

Deep Speaker Representation Learning in Speaker Verification

by

Yingke ZHU

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
in Computer Science and Engineering

August 2022, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Yingke ZHU

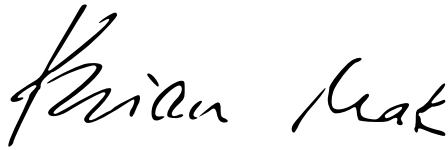
August 2022

Deep Speaker Representation Learning in Speaker Verification

by

Yingke ZHU

This is to certify that I have examined the above PhD thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.



Prof. Brian MAK, Thesis Supervisor

Department of Computer Science & Engineering

Prof. Xiaofang ZHOU

Head, Department of Computer Science and Engineering

Department of Computer Science and Engineering, HKUST

August 2022

Acknowledgments

This thesis is the culmination of my years of studying and working as a postgraduate student at HKUST. Looking back at my Ph.D. journey, I met many nice and important people that contributed to finish this thesis. I would like to take this opportunity to thank them for inspiring and supporting me all the time.

First and foremost, I would like to sincerely express my gratitude to my supervisor, Prof. Brian Mak, who has been an inspiration and a most reliable supervisor. He supported and guided me to find important things of research, and gave me freedom to work on topics I was interested in. During my Ph.D. years, we have individual meetings every week, in which we discussed my work progress, the ideas related to my research as well as problems I encountered. He is always passionate and open-minded to the challenges in research, and encourages me to keep learning and going. I deeply appreciate the time we spent in brainstorming, group seminars, repeated revisions and proofreading on my writings. I could not undertake this journey without his invaluable patience and support.

Next, I would like to thank all the committee members of my thesis proposal defence and thesis defence: Prof. Nevin Zhang, Prof. James Kwok, Prof. Yangqiu Song, Prof. Shenghui Song, Prof. Koichi Shinoda and Prof. Di Wu. I would also like to thank the professors who had taught me in the school including Prof. Mordecai Golin, Prof. Lei Chen, Prof. Ke Yi, Prof. Fangzhen Lin, Prof. Cunsheng Ding and Prof. Daniel Palomar.

I am also grateful to my advisor in Huawei Noah's Ark Lab, Dr. Tom Ko. It was a wonderful experience to work in Noah's Ark Lab. We had discussions and brainstorming whenever we achieved some progress on the project or encountered any problems in the research. His enthusiasm and passion for research also motivated me to work harder. Although my internship only last for three months, we stayed connected and kept discussing about research progress and ideas.

I would like to thank all the friends I met during my Ph.D. life, including Dongpeng Chen, Wei Li, Qing Zhang, Zhili Tan, Weiwei Lin, Pei Wang, Siqi Bao, Huan Zhao, and Shuang Qiu, who have made my life more wonderful.

Last but not least, I would like to thank my parents for their invaluable love. Their understanding and encouragement support me to go through all the difficulty times. I am always grateful to them.

Contents

Title Page	i
Authorization Page	ii
Signature Page	iii
Acknowledgments	iv
Table of Contents	v
List of Figures	viii
List of Tables	ix
Abstract	xi
1 Introduction	1
1.1 Speaker Verification	2
1.2 Speaker Modeling	3
1.3 Thesis Summary and Outline	6
2 Review of Speaker Verification Systems	8
2.1 GMM-UBM-MAP	9
2.2 Stage-wise Systems	12
2.2.1 Front-end	12
2.2.2 Back-end	17
2.3 End-to-end Systems	19
2.4 Score Normalization	21
2.4.1 Aims and Basics of Score Normalization	21
2.4.2 Score Normalization Techniques	22

3	The Baseline Speaker Verification System	24
3.1	The Speaker Verification Task	24
3.1.1	Training Corpora	24
3.1.2	Evaluation Datasets	25
3.1.3	Evaluation Metrics	27
3.2	Baseline System	28
3.2.1	Pre-processing	28
3.2.2	Speaker Modeling	30
3.2.3	Embedding Matching	33
3.3	Baseline Performance	33
4	Self-attentive Speaker Embedding Learning	34
4.1	Pooling in Speaker Verification	35
4.2	Attention Mechanism	35
4.2.1	Additive self-attention	36
4.2.2	Dot-product self-attention	36
4.3	Self-attentive Speaker Embeddings	37
4.3.1	Introduction of self-attention to pooling layers	37
4.3.2	Multi-head self-attentive speaker embeddings	38
4.3.3	Fixed-sized multi-head self-attentive speaker embeddings	39
4.3.4	Sub-vector multi-head self-attentive speaker embeddings	40
4.4	Performance	40
4.5	Analysis	45
4.5.1	Orthogonality of attention weights	45
4.5.2	Attention weights versus phonetic classes	46
5	Bayesian Self-attentive Speaker Embedding Learning	48
5.1	Theory of Stein Variational Gradient Descent	48
5.1.1	Stein's identity and Stein discrepancy	49
5.1.2	Variational inference with Stein variational gradient descent	50
5.2	Generalization of Self-attentive Speaker Embedding Learning under the Bayesian Framework	52
5.2.1	Bayesian inference perspective of the attention mechanism	52

5.2.2	Repulsive multi-head attention learning	53
5.3	Performance	54
5.4	Comparison to Deterministic Self-attentive Speaker Embeddings	57
6	Frequency-domain Pooling in Speaker Embedding Learning	60
6.1	Channel Attention	61
6.2	Frequency-domain Learning	63
6.3	Frequency-domain Pooling in Channel Attention	64
6.3.1	Discrete Cosine Transform (DCT) and Average Pooling	64
6.3.2	Multi-branch Frequency-domain Pooling	65
6.3.3	Max Frequency-domain Pooling	67
6.4	Performance	68
6.4.1	Frequency-domain pooling with a single component	68
6.4.2	Frequency-domain pooling with multiple components	70
6.5	Comparison to Self-attentive Speaker Embeddings	73
7	Summary and Future Work	75
7.1	Summary	75
7.2	Future work	77
	Reference	82
	List of Publications	91

List of Figures

1.1	Illustration of speaker verification tasks.	1
1.2	Workflow of speaker verification systems.	2
1.3	The speaker modeling module in typical speaker verification systems.	4
2.1	A roadmap of speaker verification systems.	8
2.2	Diagram of the d-vector front-end framework	14
2.3	Diagram of the x-vector front-end model	15
3.1	Length, gender and nationality distribution of speakers in VoxCeleb2.	25
3.2	The process of filterbank-based parameterization process.	29
3.3	Overview of the network structure in the baseline system.	31
4.1	Structure of the self-attention layer.	37
4.2	Heat-map of phonemes in the attentive speaker verification system.	47
6.1	SE units with global average pooling.	62
6.2	SE units with multi-branch frequency-domain pooling with K branches.	66
7.1	Architecture of self-supervised speaker embedding learning framework.	78

List of Tables

3.1	<i>Dataset statistics for VoxCeleb2</i>	25
3.2	<i>Dataset statistics for VoxCeleb1</i>	26
3.3	<i>Dataset statistics for SITW</i>	27
3.4	<i>Architecture of the speaker discriminative ResNet. T is the utterance length.</i>	32
3.5	<i>Baseline system performance on VoxCeleb1, VoxCeleb1-E and VoxCeleb1-H.</i>	33
3.6	<i>Baseline system performance on SITW evaluation set.</i>	33
4.1	<i>Results on VoxCeleb1-O, VoxCeleb1-E and VoxCeleb1-H with various self-attentive embedding learning methods.</i>	41
4.2	<i>Results on SITW evaluation set with various self-attentive embedding learning methods.</i>	42
4.3	<i>Speaker classification results of various self-attentive embedding learning methods.</i>	43
4.4	<i>Model complexity of various self-attentive embedding learning methods.</i>	44
4.5	<i>Average $O(A)$ of various multi-head systems.</i>	45
4.6	<i>Phonemes</i>	46
5.1	<i>Results on VoxCeleb1-O, VoxCeleb1-E and VoxCeleb1-H with Bayesian self-attentive embedding learning methods.</i>	55
5.2	<i>Results on SITW evaluation set with Bayesian self-attentive embedding learning methods.</i>	55
5.3	<i>Speaker classification results of Bayesian self-attentive embedding systems.</i>	56
5.4	<i>Model complexity of Bayesian self-attentive embedding learning systems.</i>	57
5.5	<i>Overall results on VoxCeleb1-O, VoxCeleb1-E and VoxCeleb1-H.</i>	58
5.6	<i>Overall results on SITW evaluation set.</i>	59
5.7	<i>Performance of state-of-the-art systems on VoxCeleb1.</i>	59
6.1	<i>Architecture of the fast SE-ResNet. T represents the training segment length.</i>	68

6.2	<i>EER results with individual frequency components pooling on Voxceleb1-O with the fast SE-ResNet framework.</i>	69
6.3	<i>minDCF results with individual frequency components pooling on Voxceleb1-O with the fast SE-ResNet framework.</i>	69
6.4	<i>Results with multiple frequency components pooling on Voxceleb1-O with the fast SE-ResNet framework.</i>	71
6.5	<i>Results with multiple frequency components pooling on Voxceleb with the SE-ResNet framework.</i>	71
6.6	<i>Results with multiple frequency components pooling on SITW with SE-ResNet framework.</i>	72
6.7	<i>Speaker classification results with multiple frequency components pooling with SE-ResNet framework.</i>	72
6.8	<i>Model complexity with multiple frequency components pooling with the SE-ResNet framework.</i>	73
6.9	<i>Overall results on VoxCeleb1-O, VoxCeleb1-E and VoxCeleb1-H.</i>	73
6.10	<i>Overall results on SITW evaluation set.</i>	73

Deep Speaker Representation Learning in Speaker Verification

Abstract

Speaker verification (SV) is the process of verifying whether an utterance belongs to the claimed speaker, based on some reference utterances.

Learning effective and discriminative speaker embeddings is a central theme in the speaker verification task. In this thesis, we focus on the speaker embedding learning issues in text-independent SV tasks, and present three methods to learn better speaker embeddings.

The first one is the self-attentive speaker embedding learning method. Usually, speaker embeddings are extracted from a speaker-classification neural network that averages the hidden vectors over all the spoken frames of a speaker; the hidden vectors produced from all the frames are assumed to be equally important. We relax this assumption and compute the speaker embedding as a weighted average of a speaker’s frame-level hidden vectors, and their weights are automatically determined by a self-attention mechanism. The effect of multiple attention heads is also investigated to capture different aspects of a speaker’s input speech.

The second method generalizes the multi-head attention in the Bayesian attention framework, where the standard deterministic multi-head attention can be viewed as a special case. In the Bayesian attention framework, parameters of each attention head share a common distribution, and the update of these parameters is related, instead of being independent. The Bayesian attention framework can help alleviate the attention redundancy problem. It also provides a theoretical understanding of the benefits of applying multi-head attention. Based on the Bayesian attention framework, we propose a Bayesian self-attentive speaker embedding learning algorithm.

The third method introduces channel attention to the embedding learning framework, and analyzes the channel attention from the perspective of frequency analysis. Frequency-domain pooling methods are then proposed to enhance the channel attention and produce better speaker embeddings.

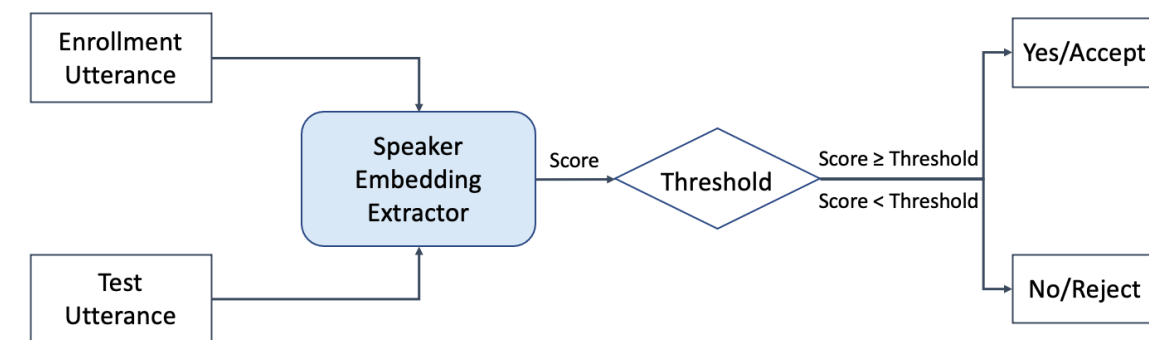
Systematic evaluation of the proposed embedding learning methods is performed on different evaluation sets. Significant and consistent improvements over state-of-the-art systems are achieved on all the evaluation datasets.

Chapter 1

Introduction

Human voices contain personal traits of a human being given his/her unique organs used for pronunciation and speaking manner, such as larynx size, vocal tract shape and speaking accent. Therefore, identification and authentication technologies based on voice biometrics have been developed. Speaker verification (SV) is the task of accepting or rejecting the identity claim of a speaker based on some given speech. It is one of the fundamental tasks of speech processing and has been widely used in real-world applications. For example, devices authentication based on voice, such as smart phones, smart homes and vehicles. It can also serve to help solve telecommunication fraud cases by identifying scammers.

Alice's registration voice.



Is that Alice's voice ?

Figure 1.1: Illustration of speaker verification tasks.

The standard speaker verification process consists of three stages: training, enrollment and evaluation. In the training stage, a large amount of speech data from many speakers are used to learn a speaker embedding extractor and build a scoring function. In the enrollment stage, a few utterances of a speaker are provided to estimate his/her speaker

model. Finally, in the evaluation stage, a new unknown utterance is scored against the speaker model estimated in the enrollment stage. The unknown utterance is considered to be produced by the claimed speaker if the evaluation score is above a pre-defined threshold, otherwise the claim is rejected. Figure 1.1 illustrates a general speaker verification task.

Speaker verification can be categorized into two broad categories according to the speech content in the enrollment and evaluation stages: text-dependent and text-independent. In text-dependent SV tasks, the speech content employed during the enrollment and evaluation stages is limited to a fixed scope. While in text-independent SV tasks, speech content is entirely unconstrained.

1.1 Speaker Verification

Speaker verification systems can be divided into two architectures based on the working criteria: end-to-end and stage-wise. A typical stage-wise system consists of a front-end module for speaker representation learning and a back-end module for speaker representation comparison. The front-end and back-end modules are trained separately. On the other hand, an end-to-end system takes a set of speech utterances as input, and directly produces the similarity scores of utterance pairs.

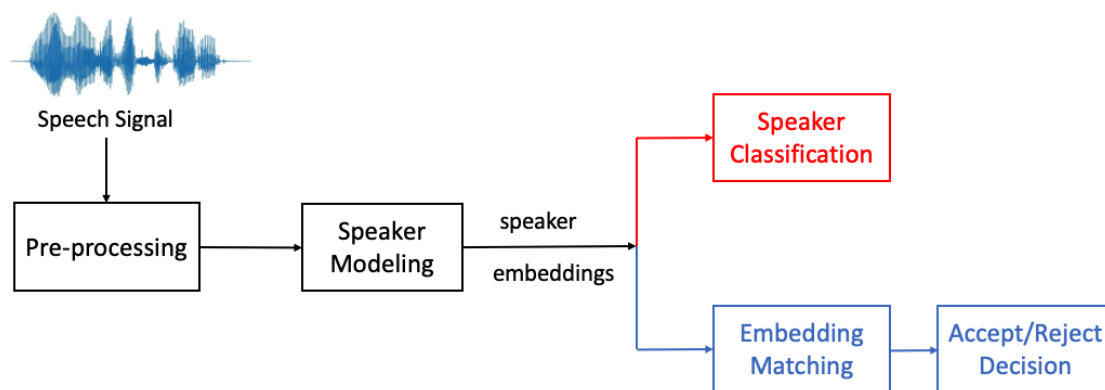


Figure 1.2: Workflow of speaker verification systems.

Figure 1.2 depicts the workflow of speaker verification systems.

- **Pre-processing:** It includes voice activity detection, speech parameterization, etc. The voice activity detection removes the non-speech portions from the input speech

signal. Speech parameterization converts speech signals to sets of feature vectors through signal processing algorithms. Filterbanks features and Mel-frequency Cepstral Coefficients (MFCC) are some widely used features.

- **Speaker Modeling:** It takes feature vectors as input, and produces representations for every speaker. These representations are also called “speaker embeddings”. Statistical methods like Gaussian Mixture Model (GMM) have been predominant over a long period of time. With the development of deep learning, a wide range of deep neural networks are developed for speaker modeling recently, such as time-delay neural networks (TDNN), long short-term memory networks (LSTM), residual networks (ResNet), etc.
- **Stage-wise systems:** In the stage-wise systems in Figure 1.2, the black parts are shared at all stages, and the red and blue parts are two subflows that are deployed at different stages. At the training stage, speaker embeddings are learned in a speaker classification task. After that, the classification layers are removed, and the learned speaker embeddings are used to train an embedding matching model. During the enrollment and evaluation stages, speaker embeddings only go through the blue part. The embedding matching module computes the similarity of embedding pairs, and decision is made according to the similarity score.
- **End-to-end systems:** End-to-end systems contain only black and blue parts in Figure 1.2. The input to the system is a set of speech utterances, and the speaker modeling module and embedding matching module are optimized together throughout the training stage.

1.2 Speaker Modeling

Speaker Modeling is an essential module in all the speaker verification systems. The goal of speaker modeling is to learn compact and effective embeddings for speakers given the speech data.

Figure 1.3 illustrated the speaker modeling module in typical speaker verification systems. It can be divided into two stages with the pooling step as the boundary: frame-level modeling before the pooling step and utterance-level modeling after the pooling

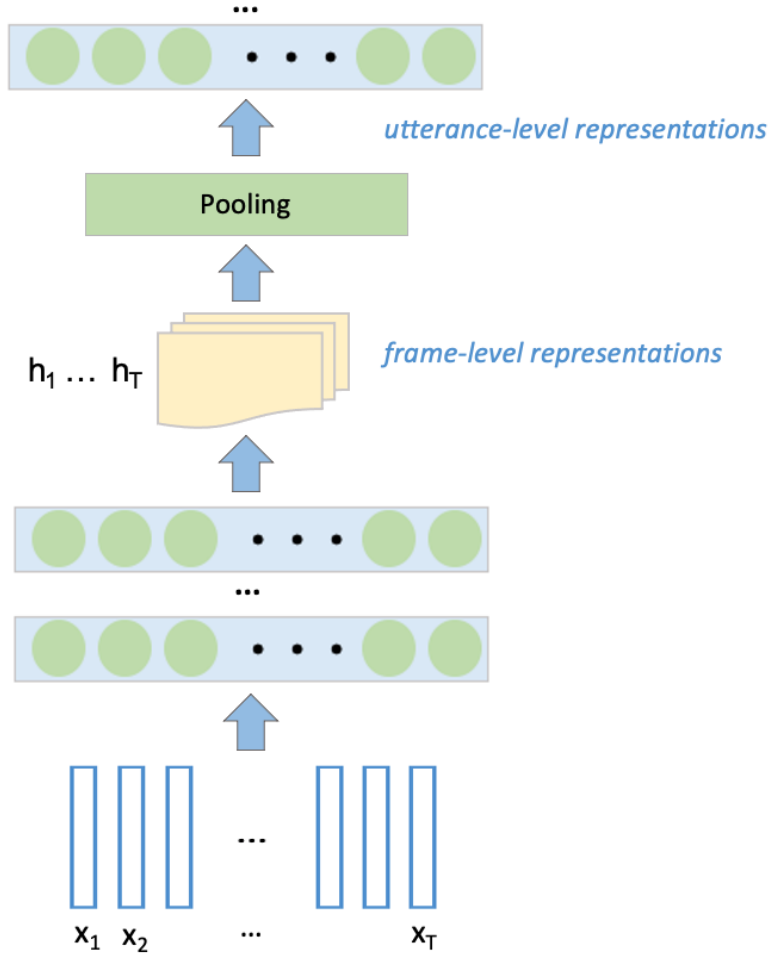


Figure 1.3: The speaker modeling module in typical speaker verification systems.

step. The inputs are a set of feature vectors computed from the speech signals at time $\{1, 2, \dots, T\}$. The frame-level modeling part deals with these feature vectors sequentially, and learns high-level representations $\{h_1, \dots, h_T\}$ for every input feature vector $\{x_1, \dots, x_T\}$. Then the pooling step gathers all the frame-level representations and produce a compact utterance-level embedding. After that, utterance-level modeling deals with utterance-level representations and produces the final speaker embeddings.

Pooling is an indispensable step in speaker modeling. Since the duration of utterances varies a lot, the length of frame-level representations can be different for every input utterance. The pooling step converts frame-level representations with various lengths to a fixed-length utterance-level representation for subsequent processing.

For a long time in the past, the pooling step simply averages over the frame-level features to produce utterance-level representations. It is simple and effective, but it also discards a large amount of information. Later, statistical pooling [1] is proposed to

utilize second-order statistics. Compared to average pooling which only reserves the first-order statistic, the statistical pooling computes mean as well as standard deviation from the frame-level features. Recent research [2] also investigated more high-order statistics information to enhance the pooling step.

While people are digging into high-order statistics of frame-level features, an underlying assumption when producing the utterance-level features keeps unchanged: all the frame-level features are equally important. In practice, a frame carries signal within a very short period of time, and different frames carry different acoustic information. Some of them may contain silence or environmental noises which can be useless when determining the identity of speakers, and some may contain distinctive phonetic information which can help speaker verification. Therefore, we propose to consider the importance of frame-level features in the pooling step. We first introduce the self-attention mechanism into the pooling step, and frame-level features are assigned with different weights when producing utterance-level representations.

By introducing the self-attention mechanism into the pooling step, we are able to weigh different frame-level features, and emphasize the frames that contain more distinctive information from a certain perspective. Apparently, speakers can be discriminated in many different aspects, especially when the speech utterances are long and contain a large amount of information. Thus, a natural extension is to explore different weighting schemes. It can be easily achieved by using multiple attention heads in the self-attention mechanism. However, we find that the system performance does not get improved when introducing more attention heads. The reason is that during the training process, there is no constraint or guidance on the weights produced by different attention heads, so they can be similar to each other and resulting in information redundancy. To avoid this issue, we investigate various techniques to improve the multi-head mechanism. We generalize the self-attention mechanism in the Bayesian framework, and propose an algorithm to encourage different attention heads to produce different weighting schemes.

Besides introducing the weighting mechanism in the pooling step, we further investigate frame-level modeling. Frame-level modeling aims to learn high-level representations for input features, and convolutional neural networks (CNNs) are the most commonly used networks. One shortcoming of CNN is its ability to model long-term interdependencies,

as the convolution operators are designed to learn only local relations. To capture long-term interdependencies, various attention modules are incorporated into the intermediate layers of CNNs. We investigate channel attention in frame-level modeling, and analyze the channel attention from the perspective of frequency analysis. Based on the frequency analysis, we propose frequency-domain pooling approaches in channel attention.

1.3 Thesis Summary and Outline

In this thesis, we focus on speaker embedding learning in text-independent SV tasks and present three learning methods.

The first one is the self-attentive speaker embedding learning method. Usually, speaker embeddings are extracted from a speaker-classification network that averages the hidden vectors over the frames of a speaker; the hidden vectors produced from all speech frames are assumed to be equally important. We relax this assumption and compute the speaker embedding as a weighted average of a speaker’s frame-level hidden vectors, and their weights are automatically determined by a structured self-attention mechanism. The effect of multiple attention heads is also investigated to capture different aspects of a speaker’s input speech.

The second method generalizes the multi-head attention in the Bayesian attention framework, where the standard deterministic multi-head attention can be viewed as a special case. In the Bayesian attention framework, parameters of each attention head share a common distribution, and the update of these parameters is related, instead of being independent as in deterministic multi-head attention. The Bayesian attention framework can help alleviate the attention redundancy problem, and it also provides a theoretical understanding of the benefits of applying multi-head attention.

The third method introduces channel attention into the speaker embedding learning framework. We analyze channel attention from the perspective of frequency analysis, and propose frequency-domain learning methods to enhance the channel attention.

The organization of this dissertation is as follows.

