RBMs and contrastive divergence are used to pre-train a DNN, the input data are used to train the first RBM, and the corresponding weights become the initial value in the first layer of the pre-trained DNN. Then, the hidden-layer activations of the first RBM, as the result of the visible input $\mathcal{V}$, are considered as the visible input of the second RBM. The process is repeated until a desirable number of layers is obtained. By stacking these RBMs, we initialize a DNN for BP fine-tuning.

Chapter 4

SPEAKER BOTTLENECK FEATURES

4.1 Introduction

In order to obtain noise robust acoustic features as discussed in Section 2.6.1, this chapter explores the use of DNNs for extracting speaker-dependent features for speaker recognition. To this end, we stack a denoising deep autoencoder [17,68], two layers of RBMs and a softmax layer to form a DNN classifier that produces posterior probabilities of speaker identities as output. However, instead of using the classifier directly for speaker identification, we used the RBM just below the softmax output layer of the DNN as the bottleneck layer for feature extraction. More precisely, bottleneck features are extracted from the RBM's outputs before sigmoid nonlinearity. The bottleneck features, which provide a low-dimensional representation of the input patterns [69], are used for training an i-vector/PLDA speaker identification system. The advantage of using the DNN as feature extractor rather than using it directly as speaker identifier is that the number of test speakers will not be limited by the number of nodes in the softmax layer.

Unlike Vincent's denoising autoencoder [17], we used noisy speech as the input and clean speech as the target output to train the denoising autoencoder. It is formed by stacking multiple layers of RBMs pre-trained by the contrastive divergence algorithm [31], followed by backpropagation. Then, two layers of RBMs were trained using the outputs of the denoising autoencoder as input. Finally, a softmax layer with number of nodes equal to the number of training speakers was put on top of the last (bottleneck) layer of the RBMs and backpropagation fine-tuning was further applied to minimize

the cross-entropy error. Therefore, the first several layers in the DNN classifier help to make the whole neural network more noise robust, while the top layers extract the speaker-dependent information from the denoisd spectra. We demonstrated that at an SNR of 0dB, the bottleneck features are slightly more robust than the standard MFCCs [26].

4.2 SNR-Adaptive Denoising Deep Autoencoder

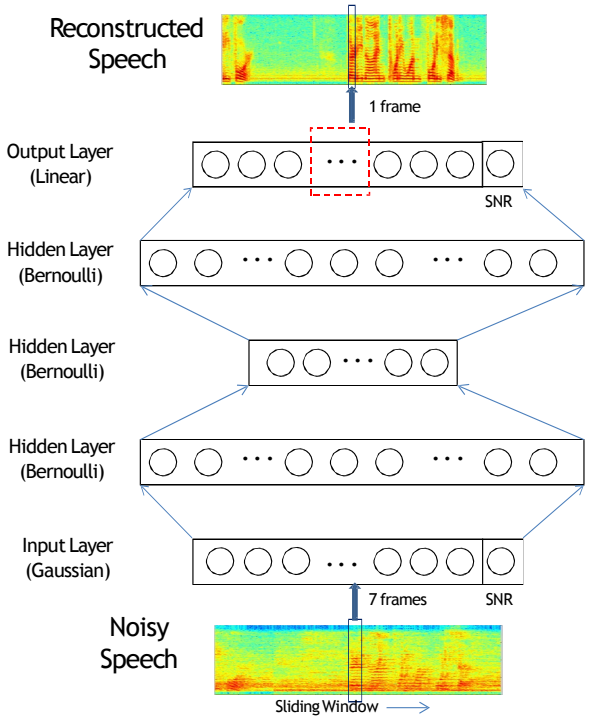

Figure 4.1: Structure of the SNR-adaptive denoising deep autoencoder.

4.2.1 Input Features

To train the denoising autoencoder, it is necessary to preprocess the input speech. On one hand, cepstral features have shown promise in previous research; on the other hand, it is intuitive to apply raw features as input to realize the potential of autoencoders in modelling speech signals. In particular, the log-spectra, the log mel-scale triangular filterbank outputs and even MFCC are candidate inputs to the autoencoder.

For the log-spectra, we performed 512-point fast Fourier transform on 8 kHz speech data, followed by taking logarithm. Due to the symmetry property of Fourier transform for real numbers, only the first 256 spectral components were used in subsequent steps.

For the log mel-scale triangular filterbank outputs, 20 triangular filterbanks from 300Hz to 3700Hz were used, and therefore the 256-dimensional spectra were reduced to 20 dimensions. After applying discrete cosine transform (DCT), we obtained the MFCCs.

For each of the input types, we packed it with another input node that represents the SNR to form the input patterns of the SNR-adaptive denosing autoencoder. The SNR node in the input may help the denoising autoencoder to realize the noise level of the input speech. An SNR node is also added to the output of the autoencoder to make input and output dimensions the same. These two nodes are also used in the testing phase. The structure of the hidden layers is identical for all input types. It has been shown [70] that it is beneficial to apply Z-norm to the input vectors. In our experiments, the SNR and the input features are normalized independently, i.e. the mean and standard deviation of the SNR and the bottleneck features were estimated separately. For example, for a random variable x, Z-norm normalizes it to x' with zero mean and unit variance:

$$x' = \frac{x - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation of x, respectively.

In addition to the preprocess techniques above, we can also use a contextual window covering several frames as the input to the DNN. For example, a sliding window covering 7 frames of mel filterbank outputs consists of $20 \times 7 = 140$ nodes. Together with the SNR node, there are 141 nodes in the input layer. Fig. 4.1 shows the architecture of the denoising autoencoder.

4.2.2 RBM Pre-training and Backpropagation Fine-tuning

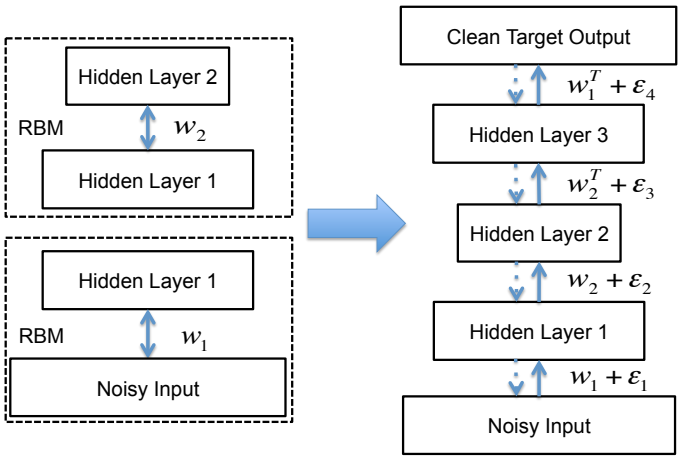

Figure 4.2: Construction of a denoising autoencoder by training two RBMs layer-by-layer and then stacking them symmetrically, followed by backpropagation fine-tuning.

The denoising autoencoder can be initialized layer-wise by using RBMs trained by the contrastive divergence algorithm [31]. After RBM pre-training, we can stack the RBMs, copy the parameters from the lower half of the deep belief network (DBN) to the upper half, and then fine-tune them by using backpropagation, as Fig. 4.2 illustrates.

In backpropagation training, we presented a mini-batch of input patterns and updated the network parameters to bring the actual outputs closer to the target

values. To equip our autoencoder with denoising ability, we used noisy speech as input and their corresponding clean counterparts as target outputs, while the error function is the squared loss, $L(z, \tilde{z}) = \|z - \tilde{z}\|_2^2$. However, we kept the SNR component in the output the same as the input in the experiments since we only focused on the speech denoising capability of this autoencoder.

Backpropagation [65] uses the chain rule to iteratively compute the error gradient of each layer, collects the gradients from top to bottom layers, and updates the weights layer by layer. In our experiments, the first three hidden layers are all Bernoulli layers, and therefore they have a sigmoid activation function. However, the output layer of the autoencoder, which aims to reconstruct the input, uses a linear activation function. The autoencoder is a key component of the DNN classifier, as Fig. 4.3 shows.

4.3 SNR-Adaptive Denoising Deep Classifier

Because of the denoising autoencoder, the DNN learns how to extract clean information from the noisy input patterns. However, our goal is to enable the DNN to extract speaker-dependent features. To this end, we construct a speaker classifier by putting two more layers of RBMs on top of the autoencoder as shown in Fig. 4.3. Finally, a softmax layer with the number of nodes equals to the number of training speakers is added to the network. The first RBM (comprising *Hidden Layer 4* and *Hidden Layer 5* in Fig. 4.3) is Gaussian-Bernoulli due to the characteristic of the autoencoder's reconstruction layer. The top-most RBM (comprising *Hidden Layer 5* and *BN Layer* in Fig. 4.3) is Bernoulli-Bernoulli. Because what we require is a classifier, the last layer (*Speaker ID* in Fig. 4.3) is a softmax layer, and therefore no RBM pre-training for this layer was applied. Its size is equal to the number of classes, which in our case is the number of training speakers.

Backpropagation is applied to fine-tune the DNN by minimising the cross entropy error. Specifically, we assume that we have N training speakers whose spectral feature

vectors and speaker labels are given by

$$\begin{aligned}\mathcal{O} &= \{\boldsymbol{o}_i^{(j)} \in \Re^D; i = 1, \ldots, M_j; j = 1, \ldots, N\} \\ \mathcal{T} &= \{\boldsymbol{t}_i^{(j)} \in \Re^N; i = 1, \ldots, M_j; j = 1, \ldots, N\},\end{aligned} \tag{4.1}$$

where $\boldsymbol{t}_i^{(j)}$'s are one-of-N vectors indicating to which speaker the spectral vector $\boldsymbol{o}_i^{(j)}$ belongs and M_j is the number of vectors from speaker j.

Specifically, for each input vector, the desired output of the nodes in the softmax layer are all zeros, except for the one that indicates the speaker to which the input vector belongs. Then, we minimize the cross-entropy error:

$$\begin{aligned}E(\mathcal{O}, \mathcal{T}) &= -\sum_{i=1}^{N}\sum_{j=1}^{M_i}\sum_{k=1}^{N} t_{i,k}^{(j)} \log(f(\boldsymbol{o}_i^{(j)})_k) \\ &= -\sum_{i=1}^{N}\sum_{j=1}^{M_i}\sum_{k=1}^{N} t_{i,k}^{(j)} \log(y_{i,k}^{(j)})\end{aligned} \tag{4.2}$$

where k indexes to the output nodes of the DNN, $f(\cdot)$ represents the mapping function of the DNN, and

$$y_{i,k}^{(j)} = f(\boldsymbol{o}_i^{(j)})_k = \frac{e^{a_k}}{\sum_{k'=1}^{N} e^{a_{k'}}} \tag{4.3}$$

represents the output of the k-th output node subject to the input vector $\boldsymbol{o}_i^{(j)}$. In Eq. 4.3, a_k is the linear activation of the k-th output neuron.

Note that there are two layers in the DNN that do not use the sigmoid function as the activation function. The first one is the reconstruction layer in the autoencoder, which uses the linear function instead; the second one is for classification, which uses the softmax function.

Fig. 4.4 shows the spectrogram and histograms of log-spectra of the clean, 0dB noisy, and denoised speech, respectively. The figure shows that the noise seriously distorted the spectral patterns of the clean speech. However, the denoising deep

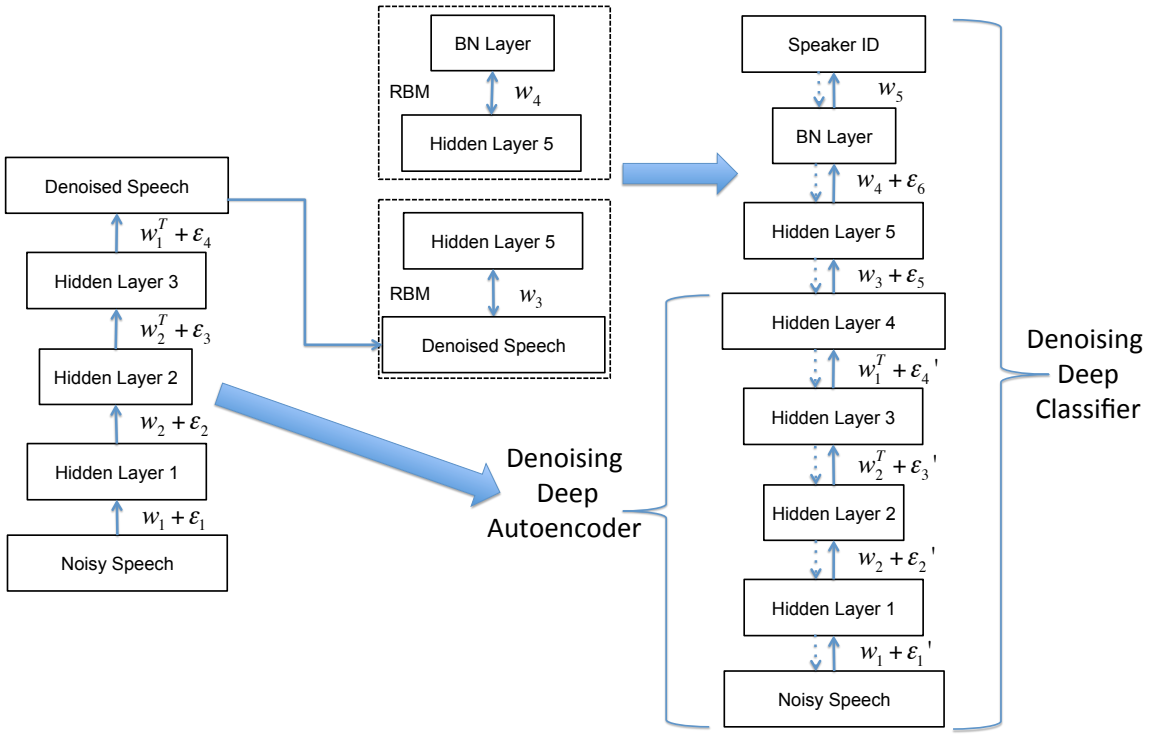

Figure 4.3: Constructing the DNN classifier and bottleneck (BN) feature extractor by stacking two RBM layers (*Hidden Layer 5* and *BN Layer*) and a softmax layer (*Speaker ID*) on top of the autoencoder (from *Noisy Speech* to *Hidden Layer 4*), followed by backprogagation fine-tuning. Note that after fine-tuning, only the features extracted from the BN layer will be used for iVector-PLDA speaker identification.

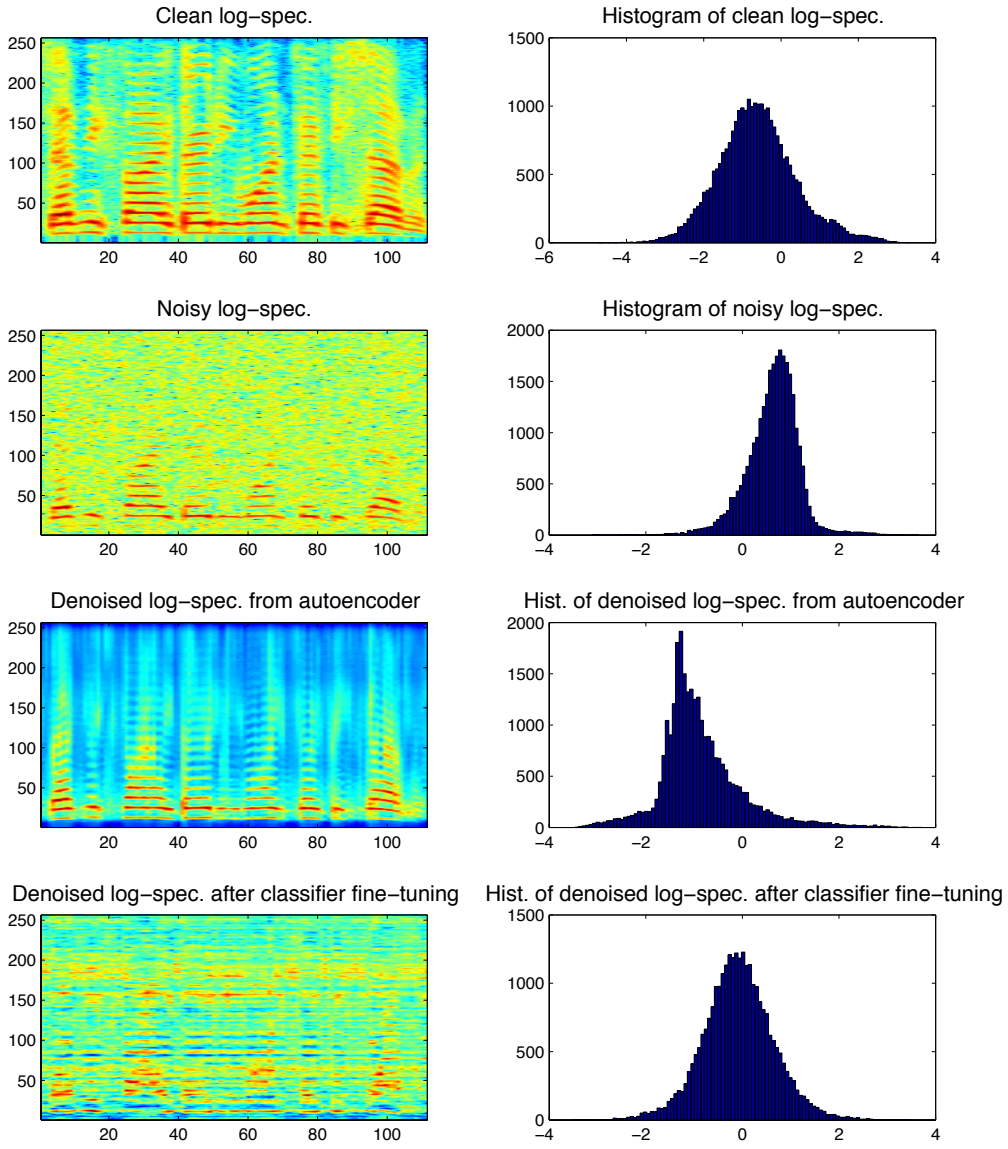

Figure 4.4: The spectrograms and histograms of log-spectra of clean, 0dB noisy, and denoised speech. The spectrograms are to show the denoising ability of DAE for reference only.

autoencoder performs very well in restoring the patterns, as demonstrated in the third panel of the figure. The histogram on the right of Fig. 4.4 shows that after denoising, the distribution of log-spectra is non-Gaussian (3rd row). However, after applying BP to fine tune the whole DNN (4th row), the output of the autoencoder follows a Gaussian-like distribution.

We trained the denoising deep classifier in an end-to-end manner. We could have trained it independently such that the upper half of the classifier is trained using the original clean speech followed by putting it on top of the DAE. However, such a setting is likely to cause the performance degradation, because it requires the DAE to be perfect. Unfortunately, in practice, the outputs of the DAE are different from (although they may be quite close to) the clean MFCCs. Therefore an end-to-end manner is more reasonable.

4.4 i-vector/PLDA Speaker Identification

The i-vector [9] and probabilistic LDA (PLDA) [13] are commonly used for speaker verification. The idea is to represent the speaker and channel characteristics of an utterance by a low-dimensional vector called the i-vector, which is essentially the posterior mean of the latent factors of a factor analysis model. Given an i-vector, the channel variability is removed by marginalizing out the channel factors in a PLDA model (which is a supervised factor analysis model) during verification. This i-vector/PLDA framework was originally designed for speaker verification, which is a binary classification problem.

In this work, we applied this framework to speaker identification, which is a multi-class problem. First, we used the utterances of all training speakers to train an i-vector extractor. Then, the i-vectors of each registered speaker in the speaker identification system were computed, one for each enrollment utterance. Also, the i-vectors of all training speakers (who can be different from the registered speakers) were extracted.

These training i-vectors together with their speaker labels were used for training a PLDA model [12, 38].

During identification, given a test utterance, an i-vector is computed. Then, the test i-vector is scored against the i-vectors of each of the registered speakers using the PLDA scoring function [14, 38] (Eq. 2.8) and the scores were averaged. For a system comprising R registered speakers, these steps give R averaged scores for each test utterance, and the speaker identity corresponds to the maximum averaged score. More precisely, denote $\boldsymbol{x}_i^j$ as the i-vector of the i-th session of the j-th registered speaker in the system. Then, given a test utterance with i-vector $\boldsymbol{x}_t$, the speaker ID of the test utterance is

$$\mathrm{ID}(\boldsymbol{x}_t) = \arg\max_{i=1}^{R} \frac{1}{N_j} \sum_{i=1}^{N_i} S_{\mathrm{PLDA}}(\boldsymbol{x}_i^{(j)}, \boldsymbol{x}_t), \tag{4.4}$$

where N_j is the number of enrollment i-vectors of speaker j, and $S_{\mathrm{PLDA}}(\boldsymbol{x}_i^{(j)}, \boldsymbol{x}_t)$ is computed by using the score function in Eq. 2.8.

