DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

Machine Learning for Text-Independent Speaker Verification

How to Teach a Machine to Recognize Human Voices

STEFANO IMOSCOPI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Abstract

The aim of speaker recognition and verification is to identify people's identity from the characteristics of their voices (voice biometrics). Traditionally this technology has been employed mostly for security or authentication purposes, identification of employees/customers and criminal investigations. During the last decade the increasing popularity of hands-free and voice-controlled systems and the massive growth of media content generated on the internet has increased the need for techniques to automatically and accurately analyse speech signals.

Speaker recognition is thus becoming a fundamental block for the smart analysis of speech in video and audio content, along with other technologies like speech recognition and diarization. Examples of useful applications of these technologies are query-by-voice, automatic subtitling and automatic metadata generation for movies and television.

In this thesis we evaluate different state-of-the-art techniques for text-independent speaker verification on a large database of read English speech (LibriSpeech ASR corpus). The different techniques are compared in terms of classification accuracy, scalability and robustness to noise.

A classification approach based on discriminatively trained Artificial Neural Networks (ANNs) is presented, showing superior classification performance to traditional generative models like Gaussian Mixture Models (GMMs) and I-vectors.

The core contribution of the thesis is a novel hybrid generative/discriminative method, using ANNs and a GMM-Universal Background Model (UBM) to obtain state-of-the-art speaker recognition results. The advantage of the new system is the possibility of using ANNs while maintaining complete scalability: an arbitrary number of new speakers can be added to the system without the need of retraining the speaker models. At the same time the system achieves very good performance, with only 0.23% Equal Error Rate (EER) in verification mode and 99.6% classification accuracy on a dataset of 2483 speakers, both male and female.

Sammanfattning

Syftet med talarigenkänning och verifiering är att identifiera människors identitet utifrån de egenskaper som karakteriserar deras röster (röstbiometri). Traditionellt har denna teknologi främst använts inom säkerhets och autentiseringsområdet för att identifiera anställda/kunder eller personer i brottsutredningar. Under det senaste decenniet har den successivt ökande populariteten för handsfree och röststyrda system och den massiva ökningen av medieinnehåll som genereras på Internet ökat behovet av tekniker för att automatiskt och noggrant analysera talsignaler.

Talarigenkänning är därmed på väg att bli en grundläggande teknologi för smart analys av tal i video och ljud, tillsammans med andra teknologier som taligenkänning och diarieföring. Exempel på användbara tillämpningar av dessa teknologier är röstbaserade användarinterface, automatisk textning och automatisk generering av metadata för film och TV.

I detta examensarbete utvärderar vi olika state-of-the-art tekniker för text oberoende talarverifiering på en stor databas av engelska ljudböcker (LibriSpeech ASR corpus). De olika teknikerna jämförs i termer av klassificeringsnoggrannhet, skalbarhet och robusthet för buller.

En klassificeringsmetod baserad på diskriminativt tränade Artificiella Neurala Nätverk (ANN) presenteras. Metoden visar överlägsen klassificeringsprestanda över traditionella generativt tränade modeller som Gaussiska mixturmodeller (GMMer) och i-vektorer.

Huvudbidraget i examensarbetet är en ny hybrid generativ/diskriminativ metod som använder en kombination av en ANN och en GMM-Universal bakgrundsmodell (UBM) för att uppnå state-of-the-art talarigenkänningsresultat . Fördelen med det nya systemet är möjligheten att använda ANN med bibehållen skalbarhet: ett godtyckligt antal nya talare kan adderas till systemet utan behov av omträning av talarmodellerna. Samtidigt uppnår systemet mycket bra prestanda, endast 0,23% Equal Error Rate (EER) i verifieringsläget och 99,6% klassificeringsnoggrannhet på ett dataset med 2483 talare, både män och kvinnor.

Acknowledgment

I would like to express my deepest gratitude to the audio mining team at Ericsson research: Sigurdur Sverrisson, Erlendur Karlsson, Volodya Grancharov, Harald Pobloth and Fredrik Jansson. Thank you for the continuous support and the fruitful discussions we had during the months of working on this project.

Thanks to my supervisor/examiner prof. Saikat Chatterjee, for being there when I needed and his wise suggestions in the middle of the project. Also thanks to prof. Markus Flierl for the supervision during my final year at KTH.

A mention also goes to the other students at Ericsson research, in particular Denis and Lorenzo, for the fun times and lunch breaks spent together that made the workplace an enjoyable place to be.

Also many thanks to all my friends from EIT Digital Master School, both in Delft and Stockholm, and my friends at Karolinska Institute. You really made this two year journey a special one and I don't know if I could have done it without you: Denis, Taha, Karina, Yue, Zhao, Sun, Andrea, Federico, Yonatahn, Lotte, Riccardo, Federica, Anna, Dennis, Lorenzo, Diana and all the others!

Finally I want to send my love to my parents, Walter and Tiziana, and my brother Lorenzo in Italy, for always being the best example they could be and for their wise support and guidance that really kept me going during all my studies abroad.

Contents

List of Figures

List of Tables

Acronyms

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DET	Detection Error Trade-off
DNN	Deep Neural Network
EER	Equal Error Rate
EM	Expectation Maximization
FAR	False Acceptance Rate
FFT	Fast Fourier Transform
FRR	False Rejection Rate
GMM	Gaussian Mixture Model
LDA	Linear Discriminative Analysis
MAP	Maximum A Posteriori
MFCC	Mel-Frequency Cepstral Coefficient
ML	Maximum Likelihood
NAG	Nesterov's Accelerated Gradient
NAP	Nuisance Attribute Projection
PDF	Probability Density Function
PLDA	Probabilistic Linear Discriminative Analysis
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
SVM	Support Vector Machine
UBM	Universal Background Model
VAD	Voice Activity Detector
VQ	Vector Quantization
WCCN	Within-Class Covariance Normalization

Chapter 1

Introduction

1.1 Motivation

The field of machine learning has received significant attention in recent years, both in academia, where it is raising interesting scientific questions and becoming an important part of artificial intelligence, and in industry, with a large number of useful applications, from image and speech recognition to recommendation systems.