In Chapter 2, the current SV systems and technologies are reviewed. After an introduction to general SV system architectures, different modules are discussed.

In Chapter 3, we describe the baseline text-independent speaker verification system,

the performance of which is used as a benchmark throughout this thesis. All aspects of the system like pre-processing, speaker modeling, embedding matching, evaluation datasets and evaluation metrics will be discussed.

Chapter 4 presents the self-attentive speaker embedding learning method. We introduce the self-attention mechanism to the speaker modeling module. The effect of the self-attention mechanism and multiple attention heads will be studied and evaluated.

In Chapter 5 we generalize the self-attentive speaker embedding learning in the Bayesian framework. An algorithm to learn repulsive attention based on Stein variational gradient descent in the Bayesian framework is proposed. The effect of Bayesian self-attentive speaker embeddings will be evaluated in various benchmark corpora.

In chapter 6 we introduce channel attention to the speaker embedding learning framework. Analysis on the channel attention module is performed. We first verify that the global average pooling in the channel attention module is a special case of frequency-domain pooling where only the lowest frequency component is utilized. Then the global average pooling is generalized to the frequency-domain pooling. Two methods are proposed to utilize multiple frequency components and enhance the representation ability of channel attention.

In Chapter 7, we summarize our embedding learning methods and our contributions. We also discuss some future work in the area of speaker embedding learning.

Chapter 2

Review of Speaker Verification Systems

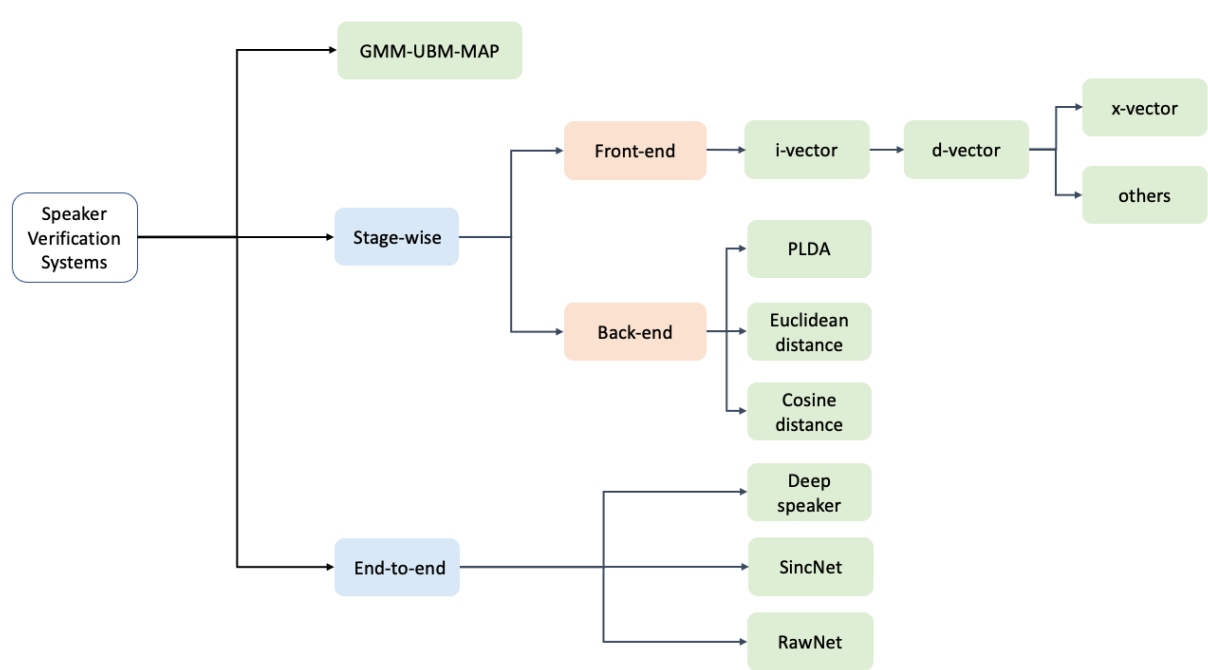


Figure 2.1: A roadmap of speaker verification systems.

In this chapter, we introduce a variety of approaches used in speaker verification systems, especially milestones in the development of speaker verification.

GMM-UBM-MAP, short for Gaussian Mixture Model-Universal Background Model with maximum a posteriori, has dominated speaker recognition area for decades. It trains a strong UBM using massive speech data first, and then derives a certain speaker’s model from the UBM with MAP adaptation given the speaker’s enrollment speech. However, the GMM-UBM-MAP suffers from the data sparsity issue. To solve this problem, the concept of “supervector”, or “speaker embedding”, is proposed. Since then representing

a speaker by a fixed-length vector receives more and more interest. We divide all the SV systems that represent speakers by fixed-length vector representations into stage-wise and end-to-end architectures according to their working criteria.

The literature review is conducted on each part based on the roadmap in Figure 2.1.

2.1 GMM-UBM-MAP

Given a piece of speech \mathbf{Y} and a hypothesized speaker \mathbf{S} , the aim of speaker verification is to determine if \mathbf{Y} is produced by \mathbf{S} . Here we always assume that \mathbf{Y} contains speech from only a single speaker. The task is basically a hypothesis test between two hypotheses:

- H_0 : \mathbf{Y} is produced by the hypothesized speaker \mathbf{S}
- H_1 : \mathbf{Y} is not produced by the hypothesized speaker \mathbf{S}

A likelihood ratio test is computed as follows to decide between the two hypotheses:

$$\frac{p(\mathbf{Y}|H_0)}{p(\mathbf{Y}|H_1)} \begin{cases} > \theta, & \text{accept } H_0, \\ < \theta, & \text{accept } H_1, \end{cases} \quad (2.1)$$

here $p(\mathbf{Y}|H_0)$ is the likelihood of hypothesis H_0 given the speech \mathbf{Y} , and θ is the decision threshold.

After pre-processing and feature extraction on speech \mathbf{Y} , a sequence of feature vectors can be obtained: $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$, where \mathbf{x}_t is a feature vector at time $t \in [1, 2, \dots, T]$. We use Ω_{hyp} to denote a model satisfying H_0 , and $\Omega_{\overline{hyp}}$ to denote a model satisfying the alternative hypothesis H_1 . The likelihood ratio statistic then becomes $\frac{p(\mathbf{X}|H_0)}{p(\mathbf{X}|H_1)}$. In practice, the logarithm version is often used instead:

$$\Lambda(\mathbf{X}) = \log p(\mathbf{X}|\Omega_{hyp}) - \log p(\mathbf{X}|\Omega_{\overline{hyp}}). \quad (2.2)$$

The model for H_0 is well defined, it can be estimated using the speech data from hypothesized speaker \mathbf{S} , while $\Omega_{\overline{hyp}}$ is hard to define since it represents the entire space of possible alternatives to the hypothesized speaker. There are two strategies to model this alternative hypothesis. The first one is to select a limited set of speaker models except the hypothesis speaker to represent the space of the alternative hypothesis. This set of speaker models is denoted as background speaker models. Given N background speaker models $\{\Omega_1, \dots, \Omega_N\}$, the alternative hypothesis model can be formulated as

$$p(\mathbf{X}|\Omega_{hyp}) = f(p(\mathbf{X}|\Omega_1), \dots, p(\mathbf{X}|\Omega_N)), \quad (2.3)$$

where $f(\cdot)$ is an aggregation function such as average or maximum. The selection of the background speakers has been studied in many works [3–6]. One major limitation of this approach is that it requires the speaker-specific background speaker sets, and has difficulty in tasks containing a large number of hypothesized speakers.

The second approach is to collect speech from a large number of speakers and train a single model, which is denoted as the universal background model (UBM) [7]. The UBM is used to cover the alternative hypothesis space. The major advantage of this approach is that the UBM is trained only once and can be used for all speakers in the evaluation stage. Based on the UBM concept, people also have investigated to use more than one background models tailored to certain sets of speakers [8, 9]. The use of a single UBM has been the predominant approach in the statistical SV models.

Besides the definition of $p(\mathbf{X}|\Omega_{hyp})$ and $p(\mathbf{X}|\Omega_{hyp})$, another important step is to select a proper likelihood function $p(\mathbf{X}|\Omega)$. The choice is highly dependent on the application types and features. For text-independent speaker verification, GMM is the most successful likelihood function.

For a F -dimensional feature vector \mathbf{x}_t at time t , the mixture density is defined in the following way:

$$p(\mathbf{x}_t|\Omega) = \sum_{c=1}^C w_c p_c(\mathbf{x}_t). \quad (2.4)$$

The density is a weighted linear combination of C unimodal Gaussian densities $p_c(\mathbf{x}_t)$, and every Gaussian density is parameterized by a $F \times 1$ mean vector $\boldsymbol{\mu}_c$ and a $F \times F$ covariance matrix $\boldsymbol{\Sigma}_c$. The weights w_c have to satisfy the constraint of summing up to 1: $\sum_{c=1}^C w_c = 1$. In practice, to improve computation efficiency, only diagonal covariance matrices are used. Compared to full covariance GMMs, diagonal-matrix GMMs also achieve better system performance in practice. The parameters of the density model are denoted as $\Omega = (w_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$, where $c = (1, \dots, C)$.

We assume that the feature vectors at each time step are independent. The log-likelihood of a model Ω for a sequence of feature vectors $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$ can be computed as:

$$\log p(\mathbf{X}|\Omega) = \frac{1}{T} \sum_{t=1}^T \log p(\mathbf{x}_t|\Omega), \quad (2.5)$$

where $\log p(\mathbf{x}_t|\Omega)$ is computed according to Equation 2.4.

The iterative expectation-maximization (EM) algorithm [10] is adopted to estimate the parameters given a collection of training vectors. The EM algorithm iteratively updates the model parameters, with the objective that increases the likelihood of the estimated model given the observed data, i.e., $p(\mathbf{X}|\Omega^{k+1}) \geq p(\mathbf{X}|\Omega^k)$ for iteration k and $k + 1$.

In the training stage, a UBM, which is a high-order Gaussian Mixture Model, is trained on utterances from a large number of speakers. It learns a distribution of features which is not dependent on the speakers, and is used in the alternative hypothesis when computing the likelihood ratio.

In the enrollment stage, the system builds one Gaussian Mixture Model for every speaker. Since the speech data from a speaker in the enrollment stage is limited, a common strategy is to adapt the well-trained speaker-independent UBM to the speaker model using Maximum a Posteriori (MAP) adaptation.

We run the EM algorithm on the well-trained UBM. In this step, we keep the covariance fixed, and only adapt the mean, because the amount of speaker data is limited. In practice, people have also tried with updating the covariance, but no further improvements are obtained. The update on mean is performed via maximum a posteriori adaptation :

$$\boldsymbol{\mu}_c^{MAP} = \alpha_c \boldsymbol{\mu}_c + (1 - \alpha_c) \boldsymbol{\mu}_c^{UBM}, \quad (2.6)$$

where:

- $\boldsymbol{\mu}_c$ represents the mean of the c -th mixture component
- $\alpha_c = \frac{n_c}{n_c + \tau_c}$ is a coefficient for the adaptation on mean
- n_c is the amount of data used for adaptation
- τ_c is a fixed relevance factor

Finally for a testing utterance, the log likelihood ratio can be computed using Equation 2.2, where Ω_{hyp} is the adapted speaker model of the claimed speaker, and $\Omega_{\overline{hyp}}$ is the UBM model.

A major limitation of GMM-UBM-MAP is the data sparsity issue in the enrollment stage. Since only very limited speech data is available for every speaker during the enrollment stage, only a limited part of the parameters in UBM are well adapted. Besides, the system is sensitive to the speaker and channel variability.

To solve these problems, researchers tend to find methods to connect all the Gaussian components in the UBM and apply some “global transform”. The concept of “super-vector” is then proposed. Basically the super-vector of an input utterance is the concatenation of means of mixture components in the adapted model $\{\mu_1^{MAP}, \dots, \mu_C^{MAP}\}$. By the introduction of the super-vector, every input utterance can be represented by a fixed-length vector, and models for classifying these super-vectors and computing similarity between super-vectors can be developed separately [11, 12].

Since then, learning fixed-length vector representations for speakers becomes mainstream in speaker verification. The representation is also called “speaker embeddings”. All the work in the following sections are developed upon the speaker embedding learning concept.

2.2 Stage-wise Systems

Stage-wise systems consist of a front-end model and a back-end model. The front-end model takes speech utterances as input and produces speaker embeddings, while the back-end model takes speaker embeddings as input and computes the similarity of pairs of embeddings. The front-end model and the back-end model are trained separately.

2.2.1 Front-end

The front-end model converts speech utterances into fixed-dimensional representation vectors, or, speaker embeddings. In the scope of statistical modeling, the “i-vector” method has been predominant for many years. With the development of deep learning, deep neural networks are adopted in speaker embedding learning. “d-vector” and “x-vector” are the millstones of deep learning-based approaches in SV. Besides, more and more speaker embedding learning models are proposed based on different neural architectures.

I-Vector

The GMM-UBM based systems performance can be affected by the intra-speaker variations and channel variations of utterances. [13] proposed the *i-vectors* based on joint factor analysis [14] to address this issue.

Given an utterance, the speaker- and channel-dependent GMM supervector can be written as:

$$\mathbf{M} = \mathbf{m} + \mathcal{T}\mathbf{w}, \quad (2.7)$$

where \mathbf{m} is a super-vector obtained from the UBM, which is speaker-independent and channel-independent; \mathcal{T} is a rectangular matrix containing all the speaker variabilities and channel variabilities, and \mathbf{w} is a random vector which follows a standard normal distribution. The components of the vector \mathbf{w} are total factors, and \mathbf{w} is named as identity vectors or i-vectors for short.

Given a sequence of feature vectors $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$ and an UBM Ω consists of C mixture components defined in a certain feature space of dimension F , the zero order, first order and centralized first-order Baum-Welch statistics for the given speech feature \mathbf{X} can be obtained by:

$$\begin{aligned} N_c &= \sum_{t=1}^T P(c|\mathbf{x}_t, \Omega) \\ F_c &= \sum_{t=1}^T P(c|\mathbf{x}_t, \Omega)\mathbf{x}_t \\ \tilde{F}_c &= \sum_{t=1}^T P(c|\mathbf{x}_t, \Omega)(\mathbf{x}_t - \mathbf{m}_c), \end{aligned} \quad (2.8)$$

where c is the Gaussian mixture index and $P(c|\mathbf{x}_t, \Omega)$ represents the probability of c -th mixture component which produces the vector \mathbf{x}_t . The i-vector for the given speech feature \mathbf{X} can be computed as follows:

$$\mathbf{w} = (\mathbf{I} + \mathcal{T}^T \Sigma^{-1} N(\mathbf{X}) \mathcal{T})^{-1} \mathcal{T}^T \Sigma^{-1} \tilde{F}(\mathbf{X}), \quad (2.9)$$

where $N(\mathbf{X})$ is a diagonal matrix with dimension $CF \times CF$. $\tilde{F}(\mathbf{X})$ is a super-vector with dimension $CF \times 1$, and it is computed by concatenating all first-order Baum-Welch statistics \tilde{F}_c for a given input \mathbf{X} . Σ is a diagonal covariance matrix with dimension $CF \times CF$; it is estimated during the factor analysis and models the additional potential variability that is not covered by the matrix \mathcal{T} .

d-Vector

The *i-vector*-based systems have been the dominant solution for speaker verification since they were proposed. In these systems, every utterance is represented by a compact feature vector, i.e., *i-vector*. Statistical methods such as joint factor analysis are employed to extract *i-vectors*.

With the great success of deep learning in various research areas, deep neural networks (DNNs) are also investigated in SV. [15] first proposed a DNN architecture to learn a compact representation of speakers.

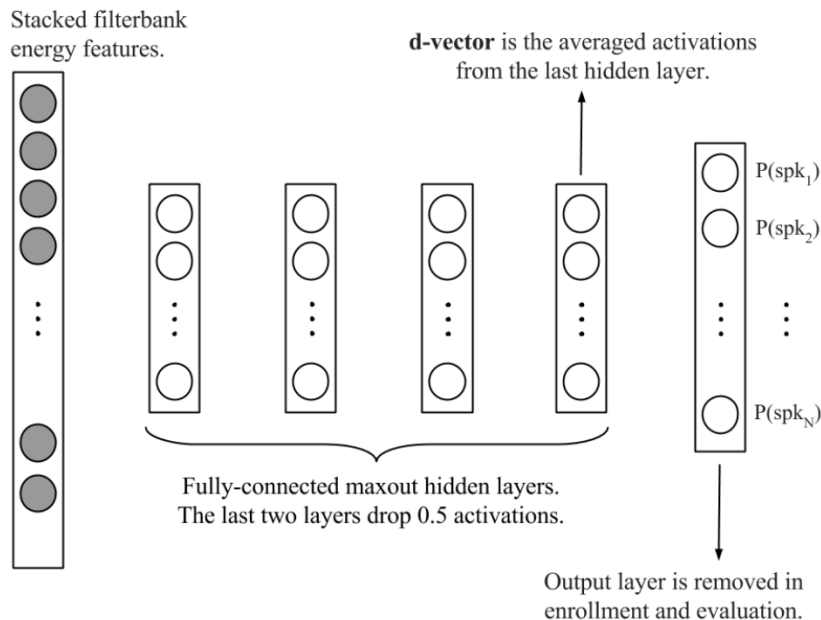


Figure 2.2: Diagram of the d-vector front-end framework

The architecture of the d-vector system is illustrated in Figure 2.2. Before feeding into the network, the input frames are stacked with its neighboring frames within a context window. The output size of the network equals to the total number of speakers in the training dataset. The overall model is trained under a speaker classification task which aims to classify speakers in the training dataset.

After the network has been trained, the accumulated activations output from the last hidden layer are used as the speaker embedding. That is, for a given utterance, the network computes the output activations of the last hidden layer for every frame, and then accumulates these activations to form the speaker embedding, the *d-vector*. The reasons to use the last hidden layer outputs include: 1. the DNN model size can be reduced by removing the output layer after training; 2. it allows using a large number of training speakers while keeping the DNN size at enrollment and evaluation time stable; 3. the output from the last hidden layer generalizes better to unseen speakers.

x-Vector

x-vector framework [16] is a milestone in speaker verification because it greatly improves the performance on various publicly available corpora. It introduces a time-delay neural network architecture to learn a high-level representation of input speech. Long-term speaker characteristics are captured by a statistical pooling method which aggregates over the input speech.

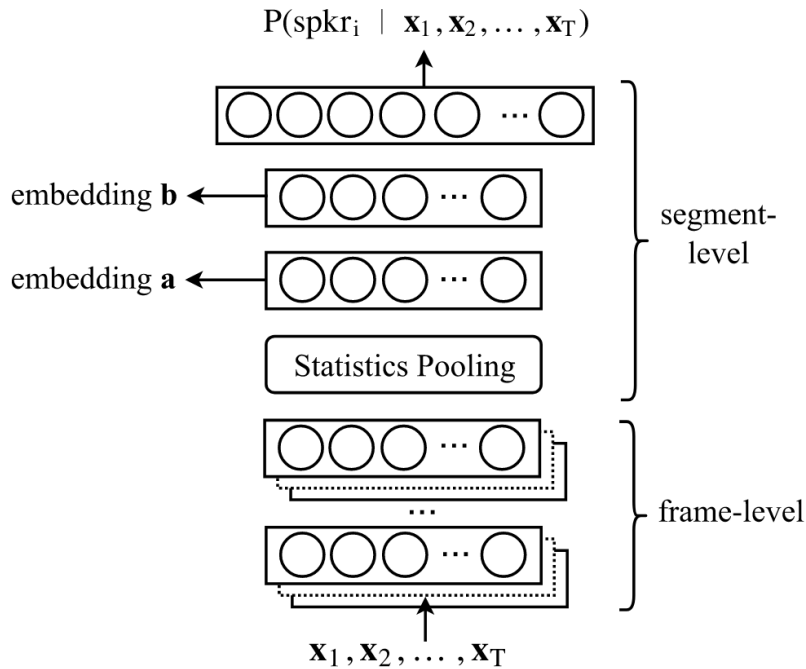


Figure 2.3: Diagram of the x-vector front-end model

The architecture of the x-vector front-end model is illustrated in Figure 2.3. The early layers of the network have a time-delay architecture, and they work at the frame level. At a certain time step t , frames are spliced together at time $\{t-2, t-1, t, t+1, t+2\}$ in the input layer. The following layers splice together the output from the previous layer at time $\{t-2, t, t+2\}$ and $\{t-3, t, t+3\}$, respectively. The layers before the statistics pooling layer has a total temporal context of 17 in this case. After that, the statistics pooling layer aggregates the output from the last frame-level layer over the whole input utterance, and computes their statistics, which are mean and standard deviation in the x-vector model. The concatenation of these statistics is then forwarded to upper layers.

The whole framework is trained to classify training speakers. After training, the output layer is removed, and output from two segment-level layers can be used as speaker embeddings, namely *x-vectors*. “X” means the place to extract speaker embeddings is not

fixed. In Figure 2.3, both “embedding **a**” and “embedding **b**” can be used as the speaker embedding.

Compared to *d-vectors*, *x-vectors* framework provides a larger temporal context utilizing the time-delay neural network (TDNN). Besides, the statistics pooling layer computes the second-order statistic in addition to the mean, which enriches the information contained in the segment-level features.

There are a lot of extension works based on the *x-vector* framework. For example, [17] proposed an extended TDNN architecture, namely E-TDNN, which provides a wider temporal context than TDNN, and it also inserts dense-connected layers between the time-delay layers to enhance the representation ability of the network. [18] developed a factorized TDNN (F-TDNN) that aimed at reducing the size of the model. The idea is to factorize the weight matrix of original TDNN layers into the product of two low-rank matrices, and the first low-rank matrix is constraint to be semi-orthogonal.

Other Deep neural network based approaches

Besides *d-vector* and *x-vector* frameworks, there are many other front-end models based on different deep neural networks such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), etc.

Residual network (ResNet) [19] is a popular architecture in speaker embedding learning. The trunk structure can be a 1-dimensional CNN with convolutions in the frequency domain, or a 2-dimension CNN with convolutions in both the time and frequency domains. Various standard ResNet architectures can be directly used as the front-end model to extract speaker embeddings [20–22]. Some work employed ResNet and its variants as the basic architecture [23–26]. For example, [23] modified the ResNet-34 to a smaller network by reducing the channel sizes in the residual blocks. [26] combined the ResNet with long short-term memory (LSTM) into a unified architecture, where LSTM was used to learn long temporal context. [27] introduced the squeeze-and-excitation block into the residual blocks, and they were combined with the TDNN architecture.

Besides ResNets, other CNN architectures have also been investigated for speaker embedding learning, such as VGGNet [28, 29], Inception Net [30], etc.

2.2.2 Back-end

The back-end model is used in the evaluation stage. It takes speaker embeddings as input and compute the similarity of pairs of embeddings.

Probabilistic Linear Discriminant Analysis

Probabilistic Linear Discriminant Analysis (PLDA) is a probabilistic version of Linear Discriminant Analysis (LDA).

LDA is a dimension reduction technique which is performed in a supervised manner. It projects the data to a lower-dimensional subspace such that in the projected subspace, data belonging to different classes are more spread out (maximizing between-class covariance) as compared to the spread within each class (minimizing the within-class covariance).

LDA works well in classification when the test data only come from the seen classes. However, in the speaker verification tasks, we want to find whether two utterances belong to the same speaker even though the model has not seen any utterances of that speaker before. If we use LDA, it will project two utterances into a subspace learned from the training data and hence will not be optimal. Probabilistic LDA is a way to address this problem.

Assume for a speaker, there are totally R utterances produced by him/her. The speaker embedding extracted from each utterance can be denoted as

$$\boldsymbol{\eta}_r = \mathbf{m} + \boldsymbol{\Phi}\boldsymbol{\beta} + \boldsymbol{\Gamma}\boldsymbol{\alpha}_r + \boldsymbol{\sigma}_r, \quad (2.10)$$

where r is the utterance index. The expression consists of two parts:

- the speaker-specific part $\mathbf{m} + \boldsymbol{\Phi}\boldsymbol{\beta}$: it models the inter-speaker variability and is utterance-independent;
- the channel-specific part $\boldsymbol{\Gamma}\boldsymbol{\alpha}_r + \boldsymbol{\sigma}_r$: it represents the channel variability, which depends on particular utterances and models the intra-speaker variability.