4.5 Experiments and Results

4.5.1 Experimental Setup

We performed speaker identification experiments based on 138 speakers in the YOHO corpus [71].[1] We used the enrollment sessions, which consist of 96 utterances per speaker, as training data. We used the verification sessions of the corpus as testing data, which have 40 utterances per speaker.[2] Totally, there are 13,248 utterances for training and 5,516 utterances for testing. Each utterance is about 3 to 4 seconds long, sampled at 8kHz, and comprises three two-digit numbers in English, e.g. 26-81-57.

[1] To minimize computation time in this pilot study, we did not use the NIST evaluation corpora. These corpora will be used in Chapters 5 and 6.

[2] Except for Speaker 277 whose only have 36 valid utterances in the verify sessions.

We used the FaNT tool to add babble noise to the original YOHO utterances at 15dB, 6dB, and 0dB. Thus, there are totally $13,248 \times 4 = 52,992$ utterances for training the DNN classifier. We used an energy-based voice activity detector [72, 73] to extract the speech regions of each utterance.

To train the autoencoder part of the classifier, each clean utterance is paired with itself and its noisy counterparts, which amounts to 13,248 clean–clean training pairs and $13,248 \times 3 = 39,744$ clean–noisy training pairs. The autoencoder comprises three hidden layers, whose weights were trained by using both noisy and clean speech data as input and only clean speech data as the target output. Each of the hidden layers contains 256 nodes. Our aim is to enable it to perform speech denoising in the spectral domain. To achieve this, we split the training data into mini-batches comprising 100 consecutive spectra and applied 30 epochs of RBM pre-training, and then used mini-batches comprising 1,000 consecutive spectra and applied 100 epochs of backpropagation fine-tuning using gradient descent with a learning rate of 0.1 and momentum of 0.6.

By using the denoised spectra from the autoencoder, we trained another two RBMs with 256 and 60 hidden nodes, respectively. Because the number of training speakers is 138, the DNN has 138 output nodes, i.e., $N = 138$ in Eq. 4.1. However, because the cross-entropy error has a larger fluctuation when fine-tuning the whole DNN, we reduced the learning rate to 0.0003.

Finally, we obtained a denoising deep classifier with structure D-256-256-256-D-256-60-138 nodes in the respective layers starting from input to output, where D represents the dimension of the spectral vectors time the number of frames in the contextual window. For example, in one of the setting in the experiments, $D = 20 \times 7 = 140$, where 20 filterbank outputs were used as the spectral vectors, and the contextual window covers 7 frames.

In the experiments, the 60-dimensional bottleneck features extracted from the second-top layer in the classifier before sigmoid non-linearity were compared with the

60-dimensional MFCC baseline features. For the former, the frame-based BN features were whitened using PCA whitening. For the latter, we computed 19 MFCCs and the log-energy for each frame. Then we packed the MFCCs and log-energy together with their first- and second-order derivatives to form a 60-dimensional acoustic vector for each frame.

For the back-end, we used the state-of-the-art i-vector [9] and probabilistic LDA (PLDA) [12]. To train an i-vector extractor, we used all of the 52,992 training utterances from the clean and the three SNR conditions to train a universal background model (UBM) with 256 Gaussians and a total variability matrix with 400 factors. For each utterance, an i-vector was extracted from the i-vector extractor [9] so that the speaker characteristics of the entire utterance is represented by this 400-dimensional vector. Given 52,992 training utterances, we have 52,992 i-vectors. Then they were used for training an SNR-independent PLDA model with 138 latent factors by grouping the i-vectors of the same speaker together. Because there are 138 training speakers, we have 138 groups of speaker-dependent i-vectors and each group comprises $96 \times 4 = 384$ i-vectors.

4.5.2 Results of Denoising BN Features

We used three types of input for the DNN: one frame of 256-dimensional log-spectra (*Log-spec*), a contextual window covering seven frames of 20-dimensional log mel-scale triangular filterbank output (*Log-mel*), and five frames of 60-dimensional MFCC (*MFC*) to generate the BN features. These three types of inputs result in Log-spec BN, Log-mel BN and MFC BN features, respectively.

Table 4.1 shows the accuracy of speaker identification, which is a bit disappointing in that only the Log-mel BN features with the Log-mel input are comparable with the standard MFCC under high SNR conditions and outperform it under low SNR conditions. Besides, it is surprising that the MFC BN features using 5 frames of MFCC as input always perform worse than the Log-mel ones and the standard MFCC. This may

Table 4.1: Comparison between MFCC and BN features.

Feature	SNR of Test Utterances			
	Clean	15dB	6dB	0dB
MFCC	**98.31%**	95.61%	90.08%	65.65%
Log-spec BN	95.56%	93.04%	83.39%	62.98%
Log-mel BN	98.21%	**96.77%**	**91.48%**	**75.29%**
MFC BN	97.44%	94.24%	86.84%	63.61%

be due to the smearing effect of the denoising autoencoder. Further investigations are warranted to investigate the reasons behind the poor performance.

The Log-spec BN feature performs slightly poorer than the MFC BN feature under both clean and noisy conditions. We suspect that the poor performance is due to the lack of contextual frames (in *Log-spec*, the size of contextual window is 1) in the input. However, we found that increasing the contextual window size to 5 leads to even poorer performance. This could be caused by the high-dimensionality of the log-spectral vectors, which forbids us to use multiple frames in the contextual window.

4.5.3 PLDA Score Combination

Because the BN features, especially the Log-mel one, perform well under low SNR conditions, we can fuse the MFCC and the BN features to improve the performance of speaker identification at the PLDA score level:

$$S_{\text{fused}} = \alpha \times S_{\text{mfcc}} + (1 - \alpha) S_{\text{bn}}, \tag{4.5}$$

where α is the fusion weight and S denotes the PLDA scores. When $\alpha = 1$, no fusion is performed and only MFCC features were used. On the other hand, when $\alpha = 0$ only

Table 4.2: Fusion of the PLDA scores based on MFCC and BN features.

Feature	Fusion Weight α (Eq. 9)	SNR of Test Utterances			
		Clean	15dB	6dB	0dB
MFCC	1.00	98.31%	95.61%	90.08%	65.65%
Log-spec BN	0.00	95.56%	93.04%	83.39%	62.98%
	0.57	99.15%	98.11%	94.13%	79.89%
Log-mel BN	0.00	98.21%	96.77%	91.48%	75.29%
	0.51	**99.56%**	**98.55%**	**95.87%**	**84.34%**
MFC BN	0.00	97.44%	94.24%	86.84%	63.61%
	0.56	98.89%	96.72%	93.31%	76.53%

BN features were considered. We varied the value of α with a step size of 0.01. As Table 4.2 illustrates, the Log-mel BN features with score fusion increase the accuracy significantly.

Chapter 5

SENONE I-VECTORS

5.1 Introduction

To address the robustness issue of i-vectors discussed in Section 2.6.2, this chapter explores and extends our early work [74] on using DNNs for extracting phonetically discriminative and noise robust bottleneck features from noisy speech and for computing senone posteriors for BN-based i-vector extraction. We have recently proposed a denoising autoencoder–deep neural network by stacking restricted Boltzmann machines (RBMs) on top of a denoising autoencoder [75], which is also introduced in Chapter 4. The whole network was trained to produce the posteriors of speaker IDs given noisy speech as input. Bottleneck features were then extracted from the RBM layer just below the output (softmax) layer. Results in [75] suggest that the DAE is very effective in suppressing the effect of noise in the input speech, making the BN features noise robust.

Similar to the DNNs in d-vectors [76] and speaker embedding [77], where a low dimensional representation of the whole utterance is extracted from a DNN directly, the DNN in [75] produces speaker posteriors. Because the DNNs of these methods are trained to produce speaker posteriors, their frame-based activations at the bottleneck layer tend to be very similar across the whole utterance. Fig. 5.1 shows the t-SNE plots of the "hypothetical" BN features derived from speaker posteriors and the BN features derived from senone posteriors. The senone-posterior derived BN features were obtained from an utterance in NIST 2010 SRE, whereas the speaker-posterior derived BN features are generated hypothetically. For the latter, because

the DNN was trained to produce speaker posteriors, the frame-based activations at the bottleneck layer tend to be very similar across the whole utterance. Therefore, the BN features from each speaker tend to form a tiny cluster as shown in Fig. 5.1. The highly packed clusters cause numerical problems when training the UBM and the T-matrix. The d-vectors and speaker embedding avoid this problem by averaging the activations across the frames of the entire utterance, which essentially bypasses the UBM training and TV matrix estimation. However, the averaging process throws away lots of speaker information in the frame-based BN vectors, which explains why the performance of d-vectors and speaker embedding is poorer than i-vectors for long utterances [76, 77].

In this chapter, we propose training the denoising DNN in [75] to produce senone posteriors instead of speaker posteriors. The advantage of this strategy is that as long as a training utterance is phonetically balanced, its BN vectors will be scattered over different regions of the BN feature space as shown in Fig 5.1, which solves the numerical problem. With the denoising capability of DAE, the network can produce noise robust BN features and robust senone posteriors for i-vector extraction. We refer to the resulting network as DAE–DNN.[1]

Experimental results on NIST 2012 SRE demonstrate that the proposed BN-based i-vectors are less susceptible to babble noise, even at 0dB. We found that no matter under the GMM i-vector framework or the senone i-vector framework, the phonetically discriminative BN features outperform the MFCCs in speaker verification tasks. This suggests that the phonetically discriminative BN features still retain speaker-specific information. Furthermore, we demonstrate that the denoising capability works for our denoised BN-based senone i-vectors rather than the denoised MFCC-based senone i-

[1]DAE–DNN represents the structure and training procedures of a DNN, where the DNN is constructed by stacking RBMs on the top of a DAE, followed by backpropagation fine-tuning. Therefore both the DNNs in Chapter 4 and Chapter 5 are named as DAE–DNN, although the DAE–DNN in Chapter 4 is trained for classifying speakers, where the DAE–DNN in Chapter 5 is trained for classifying senones.

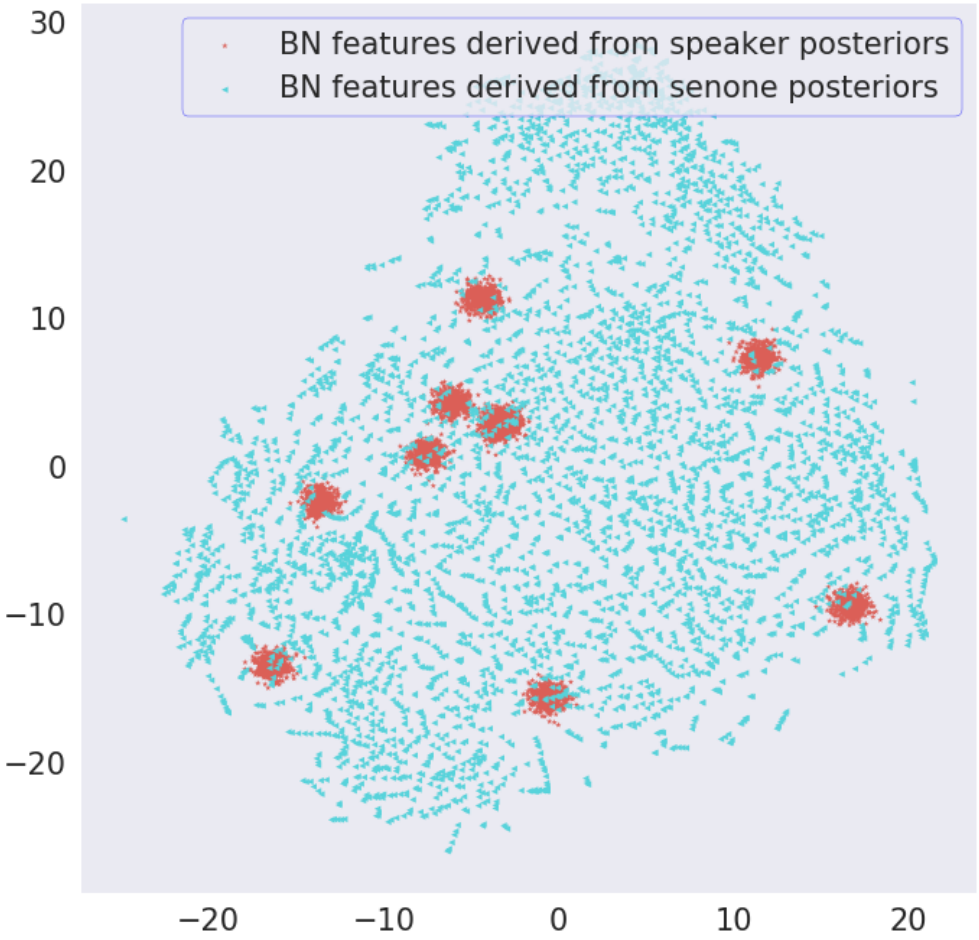

Figure 5.1: The t-SNE plots of "hypothetical" BN features derived from speaker posteriors (red dots) and BN features derived from senone posteriors (blue dots). For the former, each cluster represents the BN features from one speaker.

vectors. Specifically, by comparing the combinations of phonetically discriminative BN features and senone posteriors with and without DAE training, we validate that the DAE training is more useful for extracting phonetically discriminative BN features than estimating senone posteriors, especially under common condition 5 of NIST 2012 SRE.

5.2 Generalized I-vector Extractor

In most systems, $\{\boldsymbol{\mu}_c^{(b)}\}$ and $\{\boldsymbol{\Sigma}_c^{(b)}\}$ in Eqs. 2.4–2.7 are obtained from the UBM. However, they can also be computed from the sufficient statistics as follows:

$$\boldsymbol{\mu}_c = \frac{\sum_i \sum_t \gamma_c(\boldsymbol{o}_{i,t})\boldsymbol{o}_{i,t}}{\sum_i N_{ic}}$$

and

$$\boldsymbol{\Sigma}_c = \frac{\sum_i \sum_t \gamma_c(\boldsymbol{o}_{i,t})(\boldsymbol{o}_{i,t}-\boldsymbol{\mu}_c)(\boldsymbol{o}_{i,t}-\boldsymbol{\mu}_c)^{\intercal}}{\sum_i N_{ic}}, \quad c = 1, \ldots, C,$$

where $\boldsymbol{o}_{i,t}$ denotes the t-th frame of the F-dimensional acoustic vectors $\boldsymbol{o}_i$'s of the i-th utterance, C is the number of mixtures in the UBM, N_{ic} is the zeroth-order statistics (Eq. 2.6), and $\gamma_c(\boldsymbol{o}_{i,t})$ is the posterior of the c-th mixture (Eq. 2.7). Therefore, without the UBM, we can still estimate the T-matrix and i-vectors as long as the Baum-Welch statistics are available. In fact, only the observed vectors $\boldsymbol{o}_{i,t}$ and the mixture posteriors $\gamma_c(\boldsymbol{o}_{i,t})$ are necessary for i-vector extraction.[2] This means that we may replace the MFCCs by other types of acoustic features and estimate the mixture posteriors $\gamma_c(\mathbf{o}_{i,t})$ from other models, such as a DNN, rather than the UBM. Specifically, the acoustic feature vectors and mixture posteriors can respectively be written in more general forms:

$$\boldsymbol{o}_{i,t} = f(\boldsymbol{s}_{i,t}) \quad \text{and} \quad \gamma_c(\boldsymbol{s}_{i,t}) = P(c|\boldsymbol{s}_{i,t}), \tag{5.1}$$

[2]In some literatures, $\gamma_c(\boldsymbol{o}_{i,t})$'s are referred to as frame posteriors. But they are in fact the posterior probabilities of mixture components.

where $\boldsymbol{s}_{i,t}$ represents the speech signal in a contextual window comprising multiple frames centered at frame t and $f(\boldsymbol{s}_{i,t})$ is a function that maps acoustic vectors in $\boldsymbol{s}_{i,t}$ to $\boldsymbol{o}_{i,t}$.

5.3 DNN with Denoising Autoencoder

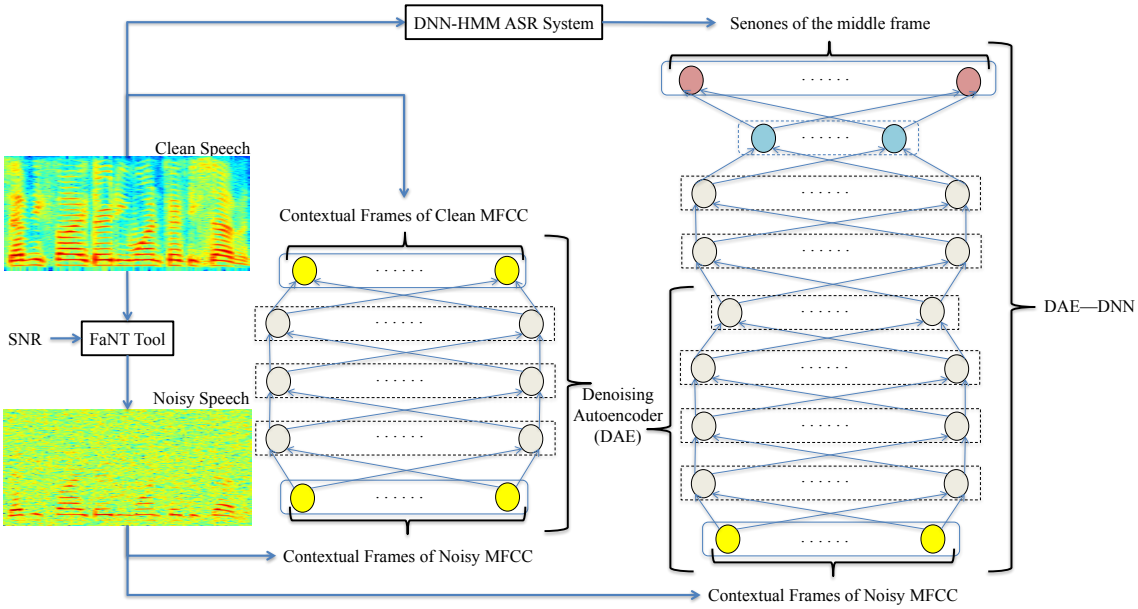

Figure 5.2: Procedure of training the Denoising Autoencoder–Deep Neural Network (DAE–DNN).

In [33], $P(c|\boldsymbol{s}_{i,t})$'s in Eq. 5.1 are given by a DNN that was trained to produce the posteriors of senones given multiple frames of MFCCs as input. In this work, we trained a DNN formed by stacking a deep belief network (DBN) on top of a denoising autoencoder [75] to improve the noise robustness of $P(c|\boldsymbol{s}_{i,t})$. The network architecture of the stacked DNN is shown in the right part of Fig. 5.2. Because of the denoising capability of the DAE and the classification capability of the DNN, we refer to the stacked DNN as denoising autoencoder–deep neural network (DAE–DNN).

Fig. 5.2 illustrates the procedure to train the DAE–DNN. To equip the autoencoder with denoising capability, we used both clean and noisy speech as input and their corresponding clean counterpart as the target output. The denoising autoencoder comprises multiple layers of restricted Boltzmann machines, which are trained layer-by-layer using the contrastive divergence algorithm [31] [78]. Only the bottom half of the RBMs need to be trained, and the upper half are the mirrored copies of the lower half due to the symmetry of the autoencoder. Since we used MFCCs as inputs to the DNN, the first RBM is a Gaussian-Bernoulli RBM and the last layer of the autoencoder is linear. The denoising autoencoder is then fine-tuned by the backpropagation algorithm to minimize the squared errors between the outputs and the clean MFCCs. In practice, we obtained the clean–noisy sample pairs by adding babble noise to clean speech using the FaNT tool [79], which will be explained in Section 5.5.1.

Once the denoising autoencoder has been trained, we built the DAE–DNN using the senone labels as the targets. By adding three layers of RBMs on top of the DAE, the network can extract the phonetic information even if the input is noisy. The details of extracting senone labels will be explained in Section 5.5.3.

To enrich the contextual information in $\mathcal{O}_i$, the vectors $\mathbf{o}_{i,t}$'s are extracted from the bottleneck layer just below the softmax layer of the DNN (the blue nodes in Fig. 5.2). More precisely, $f(\boldsymbol{s}_{i,t})$ in Eq. 5.1 represents the combined effect of the denoising operation in the DAE and the feature extraction operation in the DNN using contextual MFCCs ($\boldsymbol{s}_{i,t}$) as input. The first RBM on top of the DAE is Gaussian-Bernoulli and the last RBM is Bernoulli-Gaussian where the Gaussian hidden layer is of small size. This creates a bottleneck layer (BN) from which the low dimensional BN features can be extracted. The BN features replace the MFCCs during i-vector extraction.

Except for the BN layer and the last layer of the DAE, all hidden layers comprise sigmoid units. The output comprises softmax nodes. More specifically, assume that

there are K distinct senones, the DNN outputs are given by

$$y_k(\mathbf{x}) = \frac{e^{h_k(\mathbf{x})}}{\sum_{k'=1}^{K} e^{h_{k'}(\mathbf{x})}}, \qquad k = 1, \ldots, K,$$

where $\mathbf{x}$ is the input to the DNN, h_k is the activation of the k-th output node, and $y_k(\mathbf{x})$ is the softmax output of node k. The network is trained by minimizing the cross-entropy:

$$E(\mathcal{X}, \mathcal{Z}, \mathcal{C}) = -\sum_{r=1}^{K} \sum_{j=1}^{M_r} \sum_{k=1}^{K} z_{r,j,k} \log(y_k(\mathbf{x}_{r,j}))$$

where $\mathbf{z}_{r,j}$'s are one-of-K vectors indicating to which senone the input vector $\mathbf{x}_{r,j}$ belongs and M_r is the number of vectors from senone r. To be more precise, $\mathbf{x}_{r,j}$ comprises contextual frames of MFCCs, which has the same meaning as $\boldsymbol{s}_{i,t}$ in Eq. 5.1.