The latest trend is that of deep learning, or the use of Artificial Neural Networks (ANNs) with multiple layers of hidden neurons stacked in a hierarchy. The idea of Deep Neural Networks (DNNs) is a very old and sound one, having connections with biology and neuroscience. The brain has been shown to process information through a stacked hierarchy of feature detectors, a nice example being the mammalian visual system [1]. Unfortunately it was never possible to really explore the power of deep learning, due to the vast complexity of the training process and the large number of parameters in the model, requiring a huge amount of computational power. The field remained then confined to the use of single layer non-linear neural networks, nowadays referred to as "shallow learning".

The field of deep learning came to life in 2006 with Geoffry Hinton' seminal paper [2], in which it was shown that DNNs could be trained effectively, via an unsupervised pre-training of each inidividudal layer. A wave of renewed interest in ANNs followed leading to deep learning state-of-the-art results on many tasks, in particular huge improvements in object recognition and speech recognition. Thanks to the larger amount of training data and computational power, deep learning is flourishing, even making mainstream news when Go playing AI AlphaGO defeated 18-time Go world champion Lee Sedol [3]. New successful applications of deep learning are multiplying: despite the fact that we lack a real theoretical understanding on this technology, it works and is hugely successful on many tasks.

The goal of speaker recognition and verification is the identification of people's identity from the characteristics of their voices (voice biometrics). Traditionally this technology has been employed mostly for security or authentication purposes, identification of employees/customers and criminal investigations. In the last decade the increasing popularity of hands-free and voice-controlled systems and the massive growth of media content generated on the internet has increased the need for techniques to automatically and accurately analyse speech signals.

Speaker recognition is thus becoming a fundamental block for the smart analysis of speech in video and audio content, along with other technologies like speech recognition and diarization. Examples of useful applications of these technologies are query-by-voice, automatic subtitling and automatic metadata generation for movies and television.

The dominant approach to speaker verification has traditionally been based on generative models, in particular Gaussian Mixture Models (GMMs) [4]. More recently a new appraoch, now called the i-vector method [5], led to state-of-the-art results on many benchmark datasets. In the i-vector method each speech utterance is mapped to a low-dimensional fixed-size vector through accumulated statistics from a GMM-Universal Background Model (UBM) followed by total factor analysis.

With the majority of papers in speaker verification following generative learning and the I-vector or GMM paradigm, we felt discriminatively trained deep ANNs had not been explored in depth for this particular task. Motivated by the recent success of deep learning, the focus of the thesis has then been on ANNs and the possibility of their use to outperform existing generative learning methods.

The different techniques are compared in terms of classification accuracy, scalability and robustness to noise on a large-scale database of read English speech: the LibriSpeech ASR corpus [6].

A novel scalable approach based on ANNs in combination with a GMM-UBM is presented, showing superior classification performance to traditional GMMs while retaining the advantages of generative models in terms of scalability, parallel processing and memory requirements.

1.2 Structure of the report

The report is organized as follows. Chapter 2 gives the necessary background in speech processing and speaker verification, including a description of the traditional and state of the art techniques for speaker verification used as a comparison baseline: GMMs and i-vectors. A brief literature review of related work carried out in speaker verification is also present in this chapter.

Chapter 3 describes in detail the use of a DNN for speaker identification and verification, which is necessary to understand the training of the novel method proposed in the following chapter.

Chapter 4 presents the core contribution of the thesis, which is a scalable hybrid method for speaker verification using neural networks along with a GMM-UBM.

Chapter 5 presents and analyses the results of simulations and Chapter 6 concludes the thesis with a discussion and directions for future work.

Chapter 2

Background and Related Work

In this chapter we provide the essential background in speech processing and speaker verification/identification needed throughout the thesis. For a more in depth review of text-independent speaker recognition see [7].

2.1 Speaker Verification and Speaker Identification

Speaker recognition technologies can be divided into two major applications.

- *speaker verification*: the goal is to determine if a test utterance belongs to a single particular speaker (i.e., determining whether an unknown voice is from a particular enrolled speaker). It can be thought as a template matching, thus the decision is binary, the system can accept or reject the utterance. Examples of this are query-by-voice or voice authentication systems.
- *speaker identification*: the goal is to determine an unknown speaker identity. There is no particular target speaker, so the test utterance is compared with a big set of known enrolled speakers and the identity is recognized as the most likely voice from the set. It is a 1-to-N comparison. An example of this is identity recognition for forensic investigations, where the test utterance is compared with all the voice models in the database of potential criminals.

Another fundamental distinction is between text-dependent and text-independent speaker recognition. In *text-dependent recognition* all the utterances must be recordings of the same phrase. Knowledge about the syntactic content and temporal alignment can thus be used to improve accuracy, but the system is much more constrained and doesn't have many applications besides security authentication.

Text-independent recognition is more general, where the utterances can have variable length and different phonetic content.

2.1.1 Evaluation metrics

Speaker identification is a multi-class classification problem, so the natural evaluation metric is the classification accuracy (or error) on an held-out test set. In case of imbalanced classes, other evaluation metrics can be used, like Precision, Recall and F-Scores. See [8] for a review of performance measures of classification problems.

Speaker verification presents a trade-off between False Acceptance Rate (FAR) and False Rejection Rate (FRR), also called False Positive Rate and False Negative Rate. Given the number of false positives (FP), false negatives (FN), true positives (TP) and true negatives (TN), the evaluation metrics of interest for speaker verification are

$$FAR = \frac{FP}{FP + TN} \tag{2.1}$$

$$FRR = \frac{FN}{FN + TP} \tag{2.2}$$

The classification depends on a threshold above which the utterance is accepted as the target speaker. Moving the threshold changes the trade-off between FAR and FRR. The operating point must thus be set depending on the target application. To evaluate the accuracy of speaker verification algorithms it is common to inspect the Detection Error Trade-off (DET) curve [9] and report the Equal Error Rate (EER). This is the error rate at the operating point for which FAR and FRR are the same. The only problem with EER is the arbitrary selection of the threshold, so NIST introduced the Detection Cost Function, using its minimum value as the performance measure for speaker verification. Details can be found in [7]. Figure 2.1 shows an example comparison of multiple systems using DET curves.

In this thesis we will focus on text-independent and gender-independent recognition of English speech. The main focus will be on speaker verification, but also speaker identification will be discussed, mainly in the context of a multi-class DNN. Results will be reported in terms of classification error for the identification task and EER from DET curves for the verification task.