In detail, \mathbf{m} represents an offset estimated globally; the columns of $\boldsymbol{\Phi}$ and the columns of $\boldsymbol{\Gamma}$ comprise the basis for the speaker-specific subspace and channel-specific subspace, respectively ; $\boldsymbol{\beta}$ an $\boldsymbol{\alpha}_r$ are random vectors that follow a standard normal distribution; and $\boldsymbol{\sigma}_r$ is a residual term, it is supposed to follow the Gaussian distribution $\mathcal{N}(0, \boldsymbol{\Sigma})$.

In practice, we mainly focus on distinguishing speaker embeddings from different speakers, and the channel term may be removed. A simplified version of PLDA can be written as

$$\boldsymbol{\eta}_r = \mathbf{m} + \boldsymbol{\Phi}\boldsymbol{\beta} + \boldsymbol{\sigma}_r. \quad (2.11)$$

The parameters $\{\mathbf{m}, \boldsymbol{\Phi}, \boldsymbol{\Sigma}\}$ of the PLDA model are estimated from the training dataset with an EM algorithm [31].

Given two vectors $\mathbf{x}_1 \in \mathbb{R}^n$ and $\mathbf{x}_2 \in \mathbb{R}^n$, the verification task can be formulated as a hypothesis test between the following two hypothesis:

- H_s : \mathbf{x}_1 and \mathbf{x}_2 belong to the same speaker
- H_d : \mathbf{x}_1 and \mathbf{x}_2 belong to different speakers

The verification score based on likelihood ratio test $\log \frac{p(\mathbf{x}_1, \mathbf{x}_2 | H_s)}{p(\mathbf{x}_1, \mathbf{x}_2 | H_d)}$ can be computed as:

$$PLDA_score = \log \mathcal{N} \left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}; \begin{bmatrix} \mathbf{m} \\ \mathbf{m} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma} + \boldsymbol{\Phi}\boldsymbol{\Phi}^T & \boldsymbol{\Phi}\boldsymbol{\Phi} \\ \boldsymbol{\Phi}\boldsymbol{\Phi} & \boldsymbol{\Sigma} + \boldsymbol{\Phi}\boldsymbol{\Phi}^T \end{bmatrix} \right) - \log \mathcal{N} \left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}; \begin{bmatrix} \mathbf{m} \\ \mathbf{m} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma} + \boldsymbol{\Phi}\boldsymbol{\Phi}^T & 0 \\ 0 & \boldsymbol{\Sigma} + \boldsymbol{\Phi}\boldsymbol{\Phi}^T \end{bmatrix} \right) \quad (2.12)$$

Euclidean Distance

Given two vectors $\mathbf{x}_1 \in \mathbb{R}^n$ and $\mathbf{x}_2 \in \mathbb{R}^n$, the Euclidean distance $\|\mathbf{x}_1 - \mathbf{x}_2\|_2$ is defined by [32]:

$$\|\mathbf{x}_1 - \mathbf{x}_2\|_2 = \sqrt{\sum_{i=1}^n (\mathbf{x}_1^i - \mathbf{x}_2^i)^2}. \quad (2.13)$$

Cosine Similarity

Cosine similarity [33] is defined between two non-zero vectors. It computes the cosine result of the angle between two vector. Given two vectors $\mathbf{x}_1 \in \mathbb{R}^n$ and $\mathbf{x}_2 \in \mathbb{R}^n$, the cosine similarity can be calculated as:

$$\cos(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \times \|\mathbf{x}_2\|} = \frac{\sum_{i=1}^n \mathbf{x}_1^i \mathbf{x}_2^i}{\sqrt{\sum_{i=1}^n (\mathbf{x}_1^i)^2} \sqrt{\sum_{i=1}^n (\mathbf{x}_2^i)^2}} \quad (2.14)$$

2.3 End-to-end Systems

End-to-end SV systems take a set of utterances as input, and produce similarity scores of input pairs directly. Compared to stage-wise systems, end-to-end systems complete the front-end task and back-end task within a single architecture, and the whole system is optimized together.

Some representative end-to-end speaker verification systems include Deep speaker, SincNet, RawNet, etc.

Deep speaker

Deep speaker is a neural speaker embedding system proposed in [34]. The input to the system is usually a collection of utterances, and two neural networks are deployed to produce frame-level features: deep residual CNN and stacked gated recurrent unit (GRU). Then a pooling layer is used to produce utterance-level speaker representations followed by a length normalization layer. The system is trained with the triplet loss [35], which maximizes the distance between utterance pairs that are produced by different speakers, while minimizing the distance between embedding utterance pairs that are produced by the same speaker.

Triplet loss takes in three samples as one input tuple, including an anchor, a positive sample and a negative sample. An anchor is an utterance from a certain speaker; positive samples and negative samples are the utterances produced by the same speaker and a different speaker, respectively. During training, model parameters are updated in a way that the similarity between the negative sample and the anchor is smaller than the similarity between the positive sample and the anchor. It can be formulated as:

$$s_i^{ap} - \alpha > s_i^{an}, \quad (2.15)$$

where s_i^{ap} represents the similarity between the positive sample p and the anchor a in triplet i ; s_i^{an} represents the similarity between the negative sample n and the anchor a in triplet i ; α is a pre-defined similarity margin. The final loss function for N triplets can be formulated as:

$$L = \sum_{i=0}^N [s_i^{an} - s_i^{ap} + \alpha]_+, \quad (2.16)$$

where the operator $[\cdot]_+ = \max(\cdot, 0)$

The speaker embeddings produced by deep speaker can be directly used for various upstream applications, such as verification, identification, etc.

SincNet

In most neural-network-based SV systems, inputs to the networks are hand-crafted features obtained from specific signal processing algorithms. SincNet proposed in [36] is a CNN-based architecture that directly takes raw waveform as network input. It learns low-level representations from raw waveforms, and provides the network the possibility to capture narrow-band speaker characteristics, e.g., formants and pitch.

The first convolutional layer in SincNet consists of a set of parameterized sinc functions, which convolve with the waveform and perform as band-pass filters. Different from traditional CNNs that learn all the parameters for filters, the first layer of SincNet only learns the low and high cutoff frequencies from the training data. It offers considerable flexibility, while forcing the network to focus on a limited set of parameters that have great impact on the filters in terms of bandwidth and shape.

The major advantage of SincNet is that it greatly reduces the number of parameters in the early stage of the network, and the learned filters are more interpretable and human-readable compared to other approaches.

RawNet

RawNet [37] is another model that directly models raw waveforms. The model comprises residual blocks, a GRU layer and fully connected layers. The residual blocks in the framework are used to process inputs and produce frame-level features. After that, a GRU is employed to gather all the frame-level features, and produce a utterance-level embedding. The model is trained with the combination of three objective functions: center loss [38], speaker basis loss [39] and categorical cross-entropy loss.

Center loss \mathcal{L}_C is to minimize the intra-class covariance:

$$\mathcal{L}_C = \frac{1}{2} \sum_{i=1}^N \|\mathbf{x}_i - \mathbf{c}_{y_i}\|_2^2, \quad (2.17)$$

where \mathbf{x}_i represents the embedding of i -th input utterance, \mathbf{c}_{y_i} represents the center of class y_i , and N is the batch size.

The weight vector connecting the last hidden layer of the network and a node of the final output layer can be viewed as the basis vector for a speaker. Based on this

observation, the speaker basis loss is proposed to maximize the inter-speaker covariance. The speaker basis loss can be formulated as:

$$\mathcal{L}_{BS} = \sum_{i=1}^M \sum_{j=1, j \neq i}^M \cos(\mathbf{w}_i, \mathbf{w}_j), \quad (2.18)$$

where M is the number of speakers in the training dataset, and \mathbf{w}_i is the basis vector of the i -th speaker. The loss function for overall training is:

$$\mathcal{L} = \mathcal{L}_{CE} + \mathcal{L}_C + \mathcal{L}_{BS}, \quad (2.19)$$

where \mathcal{L}_{CE} refers to cross-entropy loss.

2.4 Score Normalization

The last step in all speaker verification systems is the decision making. A decision threshold has to be decided and used to compare the likelihood computed from the unknown speech and the claimed speaker model. The claimed speaker will be accepted if the likelihood is higher than the decision threshold, otherwise rejected.

2.4.1 Aims and Basics of Score Normalization

The decision threshold is difficult to decide since the scores vary a lot between trials. The variability of scores comes from three major sources:

- the enrollment speech can vary between speakers because of environmental conditions, speech contents, different speech duration, etc;
- the mismatch between enrollment and test speech because of intra-speaker variability;
- the quality of the test speech also have influence on the scores.

The aim of score normalization is to deal with score variability, making the decision threshold easier to decide and stable across speakers.

The basic idea of score normalization is to normalize the impostor score distributions. Given a piece of speech X and a speaker mode λ , we use $L_\lambda(X)$ to denote the similarity score of X and λ . The score normalization is applied in the following way:

$$\tilde{L}_\lambda(X) = \frac{L_\lambda(X) - \mu_\lambda}{\sigma_\lambda}, \quad (2.20)$$

where μ_λ and σ_λ are the normalization parameters, and they are estimated from the data.

The reason to choose normalizing the impostor score distribution instead of the target score distribution is that computing impostor distributions using pseudo-impostors is easy, but obtaining real target score distributions is difficult. Besides, impostor distribution can cover the most part of the score distribution variance.

2.4.2 Score Normalization Techniques

Here we summarize some commonly used score normalization techniques.

World-model and cohort-based normalizations

It is proposed in [40], the normalization takes the form as follows:

$$\tilde{L}_\lambda(X) = \frac{L_\lambda(X)}{L_{\bar{\lambda}}(X)}. \quad (2.21)$$

$L_{\bar{\lambda}}$ is the likelihood computed from a cohort of speaker models. The cohort of speakers can be a set of speakers close to speaker λ . Later this cohort of impostor models is replaced by a unique model trained on the same sets of data. This unique model is named as world-model, and the aim of introducing the world-model is mainly to reduce the computation costs.

All the score normalization methods discussed below are applied on top of this world-model normalization method.

Znorm

Znorm represents the zero normalization, and it is derived from [41]. The aim of Znorm is to scale and shift the distribution of scores between a target speaker λ and a set of impostors to the standard normal distribution. Speaker-dependent mean and variance are estimated. One advantage of Znorm is that we can estimate the normalization parameters offline.

Hnorm

Hnorm represents handset normalizaiton; it is a variant of Znorm proposed in [42]. It is especially developed for telephone speech. For telephone speech, the target speaker

models usually have different responses depending on the handset type used during speech recording. Therefore, all the speaker models are tested against handset-dependent speech produced by impostors, and the parameters of Hnorm are estimated during this process. In the evaluation time, parameters used for score normalization are decided by the type of handset used in the incoming speech.

Tnorm

Tnorm is short for test-normalization, and is proposed in [43]. It performs impostor score normalization based on the mean and variance estimations. During the evaluation stage, the testing speech is evaluated against the claimed speaker model and a set of impostor models, and the normalization parameters are then estimated. Compared to Znorm which estimates speaker-dependent mean and variance, Tnorm only depends on the test data. The same test data is used in evaluation as well we normalization parameter estimation.

HTnorm

HTnorm is a variant of Tnorm based on the handset-type information, similar to the idea of Hnorm. The normalization parameters are handset-dependent, and they are estimated by comparing every incoming speech to handset-dependent impostor models. In the evaluation stage, parameters used for score normalization are decided by the type of handset related to the claimed speaker.

Dnorm

Dnorm is proposed in [44]. It generates the pseudo-impostor data using a world model, and a Monte-Carlo-based method is used to generate a collection of targets as well as impostor data. The normalization score is computed by:

$$\tilde{L}_\lambda(X) = \frac{L_\lambda(X)}{KL2(\lambda, \bar{\lambda})}, \quad (2.22)$$

where $KL2(\lambda, \bar{\lambda})$ is the symmetrized Kullback-Leibler distance between the target model and the world model. The estimation of the KL distance is performed with the data generated by Monte-Carlo method. The advantage of Dnorm is that it does not require any data for normalization parameter estimation.

Chapter 3

The Baseline Speaker Verification System

Throughout this dissertation, our proposed methods are evaluated on a text-independent speaker verification task under noisy and unconstrained conditions. The task is chosen for two reasons: Firstly, speaker verification under noisy and unconstrained conditions is a challenging but meaningful task; methods developed for the task can be applied in real-life applications. Secondly, the datasets used in the task contain millions of utterances for over thousands of speakers, and all the data are publicly available.

In this chapter, we describe the task in detail, including various components of the baseline system, as well as its benchmark performance.

3.1 The Speaker Verification Task

The speaker verification task is text-independent and all utterances are spoken in English. The training corpora, evaluation datasets and evaluation metrics are the same for all the systems developed in this thesis.

3.1.1 Training Corpora

The training dataset used in our system is VoxCeleb2 [21]. VoxCeleb2 consists of over 1 million utterances for over 6000 celebrities extracted from YouTube videos. It is gender-balanced, and the speakers have a large variability in terms of accents, ages, professions and ethnicities. Audios in the dataset are obtained in a variety of challenging acoustic environments, including celebrity interviews on red carpets, speeches given to large

Table 3.1: *Dataset statistics for VoxCeleb2*

# of speakers	6112
# videos	150,480
# of hours	2442
# of utterances	1,128,246
Avg # of videos per speaker	25
Avg # of utterances per speaker	185
Avg length of utterances (s)	7.8

audiences in outdoor stadiums and indoor studios, excerpts from professionally shot multimedia, and even videos shot by hand-held devices. All speech data are corrupted with different kinds of noises, for example, laughter, background chatter, overlapping speech. Besides, the quality of recording equipment varies from utterance to utterance, and channel noise is also included in the recordings. Figure 3.1 shows length, gender and nationality distributions of VoxCeleb2, and Table 3.1 gives the general statistics of the dataset.

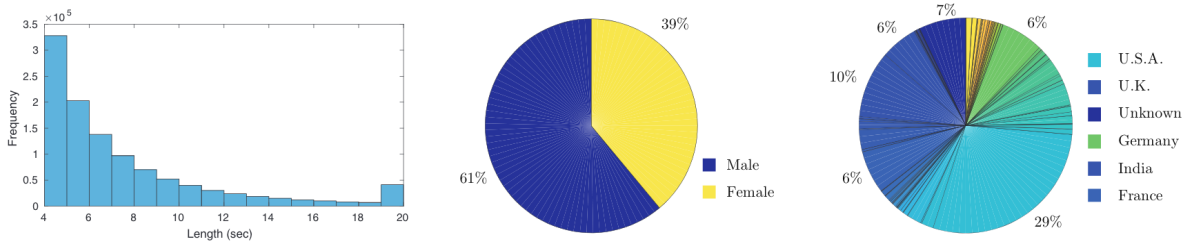


Figure 3.1: Length, gender and nationality distribution of speakers in VoxCeleb2.

To increase the diversity as well as the amount of the data for training, augmentation techniques are applied. Our strategy employs reverberation and additive noises.

Reverberation is the process of convolving room impulse responses (RIR) with audio. The RIRs simulated under different room conditions dataset [45] are used.

For additive noises, the MUSAN dataset [46] is used. MUSAN contains over 900 types of noises, 42 hours of music with different genres and 60 hours of speech collected in twelve languages.

3.1.2 Evaluation Datasets

The systems are evaluated on two public datasets: VoxCeleb1 and SITW.

Table 3.2: *Dataset statistics for VoxCeleb1*

# of speakers	1251
# videos	690
# of hours	352
# of utterances	153,516
Avg # of videos per speaker	18
Avg # of utterances per speaker	116
Avg length of utterances (s)	8.2

VoxCeleb1

VoxCeleb1[28] is collected using the same pipeline as VoxCeleb2. It has a smaller size compared to VoxCeleb2, with over 100,000 utterances from 1251 celebrities. The dataset is roughly gender-balanced with 55% male speakers, and speech segments are corrupted with various real-world noises. Table 3.2 gives the general statistics of the VoxCeleb1.

When used as an evaluation dataset, VoxCeleb1 can be further split into three different subsets:

- VoxCeleb1-O: The original Voxceleb1 dataset is partitioned into a development set and an evaluation set. VoxCeleb1-O refers to the original evaluation set.
- VoxCeleb1-E: It refers to the entire VoxCeleb1 dataset, including the development set and evaluation set.
- VoxCeleb1-H: It refers to a difficult partition of VoxCeleb1 dataset, which consists of data pairs with the same nationality and gender.

SITW

The Speakers in the Wild (SITW) is a database containing speech samples from YouTube and is well hand-annotated. It contains both single- and multi-speaker audios collected under unconstrained conditions. In our tasks, we concentrate on the single-speaker tests, where each utterance contains speech from only one speaker.

The database consists of recordings of 299 speakers, with an average of eight different sessions per person. The SITW data have a large variance in terms of speaking conditions, such as outdoor interviews, monologues, and conversational dialogues with dominant back-channels. Speeches for each speaker is acquired both from high-quality interviews and from raw audio captured on camcorders. Noises, reverberation, vocal effort

and compression artifacts in the corpus are natural characteristics of the original audio. The duration of speech is unconstrained. Table 3.3 gives the general statistics of the SITW.

Table 3.3: *Dataset statistics for SITW*

# of speakers	299
# of utterances	4841
Length of utterances for enrollment(s)	6 - 180
Length of utterances for testing (s)	6 - 180

3.1.3 Evaluation Metrics

Two kinds of errors may occur in speaker verification systems: false rejection and false acceptance. A false rejection error represents the situation when an identity claim from the target speaker is rejected. A false acceptance error represents the situation when an identity claim from an impostor is accepted. Both types of errors highly depend on the decision threshold. If we set the threshold relatively low, the system will accept all identity claims resulting in a lot of false acceptances. In contrast, if the threshold is set relatively high, the systems tends to reject most incoming claims and therefore make a lot of false rejections.

The couple (false acceptance error rate, false rejection error rate) is defined as the *operation point* of a speaker verification system. In other words, the determination of the threshold is a trade-off between these two kinds of errors.

Two evaluation metrics are used to evaluate our systems.

Equal Error Rate

Let P_{fa} denotes the false acceptance error rate and P_{fr} denotes the false rejection error rate. Equal error rate (EER) represents the operating point where $P_{fa} = P_{fr}$. The EER is a measurement of speaker verification systems in terms of their ability to separate target speakers from impostors.

Detection Cost Function

Detection cost function (DCF) can be used for comparing speaker verification systems. It takes both error rates into consideration, and weights their importance by their pre-defined costs and the prior target probability:

$$\mathcal{C}_{det} = \mathcal{C}_{fr} \times P_{fr} \times P_{target} + \mathcal{C}_{fa} \times P_{fa} \times (1 - P_{target}), \quad (3.1)$$

where P_{target} is the prior probability of the target speaker; \mathcal{C}_{fr} and \mathcal{C}_{fa} are costs given to false rejection and false acceptance, respectively. Given an SV system, \mathcal{C}_{det} can be calculated at all operation points, and the minimum among them is denoted as $\min\mathcal{C}_{det}$.

We use $\min\mathcal{C}_{det}$ as a metric to evaluate systems. We assume a prior target probability, P_{target} , of 0.01, and equal costs between false acceptance and false rejection, that is, \mathcal{C}_{fr} and \mathcal{C}_{fa} are set to be 1.

3.2 Baseline System

This section describes our baseline system based on the ResNet framework.

3.2.1 Pre-processing

The raw input utterances are pre-processed through speech parameterization that converts signals to feature vectors, voice activity detection (VAD) which removes the non-speech parts in the utterances, and data augmentation which increases the amount and diversity of the data.

Speech Parameterization

Speech parameterization is the process that transforms a speech signal into a sequence of acoustic feature vectors. This step aims at producing a more suitable and compressed representation for subsequent learning. In our systems, filterbank-based spectral parameters are used as input to subsequent models. Figure 3.2 shows the filterbank-based spectral parameterization process, and the output are filterbank features.

The speech signal is first pre-emphasized. Since the high frequencies of the spectrum are often reduced in the speech production process, here a filter is applied to enhance these high frequencies in the spectrum.

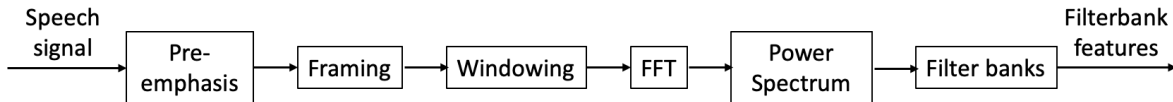


Figure 3.2: The process of filterbank-based parameterization process.

After pre-emphasis, the signals are split into short-time frames. The spectral characteristics of a signal are quasi-stationary over a short period of time, so the frequency analysis over short-time frames can obtain a good spectral representation of the signal in each frame. A common frame size in speech processing is 25ms with a 10ms stride between consecutive frames.

After framing, a window function is applied to frames. The windowing step is mainly to counteract the assumption made by the fast Fourier transform that the input is infinite long. We use a Hamming window, which is a common choice in speech processing.

Once the speech signal is windowed, the fast Fourier transform (FFT) is performed on each frame to get its power spectrum.

The final step is to apply a bank of triangular filters on a Mel-scale to the power spectrum. The basic idea of Mel-scale is having the spectral analysis more discriminative at lower frequencies while less discriminative at higher frequencies. It mimics the human perception of sound. At the end, we obtain a mel-spectrum for each frame of speech.

Voice Activity Detection (VAD)

Voice activity detection, also known as speech activity detection, is a process that detects the presence or absence of human speech. In speaker verification, silence or non-speech parts are removed after VAD since they do not contain any vocal or speaker information.

We use the energy-based VAD, which is the most commonly used method in speaker verification. We compute the energy for each frame in the spectrogram obtained in the speech parameterization step. Then a frame is considered as “non-speech” if the total energy within a window centered on the frame is smaller than a pre-defined threshold. All the non-speech frames are removed before further processing.

Data Augmentation

To increase the data diversity and improve the robustness of the model, data augmentation is often employed. In our systems, the following augmentation strategies are used:

- **babble**: Several speakers are randomly sampled from the MUSAN corpus They are then aggregated and added to the original speech signal at 13-20dB SNR.
- **music**: A music file is picked from MUSAN and added to the original signal at 5-15dB SNR. In order to match the duration, the music file may be repeated or trimmed.
- **noise**: Noises files randomly sampled from MUSAN are added at one second intervals in the entire recording at 0-15dB SNR.
- **reverberation**: The training sample is artificially reverberated via convolution with simulated RIRs.
- **SoX**¹ : Each training recording is augmented with the tempo up and tempo down methods provided in SoX.
- **FFmpeg library**² : Each training sample is augmented by alternating Opus codec or AAC codec provided in the FFmpeg library.
- **SpecAugment** [47]: It randomly masks 0 to 5 frames in the time domain and 0 to 10 channels in the frequency domain in the spectrogram of a speech segment.

The first six augmentation methods will generate new augmented samples from a given training sample, while the last method directly modifies the training sample. Totally six extra training sets are generated with these augmentation techniques.

3.2.2 Speaker Modeling

Model Architecture

The speaker modeling is based on the ResNet architecture consisting of 34 layers. The model architecture is depicted in Fig 3.3, and the detailed configuration of the network is depicted in Table 3.4.