To train the DNN, we need to collect all contextual frames of MFCCs belonging to the same senone (indexed by r). To avoid confusion, we use another symbol $\mathbf{x}$ and another set of subscripts (r and j) to highlight the grouping procedure.

5.4 Senone I-vectors

The procedures in Sections 5.2 and 5.3 produce a new variant of i-vectors: senone i-vectors. If the DAE–DNN can be integrated into the i-vector extractor, the resulting senone i-vectors should be noise robust. They should also outperform the conventional i-vectors due to the phonetic information from the BN layers.

Fig. 5.3 illustrates the procedure of senone i-vector extraction. As we have discussed in Section 5.2, only the 0th-, 1st- and 2nd-order Baum-Welch statistics are needed for T-matrix training, and the 0th- and 1st-order statistics are necessary for i-vector extraction. The key idea in this work is to replace the MFCCs by the BN features and replace the mixture posteriors from the UBM by the senone posteriors from the DAE–DNN.

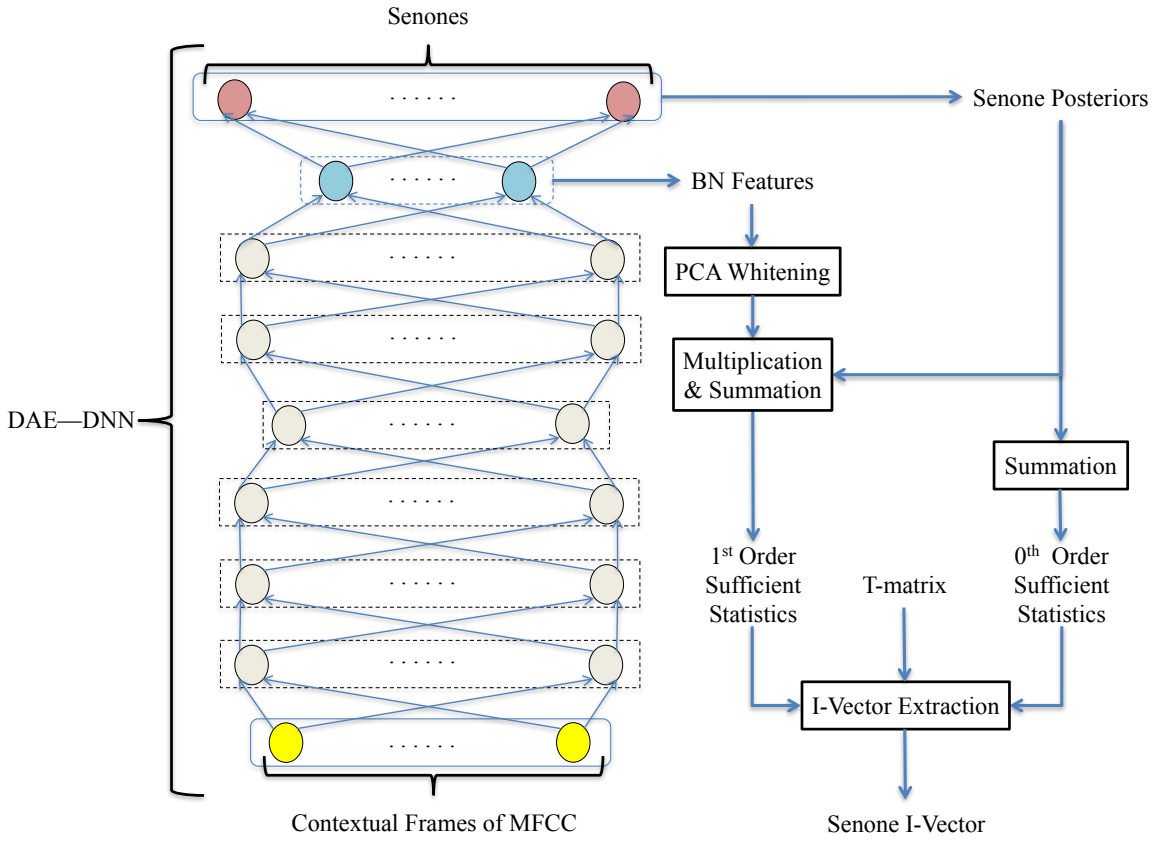

Figure 5.3: Procedure of senone i-vector extraction.

Since the BN features are highly correlated, we used principal component analysis (PCA) whitening to perform decorrelation. The decorrelation process allows us to use diagonal covariance matrices for the BN-based UBM.

Following the notation in Section 5.2, the procedure for training the T-matrix is as follows:

Step 1: Extract BN feature vectors: $\boldsymbol{o}_{i,t}^{(d)} = \mathrm{BN}(\boldsymbol{s}_{i,t})$

Step 2: Compute senone posteriors: $\gamma_c^{(d)}(\boldsymbol{s}_{i,t}) = P_{\text{DAE–DNN}}(c|\boldsymbol{s}_{i,t})$, which is the output of the c-th node in the softmax output layer.

Step 3: Compute Baum-Welch statistics:

$$\begin{aligned} N_{ic}^{(d)} &= \sum\nolimits_t P_{\text{DAE–DNN}}(c|\boldsymbol{s}_{i,t}) \\ \tilde{\boldsymbol{f}}_{ic}^{(d)} &= \sum\nolimits_t \left[P_{\text{DAE–DNN}}(c|\boldsymbol{s}_{i,t})(\mathrm{BN}(\boldsymbol{s}_{i,t}) - \boldsymbol{\mu}_c^{(d)})\right], \\ \boldsymbol{S}_{ic}^{(d)} &= \sum\nolimits_t \left[P_{\text{DAE–DNN}}(c|\boldsymbol{s}_{i,t})(\mathrm{BN}(\boldsymbol{s}_{i,t}) - \boldsymbol{\mu}_c^{(d)})\times \right. \\ &\qquad \left. (\mathrm{BN}(\boldsymbol{s}_{i,t}) - \boldsymbol{\mu}_c^{(d)})^{\intercal}\right], \end{aligned} \tag{5.2}$$

where

$$\boldsymbol{\mu}_c^{(d)} = \frac{\sum_i \sum_t P_{\text{DAE–DNN}}(c|\boldsymbol{s}_{i,t})\mathrm{BN}(\boldsymbol{s}_{i,t})}{\sum_i N_{ic}^{(d)}}.$$

Step 4: Compute the covariance matrices

$$\boldsymbol{\Sigma}_c^{(d)} = \frac{\sum_i \boldsymbol{S}_{ic}^{(d)}}{\sum_i N_{ic}^{(d)}}. \tag{5.3}$$

Step 5: Replace $\tilde{\boldsymbol{f}}_{ic}$, N_{ic} and $\boldsymbol{\Sigma}_c^{(b)}$ of Eq. 2.4 by $\tilde{\boldsymbol{f}}_{ic}^{(d)}$, $N_{ic}^{(d)}$ and $\boldsymbol{\Sigma}_c^{(d)}$ in Eq. 5.2 and

Eq. 5.3:

$$\langle \boldsymbol{w}_i|\mathcal{O}_i^{(d)}\rangle = \boldsymbol{L}_i^{-1}\sum_{c=1}^{C}\boldsymbol{T}_c^{\intercal}(\boldsymbol{\Sigma}_c^{(d)})^{-1}\tilde{\boldsymbol{f}}_{ic}^{(d)}, \tag{5.4a}$$

$$\langle \boldsymbol{w}_i\boldsymbol{w}_i{}^{\intercal}|\mathcal{O}_i^{(d)}\rangle = \boldsymbol{L}_i^{-1} + \langle \boldsymbol{w}_i|\mathcal{O}_i^{(d)}\rangle\langle \boldsymbol{w}_i|\mathcal{O}_i^{(d)}\rangle^{\intercal}, \tag{5.4b}$$

$$\boldsymbol{L}_i = \boldsymbol{I} + \sum_{c=1}^{C} N_{ic}^{(d)}\boldsymbol{T}_c^{\intercal}(\boldsymbol{\Sigma}_c^{(d)})^{-1}\boldsymbol{T}_c, \tag{5.4c}$$

where $i = 1, \ldots, N$. This constitutes the E-step of the EM algorithm.

Step 6: Replace $\tilde{\boldsymbol{f}}_{ic}$, N_{ic} and $\langle \boldsymbol{w}_i|\mathcal{O}_i\rangle$ of Eq. 2.5 by $\tilde{\boldsymbol{f}}_{ic}^{(d)}$, $N_{ic}^{(d)}$ and $\langle \boldsymbol{w}_i|\mathcal{O}_i^{(d)}\rangle$ respectively in Eq. 5.2 and Eq. 5.4 to compute the T-matrix:

$$\boldsymbol{T}_c = \left[\sum\nolimits_i \tilde{\boldsymbol{f}}_{ic}^{(d)}\langle \boldsymbol{w}_i|\mathcal{O}_i^{(d)}\rangle^{\intercal}\right] \times \left[\sum\nolimits_i N_{ic}^{(d)}\langle \boldsymbol{w}_i\boldsymbol{w}_i{}^{\intercal}|\mathcal{O}_i^{(d)}\rangle\right]^{-1}. \tag{5.5}$$

This constitutes the M-step of the EM algorithm. Go back to Step 5 with the updated T-matrix until convergency.

Once the T-matrix has been estimated, the i-vector $\langle \boldsymbol{w}_i|\mathcal{O}_i^{(d)}\rangle$ representing the i-th utterance can be computed according to Eq. 5.4a:

$$\langle \boldsymbol{w}_i|\mathcal{O}_i^{(d)}\rangle = \boldsymbol{L}_i^{-1}\sum_{c=1}^{C}\boldsymbol{T}_c^{\intercal}(\boldsymbol{\Sigma}_c^{(d)})^{-1}\tilde{\boldsymbol{f}}_{ic}^{(d)}.$$

Therefore we can combine the BN features and DNN posteriors to compute the senone i-vectors, and this combination integrates the phonetic information in the DNN into the i-vectors.

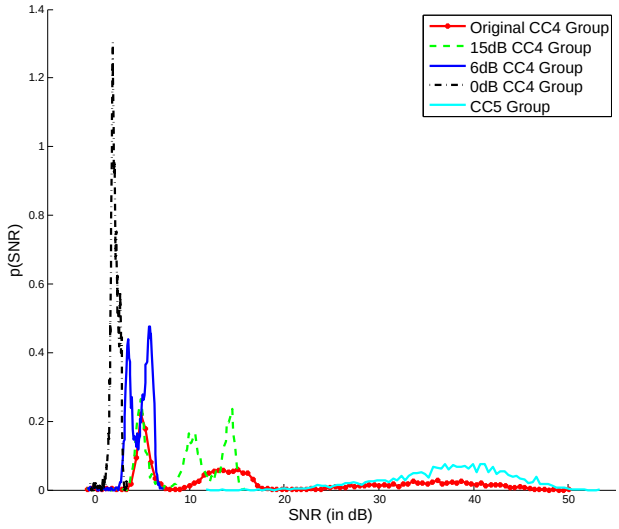

Figure 5.4: The SNR distributions of the original and noise contaminated test utterances in NIST 2012 SRE (CC4, male). For the noise contaminated utterances, babble noise was added to the original utterances at an SNR of 0dB, 6dB, and 15dB, respectively.

5.5 Experiments

5.5.1 Speech Data and Feature Extraction

Speaker verification experiments were performed on the NIST 2012 SRE under Common Condition 4 (CC4). This common condition involves 723 target speakers with 7,116 target utterances from NIST 2006–2010 SREs and 3,900 test utterances from NIST 2012 SRE, including 125,400 trials in core test. Each utterance is about 10 to 300 seconds long, sampled at 8kHz, recorded by telephone, and spoken in English. The baseline is a conventional i-vector/PLDA system, where the acoustic features are MFCCs and the mixture posteriors were obtained from GMM-based UBMs with 1024 and 2048 mixtures. The test utterances in CC4 has a wide range of SNR, from 0dB to 50dB as shown in Fig. 5.4; therefore, CC4 is appropriate for verifying the noise robustness and denoising capability of the proposed algorithm.

To verify the denoising capability under natural noisy environments, we also performed experiments on the NIST 2012 SRE under Common Condition 5 (CC5), including 62,845 trials in the core test and 1,558,788 trials in the extended test. This common condition involves the same 723 target speakers and 7,116 target utterances as in CC4; however, its 2,156 test utterances were collected in noisy environments.

To investigate the capability of various i-vector frameworks under noisy environments, we used the FaNT tool [79] to add babble noise to the target-speaker utterances and test utterances at the SNR of 15dB, 6dB, and 0dB, respectively. Therefore, we have four groups of training utterances and four groups of test utterances, with the first group being the original utterances and the last three groups having SNRs close to 15dB, 6dB, and 0dB, respectively. Hereafter, we refer to these 4 groups as SNR groups. The SNR distributions of the 4 groups of test utterances in CC4 are shown in Fig. 5.4. Note that although the target SNRs that we applied to FaNT are 0dB, 6dB, and 15dB, Fig. 5.4 shows that the peaks of the SNR distributions do not align to these targets. The misalignment is due to the discrepancy in the voice activity detection (VAD) decisions for adding noise and for measuring SNRs. Specifically, FaNT has its own VAD for estimating the amount of noise to be added to the clean signals, whereas the *measured* SNRs in Fig. 5.4 were based on the voltmeter function in FaNT and the decisions of our own noise-robust VAD [72].

Because the babble noise poses a great challenge to voice activity detection (VAD), we used the VAD decisions obtained from the original test utterances for all of the test conditions. Although this procedure may give over-optimistic performance, it avoids the complications arising from wrong VAD decisions. It also allows us to purely compare the capability of different acoustic features and frame-posterior estimation methods, as the comparisons will become meaningless when too many non-speech frames are included in the i-vector extraction processes.

Nineteen MFCCs and log-energy were computed for each 25-ms frame. Together with their 1st and 2nd derivatives, a 60-dimensional acoustic vector was obtained

every 10ms.

5.5.2 I-vector Extraction

All i-vector extractors have 500 total factors. The PLDA further reduces the speaker subspace to 150 dimensions. The GMM–UBMs and the total-variability matrixes were trained by using the utterances from the original 7,116 target telephone utterances mentioned earlier and the microphone utterances (interview speech) of the same set of target speakers in NIST 2006–2010 SREs. The PLDA model was trained by using the i-vectors derived from all of the original and noise contaminated telephone utterances and from the interview speech segments of NIST 2006–2010 SREs.

5.5.3 Senone Label Extraction[3]

We used a DNN–HMM acoustic model trained on SwitchBoard-1 release 2 to obtain the senone label for each frame. This release contains approximately 290 hours of US English telephone conversations spoken by 500 speakers. The 4,870 conversation sides were spliced into 259,890 utterances for acoustic modeling. The original DNN has 6 hidden layers with 2,048 nodes per layer, and a softmax output layer with 8,704 nodes, corresponding to 8,704 clustered states (senones). We further clustered the 8,704 senones into 2,000 senones, resulting in a DNN with 2,000 outputs nodes. The features are 13-dimensional cepstral mean-variance normalized (CMVN) MFCCs, and they were extracted from speech data every 10ms over a window of 25ms. For each frame, its neighbouring 4 frames were included and transformed by linear discriminative analysis (LDA) to 40 dimensions, followed by maximum-likelihood linear transformation. Speaker adaptation based on feature-space maximum likelihood linear regression (fMLLR) was also applied.

---

[3]This part was done by Yingke Zhu and Brian K. W. Mak of the Hong Kong University of Science and Technology.

For each frame, the fMLLR-transformed vectors of the 5 preceding and 5 succeeding frames were fed to the DNN, which outputs the posterior probabilities of different senones, and the one with the highest posterior is the senone label for the frame.

5.5.4 Training of the DAE–DNN

The input of the DAE–DNN comprises eleven 20-dimensional MFCC vectors extracted from 11 contextual frames, which amount to $20 \times 11 = 220$ input nodes. Element-wise z-norm was applied to the 220 inputs so that Gaussian-Bernoulli RBM pre-training can be applied. As shown in Fig. 5.2, the DAE has a structure 220-256-256-256-220, where the first and the last values are the numbers of inputs and outputs, respectively. Only the first two layers of the DAE needed to be pre-trained by contrastive divergence, and the last two layers were stacked by flipping the first two RBMs. The DAE's output layer uses the linear activation function. After RBM pre-training, the DAE was fine-tuned by backpropagation (BP) using the the mean squared error criterion.

After BP fine-tuning, three RBMs were put on top of the DAE, where the bottom one is a Gaussian-Bernoulli RBM and the top one is a Bernoulli-Gaussian RBM. BP fine-tuning was then applied to the combined DAE and RBMs using the 2000 senone labels (in one-hot format) as the target outputs and cross-entropy as the minimization criterion. As shown in Fig. 5.2, the final DAE–DNN has a structure 220-256-256-256-220-256-256-60-2000, where the last softmax layer has 2000 nodes and the bottleneck layer has 60 nodes. Therefore the BN features have a dimension of 60. The bottleneck layer uses the linear activation function, and all the other hidden layers use sigmoid nonlinearity.

The training set for training the DAE–DNN comprises 7,116 clean (original) utterances from NIST 2006–2010 SREs and their 15dB, 6dB, and 0dB noise contaminated versions, which amount to a total of $7,116 \times 4 = 28,464$ training utterances. These utterances were spoken by 723 target speakers in CC4 of NIST 2012 SRE. The DAE on the left of Fig. 5.2 was trained to produce the clean MFCCs of the utterances, given

the clean or noisy MFCCs as input. The DAE–DNN on the right of Fig. 5.2 was trained to produce the senone labels of the clean utterances based on the ASR–DNN mentioned in Section 5.5.3.

As the procedure in Section 5.4 and Fig. 5.3 show, we can obtain the senone i-vectors by combining BN features and senone posteriors. With the same PLDA back-end as the baseline, we can compare the performance of senone i-vectors with the conventional i-vectors.

5.5.5 Enrollment and Test Utterances

Because CC4 in 2012 SRE involves noise-contaminated test utterances, this test condition covers a wide range of SNR distribution, and we refer to this test condition as "original". In addition to this "original" test condition, we created three test conditions based on the noise contaminated test utterances by the FaNT tool as mentioned in Section 5.5.1. Specifically, for the 15dB test condition, test utterances with babble noise added at an SNR of 15dB were used for scoring, and similarly for the 6dB and 0dB test conditions. For all of the original, 15dB, 6dB and 0dB CC4 test conditions of NIST 2012 SRE, we used the original target-speaker utterances and their noise contaminated counterparts from the 6dB and 15dB SNR groups as enrollment utterances in order to keep consistency. Therefore the enrollment utterances were the same for different test conditions.

Unlike the test segments in CC4, the test segments in CC5 were intentionally collected in a noisy environment. Therefore, the noisy speech in CC5 is more realistic. The test segments in CC5 have a wide range of SNRs, from 10dB to over 40dB. Because most of the test segments in CC5 have SNR over 20dB, we only used the i-vectors of the original enrollment segments (which also have high SNR) to represent the target speakers during the scoring stage.

5.5.6 Producing Likelihood-Ratio Scores

The PLDA model for scoring was trained by the the original enrollment utterances and their noise contaminated counterparts from the 0dB, 6dB and 15dB SNR groups. For real-world deployment, it is desirable to have application-independent decision thresholds [80] such that not only the equal error rate (EER) and minimum detection cost (minDCF) are minimized, but also the actual DCF (actDCF) at specific thresholds are also small. To this end, all of the original PLDA scores were subject to score calibration to produce true likelihood-ratio scores using the Bosaris toolkit [81]:

$$S' = w_0 + w_1 S, \tag{5.6}$$

where S is the original PLDA scores. This calibration step only shifts and scales the original PLDA scores, which reduce the actDCF (primary cost) without affecting the EER and minDCF.

5.6 Results and Discussions

5.6.1 Acoustic Features and Posterior Computation

Table 5.1 shows the EER, minDCF and actDCF of various i-vector/PLDA systems that use different acoustic features and different ways of computing the senone posteriors or mixture posteriors. To study the benefit of DAE training in more details, in addition to the DAE–DNN, we also trained a DNN without DAE pre-training but with RBM pre-training, i.e., all hidden layers in Fig. 5.3 were initialized by RBMs. We refer to this network as DNN. It has a structure of 220-256-256-256-256-256-256-60-2000.