2.2 Literature Review of Machine Learning for Text-Independent Speaker Verification

A large number of learning algorithms have been applied to speaker recognition. The dominant approach for text-independent verification is based on GMMs [4, 10]. A generative type of learning is employed, where each speaker data is modelled by a probability distribution, in particular a mixture of gaussians fitted with the Expectation Maximization (EM) algorithm [11].

More recently the use of a Support Vector Machine (SVM) has been investigated, both as a stand-alone module [12, 13] or in combination with a GMM to increase accuracy [14].

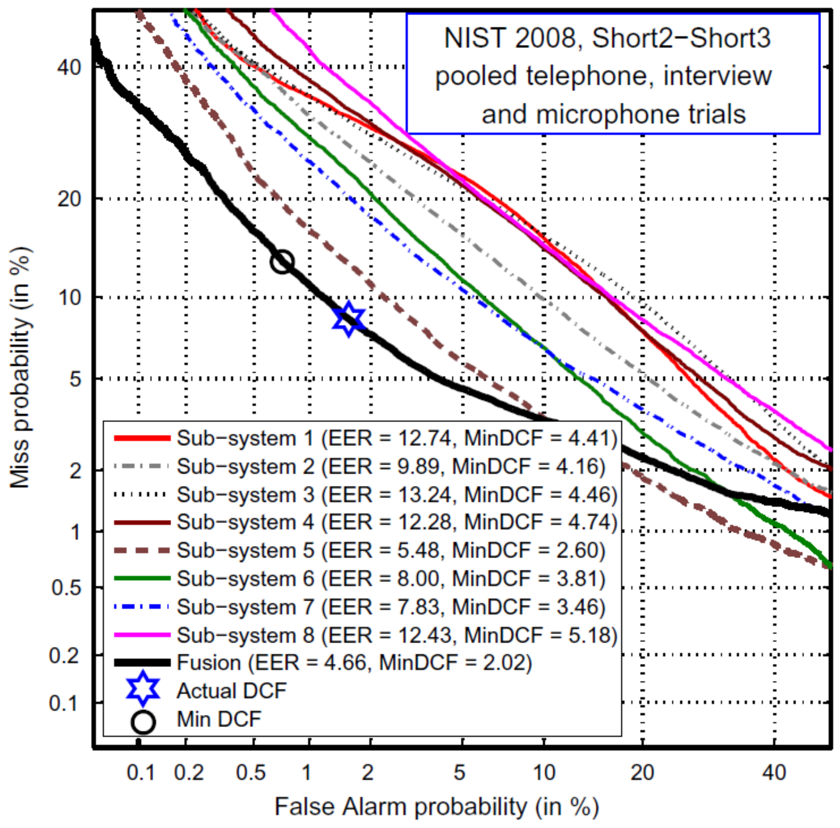


Figure 2.1: Example of DET curves comparing various speaker verification subsystems. Taken from [7].

A recent research trend has been that of supervectors. A supervector is a high- and fixed-dimensional representation of an utterance, which eliminates the problem of representing utterances with variable length. A GMM-UBM is used to map an utterance to the supervector through Maximum A Posteriori (MAP) adaptation of the means. The stacked adapted means are the supervector representation of the utterance. Supervectors in combination with SVM have shown good performance in [15].

A classic problem in speaker verification is that of intersession variability, in particular of mismatch between training and test conditions. When the test utterances are recorded through a different channel or with different environmental backgrounds the performance drops drastically. Various techniques have been applied in the SVM framework, the most successful one being Nuisance Attribute Projection (NAP) [16], a transformation that removes the directions of unwanted session variabilities from the supervector space.

Factor analysis techniques on GMM supervectors have been investigated recently to try to decouple the speaker dependent factors, which are important for speaker verification, from the channel and noise attributes. Joint factor analysis, first proposed for speaker verification in [17] and refined in [18], has recently led to the i-vector method [5], where a total variability transformation is used to map each speaker's utterances to a low dimensional vector. Each speaker is modelled by a single vector that can be used as a template for scoring using cosine distance. The i-vector method represents the current state-of-the-art method for text-independent speaker verification, in conjunction with various channel compensation techniques like Within-Class Covariance Normalization (WCCN), Linear Discriminative Analysis (LDA) and Probabilistic Linear Discriminative Analysis (PLDA) [19].

Finally ANNs have been applied to many patter recognition tasks, including speaker identification, in particular with the goal of learning features which are robust to channel variability [20]. They are also being investigated for speaker verification, usually in conjunction with an i-vector system [21, 22], where a DNN trained for speech recognition replaces the GMM in the extraction of the sufficient statistics for the i-vector extraction. A discriminatively trained DNN can also be used as an extractor of deep bottleneck features which can be used for speaker verification, as in [23].

2.3 Feature extraction

Speaker verification is fundamentally a classification problem. As in most pattern recognition problems it is necessary to represent the signals in a convenient space for classification. Speech is a complex time-varying signal containing not only speech and speaker-related content, but also information about the medium and channel through which the utterance was recorded. Indeed noise and reverberation are often responsible for the degradation of many pattern recognition systems working on speech signals. There is thus a need to extract features that retain as much speech-related information as possible, while being robust to noise. To keep complexity under control it is also desirable to have a low dimensional feature vector.

In this thesis we decided to use Mel-Frequency Cepstral Coefficients (MFCCs) [24], the standard feature used in the research community for speech and speaker recognition. They consist of a compact cepstral representation of the speech signal which retains the speaker-related characteristics we need (mostly related to the anatomy of the vocal tract) and in which the channel effects become additive and can thus be partially compensated by spectral mean subtraction.

The extraction of MFCC features is shown in Figure 2.2. The signal is first split into frames of 20ms in length with an overlap of 10ms. On such a window speech is stationary and can thus be analysed reliably in the spectral domain. An energy-based Voice Activity Detector (VAD) is employed to remove non-speech frames, and a pre-emphasis filter $1 - (\mu z^{-1})$ with $\mu = 0.98$ is applied to the input speech. Each frame is multiplied with a Hanning window to get rid of discontinuities at the boundaries of the frames that would create problems in the spectral domain. Next the signal is mapped to the frequency domain by taking a Fast Fourier Transform (FFT) and keeping the magnitude of the spectrum. The magnitude spectrum is passed throguh a Mel-spaced [25] triangular filter-bank followed by logarithmic compression. The Mel frequency scale is psychoacoustically motivated, as is the logarithmic transformation of the amplitude, thus the log-mel features retain the most perceptually critical information. Finally a Discrete Cosine Transform (DCT) is performed to compact the energy in a small number of coefficients.