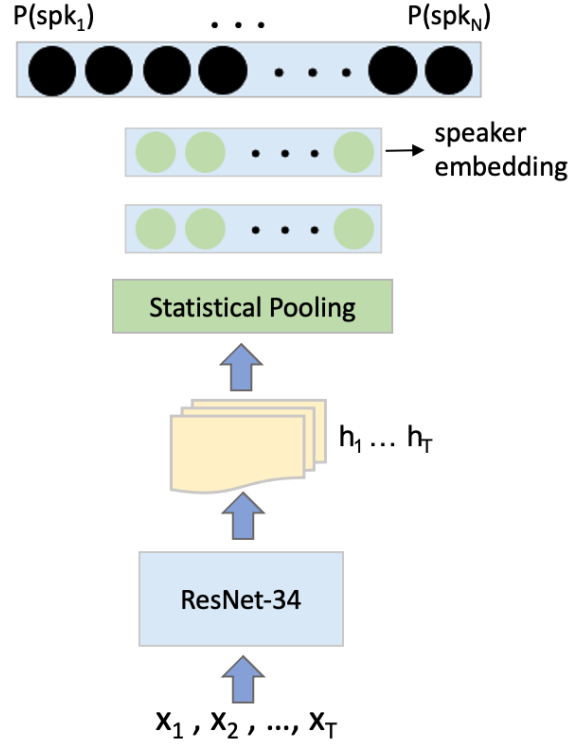


Figure 3.3: Overview of the network structure in the baseline system.

The layers before statistical pooling are constructed with residual blocks; these layers work at the frame level. The statistics pooling layer gather all the frame-level output vectors of the network, and computes their statistics: mean and standard deviation in this case. This pooling mechanism enables the network to produce a fixed-length representation from variable-length speech segments. The mean and standard deviation are concatenated together and forwarded to two additional hidden layers, and finally an additive margin softmax output layer. All neural units are rectified linear units (ReLUs). The network is trained to classify speakers in the training set.

Loss Function

Additive margin softmax [48] is used as the loss function.

The standard softmax loss is defined as follows:

$$\mathcal{L}_S = \frac{1}{N} \sum_{i=1}^N - \log \frac{e^{\mathbf{w}_{y_i}^T \mathbf{x}_i + b_{y_i}}}{\sum_{j=1}^c e^{\mathbf{w}_i^T \mathbf{x}_i + b_j}}, \quad (3.2)$$

where \mathbf{x}_i is the input to the last fully connected layer of sample i ; $y_i \in \{1, \dots, c\}$ is the class label of sample i ; c represents the total number of classes; N represents the total

¹<http://sox.sourceforge.net/>

²<http://www.ffmpeg.org/>

Table 3.4: Architecture of the speaker discriminative ResNet. T is the utterance length.

Layer	Kernel size	Stride	Output shape
Conv1	$3 \times 3 \times 32$	1×1	$T \times 64 \times 32$
Res1	$3 \times 3 \times 32$	1×1	$T \times 64 \times 32$
Res2	$3 \times 3 \times 64$	2×2	$\frac{T}{2} \times 32 \times 64$
Res3	$3 \times 3 \times 128$	2×2	$\frac{T}{4} \times 16 \times 128$
Res4	$3 \times 3 \times 256$	2×2	$\frac{T}{8} \times 8 \times 256$
Flatten	-	-	$\frac{T}{8} \times 2048$
Pooling	-	-	4096 or 4096*heads
Linear	512	-	512
Linear	256	-	256
AM-Softmax	-	-	num. of speakers

number of samples; \mathbf{w}_j and b_j are the weight vector and bias of the last fully connected layer related to class j .

In Equation 3.2, $\mathbf{w}^T \mathbf{x}$ can be written as $\|\mathbf{w}\| \|\mathbf{x}\| \cos(\theta)$, where θ is the angle between \mathbf{w} and \mathbf{x} . Therefore Equation 3.2 can be rewritten as:

$$\mathcal{L}_S = \frac{1}{N} \sum_{i=1}^N -\log \frac{e^{\|\mathbf{w}_{y_i}\| \|\mathbf{x}_i\| \cos(\theta_{y_i,i}) + b_{y_i}}}{\sum_{j=1}^c e^{\|\mathbf{w}_j\| \|\mathbf{x}_i\| \cos(\theta_{j,i}) + b_j}}, \quad (3.3)$$

By normalizing weight \mathbf{w} and input \mathbf{x} , setting the bias b to zero and using a tighter function $\psi(\theta) < \cos(\theta)$ to replace the cosine function, the softmax formulation can be modified to a margin softmax:

$$\mathcal{L}_{MS} = \frac{1}{N} \sum_{i=1}^N -\log \frac{e^{\psi(\theta_{y_i,i})}}{e^{\psi(\theta_{y_i,i})} + \sum_{j=1, j \neq y_i}^c e^{\cos(\theta_{j,i})}}. \quad (3.4)$$

Additive margin softmax adopts the following additive margin function $\psi(\theta)$:

$$\psi(\theta) = s \cdot (\cos\theta - m), \quad (3.5)$$

where m represents the margin. The additive margin softmax is finally formulated as:

$$\mathcal{L}_{AMS} = \frac{1}{N} \sum_{i=1}^N -\log \frac{e^{s \cdot (\cos\theta_{y_i,i} - m)}}{e^{s \cdot (\cos\theta_{y_i,i} - m)} + \sum_{j=1, j \neq y_i}^c e^{s \cdot (\cos\theta_{j,i})}}. \quad (3.6)$$

In our systems the margin is set to 0.2.

Training Setup

The input features are 80-dimensional filterbank features computed from a 25 ms window with a 10 ms frame shift; the energy is also included. The features are mean-normalized

over a sliding window of up to 2 seconds. VAD and data augmentation are applied before feeding the input samples to the network.

The system is trained with the stochastic gradient descent (SGD) optimizer. The initial learning rate is set to 0.01, and it is reduced when the accuracy on the development set stops improving. Weight decay is applied to all parameters in the model: the decay rate is $2e-4$ for the parameters of additive margin softmax and $2e-5$ for all the other weights. The mini-batch size is 512, and the training runs for 8 epochs.

3.2.3 Embedding Matching

After training, speaker embeddings are extracted from the last fully connected layer. The system uses cosine similarity back-end for embedding matching, and similarity scores are normalized using adaptive s-norm [49].

3.3 Baseline Performance

The baseline system performance is summarized in Table 3.5 and Table 3.6, they are used as a benchmark throughout the thesis.

Table 3.5: *Baseline system performance on VoxCeleb1, VoxCeleb1-E and VoxCeleb1-H.*

System	VoxCeleb1		VoxCeleb1-E		VoxCeleb1-H	
	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
baseline	1.12	0.151	1.34	0.143	2.38	0.222

Table 3.6: *Baseline system performance on SITW evaluation set.*

System	EER	minDCF
baseline	1.73	0.166

Chapter 4

Self-attentive Speaker Embedding Learning

In this chapter, we present our work on learning self-attentive speaker embeddings. Usually, speaker embeddings are extracted from a speaker-classification neural network that averages the hidden features over the frames of a speaker; the hidden features produced from all the frames are assumed to be equally important. We relax this assumption considering that different speech frames carry different acoustic information. For example, the frames with only silences or background noises are less useful while frames containing discriminative phonetic information contributes more when determining the speaker identities. We introduce the self-attentive mechanism into the pooling step of speaker verification systems. Speaker embeddings are computed as a weighted average of a speaker’s frame-level hidden features, and their weights are automatically determined by the self-attention mechanism.

We conjecture that the weighting scheme can emphasize frames that carry more distinctive information from a certain perspective. Since speakers can be discriminated in many different aspects, we further investigate the self-attentive mechanism with multiple attention heads. Each attention head is supposed to capture different aspects of the input speech and thus enhance speaker embedding learning. However, we find that with more attention heads, different attention heads tends to produce similar weighting patterns. To alleviate this attention redundancy issue, we investigate various techniques including the introduction of a penalty term, and splitting the input to each attention head.

4.1 Pooling in Speaker Verification

In all neural network-based SV systems, for an input utterance, the network first learns a sequence of frame-level feature vectors of variable lengths, depending on the input utterance length. Then an utterance-level speaker embedding of a fixed dimension from these frame-level features is obtained through a pooling layer. The pooling step aggregates information over the whole utterance. How to learn a better utterance-level speaker embedding through the pooling step is an essential issue in speaker verification.

In early developed systems, simple averaging was used to aggregate the frame-level features over the whole utterance [15, 50]. Suppose a speech segment of duration T produces a sequence of T frame-level features $\mathbf{H} = \{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T\}$, where \mathbf{h}_t is the hidden representation of input frame \mathbf{x}_t captured by the hidden layer before the pooling layer. The mean pooling can be formulated as:

$$\mathbf{e} = \frac{1}{T} \sum_{i=1}^T \mathbf{h}_t \quad (4.1)$$

Later in [1, 16], statistical pooling was proposed. It computes the mean as well as standard deviation from the frame-level features. The statistics vectors are then concatenated together to form the utterance-level representation. Statistical pooling obtains significant improvement by utilizing second-order information. It can be formulated as:

$$\begin{aligned} \mathbf{e} &= \frac{1}{T} \sum_{i=1}^T \mathbf{h}_t, \\ \mathbf{d} &= \sqrt{\frac{1}{T} \mathbf{h}_t \odot \mathbf{h}_t - \mathbf{e} \odot \mathbf{e}}, \end{aligned} \quad (4.2)$$

where \odot represents Hadamard product. The mean vector \mathbf{e} and standard deviation vector \mathbf{d} are concatenated to form the statistical pooling result.

In all the prior work, both average pooling and statistical pooling assigns equal weight to each frame-level feature. That is, the features produced from all frames are considered equally important.

4.2 Attention Mechanism

Attention mechanism has been a notable topic in deep representation learning. It is inspired by human biological system that focuses on only relevant information which is

helpful in solving a task at hand. In the early stage, the attention mechanism is usually embedded in recurrent networks to deal with sequential data. It has been proved to be effective in various deep learning tasks such as computer vision [51, 52], natural language processing [53, 54], and speech recognition [55, 56].

Self-attention is an attention mechanism that can capture long-range dependencies within an input sample itself. It does not require extra information and learns high-level representation by aggregating features at different positions in the input sample. The self-attention mechanism is also widely used in a variety of tasks including machine translation, reading comprehension and embedding learning [57–60]. In this thesis, we focus on the use of the self-attention mechanism in speaker verification.

There are two common ways to compute self-attention, namely, additive self-attention and dot-product self-attention.

4.2.1 Additive self-attention

The additive self-attention proposed in [59] used a one-layer feed-forward neural network to compute the alignment weights. Given an input $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the input sequence length and d is the input feature dimension, the alignment weights vector \mathbf{a} and output representation vector \mathbf{z} are calculated as:

$$\begin{aligned} \mathbf{a} &= \text{softmax}(\mathbf{v}^T \tanh(\mathbf{W}\mathbf{X}^T)), \\ \mathbf{z} &= \mathbf{a}\mathbf{X}, \end{aligned} \tag{4.3}$$

where $\mathbf{W} \in \mathbb{R}^{d_a \times d}$ and $\mathbf{v} \in \mathbb{R}^{d_a \times 1}$ are trainable attention parameters, and d_a is a hyper-parameter. The output \mathbf{z} is a sequence representation vector that is the sum of the input features weighted by the attention alignment \mathbf{a} .

4.2.2 Dot-product self-attention

Dot-product self-attention was proposed in [57]. An input feature \mathbf{X} is first mapped to three different representations: query \mathbf{Q} , key \mathbf{K} and value \mathbf{V} , then the attention alignment \mathbf{a} is calculated from the dot product of the query and key:

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}_Q, \mathbf{K} = \mathbf{X}\mathbf{W}_K, \mathbf{V} = \mathbf{X}\mathbf{W}_V, \\ \mathbf{A} &= \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right), \end{aligned} \tag{4.4}$$

and the output representation \mathbf{z} is calculated as:

$$\mathbf{Z} = \mathbf{A}\mathbf{V}, \quad (4.5)$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ are trainable attention parameters and d_k is the dimension of query and key.

4.3 Self-attentive Speaker Embeddings

4.3.1 Introduction of self-attention to pooling layers

Inspired by the structured self-attention mechanism proposed in [59] for sentence embedding, we replace the statistical pooling layer with a self-attention layer as shown in Fig.4.1. The new pooling module derives weighted means and standard deviations from the outputs of the previous hidden layer over each speech segment. The weights are learned with the self-attention mechanism to maximize speaker classification performance for the whole system.

We have compared dot-product and additive self-attention and got better results with the latter; so additive self-attention is chosen in this work.

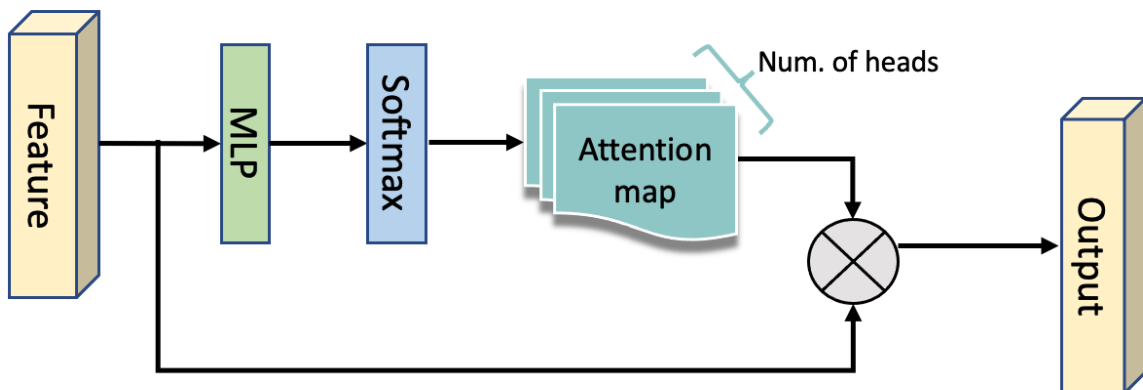


Figure 4.1: Structure of the self-attention layer.

Suppose a speech segment of duration T produces a sequence of T feature vectors $\mathbf{H} = \{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T\} \in \mathbb{R}^{d_h \times T}$, where \mathbf{h}_t is the hidden representation of input frame \mathbf{x}_t captured by the hidden layer before the pooling layer. Let the dimension of \mathbf{h}_t be d_h . Thus, the size of \mathbf{H} is $d_h \times T$. The self-attention mechanism takes the whole hidden

representation \mathbf{H} as input, and outputs an annotation matrix \mathbf{A} as follows:

$$\mathbf{A} = \text{softmax}(g(\mathbf{H}^T \mathbf{W}_1) \mathbf{W}_2), \quad (4.6)$$

where \mathbf{W}_1 is a matrix of size $d_h \times d_a$; \mathbf{W}_2 is a matrix of size $d_a \times d_r$; d_a is a hyperparameter which is usually smaller than d_h ; d_r represents the number of attention heads; $g(\cdot)$ is some activation function and ReLU is chosen here. The $\text{softmax}(\cdot)$ is performed column-wise.

Each column vector of \mathbf{A} , denoted by \mathbf{A}_r , is an annotation vector computed from the r -th attention head, and it represents the weights for different \mathbf{h}_t . Finally the weighted mean $\mathbf{E}_r \in \mathbb{R}^{d_n}$ obtained from the r -th attention head is computed as:

$$\mathbf{E}_r = \mathbf{H} \mathbf{A}_r, \quad (4.7)$$

and the weighted standard deviation of the r -th attention head $\mathbf{D}_r \in \mathbb{R}^{d_n}$ is defined as:

$$\mathbf{D}_r = \sqrt{\mathbf{H}^2 \mathbf{A}_r - (\mathbf{E}_r)^2}, \quad (4.8)$$

where $(\cdot)^2$ represents the element-wise square operation. The concatenation of weighted mean \mathbf{E}_r and weighted standard deviation \mathbf{D}_r forms the pooling output of the r -th attention head.

4.3.2 Multi-head self-attentive speaker embeddings

When the number of attention heads $d_r = 1$, the pooling output is expected to reflect an aspect of discriminative speaker characteristics in the given speech segment. Apparently, speakers can be discriminated in many different aspects, especially when a speech segment is long. Here we construct multiple attention heads to learn different aspects from a speaker’s speech by increasing d_r .

To encourage diversity in the annotation vectors so that each attention head may extract dissimilar information from the same speech segment, a penalty term P is introduced when $d_r > 1$;

$$P = \|(\mathbf{A}^T \mathbf{A} - \mathbf{I})\|_F^2, \quad (4.9)$$

where \mathbf{I} is the identity matrix and $\|\cdot\|_F$ represents the Frobenius norm of a matrix.

It uses dot product of \mathbf{A} and its transpose as the measure of redundancy. Every column annotation vector \mathbf{A}_i in \mathbf{A} can be seen as a discrete probability distribution since

the softmax function makes all the elements in an annotation vector sum up to 1. Any non-diagonal element $a_{ij}(i \neq j)$ in the $\mathbf{A}^T \mathbf{A}$ matrix corresponds to a summation over the dot product of two distributions:

$$a_{ij} = \sum_{k=1}^T \mathbf{A}_i^k \mathbf{A}_j^k \quad (4.10)$$

where \mathbf{A}_i^k is k -th element in the annotation vector \mathbf{A}_i . In the extreme case when the two probability distributions \mathbf{A}_i and \mathbf{A}_j are orthogonal, the corresponding a_{ij} will be 0; otherwise, it will have a positive value. The penalty term forces the non-diagonal elements to 0, and punishes redundancy between different annotation vectors. In the meantime, the diagonal elements are forced to be 1 so as to encourage each annotation vector \mathbf{A}_i to focus on as few number of frames as possible. The penalty term is similar to L2 regularization and is minimized together with the original cost of the whole system.

4.3.3 Fixed-sized multi-head self-attentive speaker embeddings

When we have multiple attention heads, the output from every single head is concatenated together to form the final output embedding. Therefore, the dimension of the final output increases linearly with the number of heads if we follow the mechanism in Equation 4.7 and Equation 4.8. Suppose the original dimension of hidden representation \mathbf{h}_t is d_h , and the concatenation of weighted mean and weighted standard deviation is used as the output of each attention head, the final output size of a multi-head pooling module with d_r heads will be $2 \times d_h \times d_r$.

Here we proposed an alternative of the aforementioned multi-head self-attentive mechanism which can produce fixed-dimensional output with different numbers of attention heads. To control the output dimension, after computing the annotation matrix \mathbf{A} with Equation 4.7, the original hidden representation $\mathbf{H} \in \mathbb{R}^{d_h \times T}$ is transformed by:

$$\mathbf{C} = \mathbf{W}_c \mathbf{H}, \quad (4.11)$$

where \mathbf{W}_c is a matrix of size $\frac{d_h}{d_r} \times d_h$. Then the weighted mean $\mathbf{E}_r \in \mathbb{R}^{\frac{d_h}{d_r}}$ and weighted standard deviation $\mathbf{D}_r \in \mathbb{R}^{\frac{d_h}{d_r}}$ of the r -th attention head are computed as follows:

$$\begin{aligned} \mathbf{E}_r &= \mathbf{C} \mathbf{A}_r \\ \mathbf{D}_r &= \sqrt{\mathbf{C}^2 \mathbf{A}_r - (\mathbf{E}_r)^2} \end{aligned} \quad (4.12)$$

Under this mechanism the output size of every attention head is $2 \times \frac{d_h}{d_r}$. Therefore the final output size of the multi-head module is always $2 \times d_h$, which is the same as that in a single-head self-attention module.

4.3.4 Sub-vector multi-head self-attentive speaker embeddings

In the standard multi-head self-attention mechanism discussed in section 4.3.2, every head takes the same hidden representation \mathbf{H} as input, and a penalty term is introduced to encourage diversity of heads. Sub-vector multi-head self-attention employs a different strategy to learn dissimilar heads: it makes different heads focus on different parts of the hidden vector \mathbf{H} .

Suppose we have $d_r > 1$ attention heads. Similar to Equation 4.6, the hidden representation \mathbf{H} of size $d_h \times T$ is equally split into d_r sub-matrices $\{\mathbf{H}_1, \dots, \mathbf{H}_{d_r}\}$, where the size of \mathbf{H}_r is $\frac{d_h}{d_r} \times T$. The output annotation vector of the r -th head is computed as :

$$\mathbf{A}_r = \text{softmax}(g(\mathbf{H}_r^T \mathbf{W}_1) \mathbf{w}_{2,r}), \quad (4.13)$$

where \mathbf{W}_1 is a matrix of size $\frac{d_h}{d_r} \times d_a$; $\mathbf{w}_{2,r}$ is a vector of size d_a for the r -th head. The weighted mean and weighted standard deviation are then obtained in the same way as Equation 4.7 and Equation 4.8

In sub-vector multi-head self-attention, every head takes different part of the hidden representation as input, so their output embeddings are dissimilar by nature.

4.4 Performance

The proposed self-attentive speaker embedding learning methods are deployed using the same architecture and training scheme as in the baseline system. The performance of various embedding learning methods are summarized in Table 4.1 and Table 4.2.

In the following results, ‘*baseline*’ refers to the ResNet baseline described in Section 3. The label ‘*attn-k*’ denotes the self-attentive embedding systems described in Section 4.3.2 with k attention heads. Label ‘*fs-attn-k*’ denotes the systems described in Section 4.3.3 with k fixed-sized attention heads. Label ‘*sub-attn-k*’ denotes the systems described in Section 4.3.4 with k sub-vector attention heads.

Compared to the baseline system without any attention modules, the single-head attention model obtains consistent improvement on all evaluation sets.

Table 4.1: *Results on VoxCeleb1-O, VoxCeleb1-E and VoxCeleb1-H with various self-attentive embedding learning methods.*

System	Penalty	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H	
		EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
baseline	-	1.12	0.151	1.34	0.143	2.38	0.222
attn-1	-	1.12	0.125	1.33	0.140	2.33	0.228
attn-2	✗	1.14	0.116	1.31	0.136	2.33	0.220
attn-4	✗	1.17	0.121	1.33	0.141	2.39	0.227
attn-8	✗	1.22	0.118	1.34	0.135	2.38	0.230
attn-2	✓	1.10	0.112	1.28	0.133	2.31	0.217
attn-4	✓	1.07	0.109	1.24	0.130	2.26	0.206
attn-8	✓	1.02	0.105	1.22	0.125	2.16	0.205
fs-attn-2	✗	1.16	0.119	1.34	0.137	2.32	0.215
fs-attn-4	✗	1.20	0.124	1.33	0.136	2.37	0.231
fs-attn-8	✗	1.22	0.124	1.37	0.139	2.41	0.230
fs-attn-2	✓	1.12	0.120	1.32	0.136	2.33	0.212
fs-attn-4	✓	1.10	0.118	1.28	0.134	2.31	0.210
fs-attn-8	✓	1.10	0.115	1.25	0.133	2.26	0.208
sub-attn-2	-	1.04	0.117	1.31	0.132	2.32	0.212
sub-attn-4	-	0.99	0.110	1.24	0.126	2.23	0.217
sub-attn-8	-	0.93	0.112	1.20	0.124	2.12	0.203

Multi-head systems

For the standard multi-head attention systems without the penalty term, the 2-head system achieves comparable performance to the single-head system, and the performance becomes progressively worse when the number of heads increases on the VoxCeleb dataset. Similar results are also obtained on the SITW dataset.

After introducing the penalty term, the standard multi-head systems obtain significant improvements with more attention heads. The 2-head system is 2% better in EER and 10% better in minDCF on VoxCeleb. The best performance is achieved by the 8-head system, which outperforms the single-head system in EER by 9% on VoxCeleb and 6% on SITW.

From the results of standard multi-head systems, we can see that the penalty term plays an important role, especially when the number of heads increases. It also confirms the existence of the redundancy problem in the standard multi-head mechanism. The penalty term explicitly encourages each head to produce dissimilar weights, leading to

Table 4.2: *Results on SITW evaluation set with various self-attentive embedding learning methods.*

System	Penalty	EER	minDCF
baseline	-	1.73	0.166
attn-1	-	1.72	0.156
attn-2	✗	1.72	0.162
attn-4	✗	1.77	0.166
attn-8	✗	1.75	0.168
attn-2	✓	1.70	0.158
attn-4	✓	1.61	0.157
attn-8	✓	1.62	0.152
fs-attn-2	✗	1.73	0.166
fs-attn-4	✗	1.78	0.167
fs-attn-8	✗	1.77	0.163
fs-attn-2	✓	1.72	0.164
fs-attn-4	✓	1.72	0.162
fs-attn-8	✓	1.68	0.158
sub-attn-2	-	1.77	0.164
sub-attn-4	-	1.69	0.157
sub-attn-8	-	1.64	0.156

more informative embeddings and improving the multi-head systems’ performance.