Surprisingly, a comparison between the first and the second rows of Table 5.1 suggests that MFCC–UBM with 2048 mixtures performs worse than the one with 1024 mixtures. Since the same amount of training data was used in these two mod-

Table 5.1: Performance of various i-vector/PLDA systems on NIST 2012 SRE (CC4, male speaker, core task) with test utterances contaminated with babble noise at different SNRs. DAE–DNN is DNN with DAE training (Fig. 5.2). DNN is a DNN pre-trained by RBMs. The UBM here refers to a speaker-independent GMM. Denoised MFCC is the MFCC denoised by the DAE in Fig. 5.2 (left panel). BN features: Bottleneck features obtained from the DAE–DNN (Fig. 5.3)

		Original			15dB			6dB			0dB		
Acoustic Features	Posteriors from	EER	minDCF	actDCF	EER	minDCF	actDCF	EER	minDCF	actDCF	EER	minDCF	actDCF
MFCC	UBM(1024-mix)	2.62	0.285	0.835	3.58	0.336	0.847	3.27	0.372	0.871	4.84	0.501	0.915
MFCC	UBM(2048-mix)	3.60	0.442	0.969	3.22	0.458	0.966	3.65	0.505	0.976	5.39	0.651	0.989
MFCC	DNN	1.69	0.230	**0.786**	**1.92**	0.281	0.816	2.64	0.324	**0.799**	3.16	0.474	0.878
MFCC	DAE–DNN	1.82	0.253	0.808	2.75	0.273	**0.790**	2.57	0.296	0.816	3.43	0.474	0.861
Denoised MFCC	DAE–DNN	2.46	0.339	0.933	3.45	0.331	0.907	3.74	0.387	0.924	4.73	0.643	0.942
BN Features	DAE–DNN	**1.56**	**0.218**	0.859	2.17	**0.212**	0.799	**2.01**	**0.229**	0.817	**3.07**	**0.432**	**0.852**

els, increasing the number of mixtures, i.e., increasing the learning capacity of the model, does not necessarily improve performance. Furthermore, for UBM with 2048 mixtures, the misalignment of speech frames to mixture component would be more severe under noisy conditions, causing further performance degradation. Results from the first three rows of Table 5.1 suggest that the i-vectors derived from senone posteriors obtained from the DNN outperform the i-vectors whose mixture posteriors $\gamma_c(\boldsymbol{o}_{i,t})$'s are obtained from the UBM. This confirms the findings in [28, 32, 33] that the phonetic information in the senone i-vectors is beneficial for speaker comparison. The comparison between the 3rd and the 4th rows suggest that the DAE training hurts the prediction of senones except for the 6dB case.

In Table 5.1, the denoised MFCC and the senone posteriors of each frame were extracted from the left- and right-networks of Fig. 5.2, respectively. After DAE training of the left-network, it was used for initializing the lower part of the right-network. After backpropagation fine-tuning, the right-network is able to produce the senone posteriors given a contextual window of noisy MFCCs as input. For Row 4 of Table 5.1, the right-network was asked to compute the senone posteriors given the noisy MFCCs, which are exactly the network input. As a result, there is a perfect match between the acoustic features (noisy MFCCs) and the senone posteriors. On the other hand, for Row 5 of Table 5.1, the right-network was asked to compute the senone posteriors given the denoised MFCCs, which do not agree with the network input. This causes mismatch between the senone posteriors produced by the network and the acoustic features (denoised MFCCs), which explains why the performance in Row 5 is much poorer than that in Row 4 of Table 5.1. Note that because of the backpropagation fine-tuning, the output of the lower part of the right-network in Fig. 5.2 cannot be considered as denoised MFCCs. As a result, denoised MFCCs can only be extracted from the left-network.

On the other hand, the last row suggests that the denoised senone i-vectors, in which both the BN features and posteriors are obtained from the DAE–DNN, achieve

the best performance under all of the 4 SNR conditions. The good performance of these BN-based senone i-vectors is attributed to the fact that both the BN features and the senone posteriors are obtained from the same network (the DAE–DNN). Therefore, they work very well with each other. We conjecture that there is a compromise between the robustification of BN features by the DAE and the amount of speaker information loss. For the BN-based senone i-vectors, the benefit of the former prevails.

Although we have performed score calibration by using the logistic regression function in the Bosaris toolkit [81], all the systems still have high actDCF. This is mainly caused by the wide range of SNR in the training data for estimating the calibration weights. The training data comprise the original utterances and noise contaminated utterances at 0dB, 6dB and 15dB. More advanced calibration techniques [42, 82] are needed to improve the actDCF performance.

5.6.2 Senone Posteriors vs. Mixture Posteriors for BN Features

Table 5.2: Performance of BN-based i-vector/PLDA systems on NIST 2012 SRE (CC4, male speaker, core task) with test utterances contaminated with different levels of babble noise. DAE–DNN is DNN with DAE training (Fig. 5.2). The UBM here is a speaker-independent GMM trained by using BN features.

	Original		15dB		6dB		0dB	
Posteriors from	EER	minDCF	EER	minDCF	EER	minDCF	EER	minDCF
UBM(1024-mix)	3.19	0.357	4.11	0.350	3.73	0.358	4.70	0.484
UBM(2048-mix)	1.97	**0.203**	2.63	0.245	2.58	0.239	3.70	**0.389**
DAE–DNN	**1.56**	0.218	**2.17**	**0.212**	**2.01**	**0.229**	**3.07**	0.432

Table 5.2 compares the performance of two BN-based i-vector/PLDA systems. In the first two rows, the mixture posteriors $\gamma_c(\boldsymbol{o}_{i,t})$'s were obtained from the UBM; whereas in the last row, the senone posteriors $\gamma_c(\boldsymbol{s}_{i,t})$'s were obtained from the DAE–DNN. The BN features were extracted from the DAE–DNN shown in Fig. 5.3. The

performance improves significantly when the number of UBM mixtures increases from 1024 to 2048. Because the DNN has 2000 outputs (each representing a senone), it is reasonable that the BN features have around 2000 clusters in the feature space. Due to their noise robustness, the BN features still have the similar distributions even under noisy environments. Therefore, the UBM with 2048 mixtures is more appropriate for modeling the BN features. On the other hand, each Gaussian in the 1024-mixture UBM requires to model roughly two senone clusters, which limits the performance of the BN i-vectors derived from this UBM. Under all of the 4 SNR conditions, using the posteriors from the DAE–DNN improves performance significantly. Although the performance of the system with senone posteriors drops when the test utterances become noisier, it is still superiors to the one with GMM mixture posteriors. It shows that the DAE–DNN can estimate the senone posteriors accurately under noisy conditions to some extent.

5.6.3 Importance of DAE Training

Table 5.3: Performance of various i-vector/PLDA systems on NIST 2012 SRE (CC4, male speaker, core task) with test utterances contaminated with different levels of babble noise. DAE–DNN is DNN with DAE training (Fig. 5.2). DNN has a similar structure as DAE–DNN, but without DAE training.

		Original		15dB		6dB		0dB	
BN Features from	Posteriors from	EER	minDCF	EER	minDCF	EER	minDCF	EER	minDCF
DAE–DNN	DAE–DNN	1.56	0.218	2.17	0.212	**2.01**	0.229	3.07	0.432
DAE–DNN	DNN	1.46	**0.212**	**2.08**	0.205	**2.01**	0.236	**2.90**	0.438
DNN	DAE–DNN	**1.30**	0.220	2.12	0.200	2.05	**0.227**	3.08	**0.425**
DNN	DNN	1.54	**0.212**	2.24	**0.199**	2.04	0.246	3.20	0.435

With DAE–DNN and DNN, we have four combinations of BN features and senone posteriors as shown in Table 5.3. Comparing Row 2 and Row 4 of Table 5.3 suggests that DAE training improves the performance of BN features if the posteriors are from the DNN without DAE training. Similarly, comparing Row 3 and Row 4 suggests that

DAE training becomes important for estimating the senone posteriors when the BN features are obtained from the DNN without DAE training. However, comparing Row 1 and Row 3 suggests that DAE training is not necessary for extracting the BN features if the posteriors are obtained from the DAE–DNN. On the other hand, Row 1 and Row 2 suggest that DAE training is not necessary for estimating the senone posteriors when the BN features are obtained from the DAE–DNN. In conclusion, the DAE training can benefit the senone i-vector systems if one of the two components (BN feature extraction and posterior computation) receives DAE training. Overall speaking, the senone i-vectors whose phonetically discriminative BN features are obtained from the DAE–DNN and senone posteriors are obtained from the DNN without DAE training (Row 2) achieve the best performance.

Table 5.4: Performance of various i-vector/PLDA systems on NIST 2012 SRE (CC5, male speaker, core task). DAE–DNN is DNN with DAE training (Fig. 5.2). DNN has a similar structure as DAE–DNN, but without DAE training.

		CC5 Core		CC5 Extended	
BN Features from	Posteriors from	EER	minDCF	EER	minDCF
DAE–DNN	DAE–DNN	2.18	**0.251**	**2.15**	**0.248**
DAE–DNN	DNN	**2.07**	0.261	2.16	**0.248**
DNN	DAE–DNN	2.24	0.278	2.19	0.253
DNN	DNN	2.35	0.263	2.68	0.255

The performance of BN-based senone i-vectors under CC5 of NIST 2012 SRE (male speaker) is shown in Table 5.4. The performance in Row 1 and Row 2 is significantly better than the one in Row 3 and Row 4, which suggests that DAE training benefits the BN features. However, comparisons between Row 1 and Row 2 and between Row 3 and Row 4 of Table 5.4 suggest that DAE training is more effective for cleaning up BN features than for robustifying senone posteriors. This conclusion is consistent with the one under CC4, where the senone i-vectors with BN features from DAE–DNN and senone posteriors from DNN (Row 2 in Table 5.3)

achieve the best performance in Table 5.3.

Table 5.5: Cross-entropy of DAE–DNN and DNN on the training set mentioned in Section 5.5.4 and the noise contaminated test utterances from CC4 of NIST 2012 SRE. DAE–DNN is DNN with DAE training (Fig. 5.2). DNN has a similar structure as DAE–DNN, but without DAE training.

	Training Set	15dB CC4	6dB CC4
DAE–DNN	**6.32**	6.91	6.97
DNN	6.34	**6.82**	**6.89**

To verify the conclusion that DAE training is less beneficial for senone posteriors estimation, we calculated the cross-entropy of DAE–DNN and DNN on the training set mentioned in Section 5.5.4 and on the noise contaminated test utterances from CC4 of NIST 2012 SRE. The results are shown in Table 5.5. The results show that the cross-entropy on the training set is almost the same regardless of whether DAE training is applied. However, with DAE training, the cross-entropies on the noise contaminated test utterances become higher. This suggests that while DAE training can benefit BN feature extraction, it reduces the generalization capability of the network, causing less accurate senone posteriors on the test utterances. This agrees with our conclusions in Table 5.1, Table 5.3 and Table 5.4.

The DET curves of the systems corresponding to Row 2 and Row 4 in Table 5.4 are shown in Fig. 5.5. The senone posteriors of both systems were obtained from a DNN without DAE training. The results clearly show that DAE training is beneficial to BN feature extraction, as the DET curve of "BN from DAE–DNN" is below that of "BN from DNN" for a wide range of decision thresholds.

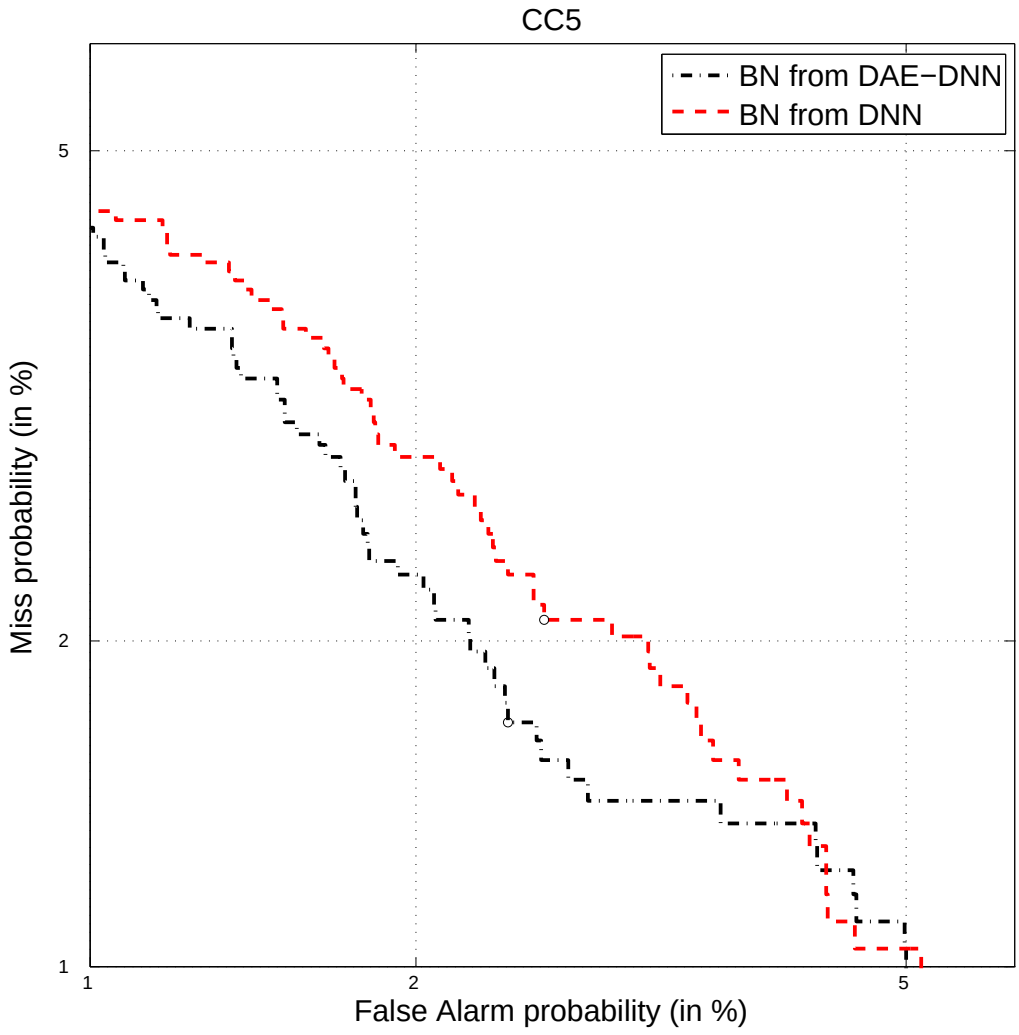

Figure 5.5: The DET performance (CC5 of NIST 2012 SRE) of two senone i-vector systems based on BN features. In the legend, "BN from DAE–DNN" and "BN from DNN" mean that the bottleneck features were obtained from a DNN with and without DAE training, respectively. They correspond to Row 2 and Row 4 in Table IV, respectively. In both cases, the senone posteriors were obtained from the DNN without DAE training.

Chapter 6

DNN-BASED SCORE CALIBRATION

6.1 Introduction

During verification, given an i-vector pair derived from the utterance of a target (claimed) speaker and the utterance of a claimant, a likelihood ratio score (namely PLDA score)

$$S = \frac{p(\text{i-vector pair}|\text{same speaker})}{p(\text{i-vector pair}|\text{different speaker})}$$

is computed to determine whether the i-vectors in the i-vector pair are from the same speaker or not. When computing the PLDA score, the difference between the speaker and channel subspaces in the PLDA model is leveraged to marginalize the channel effect on the i-vector pair. This marginalization process leads to a likelihood ratio score that is mainly dependent on speaker characteristics of the i-vector pair.

To compensate for the detrimental effect on the PLDA scores discussed in Section 2.6.3, in this chapter, we attempt to directly model the complex relationship between score shift and distorted i-vectors. Inspired by the recent findings that deep neural networks (DNNs) have a high capacity in modeling complex relationship, we trained a DNN using i-vector pairs (derived from both clean and noisy speech) as inputs and the ideal score shifts as target outputs. This method, however, requires parallel training data comprising clean and noisy i-vectors so that ideal score shifts can be computed during the training stage. To obviate this requirement, we trained a second DNN that can produce close-to-ideal (clean) PLDA scores by using i-vector pairs augmented with the PLDA score as input. To further leverage the meta data (SNR and speaker labels) that can be easily obtained from training utterances, we

used multi-task learning to train a third network whose input is identical to the second DNN but its outputs aim to achieve two tasks: regression and classification. For the former, the network was trained to produce ideal score shift, clean score, and the SNRs of target and test utterances, whereas for the latter, the network aims to classify whether the i-vector pairs come from the same speaker or from different speakers.

This chapter is organized as follows. We will introduce the previous score calibration methods in Section 6.2; based on these methods, we propose the DNN-based calibration methods in Section 6.3, where the DNNs are trained to output the calibrated score directly without estimating the score shift. Experiments on NIST 2012 SRE in Section 6.4 show that the auxiliary tasks in multi-task learning help the DNNs to find a better solution, which makes the multi-task DNNs outperform the single-task DNNs under all SNR conditions significantly.

6.2 Quality-based Score Calibration

To improve the robustness of speaker verification, Mandasari *et al.* [45,46] and Hasan *et al.* [42] proposed several quality measure functions (QMFs) to compensate for the score shift caused by background noise and short utterance duration. A QMF is a function of some quality measures such as SNR and duration that can be directly obtained from utterances. Denote S as the *uncalibrated* verification score of a target-speaker utterance and a test utterance. Also denote λ_{tgt} and λ_{tst} as the quality measures of the target-speaker and test utterances, respectively. Then, the *calibrated* score S' can be computed as follows:

$$S' = w_0 + w_1 S + Q(\lambda_{tgt}, \lambda_{tst}), \tag{6.1}$$

where $Q(\lambda_{tgt}, \lambda_{tst})$ is a QMF. In [45, 46], the QMFs were based on the duration and SNR of test speech:

$$\begin{aligned} Q_{\text{SNR}}(\text{SNR}_{tst}) &= w_2 \text{SNR}_{tst} \\ Q_{\text{Dur}}(d_{tst}) &= w_2 \log(d_{tst}) \\ Q_{\text{SNR+Dur}}(\text{SNR}_{tst}, d_{tst}) &= w_2 \text{SNR}_{tst} + w_3 \log(d_{tst}), \end{aligned} \tag{6.2}$$

where SNR_{tst} and d_{tst} are the SNR and duration of the test utterance, and w_2 and w_3 are their corresponding weight. If the effect of noise in the target utterance is also considered, Q_{SNR} becomes:

$$Q_{\text{SNR2}}(\text{SNR}_{tgt}, \text{SNR}_{tst}) = w_2 \text{SNR}_{tgt} + w_3 \text{SNR}_{tst}, \tag{6.3}$$

where SNR_{tgt} is the SNR of the target-speaker utterance. In Eqs. 6.1–6.3, the weights w_i, $i = 0, \ldots, 3$, can be estimated by logistic regression [83].

By assuming that i-vectors are acoustic-condition dependent, Ferrer *et al.* [49] and Nautsch *et al.* [47] derived a quality vector $\boldsymbol{q}$ based on the posterior probabilities of various acoustic conditions given an i-vector. Thus, each i-vector (either from target speaker or test speaker) is associated with a quality vector, and the score shift of a verification trial is a function of the quality vectors derived from the i-vectors in that trial. In [47], the function is called the function of quality estimate (FQE). Specifically, i-vectors derived from utterances of 55 combinations of different durations and SNRs were used to train 55 Gaussian models $\Lambda_j = \{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}\}_{j=1}^{55}$. Each of these Gaussian models has its own mean $\boldsymbol{\mu}_j$ estimated from the i-vectors of the respective condition, but they share the same global covariance matrix $\boldsymbol{\Sigma}$. The j-th element of $\boldsymbol{q}$ for an i-vector $\boldsymbol{x}$ is defined as the posterior of the j-th condition:

$$q_j = \frac{\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma})}{\sum_{j'} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{j'}, \boldsymbol{\Sigma})}, \ j = 1, \ldots, 55. \tag{6.4}$$

Given the i-vectors $\boldsymbol{x}_{tgt}$ and $\boldsymbol{x}_{tst}$ from a target-speaker and a test speaker, respectively, the corresponding quality vectors $\boldsymbol{q}_{tgt}$ and $\boldsymbol{q}_{tst}$ are obtained from Eq. 6.4 and the score shift can be obtained from a symmetric bilinear matrix $\boldsymbol{W}$ or cosine-distance score as follows:

$$\begin{aligned} Q_{\text{UAC}}(\boldsymbol{q}_{tgt}, \boldsymbol{q}_{tst}) &= w_2 \boldsymbol{q}_{tgt}^{\intercal} \boldsymbol{W} \boldsymbol{q}_{tst} \\ Q_{\text{qvec}}(\boldsymbol{q}_{tgt}, \boldsymbol{q}_{tst}) &= w_2 \cos(\boldsymbol{q}_{tgt}, \boldsymbol{q}_{tst}), \end{aligned} \tag{6.5}$$

where

$$\cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a}^{\intercal} \boldsymbol{b}}{\|\boldsymbol{a}\| \|\boldsymbol{b}\|}.$$

In Eq. 6.5, the elements in a quality vector are the posteriors with respect to the corresponding SNR/duration groups, and the simple functions (bilinear transformation and cosine distance) of the two quality vectors could only reflect their similarity in terms of SNR and duration. It is still very close to the QMFs in Eq. 6.2 and Eq. 6.3, where the score shift is assumed to be linear with respect to SNR of utterances and/or to the logarithm of utterance duration. As we will discuss in Section 6.4 and Fig. 6.8, the relationship between score shift and SNR and log-duration is complex, and only the SNR and log-duration information is not enough to estimate the ideal score shift. The i-vectors are essential for estimating the ideal score shift.