The 24 lowest DCT AC coefficients are kept as the cepstral representation of each speech frame. The resulting feature vectors are mean and variance normalized over the whole utterance to compensate for channel effects, as described in [26].

Our implementation of MFCC extraction is based on the one in the HTK toolkit [27]. Through this signal processing front-end, the original time-domain utterance is transformed into a time sequence of 24-dimensional vectors that are fed to the input of machine learning algorithms.

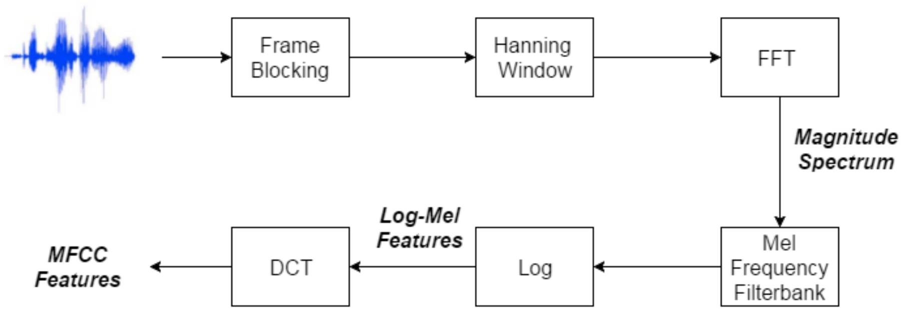


Figure 2.2: MFCC features extraction

2.4 Mathematical background

In this section we establish the core mathematical notation and background needed to understand the speaker verification problem and the learning algorithms presented in the following chapters and also to better interpret the results of the experiments.

2.4.1 Dataset notation

As described in section 2.3, feature extraction maps each frame of the time-domain speech signal $x(t)$ to a 24-dimensional vector of MFCCs. Utterance u_i is thus represented by a time sequence of $N_F(u_i)$ vectors: $X_i = \{\boldsymbol{x_{i,j}} : j = 1 : N_F(u_i)\}$, which is the input to the final trained system at test time.

The utterances are spoken by a set of N_S speakers, both female and male, each speaker s_i having a number $N_U(s_i)$ of spoken utterances.

The complete notation for the dataset is summarized in table 2.1.

2.4.2 Speaker recognition with Gaussian Mixture Models

A Gaussian Mixture Model (GMM) is a Probability Density Function (PDF) given by the weighted sum of Gaussian densities, called the components of the GMM. It is used to model the distribution of the feature vectors of each speaker in the MFCC space. It is the most used model in speaker recognition, thanks to its good modelling of a large class of sample distributions. One of the nice properties of the GMM is its ability to form smooth approximation of arbitrarily shaped PDFs. The parameters of the model are learned from the

$S = \{s_i : i = 1 : N_S\}$	Set of Speakers. The speakers, both male and female, are indexed from 1 to N_S.
$U_i = \{u_{i,j} : j = 1 : N_U(s_i)\}$	Sequence of Utterances for speaker s_i. Total number of utterances for speaker s_i is $N_U(s_i)$
$U = \{u_i : i = 1 : N_U\}$	Sequence of Utterances. Total number of utterances is N_U.
$X_i = \{\boldsymbol{x_{i,j}} : j = 1 : N_F(u_i)\}$	Sequence of Feature Vectors for utterance u_i. The number of feature vectors in utterance u_i is $N_F(u_i)$ and the length of each feature vector is L_F.
$X^i = \left\{\boldsymbol{x^i_j} : j = 1 : N_F(S_i)\right\}$	Sequence of Feature Vectors for speaker i. This is the complete sequence of all feature vectors over all utterances of speaker s_i, where the total number of feature vectors for speaker s_i is $N_F(s_i) = \sum_{j=1}^{N_U(s_i)} N_F(u_j)$.
$X = \{\boldsymbol{x_i} : i = 1 : N_F\}$	Sequence of Feature Vectors. This is the complete sequence of all feature vectors over all utterances, where the total number of feature vectors is $N_F = \sum_{i=1}^{N_U} N_F(u_i)$.

Table 2.1: Dataset Notation

training data via the Expectation Maximization (EM) algorithm [28], which achieves Maximum Likelihood (ML) estimation with an iterative process.

A GMM with M components is defined as

$$p(\boldsymbol{x}|\lambda) = \sum_{c=1}^{M} w_c \, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c}) \tag{2.3}$$

where $\boldsymbol{x}$ is a D-dimensional feature vector, $w_c, c = 1, ..., M$ are the mixture weights and $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c}), c = 1, ..., M$ are the component Gaussian densities, each of them distributed according to a D-variate probability density function of the form,

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma_c}|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu_c})^{*}\boldsymbol{\Sigma_c}^{-1}(\boldsymbol{x} - \boldsymbol{\mu_c})\right\} \tag{2.4}$$

with mean vector $\boldsymbol{\mu_c}$ and covariance matrix $\boldsymbol{\Sigma_c}$. The mixture weights must satisfy the constraint that $\sum_{c=1}^{M} w_c = 1$. The complete GMM is parametrized by the mean vectors, covariance matrices and mixture weights from all components. These parameters are collectively expressed as,

$$\lambda = \{w_c, \boldsymbol{\mu_c}, \boldsymbol{\Sigma_c}\} \quad c = 1, ..., M. \tag{2.5}$$

There are many possibilities in the parametrization of the GMM. The covariance matrices can be made diagonal to reduce the number of parameters, or a full rank covariance matrix can be shared between all components. Also the number of components must be chosen a priori. Usually the choice is made based on the amount of training data.

It must be noted that using diagonal covariances doesn't limit the representational power of GMMs. A large number of diagonal mixtures can fit complicated distributions presenting correlation between features. Usually the use of a GMM with full rank covariances or a diagonal covariance GMM with more components lead to similar results.