Fixed-sized multi-head systems

The fixed-sized multi-head mechanism uses the same way as the standard multi-head mechanism to compute the attention weights. When producing embeddings, it adjusts the input feature dimension to make the size of the final aggregate output fixed regardless of the number of attention heads.

The penalty term is still crucial in the fixed-sized multi-head mechanism. On most VoxCeleb evaluation datasets, 4 and 8-head systems without penalty terms are worse than the 2-head system, and even worse than the single head system. Similar results are also obtained on the SITW dataset. By introducing the penalty term, 4 and 8-head systems outperform the 2-head system on all evaluation datasets. The best 8-head system is 5% better in EER than the 2-head system on VoxCeleb, 3% better in EER and 5% better in minDCF than the 2-head system on SITW.

Compared to the standard multi-head systems, the fixed-sized multi-head systems

perform worse on all evaluation datasets. When the multi-head systems are trained with the penalty term, standard 2, 4 and 8-head systems are 2%, 3% and 8% better than the corresponding fixed-sized multi-head systems respectively in EER on the VoxCeleb datasets.

Sub-vector multi-head systems

Sub-vector multi-head attention systems obtain consistent improvement over the single-head attention system with an increasing number of heads on all evaluation datasets. The best sub-vector multi-head attention system with 8 heads outperforms the single-head system by 17% on VoxCeleb1, 10% on VoxCeleb1-E, 9% on VoxCeleb1-H, and 5% on SITW in terms of EER.

The sub-vector multi-head mechanism provides different information to each head and guarantees that the heads learn different embeddings from different information.

Speaker classification performance

Table 4.3: *Speaker classification results of various self-attentive embedding learning methods.*

System	Penalty	Accuracy(%)
baseline	-	93.75
attn-1	-	94.31
attn-2	✗	94.53
attn-4	✗	93.75
attn-8	✗	94.12
attn-2	✓	95.02
attn-4	✓	96.09
attn-8	✓	96.09
fs-attn-2	✗	94.33
fs-attn-4	✗	93.66
fs-attn-8	✗	93.61
fs-attn-2	✓	94.66
fs-attn-4	✓	94.61
fs-attn-8	✓	95.13
sub-attn-2	-	93.71
sub-attn-4	-	95.08
sub-attn-8	-	95.82

We also report the system performance on the speaker classification task. In the training stage, the models are trained under the speaker classification framework, and the classification accuracy is reported on the validation dataset, containing 110,000 utterances from 6112 speakers. The results are summarized in Table 4.3.

The classification accuracy is consistent with the speaker verification performance in general. The incorporation of the self-attention module helps improve the classification results compared to the baseline. For multi-head and fixed-sized multi-head systems, when the number of attention heads increases, the classification accuracy drops without the penalty term. After introducing the penalty term, the accuracy keeps growing. For sub-vector multi-head systems, the classification performance is gradually improved with more attention heads.

Model complexity

Table 4.4: *Model complexity of various self-attentive embedding learning methods.*

System	Size(M)	FLOPs(G)
baseline	33	4.26
attn-1	34	4.26
attn-2	39	4.27
attn-4	50	4.29
attn-8	72	4.33
fs-attn-2	35	4.26
fs-attn-4	35	4.26
fs-attn-8	35	4.26
sub-attn-2	33	4.26
sub-attn-4	33	4.26
sub-attn-8	33	4.26

We measure the model complexity with two metrics: model size and the number of floating point operations (FLOPs). Model size is the total number of parameters, and FLOPs represent the amount of computation required in a forward pass. The statistical results are shown in Table 4.4.

For the standard multi-head attention systems, the model size grows quickly as the number of attention heads increases. Fixed-size multi-head systems and sub-vector multi-head systems keep a relatively stable model complexity level with more attention heads.

Table 4.5: Average $O(A)$ of various multi-head systems.

System	Penalty	Average $O(A)$
attn-2	✗	0.691
attn-4	✗	0.514
attn-8	✗	0.313
attn-2	✓	0.713
attn-4	✓	0.582
attn-8	✓	0.549

In general, all the sub-vector multi-head systems maintain similar model complexity compared to the baseline system, while greatly improving the performance on speaker classification and speaker verification tasks.

4.5 Analysis

4.5.1 Orthogonality of attention weights

In multi-head self-attention systems, we have adopted various strategies to learn dissimilar representations from different heads, such as introducing penalty terms, and using sub-vector attention. To illustrate the diversity of heads learned in these multi-head systems, here we introduce a metric to measure the orthogonality of the weight vectors. Given a weight matrix A , where every column of A represents a weight vector produced by a single attention head, the gram matrix of A can be computed with $G = A^T A$, and the metric is defined as:

$$O(A) = \frac{\text{Trace}(G)}{\sum_{i,j} |G_{ij}|}, \quad (4.14)$$

where $|\cdot|$ represents the absolute value. When weight vectors of A are orthogonal, $O(A)$ achieves its maximum value of 1; when all weight vectors are the same in A , $O(A)$ achieves its minimum value of $\frac{1}{d_r}$, where d_r is the number of weight vectors in A .

Since heads in the sub-vector attention system are computed from different inputs, computing the diversity of heads makes little sense. Here we only compute the mean $O(A)$ over the whole training dataset and the results are summarized in Table 4.5.

From the results, we can see that penalty terms can improve the orthogonality of the attention weights compared to the standard multi-head systems. Besides, for systems with the same number of heads, larger $O(A)$ is, (i.e., more diverse attention weights),

better performance on the evaluation datasets is obtained.

4.5.2 Attention weights versus phonetic classes

The attention mechanism learns frame-level weights and produces weighted statistics as speaker embeddings. It is a powerful technique that offers a way to obtain a more discriminative utterance-level representation. The effectiveness of self-attentive embeddings has been demonstrated in the previous evaluation tasks, but there have been few studies explaining what the attention mechanism learns and how it helps to improve SV system performance.

Here we further investigate what is learned in the attention model. It is intuitive to assume that frames assigned with higher weights are related to certain phonetic classes which can be more discriminative among speakers. We generate the phonetic alignments using an English automatic speech recognition (ASR) model and visualize the attention weights distribution over the phonemes of Table 4.6 in a heat-map in Fig.4.2. The x-axis represents phonemes and the y-axis represents speaker indices.

Table 4.6: *Phonemes*

Vowels	Front		ae; ah; eh; iy; ih
	Middle		aa; aw; er
	Back		ow; uh; uw
Semivowels	Glides		w; y
	Liquids		l; r
Diphthongs			ay; ey; oy
Consonants	Nasals		m; n; ng
	Plosives	Voiced	b; d; g
		Unvoiced	k; p; t
	Fricatives	Voiced	dh; dx; v; z
		Unvoiced	f; hh; s; sh; th
	Affricates		ch; jh

The data used in the visualization include train-clean-100 and train-clean-360 sets from Librispeech [61], for a total of 460 hours of speech from 1172 speakers. The ASR model is the DNN model trained by following the Kaldi recipe¹. The SV model used for illustration is the single head attention model.

For each utterance, the ASR model is used to align the phoneme label for every frame, and the attentive SV model is used to compute the weight for each frame. We can get a

¹<https://github.com/kaldi-asr/kaldi/tree/master/egs/librispeech/s5>

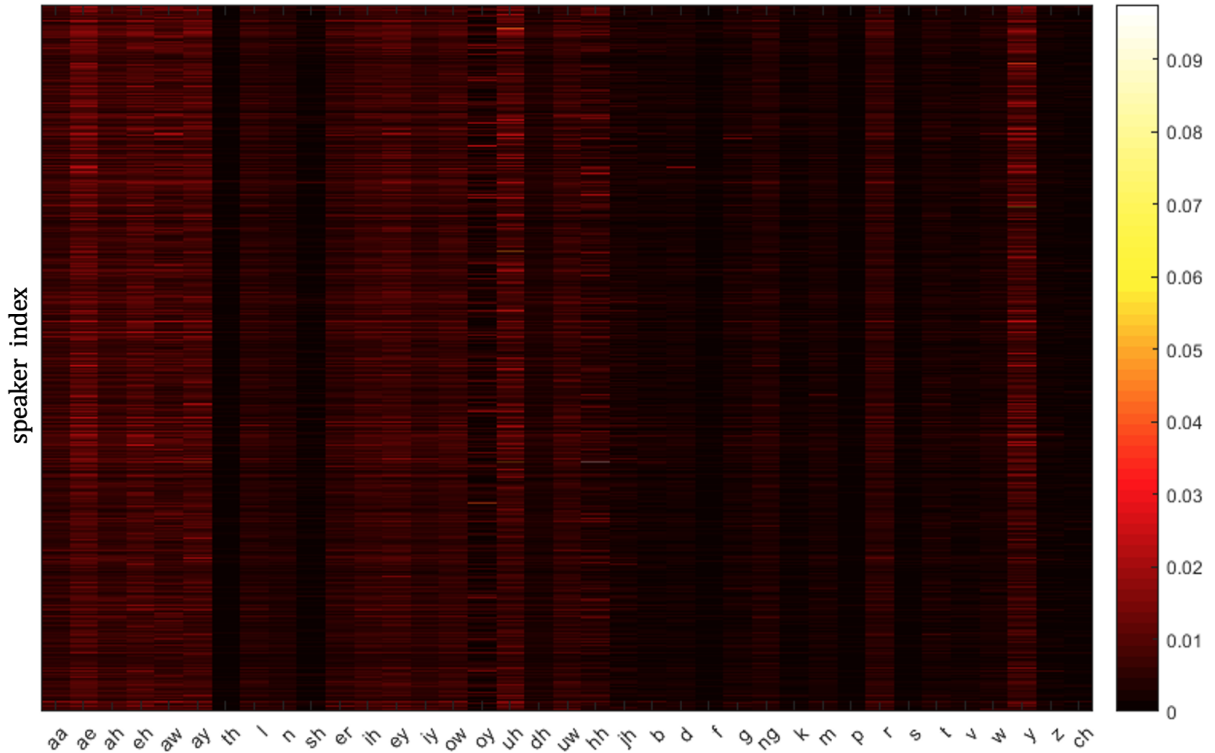


Figure 4.2: Heat-map of phonemes in the attentive speaker verification system.

list of phonemes and their corresponding weights for every utterance. Then we collect all the phonemes appearing in all utterances of a speaker, and compute the average weight for each phoneme for the speaker. The head-map is constructed as follows: the i -th row represents the average attention weights of the 39 phonemes for the i -th speaker.

Lighter parts in the heat-map mean higher weights assigned by the attention model. The complete black columns correspond to phonemes that are not present in the training data. We can see that the positions of the lighter part in each column are quite consistent across different rows. It means the attention model tends to focus on some phonemes in verifying speakers' identity. Vowels such as *ae*, *ah*, *eh*, *aw*, *ay*, *uh* and semivowels such as *r*, *y* are among the most attended phonemes. Vowels and semivowels are assigned larger weights in the attention system, meaning that they are more discriminative when determining the identity of a speaker.

Chapter 5

Bayesian Self-attentive Speaker Embedding Learning

In this Chapter, we generalize the multi-head attention in the Bayesian framework, where the standard deterministic multi-head attention can be viewed as a special case. In the Bayesian attention framework, parameters of each attention head share a common distribution, and the updates of these parameters are related, instead of being independent as in deterministic multi-head attention. Based on our framework, the attention redundancy problem is alleviated by performing Bayesian inference on attention parameters with the Stein variational gradient descent [62, 63]. During the optimization process, each attention head is forced to be far from each other in the parameter space. Besides, the Bayesian attention framework also provides a theoretical understanding of the benefits of applying multi-head attention.

5.1 Theory of Stein Variational Gradient Descent

Variational inference approximates the target distribution using a parameterized distribution by minimizing their Kullback-Leibler (KL) divergence. Current optimization techniques such as stochastic gradient descent can solve variational methods efficiently on large datasets. However, the choice of parameterized distribution for the approximation is critical. Simple distribution sets may not suffice to approximate the real posterior distributions, while sophisticated choices lead to high computation cost and optimization difficulties. Therefore, the parameterized distribution sets and the corresponding optimization algorithms have to be designed on a case-by-case basis.

[62] proposed a general variational inference algorithm that can be applied in various machine learning tasks. It performs a form of functional gradient descent on a set of particles to minimize the KL divergence between a tractable distribution and the target distribution, and drives the particles to fit the true posterior distribution. When there are multiple particles, the algorithm is a full Bayesian approach, and it degrades to gradient descent for maximizing the posterior distribution when only a single particle is used.

In this section, we briefly introduce the theory of variational inference with Stein variational gradient descent.

5.1.1 Stein’s identity and Stein discrepancy

The Bayesian self-attention framework in this work is based primarily on the Stein’s identity and Stein discrepancy.

Let $p(x)$ be a continuously differentiable distribution defined on $\mathcal{X} \subseteq \mathbf{R}^d$, and $\phi(x) = [\phi_1(x), \dots, \phi_d(x)]^T$ a smooth vector function. The Stein operator \mathcal{A}_p of the distribution p is defined as:

$$\mathcal{A}_p\phi(x) = \phi(x)\nabla_x \log p(x)^T + \nabla_x\phi(x). \quad (5.1)$$

The Stein operator of the distribution p takes the function $\phi(x)$ as the input variable, and for certain sufficiently regular $\phi(x)$, its expectation under $x \sim p$ equals to zero:

$$\mathbf{E}_{x \sim p}[\mathcal{A}_p\phi(x)] = 0, \text{ s.t. } \lim_{\|x\| \rightarrow \infty} \phi(x)p(x) = 0. \quad (5.2)$$

Equation 5.2 is the *Stein’s identity* statement. When the Stein identity holds, the function $\phi(x)$ is considered to be in the Stein class of distribution p .

If there is a different smooth distribution $q(x)$ also defined on \mathcal{X} , then the expectation of the Stein operator of the distribution p under $x \sim q$, $\mathbf{E}_{x \sim q}[\mathcal{A}_p\phi(x)]$, would no longer equal to zero for general ϕ . The magnitude of $\mathbf{E}_{x \sim q}[\mathcal{A}_p\phi(x)]$ can be used to demonstrate the difference between distributions p and q . Based on this property, the *Stein discrepancy* is defined as the maximum violation of the Stein’s identity for ϕ in some proper function set \mathcal{F} :

$$\mathbf{S}(q, p) = \max_{\phi \in \mathcal{F}} \{[\mathbf{E}_{x \sim q} \text{trace}(\mathcal{A}_p\phi(x))]^2\}. \quad (5.3)$$

The choice of the function set \mathcal{F} is critical. Simple functions will restrict the discriminative ability of the discrepancy measure, while complex choices will make subsequent computation difficult. To avoid this issue and simplify the optimization process, [64] uses kernelized Stein discrepancy (KSD) that maximizes ϕ in the unit ball of a reproducing kernel Hilbert space (RKHS). Following [64], KSD is defined as

$$\mathbf{S}(q, p) = \max_{\phi \in \mathcal{H}^d} \{[\mathbf{E}_{x \sim q} \text{trace}(\mathcal{A}_p \phi(x))]^2\}, \text{ s.t. } \|\phi\|_{\mathcal{H}^d} \leq 1, \quad (5.4)$$

where the kernel $k(x, x')$ of RKHS \mathcal{H} is in the Stein class of distribution p for any $x' \in \mathcal{X}$. The optimal solution of Equation 5.4 has a closed form as follows:

$$\begin{aligned} \phi(x) &= \frac{\phi_{q,p}^*(x)}{\|\phi_{q,p}^*(x)\|_{\mathcal{H}^d}} \\ \phi_{q,p}^*(x) &= \mathbf{E}_{x \sim q}[\mathcal{A}_p k(x, \cdot)], \end{aligned} \quad (5.5)$$

where $k(x, \cdot)$ is the kernel. $\mathbf{S}(q, p) = 0$ if and only if $p = q$ once $k(x, x')$ is strictly positive definite, and the condition is satisfied by common kernels.

5.1.2 Variational inference with Stein variational gradient descent

The standard variational inference process can be formulated in the following way:

$$q^* = \arg \min_{q \in \mathcal{Q}} \{KL(q||p)\}, \quad (5.6)$$

where $p(x)$ is the target distribution, and $q^*(x)$ is a simpler distribution found in a pre-defined distribution set $\mathcal{Q} = q(x)$. The pre-defined set \mathcal{Q} needs to satisfy the need to approximate a large class of target distributions, meanwhile it should be tractable so that the optimization problem can be solved efficiently.

Here we choose the set \mathcal{Q} to be the set of distributions of random variables with the form $z = \mathcal{T}(x)$, where $\mathcal{T} : \mathcal{X} \rightarrow \mathcal{X}$ is a smooth one-to-one transformation, and x is drawn from a tractable initial distribution $q_0(x)$. If we consider an incremental transform $\mathcal{T}(x) = x + \epsilon\phi(x)$, which is a perturbation of the identity map, where $\phi(x)$ is a smooth function that decides the perturbation direction and the scalar ϵ decides the magnitude. According to the inverse function theorem, the map \mathcal{T} is a one-to-one mapping if $|\epsilon|$ is sufficiently small.

Let $q_{[\mathcal{T}]}(z)$ be the density function of $z = \mathcal{T}(x)$ when $x \sim q(x)$. The derivative of the KL divergence w.r.t the perturbation magnitude ϵ can be calculated as

$$\nabla_{\epsilon} KL(q_{[\mathcal{T}]||p})|_{\epsilon=0} = -\mathbf{E}_{x \sim q}[\text{trace}(\phi(x)\nabla_x \log p(x)^T + \nabla_x \phi(x))]. \quad (5.7)$$

According to the Stein operator definition in Equation 5.1, Equation 5.7 can be reformulated as

$$\nabla_{\epsilon} KL(q_{[\mathcal{T}]||p})|_{\epsilon=0} = -\mathbf{E}_{x \sim q}[\text{trace}(\mathbf{A}_p \phi(x))]. \quad (5.8)$$

To find the optimal perturbation direction that gives the deepest descent on the KL divergence, we need to minimize the left part of Equation 5.8, or to maximize the following:

$$\max_{\phi} \mathbf{E}_{x \sim q}[\text{trace}(\mathbf{A}_p \phi(x))]. \quad (5.9)$$

If we restrict the optimization of Equation 5.9 in zero-centered balls of \mathcal{H}^d , it is exactly the same as the KSD Equation 5.4. So the optimal solution has the form in Equation 5.5, for which the gradient equals to the negative KSD, i.e., $\nabla_{\epsilon} KL(q_{[\mathcal{T}]||p})|_{\epsilon=0} = -\mathbf{S}(q, p)$.

This finding reveals a procedure that an initial reference distribution q_0 can be iteratively transformed to the target distribution p . We can start with an initial transform $\mathcal{T}_0^* = x + \epsilon_0 \phi_{q_0, p}^*(x)$ applied on q_0 , where ϵ_0 is the step size, and the KL divergence is decreased by an amount of $\epsilon_0 \mathbf{S}(q_0, p)$. After that a new distribution $q_1(x) = q_{0[\mathcal{T}_1]}(x)$ is generated, and we can continue applying a new transform $\mathcal{T}_1^* = x + \epsilon_1 \phi_{q_1, p}^*(x)$ on $q_1(x)$, and the KL divergence is further decreased by $\epsilon_1 \mathbf{S}(q_1, p)$. By repeating this process, we can construct a sequence of distributions $\{q_l\}_{l=1}^n$ between the initial distribution q_0 and the target distribution p :

$$\begin{aligned} q_{l+1} &= q_{l[\mathcal{T}_l^*]} \\ \mathcal{T}_l^*(x) &= x + \epsilon_l \cdot \phi_{q_l, p}^*(x). \end{aligned} \quad (5.10)$$

This eventually converges to the target distribution p with sufficiently small step-size, under which $\phi_{q_{\infty}, p}^*(x) \equiv 0$, and the transform $\mathcal{T}_{\infty}^*(x)$ reduces to the identity map.

In summary, the Stein variational gradient descent method takes a target distribution with density function $p(x)$ and a set of initial particles $\{x_i^0\}_{i=1}^n$ as inputs, and then produces a set of particles $\{x_i\}_{i=1}^n$ that approximate the target distribution by iteratively updating the particles as follows:

$$\begin{aligned} x_i^{l+1} &\leftarrow x_i^l + \epsilon_l \hat{\phi}^*(x_i^l) \\ \hat{\phi}^*(x) &= \frac{1}{n} \sum_{j=1}^n \left[k(x_j^l, x) \nabla_{x_j^l} \log p(x_j^l) + \nabla_{x_j^l} k(x_j^l, x) \right] \end{aligned} \quad (5.11)$$

where l is the iteration index and ϵ_l is the step size at the l -th iteration.

The procedure transports a set of points to match the target distribution $p(x)$, and does not depend on the initial distribution $q(0)$. There are two terms in $\phi^*(x)$ of Equation 5.11. The first term is a weighted sum of gradients of all the points, and it drives the particles towards the high probability areas of $p(x)$. The second term drives x away from its neighboring points, preventing all the points to collapse together. For the extreme case when we have only a single particle, i.e., $n=1$, the procedure in Equation 5.11 reduces to the conventional gradient ascent for maximum a posteriori (MAP) estimation for any kernel that satisfies $\nabla_x k(x, x) = 0$.

5.2 Generalization of Self-attentive Speaker Embedding Learning under the Bayesian Framework

In this section, we interpret the attention mechanism from the Bayesian inference perspective, and propose an optimization method that learns repulsive multi-head attention under the Bayesian framework.

5.2.1 Bayesian inference perspective of the attention mechanism

Let \mathbf{x} be the input feature and \mathbf{e} be the output embedding of the attention model. In single-head attention, we have $\mathbf{e} = f_{attn}(\mathbf{x}; \theta)$, where θ represents the attention parameters. In multi-head attention, we have multiple attention mappings, each computed from independent parameters. Attention mapping from different heads finally aggregate via a function $g(\cdot)$ as

$$\begin{aligned} \mathbf{e}_i &= f_{attn}(\mathbf{x}; \theta_i), \\ \mathbf{e} &= g(\mathbf{e}_1, \dots, \mathbf{e}_M), \end{aligned} \tag{5.12}$$

where M is the number of heads, and θ_i represents independent parameters for each attention head.

We generalize the deterministic transformation of \mathbf{e} into a stochastic generative process:

$$\mathbf{e} = f_{attn}(\mathbf{x}; \theta), \text{ with } \theta \sim p(\theta|\mathcal{D}) \tag{5.13}$$

where \mathcal{D} is the training data set. Bayesian inference for attention computes the distribution of the output embedding $p(\mathbf{e}|\mathbf{x}, \mathcal{D})$ for a new input \mathbf{x} and the training data set \mathcal{D} by

$$p(\mathbf{e}|\mathbf{x}, \mathcal{D}) = \int \delta_{f_{\text{attn}}(\mathbf{x}, \theta)}(\mathbf{e})p(\theta|\mathcal{D})d\theta, \quad (5.14)$$

where $\delta_x(\cdot)$ is the delta function with point mass at x . In practice, we adopt sampling methods to approximate Equation 5.14 instead of computing the integral. $p(\mathbf{e}|\mathbf{x}, \mathcal{D})$ is approximated by a set of M samples initialized from $p(\theta|\mathcal{D})$:

$$\begin{aligned} \mathbf{e}_i &= f_{\text{attn}}(\mathbf{x}; \theta_i), \text{ with } \theta_i \sim p(\theta|\mathcal{D}), \\ \mathbf{e} &= g(\mathbf{e}_1, \dots, \mathbf{e}_M). \end{aligned} \quad (5.15)$$

Equation 5.15 provides a Bayesian perspective of the multi-head attention framework. It is a more general formulation: if all parameters θ_i are independent of each other (that is, if they do not share a distribution $p(\theta|\mathcal{D})$), then Equation 5.15 is reduced to the deterministic multi-head attention as described in Equation 5.12.

5.2.2 Repulsive multi-head attention learning

In the standard deterministic multi-head attention framework, parameters of each attention head are treated independently. One long-standing challenge is making each attention head learn distinct representations, thus, reducing the information redundancy among the heads.