6.3 DNN-based Score Calibration

This chapter proposes an innovative score calibration algorithm to mitigate the limitations of the score calibration algorithms described in Section 6.2. The main idea is to use deep neural networks (DNNs) to estimate the appropriate score shift or to output clean PLDA scores given noisy i-vectors and noisy PLDA scores as input. When the DNN is used for estimating the score shift, it essentially performs *score compensation* and its role is the same as that of the function Q in Eq. 6.1. However, when the DNN is used for outputting clean PLDA scores, it essentially performs *score*

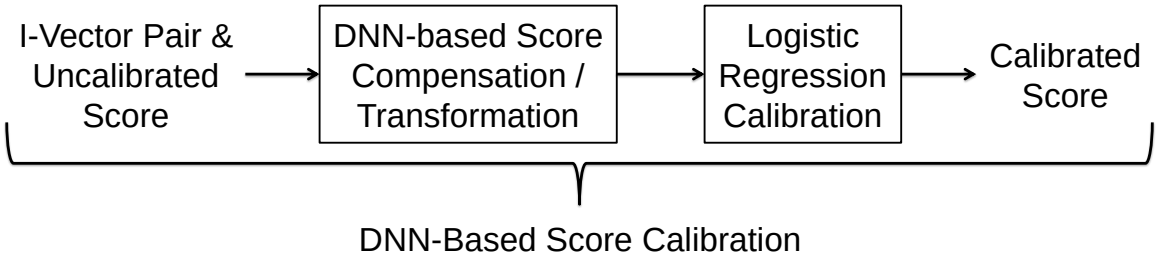

Figure 6.1: DNN-based score calibration.

transformation. For whatever roles, a further *calibration* process is essential because the DNN cannot guarantee that the resulting scores are true log-likelihood ratios. To avoid cluttering with terminologies, we collectively refer to the score compensation, transformation, and calibration processes as DNN-based score calibration. Fig. 6.1 shows the full process.[1]

6.3.1 DNN Score Compensation: Estimating Score Shifts by DNNs

The proposed algorithm uses a DNN to estimate the appropriate score shift given the target and test i-vector pairs ($\boldsymbol{x}_{tgt}$ and $\boldsymbol{x}_{tst}$) and the uncalibrated PLDA score S as shown in Fig. 6.3. Specifically, given an uncalibrated PLDA score S of a verification trial, the compensated score is given by:

$$S_1' = S + \mathrm{DNN}_1(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}, S), \tag{6.6}$$

where DNN_1 denotes the output of a DNN that receives i-vector pairs and uncalibrated scores as input. With these inputs, the DNN outputs the shift of PLDA scores due

[1] In some studies [84,85], the term *calibration* strictly referred to the process of adjusting the scores without affecting the equal error rate (EER). Here, we follow the terminology in [45–47] and relax the definition of calibration to include the processes that lead to better EER performance.

to the deviation of the acoustic conditions from the clean one:

$$\mathrm{DNN}_1(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}, S) \approx \delta_{score} = S_{cln} - S, \tag{6.7}$$

where S_{cln} is the PLDA score if both $\boldsymbol{x}_{tgt}$ and $\boldsymbol{x}_{tst}$ were derived from clean utterances. Substituting Eq. 6.7 to Eq. 6.6, we have:

$$S_1' \approx S + (S_{cln} - S) = S_{cln},$$

which means that the clean score can be recovered.

To train DNN_1, we need i-vectors derived from both clean and noise contaminated utterances where the amount of noise contamination should be varied to give a rich set of δ_{score}'s. This can be done by using the FaNT tool [79] with the target SNR set to various levels. The procedure of computing S_{cln}, S and δ_{score} at the training stage is illustrated in Fig. 6.2. Note that Fig. 6.2 depicts the situation where both of the target-speaker and test utterances are noisy. However, in real environments, there are situations where either the target-speaker utterance or the test utterance is clean, or both are clean. To accommodate these situations, some of the "noisy i-vectors" in Fig. 6.2 should be derived from clean speech. Therefore, if both of the i-vectors in the lower branch of Fig. 6.2 are derived from clean utterances, we have $S = S_{cln}$ and $\delta_{score} = 0$. This is exactly what we want the DNN to produce when the input i-vectors are clean.

As we have mentioned in Section 2.4.3, the PLDA score of i-vector pair ($\boldsymbol{x}_{tgt}$, $\boldsymbol{x}_{tst}$) can be expressed in terms of the log-likelihood ratio $\mathrm{LLR}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst})$:

$$\begin{aligned} S &= \mathrm{LLR}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}) \\ &= \boldsymbol{x}_{tgt}^{\intercal} \boldsymbol{Q} \boldsymbol{x}_{tgt} + \boldsymbol{x}_{tst}^{\intercal} \boldsymbol{Q} \boldsymbol{x}_{tst} + 2\boldsymbol{x}_{tgt}^{\intercal} \boldsymbol{P} \boldsymbol{x}_{tst} + \mathrm{const}, \end{aligned} \tag{6.8}$$

where $\boldsymbol{Q}$ and $\boldsymbol{P}$ are the matrices derived from the total covariances and across-speaker

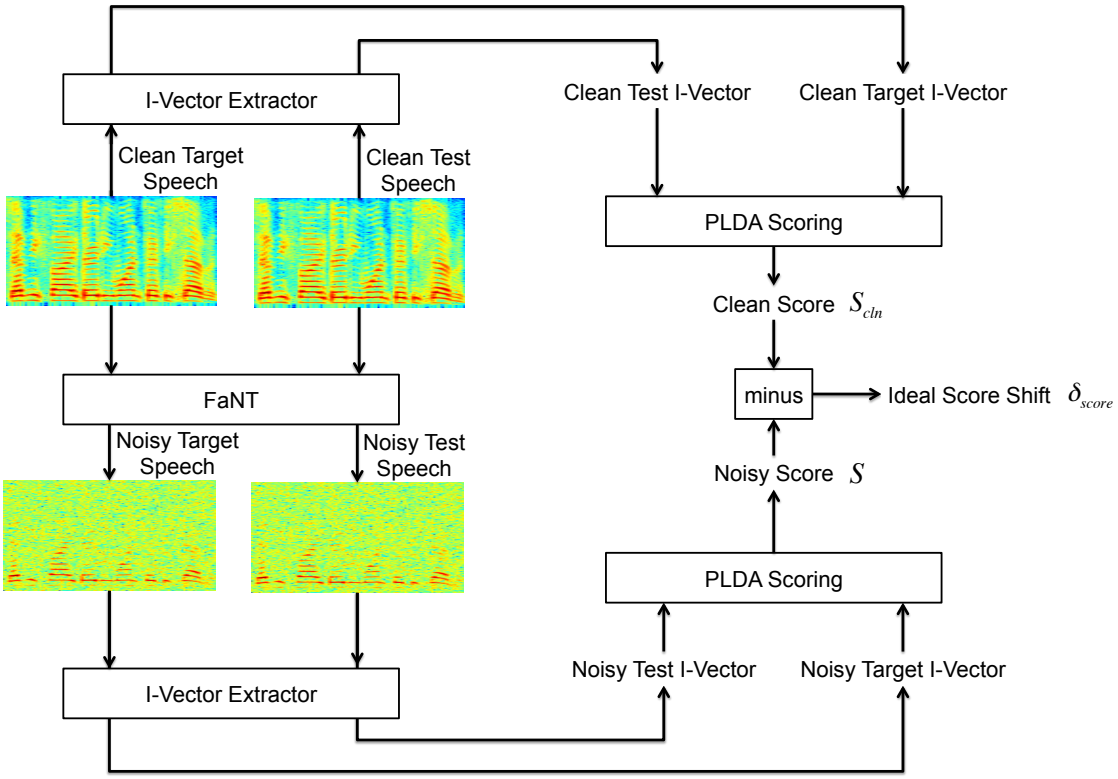

Figure 6.2: Procedure of computing clean scores, noisy scores and score shifts during the training stage. The SNRs of the target speech and the test speech can be different, and even one or both of them could be clean.

covariances of i-vectors [14]. Using Eq. 6.8, the general form of score shift is:

$$\begin{aligned}\delta_{score} &= \text{LLR}(\boldsymbol{x}_{tgt\_cln}, \boldsymbol{x}_{tst\_cln}) - \text{LLR}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}) \\ &= \boldsymbol{x}_{tgt\_cln}^{\intercal}\boldsymbol{Q}\boldsymbol{x}_{tgt\_cln} - \boldsymbol{x}_{tgt}^{\intercal}\boldsymbol{Q}\boldsymbol{x}_{tgt} \\ &\quad + \boldsymbol{x}_{tst\_cln}^{\intercal}\boldsymbol{Q}\boldsymbol{x}_{tst\_cln} - \boldsymbol{x}_{tst}^{\intercal}\boldsymbol{Q}\boldsymbol{x}_{tst} \\ &\quad + 2\boldsymbol{x}_{tgt\_cln}^{\intercal}\boldsymbol{P}\boldsymbol{x}_{tst\_cln} - 2\boldsymbol{x}_{tgt}^{\intercal}\boldsymbol{P}\boldsymbol{x}_{tst}.\end{aligned} \tag{6.9}$$

Note the difference between Eq. 6.9 and the bilinear transformation in Eq. 6.5. The score shift in Eq. 6.9 involves not only a bilinear transformation between the target-speaker and test i-vectors in its last term, but also the bilinear transformation of clean and noisy test i-vectors. If we were to know the clean test i-vector ($\boldsymbol{x}_{tst\_cln}$) for every noisy test i-vector ($\boldsymbol{x}_{tst}$), then Eq. 6.9 can be easily computed without a DNN. However, as we do not know $\boldsymbol{x}_{tst\_cln}$, we resort to relying on the DNN to learn the complex relationship between the input i-vector pairs and the score shifts.

6.3.2 DNN Score Transformation: Recovering Clean PLDA Scores by DNNs

The score calibration method in Section 6.3.1 and previous scoring methods such as QMF and FQE use the concept of score shift to compensate or calibrate the scores. However, if the clean score can be restored, the estimation of score shifts seems to be redundant. To make the restored scores close to the ideal clean scores, we can use a DNN to model the complex relationship between the i-vector pairs, noisy scores (S), and the clean scores (S_{cln}):

$$S_2' = \text{DNN}_2(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}, S) \approx S_{cln}. \tag{6.10}$$

In this model, the DNN (see Fig. 6.4) receives an i-vector pair and its corresponding noisy score as input, and it is trained to output the clean score.

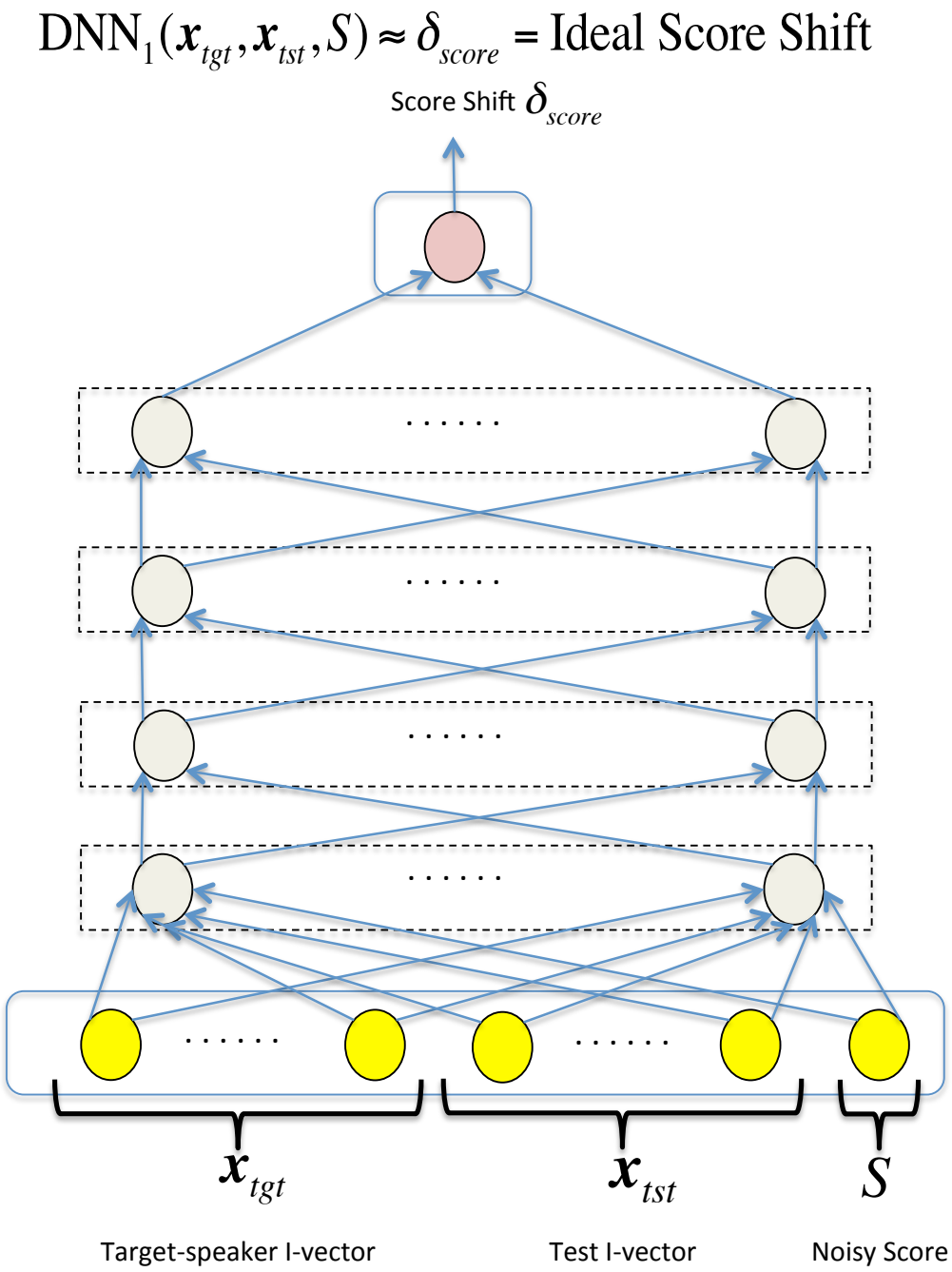

Figure 6.3: Single-task DNN with score shift as output.

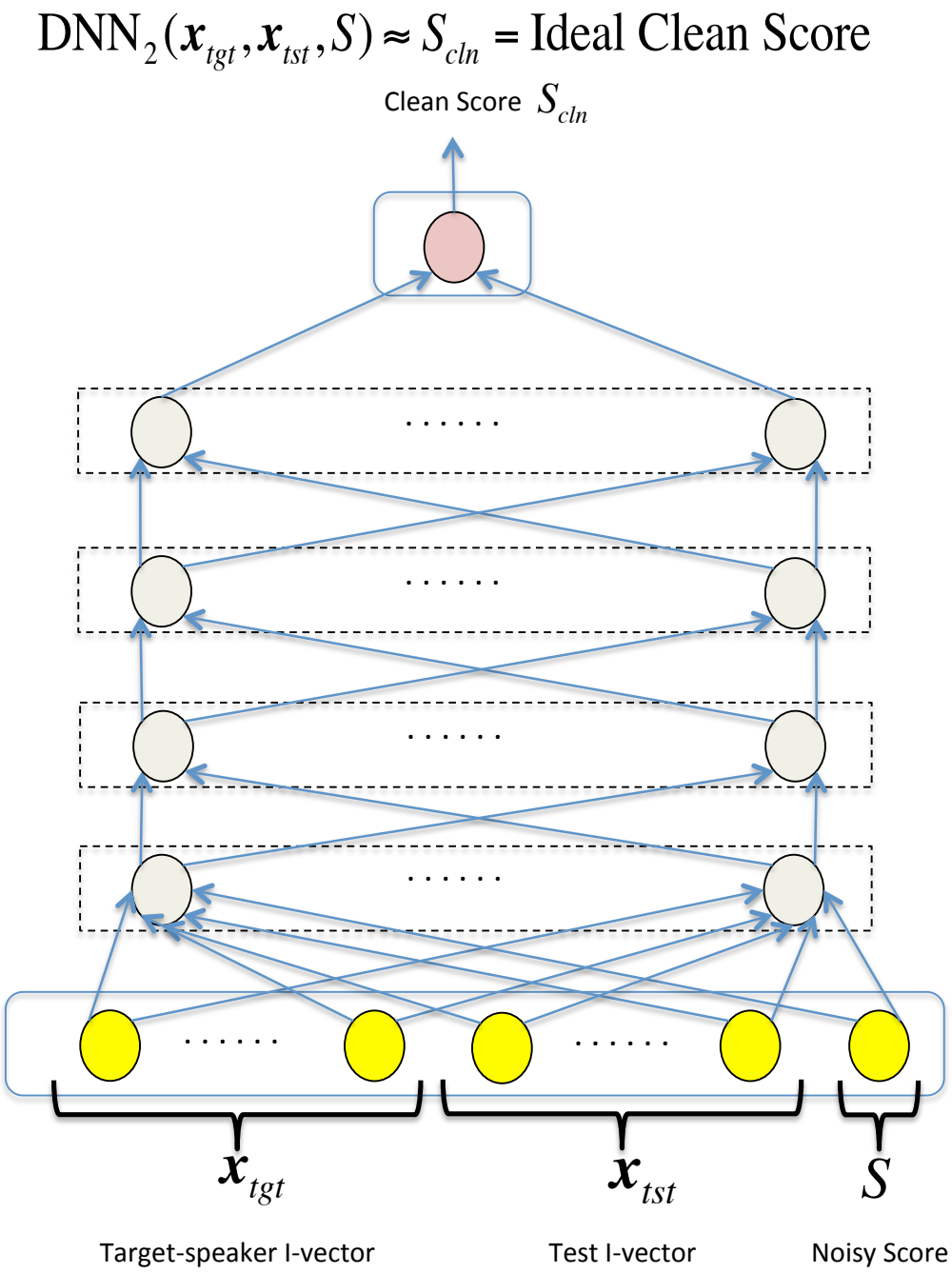

Figure 6.4: Single-task DNN that recovers the clean scores.

6.3.3 Multi-task DNNs for Score Compensation/Transformation

The DNNs in Fig. 6.3 and Fig. 6.4 have hundreds of input nodes but only one output node. Their goal is to learn a regression task to produce the desired score shifts or clean scores. During training, the squared errors in the output node will need to be propagated to hundreds of nodes in both the hidden and input layers. Our experience is that having a single source of errors makes the backpropagation (BP) of error gradients very inefficient. One possible solution to assisting the network to learn the regression task is to introduce some auxiliary tasks for the network to learn. In the literature, this is known as multi-task learning [86,87]. Therefore, a multi-task DNN with auxiliary information in the output layer may help to improve the learning efficiency.

Fig. 6.5 shows a DNN that uses multi-task learning to learn not only the main task (producing score shift δ_{score} and clean score S_{cln}) but also the auxiliary tasks (producing the SNRs of target-speaker and test utterances and same-speaker and different-speaker posteriors). To incorporate the auxiliary information, we may add auxiliary nodes to either the input layer, the output layers, or both. However, we opt for adding the auxiliary nodes to the output layers rather than to the input layer for four reasons:

1. Adding more input nodes will require the error signals from the error source to be propagated to more input nodes, which is contradictory to our goal of easing the BP training.

2. Given the large number of input nodes corresponding to the i-vector pairs, the squared errors in the output node will mainly depend on the i-vectors rather than the small number of auxiliary inputs. As a result, BP training will tend to find a network that is insensitive to the variability in the auxiliary inputs.

3. Adding auxiliary nodes to the input layer means that it is necessary to estimate

this information during the calibration stage. While some information such as SNRs of utterances can be easily estimated, others such as whether the input i-vector pair belongs to the same person or not (see Fig. 6.5) is not that trivial. In fact, the latter is the goal of the application in the first place.

4. Having more output nodes means that more error signals can be propagated to the hidden layers. The error signals from the auxiliary tasks can guide the network to learn the main task [88]. They also serve as a regularizer to avoid overfitting the main task [86].

Both the clean score S_{cln} and ideal score shift δ_{score} can be the target outputs of the multi-task DNN. Once the multi-task DNN has been trained, the calibrations defined in Eq. 6.6 and Eq. 6.10 can be obtained from the output of this DNN. Besides, according to Eq. 6.3, the SNRs of the target-speaker's utterance and the test utterance, SNR_{tgt} and SNR_{tst}, are useful for estimating the score shift. Therefore, we have 4 output nodes in the regression task as shown in Fig. 6.5. In addition to the regression task, a classification task can be added. Because our goal is to verify speakers, two classification output nodes indicating whether the input i-vector pair is from the same speaker or not is added to the network. In this chapter, the regression part of the DNN uses linear output nodes and minimum mean squared error as the optimization criterion, whereas the classification part uses softmax outputs and cross-entropy as the optimziation criterion.