The aim of ML estimation is to find the model parameters λ that maximize the likelihood of the GMM given the training data. Given a sequence of N_F training vectors $X = \{\boldsymbol{x_1}, ..., \boldsymbol{x_{N_F}}\}$ the likelihood of the data under the model, assuming independence between training vectors, can be expressed as

$$p\left(X|\lambda\right) = \prod_{j=1}^{N_F} p\left(\boldsymbol{x_j}|\lambda\right) \tag{2.6}$$

Direct maximization is not possible, but the ML parameters can be obtained iteratively using a special case of the EM algorithm. An initial model λ is initialized using Vector Quantization (VQ) and at each iteration a new model $\bar{\lambda}$ is estimated, such that $p\left(X|\bar{\lambda}\right) \geq p\left(X|\lambda\right)$. The following are the EM update formulas for a GMM with M components and diagonal covariance matrix. They have to be applied for all mixtures ($c = 1, ..., M$) and guarantee a monotonic increase in the model likelihood. The iterative process is repeated until convergence.

Mixture Weights

$$\bar{w_c} = \frac{1}{N_F}\sum_{j=1}^{N_F} \gamma_c(\boldsymbol{x_j}) \tag{2.7}$$

Means

$$\bar{\boldsymbol{\mu_c}} = \frac{\sum_{j=1}^{N_F} \gamma_c(\boldsymbol{x_j})\, \boldsymbol{x_j}}{\sum_{j=1}^{N_F} \gamma_c(\boldsymbol{x_j})} \tag{2.8}$$

Variances

$$\bar{\boldsymbol{\sigma_c}}^2 = \frac{\sum_{j=1}^{N_F} \gamma_c(\boldsymbol{x_j})\, diag(\boldsymbol{x_j}\boldsymbol{x_j}^*)}{\sum_{j=1}^{N_F} \gamma_c(\boldsymbol{x_j})} - \bar{\boldsymbol{\mu_c}}^2 \tag{2.9}$$

Where $\gamma_c(\boldsymbol{x_j})$ is the *a posteriori* probability of component c given vector $\boldsymbol{x_j}$, also called *responsibility* of component c

$$\gamma_c(\boldsymbol{x_j}) = Pr(c|\boldsymbol{x_j}, \lambda) = \frac{w_c\, \mathcal{N}(\boldsymbol{x_j}|\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c})}{\sum_{k=1}^{M} w_k\, \mathcal{N}(\boldsymbol{x_j}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \tag{2.10}$$

For further readings on the EM algorithm see [11, 29].

The first step in developing the speaker verification system is to train one GMM for each speaker s_i, with $i = 1, ..., N_S$. The EM algorithm is applied on each set of feature vectors X^i separately, obtaining N_S speaker models.

Another key component is the Universal Background Model (UBM), a GMM trained over all the feature vectors pooled from all speakers (or a large enough number of speakers). This bigger GMM models the average distribution of speech in the feature space, across a large population of speakers with many different voice biometrics (pitch, vocal tract length, gender, age). The UBM is necessary to normalize the scores of each speaker and it can be thought of as a common reference against which each speaker is discriminated.

There are many approaches to training a UBM. The simplest one is to pool together all the training data coming from all speakers and fit the UBM to it using the EM algorithm (Fig 2.3a). It's important to make this pooled training data balanced over the different classes present in the dataset. For example there should be the same amount of male and female speech when training a gender-independent UBM.

The other approach is to fit separate UBMs over the sub-populations present in the dataset, for example training one UBM for male speech and another one for female speech (Fig 2.3b). The two models are then combined after training. This approach is slightly more complicated, but one advantage is that different amounts of data can be used for training the UBMs, and the balance between them can be tuned after training when combining them in a single model.

Given the test utterance u_i and the sequence of its frames X_i, we test the utterance against two mutually exclusive hypothesis

H_0: u_i is spoken by the hypothesized target speaker.
H_1: u_i is **not** spoken by the hypothesized target speaker.

The optimal decision between the two hypothesis is given by the likelihood ratio test

$$\frac{p\,(X_i|H_0)}{p\,(X_i|H_1)} = \begin{cases} \geq \Theta, & \text{accept } H_0 \\ < \Theta, & \text{reject } H_0 \end{cases} \tag{2.11}$$

where Θ is the decision threshold, usually a speaker-independent parameter set after testing the system to obtain the desired trade-off between False Acceptance and False Rejection.

The target speaker GMM models the positive hypothesis, $p\,(X_i|H_1)$, while the UBM models the negative hypothesis, $p\,(X_i|H_1)$.

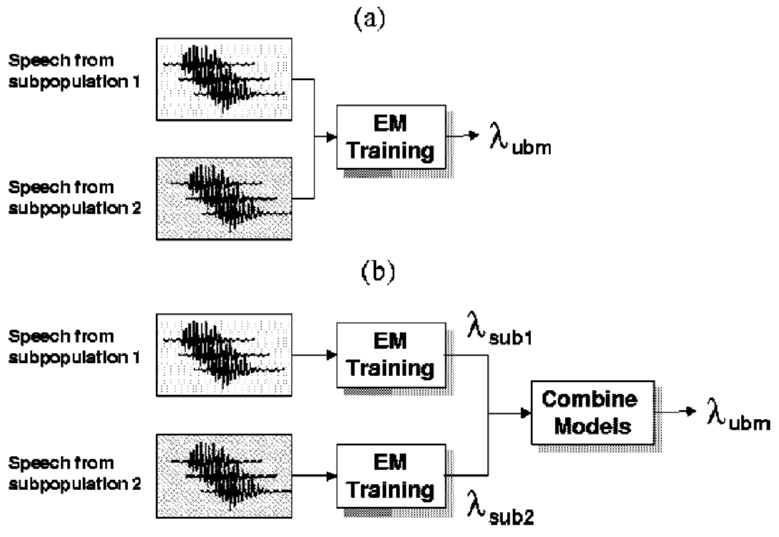


Figure 2.3: Data and model pooling approaches for creating a UBM. (a) Data from sub-populations are pooled prior to training the UBM via the EM algorithm. (b) Individual sub-population models are trained and then combined (pooled) to create final UBM. Taken from [10].