In the Bayesian attention framework, the attention parameters share a common distribution $p(\theta|\mathcal{D})$ defined over the observed dataset $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^N$, and we adopt the Stein variational gradient descent method to develop repulsive attentions. The Stein variational gradient descent takes a target distribution with density function $p(x)$ and a set of initial particles $\{x_i^0\}_{i=1}^n$ as inputs, and produces a new set of particles $\{x_i\}_{i=1}^n$ that approximate the target distribution by iteratively updating the particles in a principled procedure.

In our case, parameters of each head θ_i are considered as one particle. Thus, in a multi-head system with M heads, there will be M particles, $\{\theta_i\}_{i=1}^M$, which are updated iteratively to approximate the posterior distribution $p(\theta|\mathcal{D})$ according to Equation 5.11. The learning algorithm is illustrated in Algorithm 1.

Algorithm 1 is basically the same as standard stochastic gradient descent (SGD) except that the update for attention parameters is replaced by the Stein variational gradient

Algorithm 1: Repulsive multi-head attention learning

Data: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$

Input: Initial attention parameters with M heads $\Theta^0 = \{\theta_m^0\}_{m=1}^M$, all other model parameters Ω^0

Output: Optimized parameters $\hat{\Theta}$ and $\hat{\Omega}$

```
1 for iteration  $l$  do
2   forward the input data  $(\mathbf{x}_i, y_i)$  and get the prediction  $\hat{y}_i$ 
3   calculate the loss  $\mathcal{L}(\hat{y}_i, y_i)$  and the gradient of loss w.r.t  $\Omega^l$ :  $\varphi(\Omega^l) \leftarrow \nabla_{\Omega^l} \mathcal{L}$ 
4   for attention head  $m$  do
5      $\phi(\theta_m^l) = \frac{1}{M} \sum_{j=1}^M [k(\theta_j^l, \theta_m^l) \nabla_{\theta_j^l} \log p(\theta_j^l) + \nabla_{\theta_j^l} k(\theta_j^l, \theta_m^l)]$ 
6      $\varphi(\theta_m^l) \leftarrow \epsilon_l \phi(\theta_m^l)$ 
7   end
8   update parameters
9    $\Theta^{l+1} \leftarrow \text{Optimizer}(\Theta^l, \varphi(\Theta^l))$ 
10   $\Omega^{l+1} \leftarrow \text{Optimizer}(\Omega^l, \varphi(\Omega^l))$ 
11 end
```

descent in Equation 5.11. Therefore, the learning algorithm can be easily integrated into existing optimizers and used to train the model efficiently.

By adopting the Stein variational gradient descent, the updates of the attention parameters of different heads are related to each other. We have discussed the two terms of $\phi^*(x)$ in Equation 5.11: the first term pushes the particles to the high probability areas of $p(\theta|\mathcal{D})$, while the second term performs repulsive constraint and prevents all the particles from collapsing together into the local mode of $p(\theta|\mathcal{D})$. Therefore, the algorithm explicitly encourages repulsiveness between heads during the training process.

Besides, when the number of head $M=1$ and the kernel satisfies $\nabla_x k(x, x) = 0$ such as the RBF kernel, Algorithm 1 reduces to the standard SGD.

5.3 Performance

The Bayesian self-attentive speaker embedding learning systems have the same architecture as the deterministic multi-head self-attentive systems described in Section 4.3.3 and 4.3.4. The only difference is that when training Bayesian systems, Stein variational gradient descent method described in Algorithm 1 is applied to optimize the systems, while the deterministic multi-head self-attentive systems use standard SGD for training. The reason we choose only fixed-sized and sub-vector multi-head systems is that their model complexity remain stable with more attention heads, which is preferred in practice.

In the following results, ‘*baseline*’ refers to the ResNet baseline described in Section 3. The label ‘*attn-1*’ denotes the self-attentive embedding systems with single attention head. Labels ‘*Bayesian fs-attn-k*’ and ‘*Bayesian sub-attn-k*’ denote the Bayesian version of multi-head self-attentive speaker embedding systems with k attention heads described in Section 4.3.3 and sub-vector multi-head self-attentive speaker embedding systems with k attention heads described in Section 4.3.4, respectively.

The performances are summarized in Table 5.1 and Table 5.2.

Table 5.1: *Results on VoxCeleb1-O, VoxCeleb1-E and VoxCeleb1-H with Bayesian self-attentive embedding learning methods.*

System	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H	
	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
baseline	1.12	0.151	1.34	0.143	2.38	0.222
attn-1	1.12	0.125	1.33	0.140	2.33	0.228
Bayesian fs-attn-2	0.99	0.112	1.25	0.126	2.26	0.210
Bayesian fs-attn-4	0.91	0.113	1.18	0.122	2.18	0.205
Bayesian fs-attn-8	0.83	0.106	1.16	0.124	2.06	0.196
Bayesian sub-attn-2	1.02	0.117	1.26	0.127	2.26	0.211
Bayesian sub-attn-4	0.99	0.116	1.20	0.124	2.22	0.203
Bayesian sub-attn-8	1.03	0.108	1.20	0.125	2.10	0.202

Table 5.2: *Results on SITW evaluation set with Bayesian self-attentive embedding learning methods.*

System	EER	minDCF
baseline	1.73	0.166
attn-1	1.72	0.156
Bayesian fs-attn-2	1.71	0.161
Bayesian fs-attn-4	1.61	0.158
Bayesian fs-attn-8	1.57	0.152
Bayesian sub-attn-2	1.74	0.159
Bayesian sub-attn-4	1.70	0.157
Bayesian sub-attn-8	1.66	0.157

The Bayesian self-attentive embeddings greatly improve the performance compared to the baseline and self-attentive embeddings with a single attention head.

The Bayesian system with two heads improves the baseline by 12% in EER and 26% in minDCF on VoxCeleb1. When the number of heads increases, the performance of Bayesian fixed-sized multi-head systems keeps improving, and the best system with 8

heads is 26% better in EER and 30% better in minDCF on VoxCeleb1, 9% better in EER and 8% better in minDCF on SITW.

Similar results are also obtained in the sub-vector multi-head case. In general, the performance of Bayesian sub-vector multi-head systems keeps improving with increasing number of attention heads. The best Bayesian sub-vector multi-head system with 8 heads improves the baseline by 8% in EER and 28% in minDCF on VexCeleb1, 4% in EER and 5% in minDCF on SITW.

We also find that Bayesian fixed-sized multi-head systems consistently outperform sub-vector systems when they have the same number of attention heads.

Speaker classification performance

Table 5.3: *Speaker classification results of Bayesian self-attentive embedding systems.*

System	Accuracy(%)
baseline	93.75
attn-1	94.31
Bayesian fs-attn-2	94.88
Bayesian fs-attn-4	96.05
Bayesian fs-attn-8	96.88
Bayesian sub-attn-2	94.33
Bayesian sub-attn-4	94.86
Bayesian sub-attn-8	95.33

Besides the speaker verification performance, we also report the performance on the speaker classification task. The accuracy is computed on the validation dataset, containing 110,000 utterances from 6112 speakers. The results are summarized in Table 5.3.

The speaker classification performance is consistent with the verification performance. Under the Bayesian fixed-sized multi-head attention framework, the system obtains better classification accuracy with more attention heads. And when the number of heads are the same, the Bayesian fixed-sized multi-head systems have better performance than Bayesian sub-vector multi-head systems. The best Bayesian fixed-sized multi-head system with 8 attention heads reduces the classification error rate by over 50% compared to the baseline.

Model complexity

The model complexity measured in model size and floating point operations (FLOPs) are summarized in Table 5.4. Since the Bayesian learning algorithm only affects the back-propagation pass in the training stage, the model size and FLOPs of all the Bayesian models are the same as their corresponding deterministic versions.

We can see that Bayesian self-attentive embedding learning methods greatly improve the performance of multi-head systems on speaker classification and verification tasks. In the mean time they maintain comparable model complexity in the evaluation stage.

5.4 Comparison to Deterministic Self-attentive Speaker Embeddings

We summarize the performance of all the self-attentive speaker embedding systems in Table 5.5 and Table 5.6. Note that in Table 5.5, “*Pen.*” stands for “penalty” and “*B.*” stands for “Bayesian”.

By adopting Stein variational gradient descent, Bayesian multi-head attention systems achieve significant improvements compared to their corresponding deterministic multi-head systems. When the number of heads increases to 4 and 8, the performances of the deterministic attention systems degrade on all VoxCeleb1-O evaluation datasets compared to the single-head attention system, while the 4-head Bayesian system outperforms the single-head attention system by 19% on VoxCeleb1-O, 11% on VoxCeleb1-E and 6% on VoxCeleb1-H in terms of EER, and the 8-head Bayesian system outperforms the single-head attention system by 26% on VoxCeleb1-O, 13% on VoxCeleb1-E and 12% on

Table 5.4: *Model complexity of Bayesian self-attentive embedding learning systems.*

System	Size(M)	FLOPs(G)
baseline	33	4.26
attn-1	34	4.26
Bayesian fs-attn-2	35	4.26
Bayesian fs-attn-4	35	4.26
Bayesian fs-attn-8	35	4.26
Bayesian sub-attn-2	33	4.26
Bayesian sub-attn-4	33	4.26
Bayesian sub-attn-8	33	4.26

Table 5.5: Overall results on VoxCeleb1-O, VoxCeleb1-E and VoxCeleb1-H.

System	Pen.	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H	
		EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
baseline	-	1.12	0.151	1.34	0.143	2.38	0.222
attn-1	-	1.12	0.125	1.33	0.140	2.33	0.228
attn-2	✗	1.14	0.116	1.31	0.136	2.33	0.220
attn-4	✗	1.17	0.121	1.33	0.141	2.39	0.227
attn-8	✗	1.22	0.118	1.34	0.135	2.38	0.230
attn-2	✓	1.10	0.112	1.28	0.133	2.31	0.217
attn-4	✓	1.07	0.109	1.24	0.130	2.26	0.206
attn-8	✓	1.02	0.105	1.22	0.125	2.16	0.205
fs-attn-2	✗	1.16	0.119	1.34	0.137	2.32	0.215
fs-attn-4	✗	1.20	0.124	1.33	0.136	2.37	0.231
fs-attn-8	✗	1.22	0.124	1.37	0.139	2.41	0.230
fs-attn-2	✓	1.12	0.120	1.32	0.136	2.33	0.212
fs-attn-4	✓	1.10	0.118	1.28	0.134	2.31	0.210
fs-attn-8	✓	1.10	0.115	1.25	0.133	2.26	0.208
sub-attn-2	-	1.04	0.117	1.31	0.132	2.32	0.212
sub-attn-4	-	0.99	0.110	1.24	0.126	2.23	0.217
sub-attn-8	-	0.93	0.112	1.20	0.124	2.12	0.203
B. fs-attn-2	-	0.99	0.112	1.25	0.126	2.26	0.210
B. fs-attn-4	-	0.91	0.113	1.18	0.122	2.18	0.205
B. fs-attn-8	-	0.83	0.106	1.16	0.124	2.06	0.196
B. sub-attn-2	-	1.02	0.117	1.26	0.127	2.26	0.211
B. sub-attn-4	-	0.99	0.116	1.20	0.124	2.22	0.203
B. sub-attn-8	-	1.03	0.108	1.20	0.125	2.10	0.202

VoxCeleb1-H in terms of EER. On the SITW evaluation dataset, the best deterministic 8-head system is 6% better in EER than the single-head attention system, while Bayesian systems with 4 and 8 heads achieve 6% and 9% relative improvement in EER, respectively.

In general, Bayesian fixed-sized multi-head attention systems consistently outperform deterministic attention systems. The proposed algorithm helps learn repulsive attention heads, thus improving the speaker representation in multi-head attention systems.

For sub-vector systems, the Bayesian systems obtain comparable performance with their deterministic counterparts. Since every head in a sub-vector system already has a different input, the repulsive multi-head learning strategy is not deemed necessary.

Table 5.7 shows the performance of recent state-of-the-art SV systems on VoxCeleb1. Among all the systems, our proposed Bayesian fixed-sized multi-head attentive system achieves the best performance in most evaluation corpora.

Table 5.6: Overall results on SITW evaluation set.

System	Penalty	EER	minDCF
baseline	-	1.73	0.166
attn-1	-	1.72	0.156
attn-2	✗	1.72	0.162
attn-4	✗	1.77	0.166
attn-8	✗	1.75	0.168
attn-2	✓	1.70	0.158
attn-4	✓	1.61	0.157
attn-8	✓	1.62	0.152
fs-attn-2	✗	1.73	0.166
fs-attn-4	✗	1.78	0.167
fs-attn-8	✗	1.77	0.163
fs-attn-2	✓	1.72	0.164
fs-attn-4	✓	1.72	0.162
fs-attn-8	✓	1.68	0.158
sub-attn-2	-	1.77	0.164
sub-attn-4	-	1.69	0.157
sub-attn-8	-	1.64	0.156
Bayesian fs-attn-2	-	1.71	0.161
Bayesian fs-attn-4	-	1.61	0.158
Bayesian fs-attn-8	-	1.57	0.152
Bayesian sub-attn-2	-	1.74	0.159
Bayesian sub-attn-4	-	1.70	0.157
Bayesian sub-attn-8	-	1.66	0.157

Table 5.7: Performance of state-of-the-art systems on VoxCeleb1.

System	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H	
	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
E-TDNN	1.49	0.160	1.61	0.171	2.69	0.242
ResNet	1.31	0.154	1.38	0.163	2.50	0.233
ECAPA-TDNN	0.87	0.107	1.12	0.132	2.12	0.210
Bayesian attn-8	0.83	0.105	1.16	0.122	2.06	0.196

Chapter 6

Frequency-domain Pooling in Speaker Embedding Learning

The previous chapters work on the pooling step which produces utterance-level representations from a variable-length sequence of frame-level representations. The central idea is emphasizing on frames with discriminative information and neglecting others by weighting the frames proportionally. The self-attention mechanism is proposed to determine the weights, and it is further generalized to the Bayesian self-attention framework to learn effective multi-head attention weights.

In this chapter, we focus on the frame-level representation learning. CNN is the most commonly used network in frame-level modeling and has achieved great success. However, it lacks the ability of utilizing global contexts and capturing long-term interdependencies. We first incorporate channel attention modules into the current frame-level representation learning framework, and then analyze channel attention from the perspective of frequency analysis. We adopt discrete cosine transform (DCT) as the frequency analysis tool and show that global average pooling in channel attention is a special case of frequency-domain pooling where only the lowest frequency component is used. Finally we generalize the global average pooling in channel attention to frequency-domain pooling. Two frequency-domain pooling methods are proposed to utilize multiple frequency components.

The chapter is arranged in the following way: we revisit channel attention in Section 6.1 and briefly introduce frequency-domain learning in Section 6.2. In Section 6.3 we analyze channel attention from the perspective of frequency analysis, and propose two frequency-domain pooling methods for channel attention. Section 6.4 presents the experimental results and discussions.

6.1 Channel Attention

Convolutional neural networks (CNNs) have achieved great success in a variety of areas such as computer vision, natural language processing and speech processing, etc.

Convolution is the basic operation in CNNs. In each convolution layer, a set of filters are deployed to perform convolution on the inputs. They are trained to capture spatial and channel correlations by aggregating spatial-wise and channel-wise information within local receptive fields. By stacking layers and non-linear activation functions, CNNs are able to obtain broader receptive fields and learn high-level representations from the raw inputs.

To enhance the representation ability of CNNs, research has been conducted to make the networks go deeper and wider. [19] proposed to learn residual mappings instead of unreferenced underlying mappings, and skip connections were introduced to help optimize much deeper networks. [65–67] proposed a multi-path way to get wider networks. The inputs are split into different processing paths and each path uses filters of different sizes. The outputs from all paths are then concatenated and sent to subsequent layers. [68] generalized previous works on multi-path, and introduced a new dimension of CNNs besides depth and width, namely cardinality. Cardinality represents the size of independent paths in the network. The network performance can be improved more efficiently by increasing the cardinality compared to building deeper and wider architectures.

Another direction of improving CNNs is to investigate larger contexts. In many CNNs, the receptive fields are theoretically large enough to cover the entire input. However, [69] found that the effective size of receptive fields is actually much smaller. Besides, CNNs are inherently inefficient in modeling long-range interdependencies since the convolution operators are designed to learn local relations. To obtain rich global context information and learn long-range interdependencies, attention modules are often used in the intermediate layers of CNNs. [70] incorporated channel attention into networks with a new architecture named “Squeeze-and-Excitation”. It employs global pooling in-between the network and utilizes holistic information in all stages of a CNN in order to capture the relationship between channels.

Squeeze-and-Excitation (SE) units [70] have been widely used in various tasks and achieved significant improvements in performance for CNN. It also has been successfully applied in speaker verification frameworks [27] and gains great performance improvements.

The key idea of the SE units is to explicitly model the inter-dependencies between channels. It performs a global pooling step followed by a self-attention function on channels to adaptively recalibrate channel-wise features.

Suppose we have feature maps $\mathbf{F} \in \mathbb{R}^{C \times H \times W}$ obtained from a 2-dimensional convolutional layer, where C is the number of channels, and $H \times W$ is the size of the 2D feature map within each channel.

The *squeeze* step takes \mathbf{F} as input, and produces compressed channel descriptors $\mathbf{z} \in \mathbb{R}^C$ by applying global average pooling to each channel feature map:

$$\mathbf{z}_c = \text{GlobalPool}(\mathbf{F}_c) = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W \mathbf{F}_c(i, j), \quad (6.1)$$

where $\mathbf{F}_c \in \mathbb{R}^{H \times W}$ is the c -th channel feature map in \mathbf{F} .

The *excitation* step takes the channel descriptors \mathbf{z} as input and produces channel weights $\mathbf{s} \in \mathbb{R}^C$ by employing a channel attention mechanism:

$$\mathbf{s} = \sigma(\mathbf{W}_2 g(\mathbf{W}_1 \mathbf{z})), \quad (6.2)$$

where σ is the sigmoid activation and g is the ReLU function, $\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C}$, and $\mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}$. Basically \mathbf{W}_1 is a dimension reduction transform that reduces the input size from \mathbb{R}^C to $\mathbb{R}^{\frac{C}{r}}$ where r is a hyper-parameter denoting the reduction ratio, and \mathbf{W}_2 transforms the reduced size back to \mathbb{R}^C .

The final output of the c -th channel in the excitation step is obtained by scaling the original feature map \mathbf{F}_c with the channel weight s_c

$$\tilde{\mathbf{F}}_c = s_c * \mathbf{F}_c \quad (6.3)$$

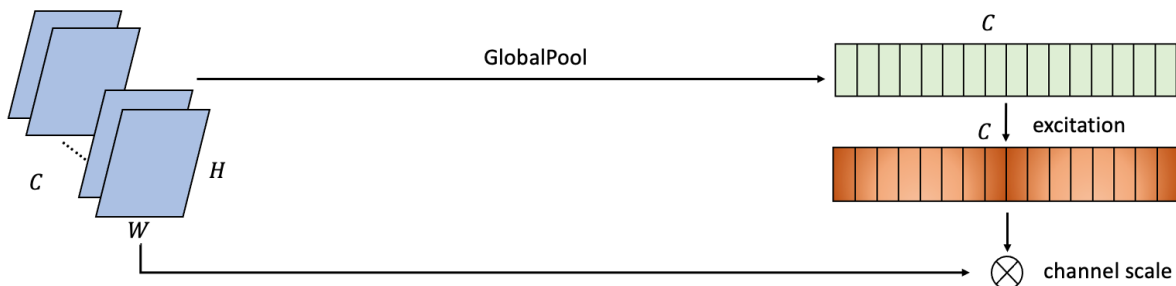


Figure 6.1: SE units with global average pooling.

Figure 6.1 illustrates an SE unit with global average pooling. In conclusion, an SE unit first applies pooling on channel features to get compressed channel descriptors, and

then maps the channel descriptors to a set of channel weights. The final output of the SE unit is obtained by re-scaling channel features with the corresponding channel weights.

Various works have been conducted on top of the channel attention architecture to obtain further improvement. They can be categorized into two directions: enhance the channel descriptor computation step, and combine the channel attention with other attention modules. For the first direction, [71] used max-pooling together with global average pooling, considering that max-pooling gives information about distinctive features within a channel. [72] and [73] utilized second-order statistics as the channel descriptors. [74] parameterized the aggregation step, and used a convolution layer to gather information in each channel. For the second direction, spatial attention is often used to model spatial dependencies. The channel attention module focuses on “what” is important and the spatial attention module focus on “where” is important. Therefore, these two types of attention modules are complementary. [71] placed the two attention modules in a sequential manner, while [75] built two parallel attention branches.

6.2 Frequency-domain Learning

Frequency analysis is a technique for decomposing functions, waveforms or signals into different frequency components. It is a powerful tool in signal processing. In speech related tasks, frequency analysis is commonly used in the feature engineering stage, producing handcrafted features from the raw input signals for subsequent learning.

Recently frequency analysis has been incorporated as a part of the deep learning frameworks. [76] formulated deep learning in the JPEG transform domain and provided a framework to learn from compressed inputs. [77] proposed the idea of learning directly from block-wise discrete cosine transform (DCT). [78] proposed a method of learning in the frequency domain. It analyzed the feature maps from the frequency perspective using DCT and adaptively selected important frequency components for further processing. The frequency domain learning methods help reduce the communication bandwidth of the networks without accuracy loss. Besides, frequency analysis has also been applied in pruning and model compression [79–81].

Frequency domain learning is also introduced to channel attention to produce more informative channel descriptors. [82] regarded the channel descriptor computation as a

compression problem, and utilized DCT to produce descriptors. It further generalized channel attention in the frequency domain, and proposed a multi-spectral channel attention framework.

6.3 Frequency-domain Pooling in Channel Attention

Channel attention consists of two steps. The first squeeze step aggregates features within channels and produces channel descriptors. The second step computes a set of weights according to channel descriptors and uses it to re-weigh channels. The aggregation step is crucial in channel attention. It needs to summarize information within the channels, while keeping the computation simple and effective.

In this section, we propose two pooling methods for channel attention based on frequency analysis. We first analyze the relationship between global average pooling and DCT in Section 6.3.1. Then we generalize the global average pooling to frequency domain pooling using DCT in Section 6.3.2 and Section 6.3.3.

6.3.1 Discrete Cosine Transform (DCT) and Average Pooling

DCT represents a finite sequence of data points by a sum of cosine functions oscillating at different frequencies. It has a strong “energy compaction” property. That is, most energies are concentrated in the lower frequency components. Therefore, DCT is widely used in signal processing and image compression.

Suppose we have a feature map $\mathbf{M} \in \mathbb{R}^{H \times W}$. The 2-dimensional DCT on \mathbf{M} is defined as:

$$\begin{aligned} \mathbf{B}_{pq} &= \alpha_p \alpha_q \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} \mathbf{M}_{hw} \cos\left(\frac{\pi(2h+1)p}{2H}\right) \cos\left(\frac{\pi(2w+1)q}{2W}\right), \\ \alpha_p &= \frac{1}{\sqrt{H}} \text{ when } p = 0, \text{ and } \sqrt{\frac{2}{H}} \text{ otherwise,} \\ \alpha_q &= \frac{1}{\sqrt{W}} \text{ when } q = 0, \text{ and } \sqrt{\frac{2}{W}} \text{ otherwise,} \end{aligned} \tag{6.4}$$

where p and q are frequency domain indices, and h and w are feature map indices, with $0 \leq p \leq H-1$, $0 \leq q \leq W-1$, $0 \leq h \leq H-1$ and $0 \leq w \leq W-1$.

If we only consider the lowest frequency component \mathbf{B}_{00} when $p = 0$ and $q = 0$, we

have,

$$\begin{aligned}
\mathbf{B}_{00} &= \frac{1}{\sqrt{H}} \frac{1}{\sqrt{W}} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} \mathbf{M}_{hw} \times \cos\left(\frac{\pi(2h+1) * 0}{2H}\right) \cos\left(\frac{\pi(2w+1) * 0}{2W}\right) \\
&= \frac{1}{\sqrt{H}} \frac{1}{\sqrt{W}} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} \mathbf{M}_{hw} \\
&= \sqrt{H} \sqrt{W} * \text{GlobalPool}(\mathbf{M}).
\end{aligned} \tag{6.5}$$

Equation 6.5 shows that the lowest frequency component of DCT is proportional to the global average pooling, and we can treat the global average pooling as a special case of DCT. According to the “energy compaction” property of DCT, it is reasonable to use only the lowest frequency component to represent the whole feature map.