The outputs of a multi-task DNN with 4 regression nodes and 2 classification nodes are the concatenation of two vectors:

$$\begin{aligned} &\mathrm{DNN}_3(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}, S) \approx \\ &\big[\underbrace{[\delta_{score}, S_{cln}, \mathrm{SNR}_{tgt}, \mathrm{SNR}_{tst}]}_{\text{Regression}}, \underbrace{[p^+, p^-]}_{\text{Classification}}\big]^\intercal, \end{aligned} \tag{6.11}$$

where p^+ and p^- are the posterior probabilities of same-speaker and different-speaker

hypotheses, respectively. Similar to the notation in Eq. 6.7, Eq. 6.11 means that the DNN uses the i-vector pair $(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst})$ and the original score S as input. With the multi-task learning strategy, the network outputs the score shift δ_{score}, the clean score S_{cln}, the SNRs of target-speaker speech and test speech SNR_{tst}, and the posterior probabilities (p^+ and p^-).

During score compensation and transformation, only the score shift and clean score produced by the multi-task DNN will be used:

$$\text{DNN}_{3,shift}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}, S) \approx \delta_{score}, \tag{6.12}$$

and

$$\text{DNN}_{3,cln}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}, S) \approx S_{cln}. \tag{6.13}$$

Therefore, we have

$$\begin{aligned} S'_3 &= S + \text{DNN}_{3,shift}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}, S) \\ &\approx S_{cln}, \end{aligned} \tag{6.14}$$

and

$$\begin{aligned} S'_4 &= \text{DNN}_{3,cln}(\boldsymbol{x}_{tgt}, \boldsymbol{x}_{tst}, S) \\ &\approx S_{cln}. \end{aligned} \tag{6.15}$$

6.3.4 Producing Likelihood-Ratio Scores

Our goal is to use DNNs to estimate the ideal clean scores or ideal score shifts in order to improve the performance in terms of equal error rate (EER) and minimum detection cost (minDCF). Therefore, the DNNs are not trained to produce *true* likelihood ratios

$$\mathrm{DNN}_{3,shift}(\boldsymbol{x}_{tgt},\boldsymbol{x}_{tst},S)\approx\delta_{score}=\text{Ideal Score Shift}$$

$$\mathrm{DNN}_{3,cln}(\boldsymbol{x}_{tgt},\boldsymbol{x}_{tst},S)\approx S_{cln}=\text{Ideal Clean Score}$$

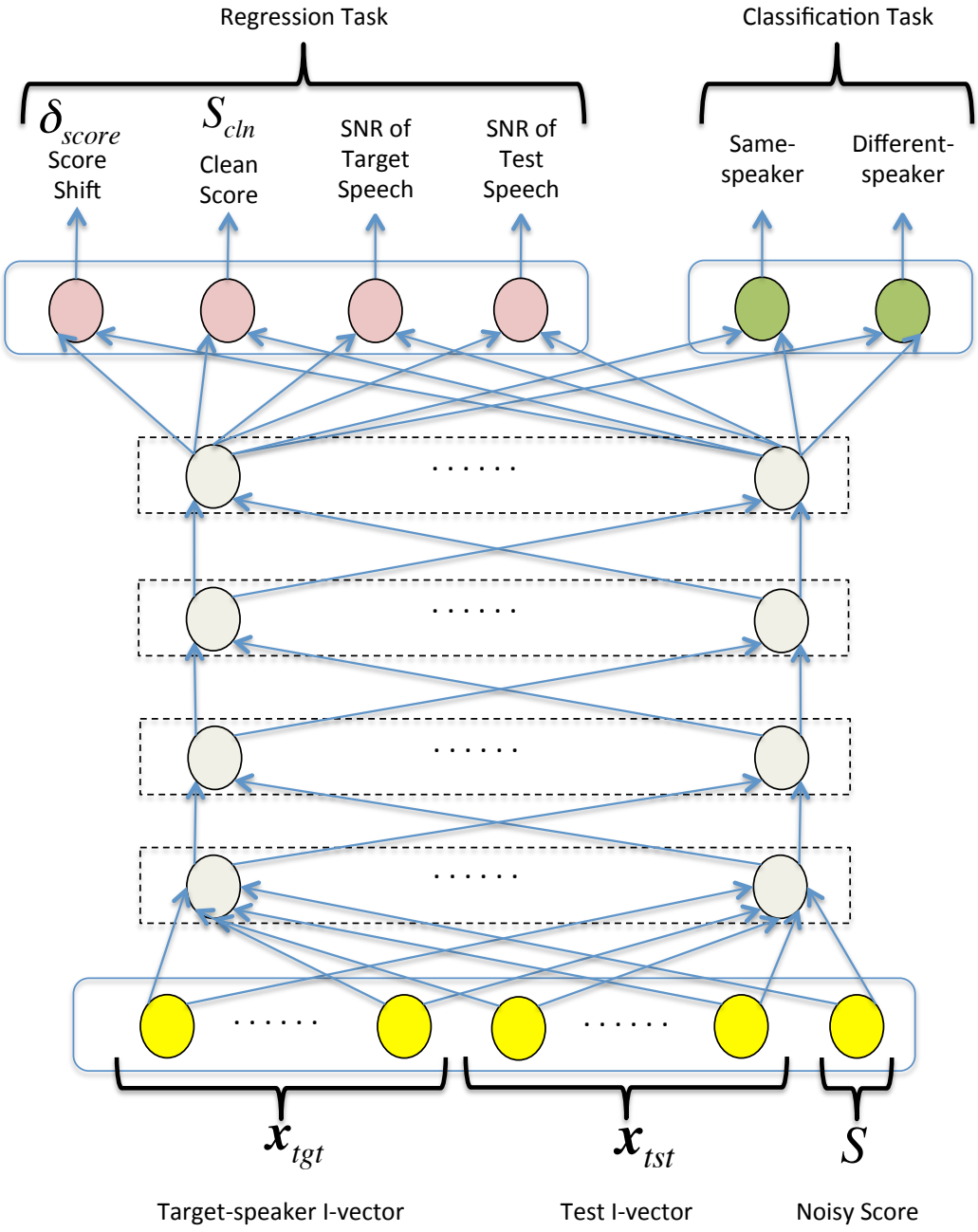

Figure 6.5: Multi-task DNN with classification and regression capacities.

so that the decision thresholds for which these performance metrics are minimized are application dependent.

For real-world deployments, it is desirable to have application-independent decision thresholds [80] such that not only the EER and minDCF are minimized, but also the actual DCF (actDCF) or C_{primary} at specific thresholds are also small. To this end, all of the *compensated/transformed* scores are subject to further calibration to produce true likelihood-ratio scores using the logistic regression (LR) in the Bosaris toolkit [81]:

$$S'' = w_0 + w_1 S'_i, \tag{6.16}$$

where S'_i, $i = 1, 2, 3$, and 4, are the compensated/transformed scores in Eqs. 6.6, 6.10, 6.14 and 6.15, respectively. This calibration step only shifts and scales the DNN calibrated scores, which reduce the actDCF without affecting the EER and minDCF.

6.4 Experiments

6.4.1 Experimental Setup

Score calibration experiments were conducted on the NIST 2012 SRE under Common Condition 4 (CC4, male). This evaluation condition involves 723 male target speakers, with a total of 7,116 telephone utterances for enrollment and 3,900 telephone utterances for performance evaluation. Totally, there are 2,775 target-speaker trials and 122,624 impostor trials. The duration of these utterances ranges from 10 to 300 seconds (before voice activity detection (VAD)). All utterances were spoken in English. These utterances cover a wide range of SNR, from 0dB to 30dB as shown in Fig. 5.4.

To investigate the capability of various calibration methods under noisy environments, we used the FaNT tool [79] to add babble noise to the target-speaker utterances and test utterances at an SNR of 15dB, 6dB, and 0dB, respectively. Therefore, we have four groups of training utterances and four groups of test utterances, with the

first group being the original utterances and the last three groups having SNRs close to 15dB, 6dB, and 0dB, respectively. Hereafter, we refer to these 4 groups as SNR groups. The SNR distributions of the 4 groups of test utterances in CC4 are shown in Fig. 5.4. Note that although the target SNRs that we applied to FaNT are 0dB, 6dB, and 15dB, Fig. 5.4 shows that the peaks of the SNR distributions do not align to these targets. The misalignments are due to the discrepancy in the VAD decisions for adding noise and for measuring SNRs. Specifically, FaNT has its own VAD for estimating the amount of noise to be added to the clean signals, whereas the *measured* SNRs in Fig. 5.4 were based on the voltmeter function in FaNT and the decisions of our noise-robust VAD [72].

The whole training set comprises $7116 \times 4 = 28,464$ target-speaker utterances from 723 target speakers in CC4 of NIST 2012, leading to $28464^2 \approx 810$ million i-vector pairs. Using the procedure shown in Fig. 6.2, these i-vector pairs give rise to 810 million PLDA scores and score shifts. A random subset of these scores and score shifts were selected for training the DNNs in Fig. 6.3 to Fig. 6.5 (see Section 6.4.2) and for estimating the calibration weights in Eqs. 6.1, 6.3 and 6.16. To ensure that all DNNs were trained by the same set of i-vector pairs and PLDA scores, only one PLDA model was trained. Specifically, it was trained by using the utterances from the 4 SNR groups mentioned earlier and the i-vectors derived from the microphone utterances (interview speech) of the same set of target speakers in NIST 2006–2010 SREs.

The evaluation protocol of the NIST 2012 SRE defines the target trials and impostor trials (in the `.ndx` files). For each trial, a target speaker is defined but not his/her enrollment utterances. It is up to the evaluator to select the appropriate enrollment utterances for each trial. Because CC4 in 2012 SRE involves noise contaminated test utterances, we used the original target-speaker utterances and their noise contaminated counterparts from the 6dB and 15dB SNR groups as enrollment utterances. In the sequel, we refer to this test condition as "original". In addition to this "original"

test condition, we created three test conditions based on the noise contaminated test utterances. Specifically, for the 15dB test condition, test utterances at the SNR of 15dB were used for scoring, and similarly for the 6dB and 0dB test conditions. The enrollment utterances were different for different test conditions. Specifically, for the 15dB test condition, the enrollment utterances were obtained from the 15dB and 6dB SNR groups; for the 6dB and 0dB test conditions, the enrollment utterances were respectively obtained from the 6dB and 0dB SNR groups.

The weights of the QMF in Eq. 6.3 and the calibration weights in Eq. 6.1 and Eq. 6.16 were trained by using 1.5 million same-speaker utterance pairs and 400 million different-speaker utterance pairs derived from the target speakers in the four SNR groups. For Eq. 6.1 and Eq. 6.3, using the logistic regression program in the FoCal toolkit, we obtained the weights $w_0 = -21.5197$, $w_1 = 0.1966$, $w_2 = 0.1284$ and $w_3 = 0.1284$, leading to:

$$S' = 0.1966S + 0.1284\text{SNR}_{tst} + 0.1284\text{SNR}_{tgt} - 21.5197.$$

Speech regions in the speech files were extracted by using a two-channel voice activity detector [72]. 19 MFCCs together with energy plus their 1st and 2nd derivatives were extracted from the speech regions, followed by cepstral mean normalization and feature warping with a window size of 3 seconds. A 60-dim acoustic vector was extracted every 10ms, using a Hamming window of 25ms.

6.4.2 DNN Training

To highlight the advantages of multi-task learning, a multi-task DNN (Fig. 6.5) that implements Eq. 6.12 and Eq. 6.13 was compared with two single-task DNNs (Fig. 6.3 and Fig. 6.4) that implement Eq. 6.6 and Eq. 6.10. For the single-task DNNs, we trained restricted Boltzmann machines (RBM) layer-by-layer using the contrastive divergence algorithm [31, 78]; then we fine-tuned the networks using the backpropa-

gation algorithm with sigmoid nonlinearity in the hidden layers. Mini-batch gradient descent with a batch size of 1000 was used. The learning rates for the classification task and regression task are 0.005 and 0.05, respectively. Because there are over 810 million i-vectors pairs that can be used for training, to speed up the training process, 3.2 million pairs were randomly chosen for every 10 iterations of backpropagation training. Totally, we applied 200 iterations of backpropagation to fine-tune the networks.

The RBM at the bottom layer has Gaussian visible nodes and Bernoulli hidden nodes. The remaining RBMs use Bernoulli distributions in both visible and hidden layers. Both the inputs and desired outputs (except for the classification outputs in Fig. 6.5) of the DNNs were processed by z-normalization. The last layer of the classification part in Fig. 6.5 were initialized with small random weights. All DNNs have 4 hidden layers, with each layer comprising 256 hidden nodes.

The multi-task DNN (DNN_3) was trained in a slightly different manner. Because having a balanced training set is beneficial for the network to learn the classification task, for every iteration we extracted 700,000 same-speaker i-vector pairs and 700,000 different-speaker i-vector pairs for training. We applied 300 iterations of backpropagation to fine-tune DNN_3.

6.4.3 Denoised Senone I-vectors

We used a senone i-vector/PLDA system [74] to produce the uncalibrated (noisy) scores, which form our baseline results. The system is equipped with a denoising deep classifier that extracts frame-based bottleneck features from the MFCCs of utterances. The deep classifier is formed by stacking two layers of RBMs on top of a denoising autoencoder (DAE) [75]. This structure allows us to extract bottleneck features and estimate the posterior of senones for i-vector extraction [32], which effectively incorporates phonetic information into the senone i-vectors.

We followed the standard procedure to pre-process the i-vectors for Gaussian

PLDA modeling. Specifically, the 500-dimensional senone i-vectors were whitened by within-class covariance normalization (WCCN) [89], which normalizes the covariance of i-vectors, and length normalization [14], followed by linear discriminant analysis to reduce the dimension to 200 and variance normalization by WCCN [90]. These 200-dimensional i-vectors were input to the PLDA model and the DNNs.

6.5 Results and Disscusions

6.5.1 Distributions of PLDA Scores and Score Shifts

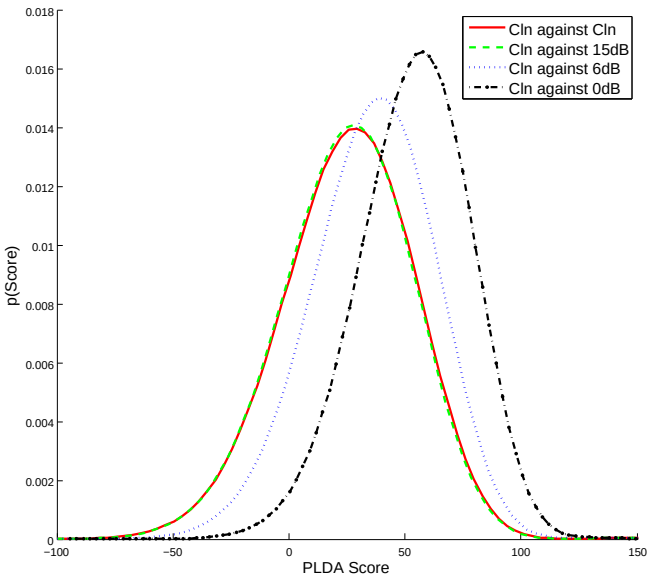

Figure 6.6: The distributions of PLDA scores produced by scoring clean target-speaker i-vectors against noisy test i-vectors. Each distribution corresponds to one group of test i-vectors whose utterances have SNRs belonging to one of the four groups: Clean (Cln), 15dB, 6dB, and 0dB.

To investigate the property of PLDA scores under various background noise levels, we scored clean target-speaker i-vectors against noisy test i-vectors and plotted the distributions of the resulting scores. Fig. 6.6 shows the distributions of these uncalibrated scores under four background noise levels of test utterances: Clean, 15dB,

6dB, and 0dB. Evidently, the scores tend to be larger and their variances tend to be smaller when the background noise level increases.

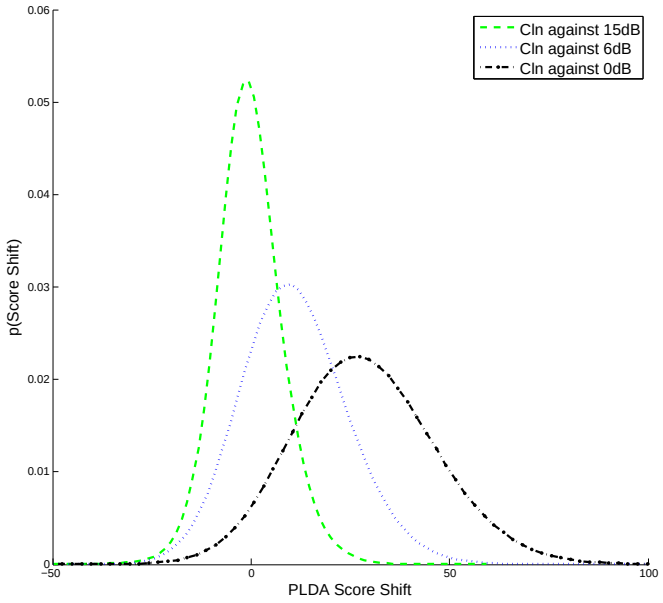

Figure 6.7: The distributions of ideal score shifts ($S - S_{cln}$) for 3 SNR conditions of the test utterances. As indicated in the legend, the target-speaker utterances are always clean. The clean scores S_{cln} were obtained by scoring clean target-speaker i-vectors against clean test i-vectors.

Because our goal is to use DNNs to compute the ideal score shifts, it is of interest to inspect the relationship between the ideal score shifts and test utterances' SNR. To this end, we plot the distributions of ideal score shifts ($S - S_{cln}$) under three SNR conditions for the test utterances in Fig. 6.7 and against all SNRs in the test utterances in Fig. 6.8. Interestingly, the score shifts exhibit a large variability when the SNR of test utterances is very low (0dB). This high variability is definitely not because of the high variability in SNR, as evident in Fig. 5.4 where the SNR distribution of test utterances is very narrow near 0dB. Quite the opposite, the high SNR variability in the 15dB group shown in Fig. 5.4 leads to the least variability in score shifts (green dashed curve) in Fig. 6.7. Therefore, at low SNR, the score shifts will become more difficult to estimate, which demonstrates a major drawback of the methods

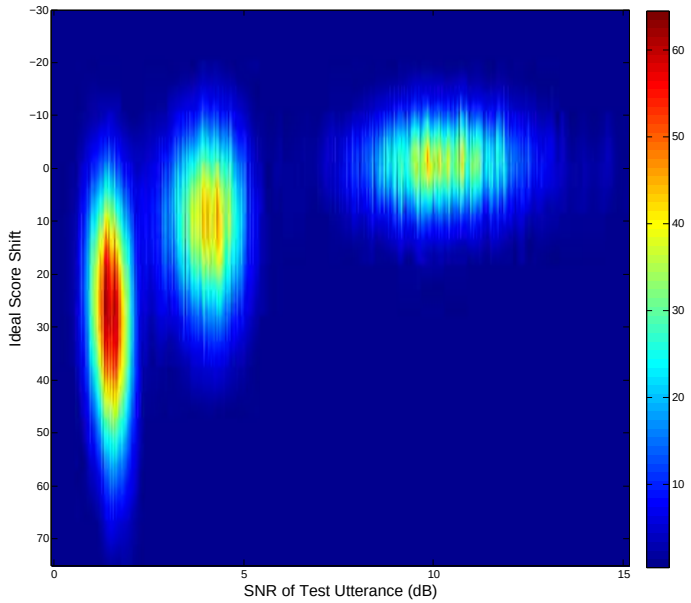

Figure 6.8: The distributions of score shifts with respect to the SNR of test utterances when the target-speaker utterances is clean. The SNRs follow the distribution shown in Fig. 5.4, and the score shifts follow the distribution shown in Fig. 6.7. The figure shows that the relationship between SNRs and score shifts is non-linear, and that at low SNR, the variability of score shifts is very large.

that entirely rely on SNR of utterances (e.g., QMF in Eq. 6.3). In theory, the FQE in Eq. 6.5 is better in the sense that it does not use SNR information directly but instead uses it implicitly through the i-vectors and the Gaussian models. However, whether the bilinear transformation and cosine distance can accurately estimate the score shift at high background noise level is unclear. As demonstrated in Fig. 6.8, the relationship between score shifts and utterances' SNR are fairly complex and definitely non-linear.

Because i-vectors are noise-level dependent [40], it makes sense to directly predict the score shifts from i-vectors rather than implicitly through the Gaussian models of the i-vectors as in FQE. Therefore, we advocate that through multi-task supervised learning, the DNNs can estimate the score shifts accurately and even recover the clean scores. This is supported by the results to be presented next.