In practice the frames X_i of the utterance u_i are assumed independent over time and the log-likelihood is used to accumulate the scores and obtain the log-likelihood of the whole utterance.

$$\log p\left(X_i|\lambda\right) = \sum_{j=1}^{N_F(u_i)} \log p\left(\boldsymbol{x_j}|\lambda\right) \tag{2.12}$$

The log-likelihood ratio is defined as

$$\Lambda\left(X_i\right) = \log p\left(X_i|\lambda_{H_0}\right) - \log p\left(X_i|\lambda_{H_1}\right) \tag{2.13}$$

and the decision, given the threshold $\theta = \log \Theta$, is taken according to

$$\Lambda\left(X_i\right) = \begin{cases} \geq \theta, & \text{accept } H_0 \\ < \theta, & \text{reject } H_0 \end{cases} \tag{2.14}$$

The same system can be used to perform closed-set speaker identification, in which case the most likely speaker for test utterance u_i is given by

$$\hat{s} = \arg\max_j \log p\left(X_i|\lambda_{s_j}\right) \quad j = 1, ..., N_S \tag{2.15}$$

where λ_{s_j} is the GMM of speaker j.

Much research has been conducted in order to improve the performance of the GMM-UBM system for speaker verification. In particular MAP adaptation of speaker models from the UBM has been shown to be promising [10]. Also many different scoring techniques have been tried to compensate for channel variability and other mismatched conditions between training and test data.

For further reference on these techniques and GMMs for speaker recognition we refer to [4, 30–32].
Figure 2.4 and Figure 2.5 show the scoring of a speech utterance in speaker verification and speaker identification mode respectively.

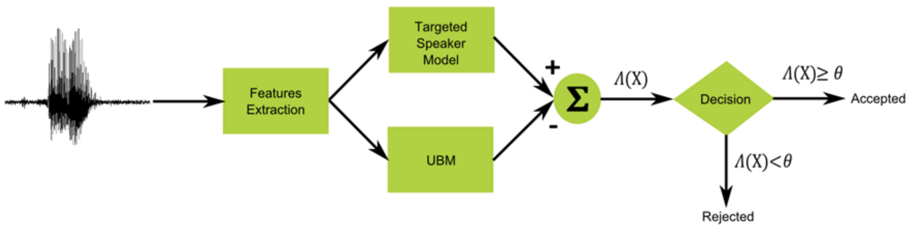


Figure 2.4: GMM-UBM based speaker verification

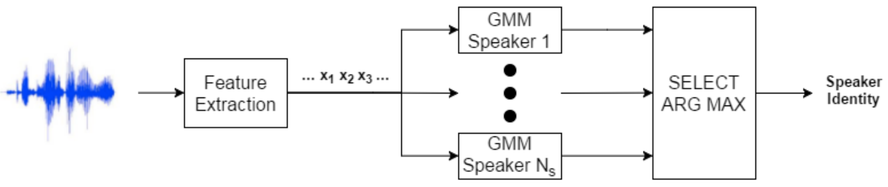


Figure 2.5: GMM based speaker identification

In our case we trained a GMM-UBM following the data pooling approach of Figure 2.3a. We created a balanced training set, taking 10 to 15 utterances per speaker from a set of 330 speakers with same amount of female and male speakers. The pooled training data for the UBM consisted of approximately 20 hours of speech. We chose a GMM-UBM with 1024 components and diagonal covariances.

For the individual speaker models we decided to use GMMs with 32 components and diagonal covariances Σ_i, resulting in around 80 training examples per parameter.

2.4.3 Speaker recognition with i-vectors

The goal of the i-vector method is to represent each speaker with a fixed-dimensional vector and perform classification with a simple distance measure in the i-vector space.

This is achieved by performing factor analysis on the supervectors, mapping each utterance to a low-dimensional space. This space is named the total variability space because it models both speaker and channel variabilities. Techniques like LDA and WCCN are used to obtain features with strong inter-speaker variability and low sensitivity to other factors like channel conditions or within speaker variabilities.

The first step is to use the GMM-UBM to extract Baum-Welch statistics for each utterance. These statistics are necessary to construct the super-vectors and are defined as

$$N_c(u_i) = \sum_{j=1}^{N_F(u_i)} \gamma_c(\boldsymbol{x_{i,j}}) \tag{2.16}$$

$$F_c(u_i) = \sum_{j=1}^{N_F(u_i)} \gamma_c(\boldsymbol{x_{i,j}})\boldsymbol{x_{i,j}} \tag{2.17}$$

$$S_c(u_i) = diag\left(\sum_{j=1}^{N_F(u_i)} \gamma_c(\boldsymbol{x_{i,j}})\boldsymbol{x_{i,j}}\boldsymbol{x^*_{i,j}}\right) \tag{2.18}$$

with $\gamma_c(\boldsymbol{x_{i,j}})$ as the posterior probability that feature vector $\boldsymbol{x_{i,j}}$ (j-th vector for utterance i) is generated by the mixture component c calculated with the UBM, as shown in equation 2.10.
N_c, F_c and S_c are the 0^{th}, 1^{st} and 2^{nd} order Baum-Welch statistics respectively.

The 1^{st} order statistics, relative to the mean-values of the feature vectors under soft alignment with each UBM component, are stacked to form the super-vector representation. The super-vector for utterance i is defined as:

$$M_i = \begin{bmatrix} F_1(u_i) \\ \vdots \\ F_c(u_i) \\ \vdots \\ F_{N_C}(u_i) \end{bmatrix} \tag{2.19}$$

The dimension of the supervectors is $N_M = N_C L_F$.

The i-vector method models the supervectors for the different utterances as a linear combination of latent variables (the total factors) from a subspace of much smaller dimensionality, thus achieving a great dimensionality reduction. The decomposition of supervector M_i is:

$$M_i = M + Tw_i \tag{2.20}$$

where M is the speaker and channel-independent supervector, which can be obtained from the UBM by stacking the mean vectors of each mixture component. M_i and M are supervectors of dimension $N_M \times 1$.