6.3.2 Multi-branch Frequency-domain Pooling

We have shown that global average pooling is a special case of DCT where only the lowest frequency component is considered. It is natural to generalize the average pooling to frequency-domain pooling and utilize more frequency components.

In our work, we consider frequency-domain pooling in 2D-ResNets with SE units. Within each SE unit, we have feature maps $\mathbf{F} \in \mathbb{R}^{C \times H \times W}$ as inputs. When computing the channel descriptors, instead of applying global average pooling within each channel, we introduce multiple channel attention branches.

Figure 6.2 illustrates an SE unit with multi-branch frequency-domain pooling with K branches. Let $k \in \{1, 2, \dots, K\}$ be the index of attention branches, and every branch uses a certain frequency component defined by the 2D frequency domain indices (p_k, q_k) as formulated in Equation 6.4, where $0 \leq p_k \leq H-1$ and $0 \leq q_k \leq W-1$. The input feature map \mathbf{F} is forwarded to every branch, and the k -th branch first reduces the channel size of the original feature map to $\mathbf{F}^k \in \mathbb{R}^{\frac{C}{K} \times H \times W}$ using 1×1 convolutional blocks. This step aims at reducing the channel size of feature maps according to the number of attention branches, so that the overall network size remains the same. Then 2D-DCT is applied to every channel of feature map \mathbf{F}^k , the component with frequency domain indices (p_k, q_k)

is used as the descriptor of each channel:

$$\begin{aligned}
\mathbf{z}^k &= \text{FreqPool}_k = \mathbf{B}_{p_k, q_k}(\mathbf{F}^k) \\
&= \alpha_{p_k} \alpha_{q_k} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} \mathbf{F}_{:,h,w}^k \cos\left(\frac{\pi(2h+1)p_k}{2H}\right) \cos\left(\frac{\pi(2w+1)q_k}{2W}\right), \\
\alpha_{p_k} &= \frac{1}{\sqrt{H}} \text{ when } p_k = 0, \text{ and } \sqrt{\frac{2}{H}} \text{ otherwise,} \\
\alpha_{q_k} &= \frac{1}{\sqrt{W}} \text{ when } q_k = 0, \text{ and } \sqrt{\frac{2}{W}} \text{ otherwise,}
\end{aligned} \tag{6.6}$$

where $\mathbf{z}^k \in \mathbb{R}^{\frac{C}{K}}$ represents channel descriptors obtained from the k -th attention branch. The output from all the branches are then concatenated together to get the final channel descriptors $\mathbf{z} \in \mathbb{R}^C$:

$$\mathbf{z} = \text{Concatenate}(\mathbf{z}^1, \mathbf{z}^2, \dots, \mathbf{z}^K) \tag{6.7}$$

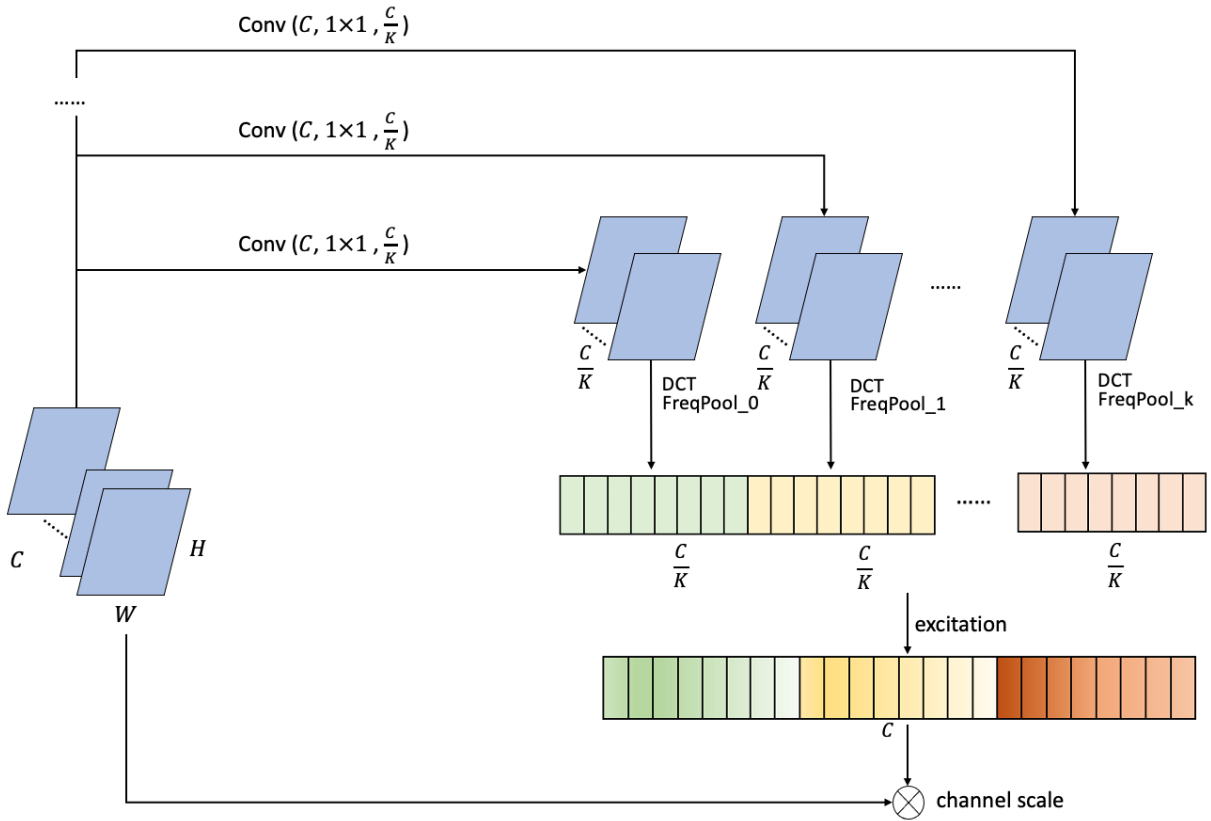


Figure 6.2: SE units with multi-branch frequency-domain pooling with K branches.

Then the excitation step takes channel descriptors \mathbf{z} as input and produces channel weights $\mathbf{s} \in \mathbb{R}^C$ as formulated in Equation 6.2. The final output of every channel is obtained by weighting the channel feature map with the corresponding channel weights following Equation 6.3.

The multi-branch frequency-domain pooling method makes use of multiple frequency components by constructing multiple attention branches. It can preserve more information compared to the global average pooling and thus produce better channel descriptors. But it has a major drawback: we need to pre-define the frequency components used in every attention branch, which requires prior knowledge about the effectiveness of every single frequency component. Besides, the multi-branch pooling method introduces additional parameters in order to reduce the input channel size before forwarding them to attention branches.

6.3.3 Max Frequency-domain Pooling

To address the aforementioned shortcomings of multi-branch pooling, we further propose the max frequency-domain pooling method. Instead of manually selecting the frequency component for every attention branch, we remove the multi-branch structure, and apply DCT on every channel feature map, then use the maximum frequency component as the corresponding channel descriptor. We argue that the maximum frequency component contains clue about distinctive features and is able to represent the channel.

The max frequency-domain pooling can be formulated in the following way: for an input feature map $\mathbf{F} \in \mathbb{R}^{C \times H \times W}$, 2D-DCT is applied on every channel feature map $\mathbf{F}_c \in \mathbb{R}^{H \times W}$ according to Equation 6.4, where c is the channel index. Among all the frequency components \mathbf{B}_{pq} , the maximum one is selected as the channel descriptor \mathbf{z}_c :

$$\mathbf{z}_c = \text{FreqPool}(\mathbf{F}_c) = \text{maximum}\{\mathbf{B}_{pq}(\mathbf{F}_c)\} \quad (6.8)$$

Then all the channel descriptors \mathbf{z}_c are used to compute channel weights \mathbf{s}_c following the excitation step in Equation 6.2, and the output of c -th channel can be obtained by scaling the original feature map \mathbf{F}_c with the channel weight \mathbf{s}_c

$$\tilde{\mathbf{F}}_c = \mathbf{s}_c * \mathbf{F}_c. \quad (6.9)$$

Compared to the multi-branch frequency-domain pooling method, the max pooling strategy does not require any prior knowledge about the frequency components. We do not need to design the number of attention branches in the system. It also avoid introducing addition parameters.

6.4 Performance

In previous chapters, the ResNet framework presented in Chapter 3 serves as the baseline system. In this chapter, we first incorporate the channel attention mechanism by introducing SE units into the frame-level modeling part of the ResNet framework, and then enhance the channel attention with frequency-domain pooling methods. The ResNet framework with SE units is named as SE-ResNet.

6.4.1 Frequency-domain pooling with a single component

Table 6.1: *Architecture of the fast SE-ResNet. T represents the training segment length.*

Layer	Kernel size	Stride	Output shape
Conv1	$7 \times 7 \times 16$	2×1	$16 \times 20 \times T$
SE-Res1	$3 \times 3 \times 16$	1×1	$16 \times 20 \times T$
SE-Res2	$3 \times 3 \times 32$	2×2	$32 \times 10 \times (T/2)$
SE-Res3	$3 \times 3 \times 64$	2×2	$64 \times 5 \times (T/4)$
SE-Res4	$3 \times 3 \times 128$	1×1	$128 \times 5 \times (T/4)$
Flatten	-	-	$640 \times (T/4)$
Attentive mean pooling	-	-	640
Linear	256	-	256

Before making use of multiple frequency components for pooling in the channel attention module, we first need to verify the effectiveness of every single frequency component. Since the smallest intermediate feature map in the SE-ResNet is 8×8 , we need to divide the 2D DCT frequency space into 8×8 parts, there will be totally 64 frequency components. To save time, we conduct these experiments with a smaller SE-ResNet framework instead. The network architecture is depicted in Table 6.1. Basically it halves the output channel size in every layer and employs a larger kernel size in the first layer, and we name it as *fast SE-ResNet*. With fast SE-ResNet, we divide the 2D DCT frequency space into 5×5 parts according to the smallest feature map size, and use 25 frequency components individually in the channel attention.

All the results are shown in Table 6.2 and Table 6.3. *FP* denotes frequency domain pooling, and *FP*(i,j) denotes frequency-domain pooling using the frequency component \mathbf{B}_{ij} in Equation 6.4.

In general, when used alone, many frequency components can achieve performance comparable to the global average pooling (FP(0,0) component), especially the low-frequency

Table 6.2: *EER results with individual frequency components pooling on Voxceleb1-O with the fast SE-ResNet framework.*

EER	FP(,0)	FP(,1)	FP(,2)	FP(,3)	FP(,4)
FP(0,)	2.23	2.31	2.42	2.38	2.32
FP(1,)	2.04	2.33	2.23	2.27	2.40
FP(2,)	2.06	2.31	2.44	2.41	2.29
FP(3,)	2.14	2.32	2.26	2.46	2.32
FP(4,)	2.15	2.31	2.37	2.25	2.38

Table 6.3: *minDCF results with individual frequency components pooling on Voxceleb1-O with the fast SE-ResNet framework.*

minDCF	FP(,0)	FP(,1)	FP(,2)	FP(,3)	FP(,4)
FP(0,)	0.178	0.184	0.192	0.185	0.185
FP(1,)	0.153	0.188	0.181	0.180	0.193
FP(2,)	0.159	0.194	0.188	0.196	0.176
FP(3,)	0.168	0.162	0.169	0.183	0.181
FP(4,)	0.169	0.167	0.184	0.173	0.186

components. It confirms the feasibility of utilizing more frequency components for pooling, and they indeed contain helpful information.

We further analyze the frequency components along different dimensions. In our experimental settings, the height(h)-dimension is the frequency dimension within each channel, and the width(w)-dimension is the time dimension. If we look at all the components along FP(,0), both EER and minDCF achieve the best results in general. Besides, it is worth noting that all the high-frequency components along FP(,0) achieve better performance than FP(0,0), which is the global average pooling. According to Equation 6.4, FP(p ,0) where $0 \leq p \leq 4$ is computed by

$$\begin{aligned}
 FP(p, 0) &= \alpha_p \frac{1}{\sqrt{W}} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} \mathbf{M}_{hw} \cos\left(\frac{\pi(2h+1) * p}{2H}\right) \\
 &= \alpha_p \frac{1}{\sqrt{W}} \sum_{h=0}^{H-1} \cos\left(\frac{\pi(2h+1) * p}{2H}\right) \sum_{w=0}^{W-1} \mathbf{M}_{hw},
 \end{aligned} \tag{6.10}$$

where α_p is a constant. Basically it sums over the time dimension first, then applies 1D-DCT along the frequency dimension and takes different frequency components as the pooling result. From the results, we can see that when applying 1D-DCT along only the frequency dimension, all the frequency components give better or comparable performance

to the average pooling. It reveals that high-frequency information along the frequency dimension is helpful in the subsequent feature learning.

On the other hand, if we look at all the components along the time dimension, such as FP(0,q), the EER drops quickly starting from FP(0,1). Similar results can also be found along FP(1,q), FP(2,q), etc. Here we take FP(0,q) as an example for analysis. According to Equation 6.4, FP(0,q) where $0 \leq q \leq 4$ is computed by

$$\begin{aligned} FP(0, q) &= \alpha_q \frac{1}{\sqrt{H}} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} \mathbf{M}_{hw} \cos\left(\frac{\pi(2w+1) * q}{2W}\right) \\ &= \alpha_q \frac{1}{\sqrt{H}} \sum_{w=0}^{W-1} \cos\left(\frac{\pi(2w+1) * q}{2W}\right) \sum_{h=0}^{H-1} \mathbf{M}_{hw}, \end{aligned} \quad (6.11)$$

where α_q is a constant. It sums over the frequency dimension first, then applies 1D-DCT along the time dimension and takes different frequency components for pooling. The results show that high-frequency information along the time dimension is much less effective compared to the lowest frequency component. The reason might be that high-frequency components along the time dimension usually contain noises, which are harmful for subsequent feature learning.

According to the results on frequency-domain pooling with single frequency component, we can conclude that in speaker embedding learning, high-frequency components from DCT along the frequency dimension is helpful, while high-frequency components from DCT along the time dimension contain noises information which can be harmful to subsequent feature learning. Therefore, we can simplify the 2D DCT to the 1D version, reducing the computation cost without affecting the system performance. In the following experiments, for every input feature map, we first average over the time dimension, and then apply 1D DCT to get the frequency components.

6.4.2 Frequency-domain pooling with multiple components

We combined different frequency-domain components for pooling as proposed in Section 6.3.2 and 6.3.3. For multi-branch pooling, we choose the best frequency-components according to the performance in Table 6.2 and Table 6.3.

Overall results are summarized in Table 6.4. The term *Global Avg* denotes global average pooling in the SE units, *Adaptive* denotes the max frequency-domain pooling, *Multi-branch-k* denotes multi-branch frequency-domain pooling with k frequency components, and *Multi-branch-all* denotes multi-branch frequency-domain pooling with all

Table 6.4: *Results with multiple frequency components pooling on Voxceleb1-O with the fast SE-ResNet framework.*

SE pooling	EER	minDCF
Global Avg	2.23	0.178
Multi-branch-2	2.08	0.165
Multi-branch-4	1.88	0.155
Multi-branch-all	1.83	0.152
Adaptive	1.96	0.156

Table 6.5: *Results with multiple frequency components pooling on Voxceleb with the SE-ResNet framework.*

SE pooling	Voxceleb1-O		Voxceleb-E		Voxceleb-H	
	EER	minDCF	EER	minDCF	EER	minDCF
Global Avg	1.10	0.087	1.23	0.085	2.39	0.154
Adaptive	1.01	0.085	1.12	0.081	2.12	0.148
Multi-branch-4	0.88	0.081	1.02	0.077	2.07	0.144
Multi-branch-8	0.83	0.080	0.98	0.072	2.04	0.141

frequency components which is five in the fast SE-ResNet framework.

In general, systems with frequency-domain pooling outperform the baselines with global average pooling. The max frequency-domain pooling is better than average pooling by 12% in both EER and minDCF. The multi-branch frequency-domain pooling with two branches improves the EER and minDCF by 7%. By increasing the attention branches, the system with four branches outperforms the baseline by 16% in EER and 13% in minDCF.

We further verify the frequency-domain pooling with multiple components in the standard SE-ResNet framework, and report results on VoxCeleb and SITW evaluation datasets. The experimental results are summarized in Table 6.5 and 6.6.

Similar results are obtained with the larger network architecture. Both frequency-domain pooling methods outperform the global average pooling consistently. The max frequency-domain pooling improves the baseline system by 8%, 9%, 11% and 6% in terms of EER on VoxCeleb1-O, VoxCeleb1-E, VoxCeleb1-H and SITW, respectively. The multi-branch frequency-domain pooling method with four attention branches achieves around 20% improvement in EER and 7% in minDCF on VoxCeleb1-O, and it also improves the performance by around 10% on SITW.

We also report the performance on the speaker classification task with the standard

Table 6.6: *Results with multiple frequency components pooling on SITW with SE-ResNet framework.*

SE pooling	EER	minDCF
Global Avg	1.66	0.105
Adaptive	1.56	0.102
Multi-branch-4	1.50	0.096
Multi-branch-8	1.48	0.093

Table 6.7: *Speaker classification results with multiple frequency components pooling with SE-ResNet framework.*

SE pooling	Accuracy(%)
Global Avg	98.31
Max-freq	98.73
Multi-branch-4	99.05
Multi-branch-8	99.13

SE-ResNet framework. The classification accuracy is computed on the validation dataset, containing 110,000 utterances from 6112 speakers. The results are summarized in Table 6.7. The speaker classification performance is consistent with the verification performance. The max frequency-domain pooling improves the global average pooling by 25%, and the multi-branch pooling with 8 attention branches is 48% better than the average pooling.

Model complexity

The model complexity measured in model size and floating point operations (FLOPs) are summarized in Table 6.8. The max frequency-domain pooling method maintains the same model size and FLOPs. The multi-branch frequency-domain pooling methods bring larger model sizes and FLOPs, but the model complexity does not increase as the number of branches increases.

In general, the max frequency-domain pooling method can improve the system performance consistently, and it does not introduce any hyper-parameters. It can be used as a plug-in module to channel attention and gives performance improvements. For the multi-branch frequency-domain pooling method, more attention branches bring better system performance, but in the meantime, the computation cost gets higher.

Table 6.8: *Model complexity with multiple frequency components pooling with the SE-ResNet framework.*

SE pooling	Size(M)	FLOPs(G)
Global Avg	43	7.66
Max-freq	43	7.66
Multi-branch-4	48	7.84
Multi-branch-8	48	7.84

6.5 Comparison to Self-attentive Speaker Embeddings

In Chapter 4 and Chapter 5 we focus on the pooling step between frame-level modeling and utterance-level modeling, and proposed pooling methods utilizing self-attention mechanism and Bayesian self-attention algorithms. In this chapter, we concentrate on the frame-level modeling. We first introduce channel attention into the frame-level modeling, and then enhance the channel attention with frequency-domain pooling methods.

Table 6.9: *Overall results on VoxCeleb1-O, VoxCeleb1-E and VoxCeleb1-H.*

System	VoxCeleb1-O		VoxCeleb1-E		VoxCeleb1-H	
	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
baseline	1.12	0.151	1.34	0.143	2.38	0.222
attn-1	1.12	0.125	1.33	0.140	2.33	0.228
attn-8	1.02	0.105	1.22	0.125	2.16	0.205
sub-attn-8	0.93	0.112	1.20	0.124	2.12	0.203
B. attn-8	0.83	0.106	1.16	0.124	2.06	0.196
attn-1+ CA	1.10	0.087	1.23	0.085	2.39	0.154
+Multi-branch-8	0.83	0.080	0.98	0.072	2.04	0.141

Table 6.10: *Overall results on SITW evaluation set.*

System	EER	minDCF
baseline	1.73	0.166
attn-1	1.72	0.156
attn-8	1.62	0.152
sub-attn-8	1.64	0.156
Bayesian attn-8	1.57	0.152
attn-1 + CA	1.66	0.105
+Multi-branch-8	1.48	0.093

To compare the results of frequency-domain pooling methods to previous self-attentive speaker embeddings, we summarize the system performance in Table 6.9 and Table 6.10.

‘*baseline*’ refers to the system described in Chapter 3; ‘*attn-k*’ refers to the self-attentive speaker embedding systems described in Section 4.3.2 with k attention heads; ‘*sub-attn-8*’ denotes the system described in Section 4.3.4 with 8 sub-vector attention heads; ‘*B.attn-8*’ denotes the Bayesian self-attentive system described in Chapter 5 with 8 heads; ‘*attn-1 + CA*’ refers to the 1-head self-attentive speaker embedding systems with channel attention incorporated; ‘*+Multi-branch-8*’ denotes the system utilizing multi-branch frequency-domain pooling with 8 branches on top of the system ‘*attn-1 + CA*’.

The self-attentive speaker embeddings and Bayesian self-attentive speaker embeddings have greatly improved the baseline system performance by utilizing multiple attention heads in the pooling stage. The best performance on all the evaluation datasets are achieved by the Bayesian self-attentive speaker embeddings with 8 attention heads. By incorporating channel attention, we first improve the self-attention speaker embedding system with a single attention head, especially on the minDCF. On all the VoxCeleb1 and SITW evaluation datasets, the channel attention has helped improve the minDCF by over 30%. We further introduce frequency-domain pooling methods inside the channel attention module, and the best multi-branch system with 8 branches improve the EER by 26% on VoxCeleb1-O and VoxCeleb1-E, 12% on VoxCeleb1-H and 14% on SITW compared to the baseline.

Chapter 7

Summary and Future Work

In this chapter we summarize our work in learning speaker embeddings, and discuss some future directions for speaker verification.

7.1 Summary

In this thesis, we mainly work on learning better speaker embeddings in text-independent speaker verification tasks. We focus on two directions to enhance the speaker embedding learning:

- introducing self-attention mechanism into the pooling stage between frame-level modeling and utterance-level modeling;
- introducing and enhancing channel attention in the frame-level modeling.

For the first direction, the pooling stage is to aggregate information from a sequence of frame-level features and produce utterance-level representations. We introduced the self-attention mechanism into this pooling step, and also investigated the multi-head self-attention mechanism. The multi-head self-attention mechanism aims to produce more discriminative and informative speaker representations. One major issue in the multi-head self-attention mechanism is information redundancy. We investigated two techniques that can alleviate the problem. The first one is introducing a penalty term that explicitly encourages the attention heads to learn dissimilar attention weights, and the second one is to provide each attention head with a different input.

We further generalized the deterministic multi-head self-attention to a Bayesian self-attention framework, and proposed an algorithm based on Stein variational gradient descent to learn repulsive attention heads under the Bayesian self-attention framework.

All the multi-head self-attention SV systems were evaluated on VoxCeleb1 and SITW. Bayesian multi-head self-attention systems obtained significant and consistent improvement over other multi-head self-attention systems on all evaluation datasets.

Besides, we analyzed the diversity of attention weights produced by different multi-head self-attention mechanisms, and its relation to the system performance. We found that multi-head systems that produce more diverse attention weights would have better performance in general. We also tried to analyze the preference of the attention mechanism in SV systems on the phonemes in the input utterances. We found that the attention model tends to focus on vowels and semivowels when verifying speakers' identities.

For the second direction, we incorporated channel attention into the frame-level modeling stage of the network, providing global information and models channel interdependencies. We analyzed the pooling step in the channel attention modules, and proposed two frequency-domain pooling methods to further enhance channel attention.