6.5.2 Sensitivity of Score Output to Score Input

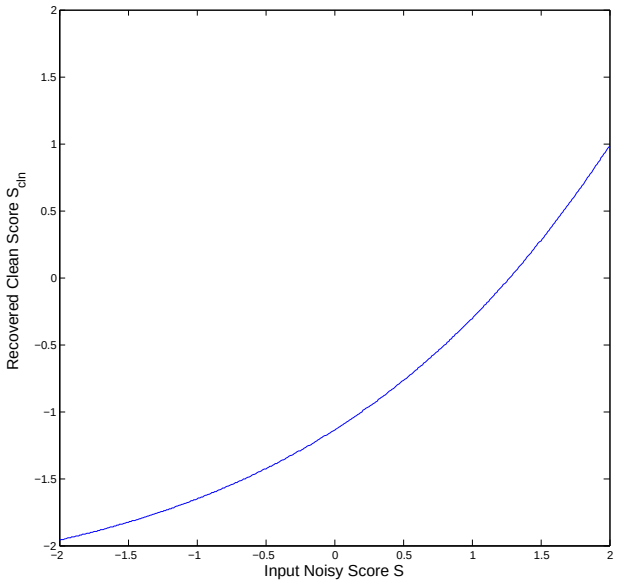

Figure 6.9: Graph showing the relationship between the recovered clean scores S'_4 in Eq. 6.15 produced by a multi-task DNN and the noisy input scores S when the i-vector pair is fixed. The input i-vector pair is a fixed non-target pair. Both S and S'_4 were normalized by the same set of z-norm parameters. The DNN is a multi-task one with 4 hidden layers.

For 200-dimensional pre-processed i-vectors, the number of input nodes corresponding to the i-vector pairs is 400, whereas there is only one score input node. This large ratio may cause the network insensitive to the noisy scores. To find out if it is the case, we plotted the recovered clean scores S'_4 (Eq. 6.15) produced by a multi-task DNN against the input noisy scores S. To focus on the sensitivity of S'_4 with respect to S, the i-vector pair was fixed to an arbitrary non-target pair. Surprisingly, as shown in Fig. 6.9, the recovered scores are fairly sensitive to the noisy input scores despite of the large ratio between the two types of input nodes. This result, in fact, agrees with our recent finding that the input noisy scores play an important role in recovering the clean scores [82].

6.5.3 Comparing Various Calibration Methods

Table 6.1: Performance of various score calibration methods in NIST 2012 SRE (CC4, male speaker, core task) with test utterances contaminated with different levels of babble noise. All networks have 4 hidden layers. LR: Logistic regression.

	Original			15dB		
Score Calibration Method	EER(%)	minDCF	actDCF	EER(%)	minDCF	actDCF
Baseline (LR on noisy PLDA scores)	1.56	0.218	0.855	2.27	0.225	0.778
SNR-dep Score Shift (Eq. 6.3)	1.68	0.209	0.780	2.24	0.215	0.770
Score Shift by Multi-task DNN (Eq. 6.12)	1.65	**0.178**	0.694	2.32	**0.203**	0.626
Recovered Clean Score by Multi-task DNN (Eq. 6.13)	**1.50**	0.189	**0.517**	**2.21**	0.211	**0.455**
	6dB			0dB		
Score Calibration Method	EER(%)	minDCF	actDCF	EER(%)	minDCF	actDCF
Baseline (LR on noisy PLDA scores)	2.29	0.276	0.749	5.37	0.753	0.779
SNR-dep Score Shift (Eq. 6.3)	2.28	0.269	0.811	5.35	0.754	0.794
Score Shift by Multi-task DNN (Eq. 6.12)	2.34	**0.231**	0.604	4.00	0.488	0.547
Recovered Clean Score by Multi-task DNN (Eq. 6.13)	**2.16**	0.243	**0.470**	**3.48**	**0.409**	**0.516**

Table 6.1 shows the performance of various score calibration strategies, including SNR-dependent score shifts (Eq. 6.3) and recovering clean scores by multi-task DNNs (Eq. 6.13). The baseline refers to using the noisy PLDA scores for computing EER and minimum detection cost function (minDCF) and using logistic regression for computing the actual DCF (actDCF). The results show that the proposed method achieves the best performance across all of the SNR levels. At 0dB, it also outperforms the baseline significantly.

Fig. 6.10 shows the normalized Bayes error rates of the minDCF and actDCF of various systems as a function of effective target prior. Among all systems, the one based on score shift computed by the multi-task DNN (green) has a very small margin between the actDCF and minDCF.

The results of the same experiments on CC5 of NIST 2012 SRE are shown in Table 6.2. Unlike the results in CC4, DNN calibration does not show obvious advantage as compared to linear calibration. We suspect that this is because most of the utterances in CC5 have higher SNRs than those in CC4 (see Fig. 5.4). When the SNR is high, the benefit of DNN score calibration diminishes. Also, since the utterances in

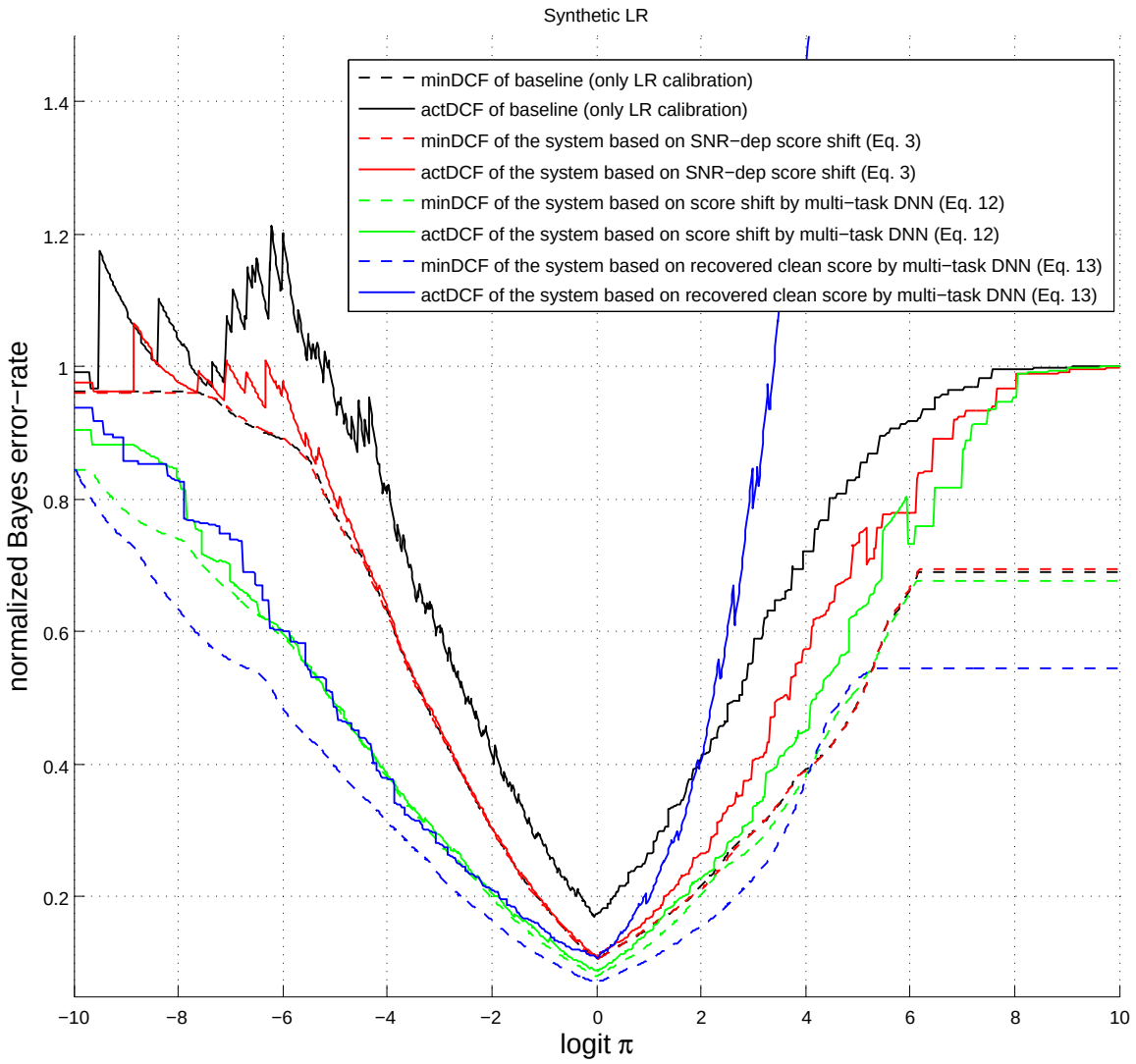

Figure 6.10: Normalized bayes error rate plots showing that the minDCF and actDCF of different systems as a function of effective target prior in NIST 2012 SRE (CC4, male speaker, core task) contaminated with babble noise at an SNR of 0dB.

Table 6.2: Performance of various score calibration methods in NIST 2012 SRE (CC5, male speaker, core task). The DNN has 4 hidden layers. LR: Logistic regression.

	Original		
Score Calibration Method	EER(%)	minDCF	actDCF
Baseline (LR on noisy PLDA scores)	**2.48**	0.267	0.861
Score Shift by Multi-task DNN (Eq. 6.12)	2.54	0.260	**0.640**
Recovered Clean Score by Multi-task DNN (Eq. 6.13)	2.51	**0.242**	0.716

CC5 were collected in a noisy environment while the DNN was trained by data with artificially added noise, the DNN may be weak to deal with the natural noises and the Lombard effect.

6.5.4 Single-task vs. Multi-task

Table 6.3: Performance of single-task and multi-task DNNs for estimating the score shifts and recovering the clean score in NIST 2012 SRE (CC4, male speaker, core task) with test utterances contaminated with different levels of babble noise. All networks have 4 hidden layers.

	Original		15dB		6dB		0dB	
Score Calibration Method	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
Score Shift by Single-task DNN (Eq. 6.7)	2.26	0.267	3.25	0.285	3.27	0.322	4.83	0.538
Score Shift by Multi-task DNN (Eq. 6.12)	**1.65**	**0.178**	**2.32**	**0.203**	**2.34**	**0.231**	**4.00**	**0.483**
Recovered Clean Score by Single-task DNN (Eq. 6.10)	7.35	0.693	8.11	0.643	8.06	0.733	12.25	0.989
Recovered Clean Score by Multi-task DNN (Eq. 6.13)	**1.50**	**0.189**	**2.21**	**0.211**	**2.16**	**0.248**	**3.48**	**0.409**

Table 6.3 compares the performance between single-task DNNs and multi-task DNNs. Regardless of which output to be used (the score shift output in Eq. 6.6 or the direct score output in Eq. 6.10), the multi-task DNN performs much better than the single-task DNN.

A comparison between the first row of Table 6.3 (Score Shift by Single-Task DNN)

and the first and second rows of Table 6.1 reveals that only under very noisy conditions (0dB), the calibrated scores produced by the single-task DNN (Eq. 6.7) can improve performance; in all other conditions, this approach performs even poorer than the ones without score calibration or the conventional approach where the score shift is linear with respect to utterances' SNR. This result suggests that estimating the score shifts *entirely* from the i-vector pairs and noisy scores (Eq. 6.7) is not a good idea. Under the clean condition, the score shifts estimated by the DNN in Eq. 6.7 have detrimental effect on the uncalibrated scores. However, the good performance in the second row of Table 6.3 suggests that once the auxiliary information is added to the network, the score shifts estimated by the multi-task DNN become very close to the ideal ones.

Table 6.3 also allows us to compare the performance of using the DNN to estimate the score shift (δ_{score}, Fig. 6.3) and using the DNN to recover the clean scores (S_{cln}, Fig. 6.4). Specifically, for the single-task case, Row 1 and Row 3 of Table 6.3 show that the scores recovered by the DNNs are so wrong that the error is 3 to 4 times that of the baseline (c.f. Table 6.1 and the third row of Table 6.3), while the score shifts by the DNNs are comparable to the baseline (c.f. Table 6.1 and the first row of Table 6.3). For the multi-task case, Row 2 and Row 4 of Table 6.3 suggest that it is better to recover the clean scores directly than to estimate the score shifts, provided that multi-task learning is used to train the DNNs.

Fig. 6.11 compares the DET performance of the baseline against the multi-task DNNs. The results clearly suggest that recovering the clean scores by DNNs leads to superior performance across a wide range of decision thresholds.

6.5.5 Generalization Capability

Using the i-vectors from target speakers to train the PLDA model and the calibration DNNs may lead to overfitting on target speakers. In NIST 2012 SRE, the enrollment utterances of target speakers come from previous NIST SREs. As these utterances

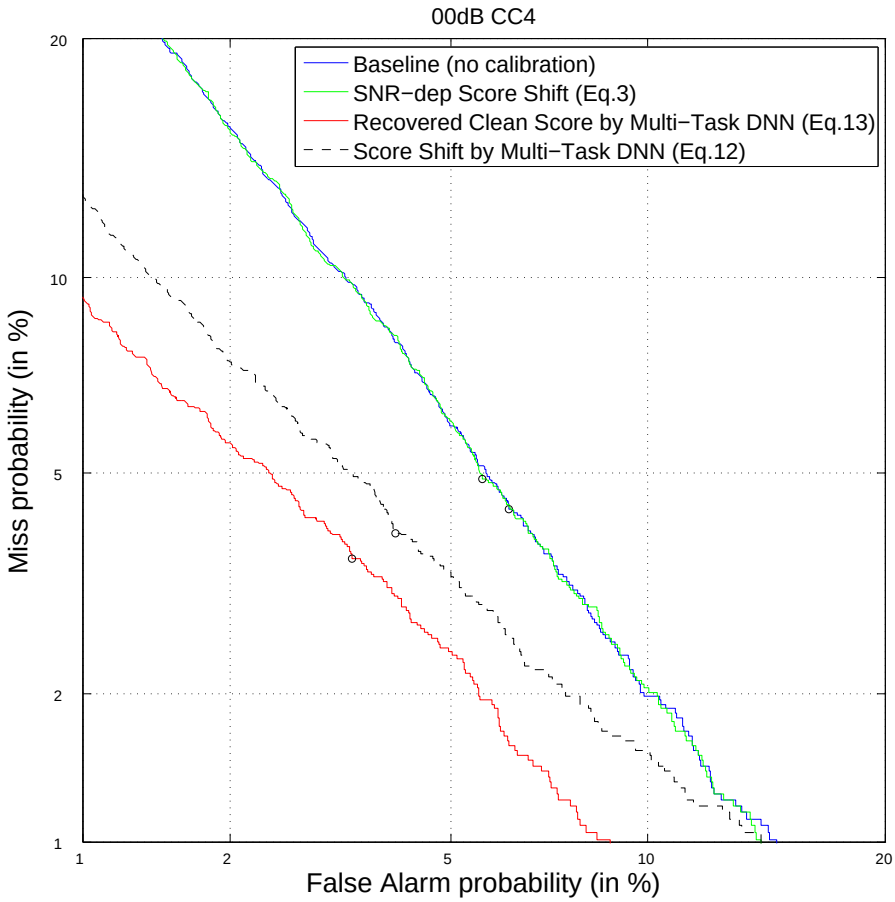

Figure 6.11: The DET curves of the four systems in NIST 2012 SRE (CC4, male speaker, core task). Test utterances were contaminated with babble noise at an SNR of 0dB.

Table 6.4: Performance of various score calibration methods in a subset of NIST 2012 SRE (CC4, male speaker, core task) with test utterances contaminated with different levels of babble noise. The speech from 500 speakers was used to train the multi-task DNN as in Fig. 6.5, and the 38,820 trials in CC4 of the other 223 speakers were used in the test trials. The DNN has 4 hidden layers. LR: Logistic regression.

	Original			15dB		
Score Calibration Method	EER(%)	minDCF	actDCF	EER(%)	minDCF	actDCF
Baseline (LR on noisy PLDA scores)	1.21	0.194	0.867	1.96	0.214	0.794
Score Shift by Multi-task DNN (Eq. 6.12)	0.94	**0.168**	**0.751**	1.96	**0.206**	**0.698**
Recovered Clean Score by Multi-task DNN (Eq. 6.13)	**0.82**	0.188	0.777	**1.57**	0.214	0.726
	6dB			0dB		
Score Calibration Method	EER(%)	minDCF	actDCF	EER(%)	minDCF	actDCF
Baseline (LR on noisy PLDA scores)	1.89	**0.266**	0.755	5.09	0.709	0.722
Score Shift by Multi-task DNN (Eq. 6.12)	1.65	0.274	**0.673**	**3.67**	**0.507**	**0.634**
Recovered Clean Score by Multi-task DNN (Eq. 6.13)	**1.55**	0.309	0.728	3.70	0.548	0.704

were also used for training the PLDA model and the DNN, there may be chance that they can only work for these target speakers. To demonstrate the generalization capability of the multi-task DNNs, we selected 500 target speakers from CC4 and used their i-vectors and PLDA scores (scores between same targets and between different targets) to train a multi-task DNN as in Fig. 6.5. The target and non-target trials (totally 38,820) derived from the remaining 223 target-speakers were then used for performance evaluation. To the PLDA model and the DNN, these 223 speakers are unseen speakers. As shown in Table 6.4, the multi-task DNNs perform significantly better than the baseline. More importantly, the performance in Table 6.4 is comparable to and in many cases better than that in Table 6.1. This suggests that both the PLDA model and DNN can generalize to unseen speakers and that using the enrollment utterances for training does not lead to overfitting.

6.5.6 Different Numbers of Hidden Layers

Table 6.5: Performance of single-task and multi-task DNNs with different numbers of hidden layers for estimating the score shifts (Eq. 6.7 and 6.12) in NIST 2012 SRE (CC4, male speaker, core task). The test utterances were contaminated with different levels of babble noise.

		Original		15dB		6dB		0dB	
Network Type	No. of Hidden Layers	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
Single-task	1	2.39	0.322	3.49	0.342	3.65	0.413	5.75	0.676
	2	**2.14**	**0.264**	**3.21**	0.298	**3.18**	0.339	4.87	0.575
	3	2.50	0.296	3.70	0.330	3.79	0.398	5.68	0.632
	4	2.26	0.267	3.25	**0.285**	3.27	**0.322**	**4.83**	**0.538**
Multi-task	1	**1.55**	0.202	2.16	0.208	2.21	0.242	4.49	0.624
	2	**1.55**	0.187	2.17	**0.197**	2.14	**0.228**	4.04	0.554
	3	**1.55**	0.193	**2.11**	0.201	**2.08**	0.239	4.20	0.571
	4	1.65	**0.178**	2.32	0.203	2.34	0.231	**4.00**	**0.488**

All of the DNNs in Tables 6.1 and 6.3 comprise four hidden layers. It is of interest to see how they perform if the number of hidden layers varies. Table 6.5

Table 6.6: Performance of single-task and multi-task DNNs with different numbers of hidden layers for recovering the direct clean scores (Eq. 6.10 and 6.13) in NIST 2012 SRE (CC4, male speaker, core task). The test utterances were contaminated with different levels of babble noise.

		Original		15dB		6dB		0dB	
Network Type	No. of Hidden Layers	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
Single-task	1	**3.79**	**0.634**	**4.89**	**0.580**	**5.10**	**0.679**	**9.09**	**0.957**
	2	5.02	0.719	5.93	0.675	6.38	0.750	10.81	0.980
	3	7.81	0.779	8.86	0.741	9.52	0.818	14.27	0.997
	4	14.61	0.910	15.41	0.881	17.05	0.942	21.44	1.002
Multi-task	1	1.50	0.234	**2.08**	0.222	**2.11**	0.278	4.48	0.605
	2	**1.48**	0.199	2.37	**0.210**	2.14	**0.241**	3.83	0.502
	3	1.71	**0.187**	2.53	0.213	2.34	0.243	3.58	0.448
	4	1.50	0.189	2.21	0.211	2.16	0.248	**3.48**	**0.409**

and Table 6.6 show the performance of single-task and multi-task DNNs with different number of hidden layers for estimating the score shifts and recovering the clean scores, respectively. Theoretically, a network with a deeper structure should posses a larger capacity to perform the mapping task. However, the results do not show such trend, especially for the single-task DNNs in Table 6.5 where they were used for estimating the score shifts. In particular, the single-task DNN with 3 hidden layers performs worse than those with 2 and 4 hidden layers. In Table 6.6, the performance of single-task DNNs (for recovering clean scores) becomes worse when the number of hidden layers increases. But this phenomenon does not occur in the multi-task DNNs. Further investigations on the error profiles during the BP training process reveal that the training errors of single tasks DNNs increase when the number of hidden layers increases. This contradicts to the common belief that networks with a higher capacity should produce a lower training error. One possible cause of this contradiction is that the networks are stuck at bad local minima. The multi-task DNNs do not suffer from this problem, primarily because the auxiliary output nodes can introduce *additional* errors to help the network to learn the main task [86].