The assumption is that for a randomly chosen utterance u_i, M_i is Gaussian distributed with mean M and covariance matrix B. Since M is completely determined by the UBM, the problem is how to estimate the supervectors covariance matrix B. Despite having very high dimensionality, in practice B is a low rank matrix (the rank is bounded by the number of training speakers), which allows to derive an exact solution to the estimation problem.

For any i, it must hold that

$$M_i \sim \mathcal{N}(M, B) \tag{2.21}$$

If N_I is the rank of B, we can approximate it with a rectangular matrix of smaller dimensionality as follows:

$$B = TT^* \tag{2.22}$$

where T is defined as the total variability matrix of dimension $N_M \times N_I$ and rank N_I, where N_I is much smaller than N_M, typically 200-300. This implies there is a unique vector w_i such that equation 2.20 holds true. The assumption of equation 2.21 is equivalent to saying that w_i is a random vector of size $N_I \times 1$, having a standard normal distribution $\mathcal{N}(0, I)$. Its elements are the total factors and we refer to this new vector as the identity vector of utterance i (i-vector for short).

Let us now define Σ as a diagonal covariance matrix of dimension $N_M \times N_M$, which models the residual variability not captured by the total variability matrix T. To explain this further, if X_i is a feature vector from utterance u_i and mixture component c, we have that

$$X_i = M_{i,c} + E \tag{2.23}$$

where the residual E follows the distribution $\mathcal{N}(0, \Sigma_c)$. The subscript c refers to subvectors corresponding to mixture component c. Σ is therefore a bock matrix composed of diagonally stacking the submatrices Σ_c.

The goal in training the i-vector model is to get ML estimates of the matrices T and Σ, which can then be used to extract the i-vectors w_i for each utterance u_i. The objective function to be maximized over (T, Σ) is the log-likelihood of the data X_i, denoted as:

$$\prod_{u_i} P_{T,\Sigma}(X_i) \tag{2.24}$$

where the product extends over all utterances in the training set. A closed form expression for this likelihood, as well as the complete derivation and the proofs of the EM algorithm to maximize it, is given in [33]. We will now focus on presenting the notation and core equations that are necessary to implement this EM algorithm, in order to train the matrix T and extract the i-vector w_i for each utterance.

Given I the L_F-dimensional identity matrix, let us define a new vectorized notation for the Baum-Welch statistics.

$$NN(u_i) = \begin{bmatrix} N_1(u_i)I & & & & \\ & \ddots & & & \\ & & N_c(u_i)I & & \\ & & & \ddots & \\ & & & & N_{N_C}(u_i)I \end{bmatrix} \tag{2.25}$$

$$FF(u_i) = \begin{bmatrix} \tilde{F}_1(u_i) \\ \vdots \\ \tilde{F}_c(u_i) \\ \vdots \\ \tilde{F}_{N_C}(u_i) \end{bmatrix} \tag{2.26}$$

$$SS(u_i) = \begin{bmatrix} \tilde{S}_1(u_i) & & & & \\ & \ddots & & & \\ & & \tilde{S}_c(u_i) & & \\ & & & \ddots & \\ & & & & \tilde{S}_{N_C}(u_i) \end{bmatrix} \tag{2.27}$$

$NN(u_i)$ is a diagonal matrix of dimension $N_M \times N_M$, while vector $FF(u_i)$ and matrix $SS(u_i)$ are obtained from the *centralized* Baum–Welch statistics $\tilde{F}_c(u_i)$ and $\tilde{S}_c(u_i)$, defined as

$$\tilde{F}_c(u_i) = F_c(u_i) - N_c(u_i)m_c \tag{2.28}$$

$$\tilde{S}_c(u_i) = S_c(u_i) - diag\left(F_c(u_i)m_c^* + m_cF_c(u_i)^* - N_c(u_i)m_cm_c^*\right) \tag{2.29}$$

where m_c is the mean of UBM mixture component c.

Given an initial random estimate for the total variability matrix T, the i-vector for utterence i is obtained as a MAP estimate using the following equations:

$$L_i = I + T^*\Sigma^{-1}NN(u_i)T \tag{2.30}$$

$$w_i = L_i^{-1}T^*\Sigma^{-1}FF(u_i) \tag{2.31}$$

Given the i-vectors w_i for the complete set of utterances, the total variability matrix T can be updated, where each row of T is estimated through a least squares estimation as explained in Proposition 3 in [33]. First some additional statistics are accumulated across the whole set of utterances:

$$A_c = \sum_{i=1}^{N_U} N_c(u_i) L_i^{-1} \tag{2.32}$$

$$\Phi = \sum_{i=1}^{N_U} FF(u_i) \left(L_i^{-1} T^* \Sigma_M^{-1} FF(u_i) \right)^* \tag{2.33}$$

$$NN = \sum_{i=1}^{N_U} NN(u_i) \tag{2.34}$$

Then the T matrix estimate is updated as follows

$$T = \begin{bmatrix} T_1 \\ \vdots \\ T_c \\ \vdots \\ T_{N_C} \end{bmatrix} = \begin{bmatrix} A_1^{-1}\Phi_1 \\ \vdots \\ A_c^{-1}\Phi_c \\ \vdots \\ A_{N_C}^{-1}\Phi_{N_C} \end{bmatrix} \quad \textit{where} \quad \Phi = \begin{bmatrix} \Phi_1 \\ \vdots \\ \Phi_c \\ \vdots \\ \Phi_{N_C} \end{bmatrix} \tag{2.35}$$

Optionally, the covariance matrix of the residuals Σ can also be updated

$$\Sigma = NN^{-1} \left(\sum_{i=1}^{N_U} SS(u_i) - diag\left(\Phi T^*\right) \right) \tag{2.36}$$

or it can be kept constant, after initializing it by diagonally stacking the covariance matrices of the UBM.
By iterating between the update of T and w_i the estimate of T will converge in a few iterations, after which the final i-vectors for each utterance can be extracted using equations (2.30) and (2.31). It can be proven that this process iteratively solves the least-squares problem defined as

$$\min_{w_i} \|FF(u_i) - Tw_i\|^2 \tag{2.37}$$

After obtaining one i-vector for each utterance, a single speaker model can be built by averaging all i-vectors of the same speaker. The target and test i-vectors can then be compared using a similarity score called cosine distance:

$$d_{cos}\left(w_1, w_2\right) = \frac{\langle w_1, w_2 \rangle}{\|w_1\|\|w_2\|} = \cos(\alpha_{w_1,w_2}) \tag{2.38}$$

If the two i-vectors point in the same direction the cosine distance takes the highest possible value of 1, while when they point in opposite direction it takes the minimum value of -1. The score is therefore sensitive only to the angle α between the two i-vectors, regardless of their length.