Our contributions include:

- introducing the self-attention mechanism to the final pooling step in speaker embedding learning;
- generalizing the deterministic multi-head attention to a Bayesian attention framework;
- proposing an algorithm to learn repulsive attention based on Stein variational gradient descent in the Bayesian attention framework. The proposed algorithm can be easily integrated into current optimizers;
- introducing channel attention into the early stage in the speaker embedding learning framework, and proposed two methods based on frequency-domain pooling to enhance the channel attention;
- evaluating proposed speaker embeddings in various benchmark corpora. Our proposed speaker embedding learning methods achieve better performance than state-of-the-art systems.

7.2 Future work

In this thesis, we have presented a complete speaker embedding learning framework, and proposed three learning methods. The results on various evaluation datasets are encouraging. Nonetheless, more work could be done to verify our current findings and build better systems. The following are some interesting topics we will be pursuing in the future.

Frequency-domain pooling

We are going to further investigate the frequency-domain pooling methods. In our current multi-branch frequency-domain pooling methods, we are only able to experiment with a limited number of branches. We are interested in how the performance changes when the number of branches increases. Besides, the current multi-branch frequency-domain pooling methods introduce additional parameters when splitting the inputs into different branches. We are going to explore the parameter-free ways.

In addition, channel attention and self-attentive pooling work on different parts of the speaker embedding learning framework. Channel attention and frequency-domain pooling methods work on the frame-level modeling, while self-attentive pooling is after the frame-level modeling. Therefore, these embedding learning methods are complementary. We will combine the proposed methods together to build a better speaker embedding learning framework.

Self-supervised speaker embedding learning

The mainstream of building speaker verification systems relies on supervised learning approach, which requires a large amount of labeled data. Although the supervised methods have achieved state-of-the-art system performance, leveraging the unlabeled speech data is still an attractive research topic. In many practical scenarios, collecting sufficient annotated training data is a non-trivial task. On the other hand, there are massive speech data without speaker identity labels, for example, videos in public social media like YouTube, speech data collected for automatic speech recognition tasks, etc. Self-supervised learning methods have been investigated for learning speaker representations utilizing unlabeled data.

One direction of self-supervised speaker embedding learning is based on the idea of input reconstruction. The first successful framework is proposed in [83]. It designs an encoder-decoder structure. The encoder takes a speech segment from a certain speaker as input, and learns a speaker embedding. The decoder takes the learned speaker embedding, together with a random input speech segment from the same speaker and its corresponding phone sequence as input, and learns to reconstruct the input speech segments in the frame level.

The key idea is to reconstruct the frames of a random speech segment based on its corresponding speaker embedding and phonetic sequence. The speaker embedding contains speaker-related information, and the phonetic sequence can be obtained from a speech recognition model trained independently. Under this framework, the learned speaker embeddings are encouraged to discard the irrelevant phonetic information, and retain only distinct speaker characteristics.

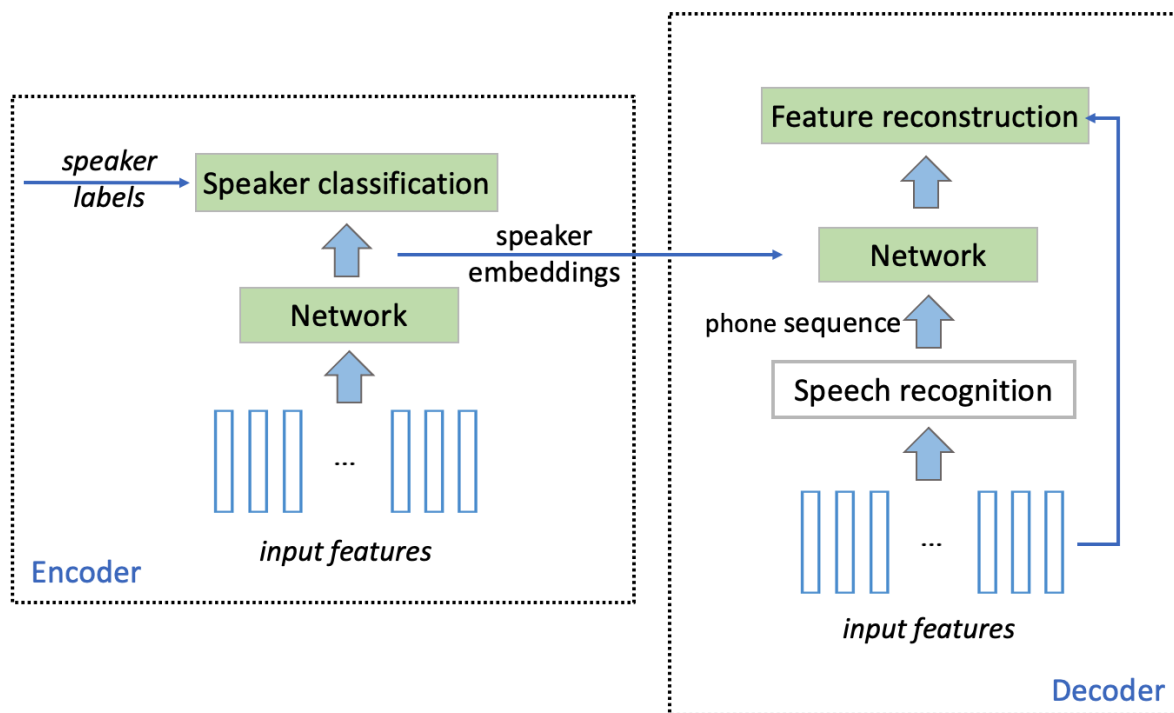


Figure 7.1: Architecture of self-supervised speaker embedding learning framework.

The overall architecture is illustrated in Figure 7.1, and the green parts are the network layers that need to be trained. The framework can be trained in a pure self-supervised way: the speaker labels and the speaker classification part can be removed, and the feature reconstruction task is trained with the mean-squared error loss. Moreover, it can be easily combined with supervised training approaches when speaker labels are available. The

speaker classification task is trained with the cross-entropy loss, and it can be optimized with the feature reconstruction loss jointly.

The framework does not require any exact speaker labels for speech segments. The only constraint is that the input speech segment to the encoder and the decoder should belong to the same speaker, which is easy to satisfy. In practice, we often have a large amount of speech data, while only a fraction of them are labeled with speaker identity. The framework can be trained in a semi-supervised manner in this case with limited speaker labels.

Another direction of self-supervised speaker embedding learning is based on contrastive learning methods. The objective of contrastive learning is to minimize the intra-class variation while maximizing the inter-class differences. A typical contrastive learning framework consists of three modules: (1) An input construction module that prepares positive and negative input pairs. Given an input sample, i.e. the anchor, a positive pair consists of the anchor and another sample from the same speaker, and a negative pair consists of the anchor and another sample from a different speaker. (2) An encoder that encodes the input samples and generates compact speaker embeddings. (3) A contrastive loss module that maximizes the similarity of positive pairs and minimizes the similarity of negative pairs. One commonly used loss is the InfoNCE loss [84].

The first input construction module plays a crucial role. To make contrastive learning work, it should generate sufficient correct positive and negative pairs, in the mean time, it needs to be efficient in terms of time and memory cost. In practice, positive pairs are usually obtained by data augmentation methods. The augmentation technique can transfer an input sample into a different view by adding noises, reverberation, speed perturbation, etc. The augmented sample together with the original anchor sample can form a positive pair. Negative pairs are usually constructed within a mini-batch, any two different input samples can be treated as a negative pair.

To learn better embeddings, contrastive learning prefers a large number of negative pairs. The size of negative pairs constructed in the usual way is limited by the mini-batch size. An alternative way is to maintain a separate dynamic queue to store a large number of negative pairs [85]. Another problem of negative pair construction is the class collision issue. Since we assume that different samples come from different speakers, negative pairs are randomly sampled. In practice, this assumption may not hold and result in wrong

pairs. The prototypical memory bank [86] is proposed to alleviate this problem. The key idea is that in every epoch, we first cluster all the training samples, then negative pairs are only sampled from different classes.

Recent research on self-supervised learning methods has helped bridge the gap between fully unsupervised learning and supervised learning. Besides, they can always be used together with the supervised learning methods when speaker labels are available, and further improve the system performance. Our proposed speaker embedding learning methods can be adopted in the encoding modules of self-supervised learning frameworks. We are interested in investigating the encoding network as well as new architectures for self-supervised speaker embedding learning.

Spoofing aware speaker verification

“Spoofing” refers to an attack on speaker verification systems with fake biometrics from a valid person, for example, audio playback, speech obtained by voice conversion, synthesized speech, etc. Anti-spoofing is the task of identifying and rejecting these fake biometrics. Anti-spoofing is an essential part of speaker verification in practice. Usually, the anti-spoofing module is placed before the verification system. It identifies whether the incoming speech is produced by real human beings, and only valid speech will be passed to the subsequent verification system.

Anti-spoofing systems and speaker verification systems are considered and trained separately all long. It is reasonable since they are two quite different tasks. Anti-spoofing is basically a binary classification problem: it only needs the fake/real labels for speech, and does not require any speaker labels. Therefore, for all state-of-the-art speaker verification systems, spoofing attacks are not considered in the training stage. When used in practical application scenarios, the verification system needs to be deployed together with a separate anti-spoofing system. Otherwise, the accuracy of the verification system can drop dramatically.

Recently people start to consider jointly training the anti-spoofing system and the speaker verification system as an integrated system, with the hope that better performance can be obtained when these two systems are optimized together.

One straightforward way might be the ensemble model. We can make use of a pre-trained anti-spoofing model and a pre-trained speaker verification model. The ensemble

model takes the embeddings produced by two pre-trained models as input, and is trained to identify whether the input belongs to the claimed speaker.

Another solution can be training the system in a multi-task fashion with multiple output layers. The system is trained to identify fake/real speech and classify speakers at the same time.

We are interested to investigate how our embedding learning methods perform when anti-spoofing is considered in the speaker verification tasks. It is meaningful to develop embedding learning methods for spoofing-aware verification and build robust speaker verification systems.

References

- [1] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, “X-vectors: Robust DNN embeddings for speaker recognition,” 2018.
- [2] L. You, W. Guo, L. Dai, and J. Du, “Multi-task learning with high-order statistics for x-vector based text-independent speaker verification,” *arXiv preprint arXiv:1903.12058*, 2019.
- [3] A. Higgins, L. Bahler, and J. Porter, “Speaker verification using randomized phrase prompting,” *Digital signal processing*, vol. 1, no. 2, pp. 89–106, 1991.
- [4] A. E. Rosenberg, J. DeLong, C.-H. Lee, B.-H. Juang, and F. K. Soong, “The use of cohort normalized scores for speaker verification.” in *ICSLP*, vol. 92, 1992, pp. 599–602.
- [5] D. A. Reynolds, “Speaker identification and verification using Gaussian mixture speaker models,” *Speech communication*, vol. 17, no. 1-2, pp. 91–108, 1995.
- [6] T. Matsui and S. Furui, “Similarity normalization method for speaker verification based on a posteriori probability,” in *Proc. ESCA Workshop on Automatic Speaker Recognition Identification Verification. pp59-62 Switzerland*, 1994.
- [7] D. A. Reynolds, “Comparison of background normalization methods for text-independent speaker verification,” in *Fifth European Conference on Speech Communication and Technology*, 1997.
- [8] A. E. Rosenberg and S. Parthasarathy, “Speaker background models for connected digit password speaker verification,” in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, vol. 1. IEEE, 1996, pp. 81–84.

- [9] L. P. Heck and M. Weintraub, “Handset-dependent background models for robust text-independent speaker recognition,” in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2. IEEE, 1997, pp. 1071–1074.
- [10] 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. 1–22, 1977.
- [11] 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.
- [12] W. M. Campbell, D. E. Sturim, D. A. Reynolds, and A. Solomonoff, “SVM based speaker verification using a GMM supervector kernel and NAP variability compensation,” in *2006 IEEE International conference on acoustics speech and signal processing proceedings*, vol. 1. IEEE, 2006, pp. I–I.
- [13] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, “Front-end factor analysis for speaker verification,” vol. 19, no. 4, pp. 788–798, 2011.
- [14] P. Kenny, G. Boulianne, P. Ouellet, and P. Dumouchel, “Joint factor analysis versus eigenchannels in speaker recognition,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 4, pp. 1435–1447, 2007.
- [15] E. Variani, X. Lei, E. McDermott, I. Moreno, and J. Gonzalez-Dominguez, “Deep neural networks for small footprint text-dependent speaker verification,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 4052–4056.
- [16] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khudanpur, “Deep neural network embeddings for text-independent speaker verification,” pp. 999–1003, 2017.
- [17] D. Snyder, D. Garcia-Romero, G. Sell, A. McCree, D. Povey, and S. Khudanpur, “Speaker recognition for multi-speaker conversations using x-vectors,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 5796–5800.

- [18] D. Povey, G. Cheng, Y. Wang, K. Li, H. Xu, M. Yarmohammadi, and S. Khudanpur, “Semi-orthogonal low-rank matrix factorization for deep neural networks.” in *Interspeech*, 2018, pp. 3743–3747.
- [19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [20] Y.-Q. Yu, L. Fan, and W.-J. Li, “Ensemble additive margin softmax for speaker verification,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6046–6050.
- [21] J. S. Chung, A. Nagrani, and A. Zisserman, “Voxceleb2: Deep speaker recognition,” 2018.
- [22] Z. Wang, K. Yao, X. Li, and S. Fang, “Multi-resolution multi-head attention in deep speaker embedding,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6464–6468.
- [23] W. Xie, A. Nagrani, J. S. Chung, and A. Zisserman, “Utterance-level aggregation for speaker recognition in the wild,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 5791–5795.
- [24] S. Yadav and A. Rai, “Frequency and temporal convolutional attention for text-independent speaker recognition,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6794–6798.
- [25] D. Garcia-Romero, G. Sell, and A. Mccree, “Magneto: X-vector magnitude estimation network plus offset for improved speaker recognition,” in *Proc. Odyssey 2020 the speaker and language recognition workshop*, 2020, pp. 1–8.
- [26] Y. Zhao, T. Zhou, Z. Chen, and J. Wu, “Improving deep CNN networks with long temporal context for text-independent speaker verification,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6834–6838.

- [27] B. Desplanques, J. Thienpondt, and K. Demuynck, “ECAPA-TDNN: Emphasized channel attention, propagation and aggregation in TDNN based speaker verification,” *arXiv preprint arXiv:2005.07143*, 2020.
- [28] A. Nagrani, J. S. Chung, and A. Zisserman, “Voxceleb: a large-scale speaker identification dataset,” 2017.
- [29] S. Yadav and A. Rai, “Learning discriminative features for speaker identification and verification.” in *Interspeech*, 2018, pp. 2237–2241.
- [30] C. Zhang, K. Koishida, and J. H. Hansen, “Text-independent speaker verification based on triplet convolutional neural network embeddings,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 9, pp. 1633–1644, 2018.
- [31] S. J. Prince and J. H. Elder, “Probabilistic linear discriminant analysis for inferences about identity,” in *2007 IEEE 11th international conference on computer vision*. IEEE, 2007, pp. 1–8.
- [32] P.-E. Danielsson, “Euclidean distance mapping,” *Computer Graphics and image processing*, vol. 14, no. 3, pp. 227–248, 1980.
- [33] G. Qian, S. Sural, Y. Gu, and S. Pramanik, “Similarity between Euclidean and cosine angle distance for nearest neighbor queries,” in *Proceedings of the 2004 ACM symposium on Applied computing*, 2004, pp. 1232–1237.
- [34] C. Li, X. Ma, B. Jiang, X. Li, X. Zhang, X. Liu, Y. Cao, A. Kannan, and Z. Zhu, “Deep speaker: an end-to-end neural speaker embedding system,” *arXiv preprint arXiv:1705.02304*, 2017.
- [35] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.
- [36] M. Ravanelli and Y. Bengio, “Speaker recognition from raw waveform with SincNet,” in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 1021–1028.

- [37] J.-w. Jung, H.-S. Heo, J.-h. Kim, H.-j. Shim, and H.-J. Yu, “RawNet: Advanced end-to-end deep neural network using raw waveforms for text-independent speaker verification,” *arXiv preprint arXiv:1904.08104*, 2019.
- [38] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, “A discriminative feature learning approach for deep face recognition,” in *European conference on computer vision*. Springer, 2016, pp. 499–515.
- [39] H.-S. Heo, J.-w. Jung, I.-H. Yang, S.-H. Yoon, H.-j. Shim, and H.-J. Yu, “End-to-end losses based on speaker basis vectors and all-speaker hard negative mining for speaker verification,” *arXiv preprint arXiv:1902.02455*, 2019.
- [40] D. A. Reynolds, “An overview of automatic speaker recognition technology,” in *2002 IEEE international conference on acoustics, speech, and signal processing*, vol. 4. IEEE, 2002, pp. IV–4072.
- [41] K.-P. Li and J. E. Porter, “Normalizations and selection of speech segments for speaker recognition scoring,” in *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*. IEEE Computer Society, 1988, pp. 595–596.
- [42] D. A. Reynolds, “The effects of handset variability on speaker recognition performance: Experiments on the Switchboard corpus,” in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, vol. 1. IEEE, 1996, pp. 113–116.
- [43] R. Auckenthaler, M. Carey, and H. Lloyd-Thomas, “Score normalization for text-independent speaker verification systems,” *Digital Signal Processing*, vol. 10, no. 1-3, pp. 42–54, 2000.
- [44] M. Ben, R. Blouet, and F. Bimbot, “A Monte-Carlo method for score normalization in automatic speaker verification using kullback-leibler distances,” in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. IEEE, 2002, pp. I–689.
- [45] T. Ko, V. Peddinti, D. Povey, M. L. Seltzer, and S. Khudanpur, “A study on data augmentation of reverberant speech for robust speech recognition.” IEEE, 2017, pp. 5220–5224.

- [46] D. Snyder, G. Chen, and D. Povey, “Musan: A music, speech, and noise corpus,” *arXiv preprint arXiv:1510.08484*, 2015.
- [47] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, “SpecAugment: A simple data augmentation method for automatic speech recognition,” *arXiv preprint arXiv:1904.08779*, 2019.
- [48] F. Wang, J. Cheng, W. Liu, and H. Liu, “Additive margin softmax for face verification,” *IEEE Signal Processing Letters*, vol. 25, no. 7, pp. 926–930, 2018.
- [49] Z. N. Karam, W. M. Campbell, and N. Dehak, “Towards reduced false-alarms using cohorts,” in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 4512–4515.
- [50] G. Heigold, I. Moreno, S. Bengio, and N. Shazeer, “End-to-end text-dependent speaker verification,” 2016, pp. 5115–5119.
- [51] V. Mnih, N. Heess, A. Graves *et al.*, “Recurrent models of visual attention,” in *Advances in neural information processing systems*, 2014, pp. 2204–2212.
- [52] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” in *International conference on machine learning*, 2015, pp. 2048–2057.
- [53] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.
- [54] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
- [55] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-based models for speech recognition,” *arXiv preprint arXiv:1506.07503*, 2015.
- [56] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 4960–4964.

- [57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.
- [58] J. Cheng, L. Dong, and M. Lapata, “Long short-term memory-networks for machine reading,” *arXiv preprint arXiv:1601.06733*, 2016.
- [59] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, “A structured self-attentive sentence embedding,” 2017.
- [60] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, “A decomposable attention model for natural language inference,” *arXiv preprint arXiv:1606.01933*, 2016.
- [61] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, “Librispeech: an ASR corpus based on public domain audio books,” in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.
- [62] Q. Liu and D. Wang, “Stein variational gradient descent: A general purpose bayesian inference algorithm,” *arXiv preprint arXiv:1608.04471*, 2016.
- [63] B. An, J. Lyu, Z. Wang, C. Li, C. Hu, F. Tan, R. Zhang, Y. Hu, and C. Chen, “Repulsive attention: Rethinking multi-head attention as bayesian inference,” *arXiv preprint arXiv:2009.09364*, 2020.
- [64] Q. Liu, J. Lee, and M. Jordan, “A kernelized Stein discrepancy for goodness-of-fit tests,” in *International conference on machine learning*. PMLR, 2016, pp. 276–284.
- [65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [66] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.
- [67] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, Inception-Resnet and the impact of residual connections on learning,” in *Thirty-first AAAI conference on artificial intelligence*, 2017.

- [68] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.
- [69] W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 29, 2016.
- [70] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.
- [71] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “CBAM: Convolutional block attention module,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.
- [72] Y. Chen, Y. Kalantidis, J. Li, S. Yan, and J. Feng, “A²-nets: Double attention networks,” *Advances in neural information processing systems*, vol. 31, 2018.
- [73] Z. Gao, J. Xie, Q. Wang, and P. Li, “Global second-order pooling convolutional networks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3024–3033.
- [74] J. Hu, L. Shen, S. Albanie, G. Sun, and A. Vedaldi, “Gather-excite: Exploiting feature context in convolutional neural networks,” *Advances in neural information processing systems*, vol. 31, 2018.
- [75] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu, “Dual attention network for scene segmentation,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3146–3154.
- [76] M. Ehrlich and L. S. Davis, “Deep residual learning in the JPEG transform domain,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3484–3493.
- [77] L. Gueguen, A. Sergeev, B. Kadlec, R. Liu, and J. Yosinski, “Faster neural networks straight from JPEG,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.

- [78] K. Xu, M. Qin, F. Sun, Y. Wang, Y.-K. Chen, and F. Ren, “Learning in the frequency domain,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1740–1749.
- [79] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, “Compressing convolutional neural networks in the frequency domain,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1475–1484.
- [80] Y. Wang, C. Xu, C. Xu, and D. Tao, “Packing convolutional neural networks in the frequency domain,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 10, pp. 2495–2510, 2018.
- [81] Z. Liu, J. Xu, X. Peng, and R. Xiong, “Frequency-domain dynamic pruning for convolutional neural networks,” *Advances in neural information processing systems*, vol. 31, 2018.
- [82] Z. Qin, P. Zhang, F. Wu, and X. Li, “FcaNet: Frequency channel attention networks,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 783–792.
- [83] T. Stafylakis, J. Rohdin, O. Plchot, P. Mizera, and L. Burget, “Self-supervised speaker embeddings,” *arXiv preprint arXiv:1904.03486*, 2019.
- [84] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding,” *arXiv preprint arXiv:1807.03748*, 2018.
- [85] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9729–9738.
- [86] W. Xia, C. Zhang, C. Weng, M. Yu, and D. Yu, “Self-supervised text-independent speaker verification using prototypical momentum contrastive learning,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 6723–6727.

List of Publications

Journal papers

1. **Yingke Zhu**, Brian Mak, "Bayesian self-attentive speaker embeddings for text-independent speaker verification," *submitted to IEEE/ACM Transactions on Audio, Speech, and Language Processing*.
2. Zhili Tan, Man-Wai Mak, Brian Mak, **Yingke Zhu**, "Denoised Senone I-Vectors for Robust Speaker Verification," in *IEEE Transactions on Audio, Speech and Language Processing*, 2018.

Conference papers

1. **Yingke Zhu**, Brian Mak, "Orthogonal training for text-independent speaker verification," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2020.
2. **Yingke Zhu**, Brian Mak, "Orthogonality regularizations for end-to-end speaker verification," in *Odyssey: The Speaker and Language Recognition Workshop*, 2020.
3. **Yingke Zhu**, Tom Ko, Brian Mak, "Mixup Learning Strategies for Text-independent Speaker Verification," in *Proceedings of Interspeech*, 2019.
4. **Yingke Zhu**, Tom Ko, David Snyder, Brian Mak, Daniel Povey, "Self-Attentive Speaker Embeddings for Text-Independent Speaker Verification," in *Proceedings of Interspeech*, 2018.
5. **Yingke Zhu**, Brian Mak, "Speeding Up Softmax Computations in DNN-based Large Vocabulary Speech Recognition by Senone Weight Vector Selection," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2017.

6. **Yingke Zhu**, Brian Mak, "An Investigation of Adaptation Techniques for Building Acoustic Models for Hearing-impaired Children in a CAPT Application," in *Proceedings of the International Symposium of Chinese Spoken Language Processing*, 2016.
7. Zhili Tan, **Yingke Zhu**, Man-Wai Mak, Brian Mak, "Senone I-Vectors for Robust Speaker Verification," in *Proceedings of the International Symposium of Chinese Spoken Language Processing*, 2016.