As discussed in [91], when a DNN becomes deeper, its accuracy may get saturated;

further increase in the depth will lead to performance degradation. This phenomenon can also be observed in Table 6.6. In our case, this phenomenon is not caused by overfitting, because adding more layers leads to higher training error; instead, vanishing gradients and slow convergence are more likely to be the cause. One possible way to alleviate this difficulty is to use residual networks [91]. A special property of residual networks is that they are trained to produce the difference between the desired mapping function $f(\mathbf{x})$ and the input $\mathbf{x}$ instead of producing $f(\mathbf{x})$. With this special arrangement, networks can enjoy performance gain from increased depth. Eq. 6.7 is similar to the residual function in [91], where the score shift δ_{score} is indeed the residual between the desired clean score S_{cln} and the input noisy score S. This explains why the DNNs that estimate score shifts (Fig. 6.3) are easier to train than the ones that recover clean scores (Fig. 6.4).

Chapter 7

CONCLUSIONS AND FUTURE WORKS

7.1 Conclusions

Chapter 4 has shown that Log-mel BN features from denoising deep classifier are noise robust under low SNR environments. The noise robustness is mainly attributed to the autoencoder's denoising ability, which facilitates the upper hidden layers of the DNN to extract more noise robust bottleneck features. The BN features are comparable with the standard MFCC, and they are complementary to each other, leading to significant performance gain after fusing the MFCC- and BN-based PLDA scores.

In the experiment, we did not use a contextual window for the 256-dimensional Log-spec input, since the input dimension is very high when compared with other types of input. Even without contextual window, the performance of the Log-spec BN features is still comparable with that of the Log-mel ones, but the performance under low SNR conditions drops significantly.

The poor performance of the MFC BN features is surprising. The reason is possibly that the distribution of the MFCC is more complex than a single Gaussian (otherwise we do not need GMM for speaker recognition), and therefore the Gaussian-Bernoulli RBM cannot model the input patterns properly. Some other preprocessing techniques, e.g. the feature warping [67], deserve a try for the MFCC input.

Our preliminary experiment is done on the YOHO corpus, whose speech contents do not have much phonetic variation. In future work, experiments on NIST datasets are necessary.

In **Chapter 5**, we demonstrated that robust BN features and frame posteriors can be obtained from a denoising autoencoder–deep neural network (DAE–DNN) formed by the combination of a denoising autoencoders (DAE) and a deep neural network (DNN). The DAE provides a good initial condition for the backpropagation to find a DNN that can suppress noise in MFCC vectors and enforces the frame alignments to respect the phonetic context of input speech. No matter under the GMM i-vector or the senone i-vector frameworks, the phonetically discriminative BN features outperform MFCCs in the speaker verification tasks, which suggests that the phonetically discriminative BN features still contain speaker information. Furthermore, we demonstrated that the denoising capability of the DAE is beneficial to the BN features in senone i-vectors but not to the denoised-MFCC-based senone i-vectors.

By comparing different combinations of BN features and senone posteriors (with and without DAE training), we also validated that the DAE training is more beneficial for BN features extraction than for senone posteriors estimation. It was found that the BN features extracted from DAE–DNN is rich in speaker information, while the DNN without DAE training is good at estimating the speaker-independent senone posteriors. The DAE training helps DAE–DNN to keep speaker information until the fifth layer in DAE–DNN, while DNN without DAE training can estimate speaker-independent senone posteriors more accurately, but loses more speaker information because no part of the network is trained to keep speaker information.

Overall speaking, our experiment results show that the demonised senone I-vectors whose BN features and senone posteriors are both extracted from a DAE–DNN are comparable with the one whose senone posteriors are estimated by a DNN, but the system involves only one neural network and thus has the advantage in accelerating the extraction of i-vectors and system implementation.

In **Chapter 6**, we proposed several DNN-based score calibration algorithms, where the calibrated scores and score shifts are estimated from the i-vector pairs of verification trials. The conventional calibration methods such as quality measure

functions (QMF) assume that score shift caused by background noise is a linear function of the utterance's SNR. Our results, however, suggest that the score shift is nonlinearly related to SNR. Also, QMFs are deterministic functions of SNR in that when the SNR is fixed, the score shift will also be fixed. Our results, however, show that the lower the SNR, the larger the variability in the ideal score shift, which suggests that SNR alone is not adequate for estimating the score shift. These observations motivate us to use more flexible models such as DNNs to model the complex relationship between the i-vector pairs, uncalibrated scores, and score shifts. This chapter has shown that DNNs are flexible enough for recovering the clean PLDA scores directly, allowing us to skip the score-shift estimation entirely. By introducing auxiliary tasks to the DNNs through multi-task learning, we demonstrated that the resulting DNNs can learn the main task better, which is supported by their superior performance across a wide-range of SNRs.

Overall speaking, DNNs can be integrated into different processing stages of a speaker verification system, including i-vector extraction (front-end) and PLDA modeling (back-end). For the i-vector front-end, the discriminative training helps the DNNs to suppress the environmental noise in the frame-based acoustic features and to integrate the speaker and phonetic information into i-vectors. For the PLDA back-end, the calibration DNNs are able to suppress the noise variabilities in the distorted i-vector pairs and recover the ideal clean scores.

7.2 Future Work

Despite the promising results, the proposed score calibration methods in Chapter 6 have some weaknesses. First, the methods are more computationally demanding than the conventional ones because of the DNNs. However, the extra computation time is insignificant when compared with the time for i-vector extraction. This is especially the case for long test utterances as the complexity of i-vector extraction is proportional

to the number of speech frames in the utterances. Second, the DNNs are not trained to produce true likelihood ratios; therefore further calibration is essential. Third, the DNNs require clean and noisy utterance pairs for training, whereas conventional methods such as QMFs and FQEs do not have this requirement.

The multi-task DNNs have five auxiliary output nodes, three for the SNR information and two for the speaker information. The current study did not investigate which of the auxiliary information is more important or helpful. Also, there may be other auxiliary information, such as the duration of utterances, that is also useful. These are the interesting directions that are worth investigation in the future.

In this work, both of the training and testing datasets cover a wide range of SNR, meaning that the DNNs were trained to handle test utterances with different SNRs. Because the SNR ranges are the same for both training and testing, the behaviour of the DNNs under unseen SNR is unclear. In future work, it is of interest to use one SNR group for training a DNN and test it on anther SNR group to investigate the generalization capability of the DNN under SNR mismatch conditions. Currently, utterances were contaminated with babble noise only. The robustness of DNN-based calibration for other types of noise and for reverberated speech also are worth investigation.

We adopted the DNN training procedure in [68]. Over the years, a number of advanced training procedures have been developed. For example, the RBM pre-training can be replaced by discriminative pre-training [92]. The stochastic gradient descent can also be enhanced by using adaptive moment estimation (Adam) [93]. It is also possible to replace the sigmoid activation by other nonlinear functions, such as hyperbolic tangent and softsign, to avoid saturation at the top hidden layer [94] or to achieve better performance [95]. Furthermore, it has been found that rectifier nonlinearity, such as ReLU, is beneficial for acoustic modeling [96]. Training can be sped up by using batch normalization [97]. Deeper networks can be trained by using the strategy in residual networks [91]. These modern DNN training methods

can potentially improve the performance of the DNN-score calibration techniques proposed in this chapter.

BIBLIOGRAPHY

[1] D. A. Reynolds, "An overview of automatic speaker recognition technology," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2002, vol. 4, pp. 4072–4075.

[2] J. H. L. Hansen and V. Varadarajan, "Analysis and compensation of Lombard speech across noise type and levels with application to in-set/out-of-set speaker recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 2, pp. 366–378, 2009.

[3] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[4] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[6] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digital Signal Processing*, vol. 10, no. 1, pp. 19 – 41, 2000.

[7] M. W. Mak and J. T. Chien, *Machine Learning for Speaker Recognition*, Cambridge University Press, 2019.

[8] W. M. Campbell, D. E. Sturim, and D. A. Reynolds, "Support vector machines using GMM supervectors for speaker verification," *IEEE Signal Processing Letters*, vol. 13, no. 5, pp. 308–311, 2006.

[9] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2011.

[10] M. W. Mak, "Lecture Notes on Factor Analysis and I-Vectors," Tech. Rep., Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, 2017, Accessed: 2018-01-11.

[11] P. Kenny, "Joint factor analysis of speaker and session variability: Theory and algorithms," *CRIM, Montreal,(Report) CRIM-06/08-13*, 2005.

[12] S. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *IEEE International Conference on Computer Vision (ICCV)*, 2007, pp. 1–8.

[13] P. Kenny, "Bayesian speaker verification with heavy-tailed priors," in *Proc. Odyssey*, 2010.

[14] D. Garcia-Romero and C. Y. Espy-Wilson, "Analysis of i-vector length normalization in speaker recognition systems," in *Proc. Interspeech*, 2011, pp. 249–252.

[15] Laurens van der Maaten and Geoffrey Hinton, "Visualizing data using t-SNE," *The Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[16] T. Hasan and J. H. L. Hansen, "Acoustic factor analysis for robust speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 4, pp. 842–853, 2013.

[17] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[18] J. H. Liu, W. Q. Zheng, and Y. X. Zou, "A robust acoustic feature extraction approach based on stacked denoising autoencoder," in *IEEE International Conference on Multimedia Big Data (BigMM)*, 2015, pp. 124–127.

[19] H. Yamamoto and T. Koshinaka, "Denoising autoencoder-based speaker feature restoration for utterances of short duration," in *Proc. Interspeech*, 2015, pp. 1052–1056.

[20] T. Pekhovsky, S. Novoselov, A. Sholohov, and O. Kudashev, "On autoencoders in the i-vector space for speaker recognition," in *Proc. Odyssey*, 2016, pp. 217–224.

[21] S. Novoselov, T. Pekhovsky, O. Kudashev, V. Mendelev, and A. Prudnikov, "Non-linear PLDA for i-vector speaker verification," in *Proc. Interspeech*, 2015, pp. 214–218.

[22] H. Xing and J. H. L. Hansen, "Single sideband frequency offset estimation and correction for quality enhancement and speaker recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 1, pp. 124–136, 2017.

[23] R. Saeidi, P. Alku, and T. Backstrom, "Feature extraction using power-law adjusted linear prediction with application to speaker recognition under severe vocal effort mismatch," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 1, pp. 42–53, 2016.

[24] W. Geng, J. Li, S. Zhang, X. Cai, and B. Xu, "Multilingual tandem bottleneck feature for language identification," in *Proc. Interspeech*, 2015, pp. 413–417.

[25] Y. Tian, M. Cai, L. He, and J. Liu, "Investigation of bottleneck features and multilingual deep neural networks for speaker verification," in *Proc. Interspeech*, 2015, pp. 1151–1155.

[26] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543–565, 2012.

[27] C. Yu, A. Ogawa, M. Delcroix, T. Yoshioka, T. Nakatani, and J. H. L. Hansen, "Robust i-vector extraction for neural network adaptation in noisy environment," in *Proc. Interspeech*, 2015, pp. 2854–2857.

[28] F. Richardson, D. A. Reynolds, and N. Dehak, "Deep neural network approaches to speaker and language recognition," *IEEE Signal Processing Letters*, vol. 22, no. 10, pp. 1671–1675, 2015.

[29] A. K. Sarkar, C.-T. Do, V.-B. Le, and C. Barras, "Combination of cepstral and phonetically discriminative features for speaker verification," *IEEE Signal Processing Letters*, vol. 21, no. 9, pp. 1040–1044, 2014.

[30] W. M. Campbell, "Using deep belief networks for vector-based speaker recognition," in *Proc. Interspeech*, 2014, pp. 676–680.

[31] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[32] Y. Lei, N. Scheffer, L. Ferrer, and M. McLaren, "A novel scheme for speaker recognition using a phonetically-aware deep neural network," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 1695–1699.

[33] L. Ferrer, Y. Lei, M. McLaren, and N. Scheffer, "Study of senone-based deep neural network approaches for spoken language recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 1, pp. 105–116, 2016.

[34] D. Garcia-Romero and A. McCree, "Insights into deep neural networks for speaker recognition," in *Proc. Interspeech*, 2015, pp. 1141–1145.

[35] S. Cumani and P. Laface, "Nonlinear i-vector transformations for PLDA-based speaker recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 4, pp. 908–919, 2017.

[36] T. Hasan, S. O. Sadjadi, G. Liu, N. Shokouhi, H. Bořil, and J. H. L. Hansen, "CRSS systems for 2012 NIST speaker recognition evaluation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 6783–6787.

[37] D. Garcia-Romero, X. Zhou, and C. Y Espy-Wilson, "Multicondition training of Gaussian PLDA models in i-vector space for noise and reverberation robust speaker recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 4257–4260.

[38] N. Li and M. W. Mak, "SNR-invariant PLDA modeling in nonparametric subspace for robust speaker verification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 10, pp. 1648–1659, 2015.

[39] N. Li and M. W. Mak, "SNR-invariant PLDA with multiple speaker subspaces," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 5565–5569.

[40] M. W. Mak, X. M. Pang, and J. T. Chien, "Mixture of PLDA for noise robust i-vector speaker verification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 1, pp. 130–142, 2016.

[41] N. Li, M. W. Mak, and J. T. Chien, "DNN-driven mixture of PLDA for robust speaker verification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. PP, no. 99, pp. 1371–1383, 2017.

[42] T. Hasan, R. Saeidi, J. H. L. Hansen, and D. A. Van Leeuwen, "Duration mismatch compensation for i-vector based speaker recognition systems," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 7663–7667.

[43] Q. Hong, L. Li, M. Li, L. Huang, L. Wan, and J. Zhang, "Modified-prior PLDA and score calibration for duration mismatch compensation in speaker recognition system," in *Proc. Interspeech*, 2015.

[44] A. Shulipa, S. Novoselov, and Y. Matveev, "Scores calibration in speaker recognition systems," in *International Conference on Speech and Computer*, 2016, pp. 596–603.

[45] M. I. Mandasari, R. Saeidi, M. McLaren, and D. A. Van Leeuwen, "Quality measure functions for calibration of speaker recognition systems in various duration conditions," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 11, pp. 2425–2438, 2013.

[46] M. I. Mandasari, R. Saeidi, and D. A. Van Leeuwen, "Quality measures based calibration with duration and noise dependency for speaker recognition," *Speech Communication*, vol. 72, pp. 126 – 137, 2015.

[47] A. Nautsch, R. Saeidi, C. Rathgeb, and C. Busch, "Robustness of quality-based score calibration of speaker recognition systems with respect to low-SNR and short-duration conditions," in *Proc. Odyssey*, 2016, pp. 358–365.

[48] A. Ortega J. Villalba, A. Miguel and E. Lleida, "Bayesian networks to model the variability of speaker verification scores in adverse environments," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 12, pp. 2327–2340, 2016.

[49] L. Ferrer, L. Burget, O. Plchot, and N. Scheffer, "A unified approach for audio characterization and its application to speaker recognition," in *Proc. Odyssey*, 2012, pp. 317–323.

[50] O. Ghahabi and J. Hernando, "Deep learning backend for single and multisession i-vector speaker recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 4, pp. 807–817, 2017.

[51] Z. Tang, L. Li, D. Wang, and R. Vipperla, "Collaborative joint training with multitask recurrent model for speech and speaker recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 3, pp. 493–504, 2017.

[52] S. Cumani and P. Laface, "Large-scale training of pairwise support vector machines for speaker recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 11, pp. 1590–1600, 2014.

[53] D. Snyder, P. Ghahremani, D. Povey, D. Garcia-Romero, Y. Carmiel, and S. Khudanpur, "Deep neural network-based speaker embeddings for end-to-end speaker verification," in *Spoken Language Technology Workshop (SLT), 2016 IEEE*. IEEE, 2016, pp. 165–170.

[54] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[55] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[56] H. Ze, A. Senior, and M. Schuster, "Statistical parametric speech synthesis using deep neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 7962–7966.

[57] Z. H. Ling, S. Y. Kang, H. Zen, A. Senior, M. Schuster, X. J. Qian, H. M. Meng, and L. Deng, "Deep learning for acoustic modeling in parametric speech generation: A systematic review of existing techniques and future trends," *Signal Processing Magazine, IEEE*, vol. 32, no. 3, pp. 35–52, 2015.

[58] P. Hamel and D. Eck, "Learning features from music audio with deep belief networks," in *ISMIR*, 2010, pp. 339–344.

[59] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4960–4964.

[60] T. Sercu, C. Puhrsch, B. Kingsbury, and Y. LeCun, "Very deep multilingual convolutional neural networks for LVCSR," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4955–4959.

[61] J. Li, A. Mohamed, G. Zweig, and Y. Gong, "Exploring multidimensional LSTMs for large vocabulary ASR," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4940–4944.

[62] L. Lu, X. Zhang, and S. Renais, "On training the recurrent neural network encoder-decoder for large vocabulary end-to-end speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 5060–5064.

[63] Z. Meng, S. Watanabe, J. R. Hershey, and H. Erdogan, "Deep long short-term memory adaptive beamforming networks for multichannel robust speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 271–275.

[64] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[65] D.E. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Cognitive Modeling*, vol. 5, 1988.

[66] M. W. Mak, "Lecture Notes on Backpropagation," Tech. Rep., Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, 2015, Accessed: 2018-01-11.

[67] J. Pelecanos and S. Sridharan, "Feature warping for robust speaker verification," in *Proc. Odyssey*, 2001, pp. 213—218.

[68] G. Hinton and R.R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[69] S. Yaman, J. Pelecanos, and R. Sarikaya, "Bottleneck features for speaker recognition," in *Proc. Odyssey*, 2012, vol. 12, pp. 105–108.

[70] E. Turajlic and O. Bozanovic, "Neural network based speaker verification for security systems," in *Telecommunications Forum (TELFOR)*, 2012, pp. 740–743.

[71] J. Campbell, "Testing with the YOHO CD-ROM voice verification corpus," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1995, vol. 1, pp. 341–344.

[72] M. W. Mak and H. B. Yu, "A study of voice activity detection techniques for NIST speaker recognition evaluations," *Computer Speech and Language*, vol. 28, no. 1, pp. 295–313, 2014.

[73] H. B. Yu and M.W. Mak, "Comparison of voice activity detectors for interview speech in NIST speaker recognition evaluation," in *Proc. Interspeech*, 2011, pp. 2353–2356.

[74] Z. L. Tan, Y. K. Zhu, M. W. Mak, and B. Mak, "Senone i-vectors for robust speaker verification," in *ISCSLP*, 2016.

[75] Z. L. Tan and M. W. Mak, "Bottleneck features from SNR-adaptive denoising deep classifier for speaker identification-adaptive denoising deep classifier for speaker identification," in *APSIPA ASC*, 2015, pp. 1035–1040.

[76] E. Variani, X. Lei, E. McDermott, I. Lopez-Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 4080–4084.

[77] G. Bhattacharya, J. Alam, and P. Kenny, "Deep speaker embeddings for short-duration speaker verification," in *Proc. Interspeech*, 2017, pp. 1517–1521.

[78] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in Neural Information Processing Systems*, vol. 19, pp. 153–160, 2007.

[79] H. Hirsch, "FaNT – Filtering and noise adding tool," Tech. Rep., Department of Electrical Engineering and Computer Science, Hochschule Niederrhein, University of Applied Sciences, 2005, Accessed: 2018-01-11.

[80] D. Leeuwen and Niko Brümmer, "The distribution of calibrated likelihood-ratios in speaker recognition," in *Proc. Interspeech*, 2013, pp. 1619–1623.

[81] N. Brümmer and E. de Villiers, "The BOSARIS toolkit: Theory, algorithms and code for surviving the new DCF," *CoRR*, vol. abs/1304.2865, 2013.

[82] Z. L. Tan and M. W. Mak, "I-vector DNN scoring and calibration for noise robust speaker verification," in *Proc. Interspeech*, 2017, pp. 1562–1566.

[83] N. Brümmer, A. Swart, and D. Van Leeuwen, "A comparison of linear and non-linear calibrations for speaker recognition," in *Proc. Odyssey*, 2014, pp. 14–18.

[84] N. Brümmer and G. Doddington, "Likelihood-ratio calibration using prior-weighted proper scoring rules," in *Proc. Interspeech*, 2013, pp. 1976–1980.

[85] N. Brummer and D. Garcia-Romero, "Generative modelling for unsupervised score calibration," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 1680–1684.

[86] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias," *Machine Learning*, vol. 28, pp. 41–75, 1997.

[87] D. Chen and B. Mak, "Multitask learning of deep neural networks for low-resource speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 7, pp. 1172–1183, 2015.

[88] G. Hinton, "Learning distributed representations of concepts," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 1986, vol. 1, pp. 1–12.

[89] M. Mandasari M. McLaren and D. Leeuwen, "Source normalization for language-independent speaker recognition using i-vectors," in *Proc. Odyssey*, 2012, pp. 55–61.

[90] A. Hatch, S. Kajarekar, and A. Stolcke, "Within-class covariance normalization for SVM-based speaker recognition," in *International Conference on Spoken Language Processing*, 2006, pp. 1471–1474.

[91] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[92] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2011, pp. 24–29.

[93] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015, pp. 1–13.

[94] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Aistats*, 2010, vol. 9, pp. 249–256.

[95] B. Karlik and V. Olgac, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[96] A. Maas, A. Hannun, and A. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, 2013, vol. 30, No. 1.

[97] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, 2015, pp. 448–456.