The scoring of i-vectors is usually preceded by channel compensation techniques. LDA, followed by WCCN and length normalization has shown to lead to highest verification accuracy, as described in [5]. A good visualization of the effect of these techniques is shown in fig 2.6. The i-vectors become much more compact in the feature space, ideally being sorted by angle on a hypersphere around the origin (fig. 2.6b), which is ideal to discriminate between speaker models using the cosine distance.

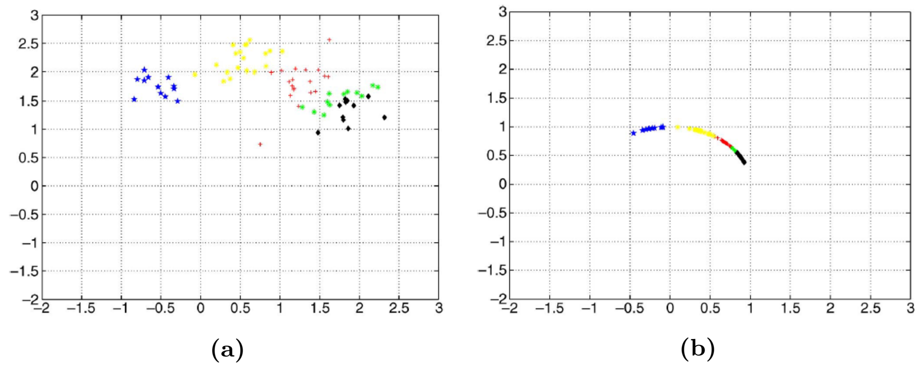


Figure 2.6: I-vector transformations for channel compensation. Taken from [5].
(a) I-vectors of five speakers after two dimensions LDA projection.
(b) I-vectors of five speakers after two dimensions LDA/WCCN projection and length normalization.

Denoting with ω_t and ω_s the channel-compensated i-vectors of the test and target speaker respectively, the verification decision is taken according to

$$d_{cos}\left(\omega_t, \omega_s\right) = \begin{cases} \geq \theta, & \text{accept } H_0 \\ < \theta, & \text{reject } H_0 \end{cases} \tag{2.39}$$

where θ is a speaker-independent threshold.

The formula for speaker recognition is

$$\hat{s} = \arg\max_j \, d_{cos}\left(\omega_t, \omega_j\right) \quad j = 1, ..., N_S \tag{2.40}$$

More work has been carried out to improve the i-vector extraction and speed-up the training of the T matrix. Also PLDA for channel compensation has been proposed in [34] and led to state-of-the-art performance on many benchmark datasets.

For further reading on the i-vector model we refer to [5, 35, 36]. For more advanced scoring techniques and improvements of PLDA see [37, 38] and [39] for optimized i-vector extraction.

Chapter 3

Deep Neural Network for Speaker Recognition

Motivated by the recent successes of deep learning on many classification tasks, we decided to explore the use of a DNN to solve the speaker identification problem.

3.1 Overview of the Field

ANNs are a family of models inspired by the brain which can learn how to approximate arbitrary multi-dimensional functions that may present complex non-linear relationships between input and output. The history of ANNs dates back to 1958, when Frank Rosenblatt created the perceptron [40], but it's only with the introduction of the backpropagation algorithm [41] by Paul Werbos in 1974 that ANNs could be trained effectively and started to be applied to many real-world pattern recognition tasks. In 1991 Kurt Hornik showed that the standard multilayer feedforward network with as few as a single hidden layer is a universal function approximator, provided this layer has a sufficient number of hidden units [42]. A graphical representation of a feedforward ANN with a single hidden layer is given in fig 3.1.

Most researchers believed that using a larger number of hidden layers could be advantageous. This is strongly motivated also by findings in neuroscience, where it is well known that the brain processes information through a stacked architecture of layers organized in a hierarchy. In the example of the mammalian visual system, the brain processes information with 6 layers of neurons, named V1 to V6. V1 contains simple receptive fields, detecting features like oriented edges, while neurons in the deeper layers V5 and V6 fire when presented with more abstract and specific patterns, for example faces or a particular object [1].

Despite this knowledge, it remained a challenge to train DNNs. There were two main problems:

- The non-convexity of the cost function to be minimized during training. These had long been one of the main theoretical disadvantages of ANNs and in particular it was really hard to make DNNs converge to a good local minima.

- The huge number of parameters and thus computational power needed to train the models on real-world datasets. Also bigger and deep networks require more training data in order to learn useful mappings without overfitting.

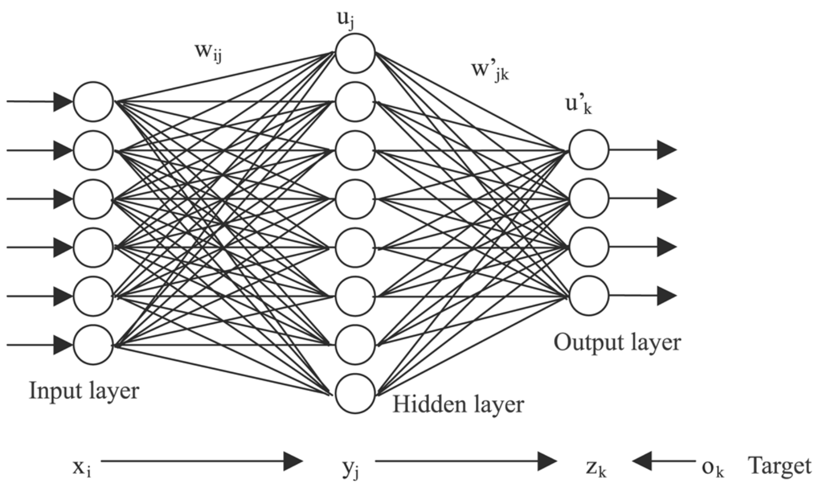


Figure 3.1: Feedforward ANN with one hidden layer

It was only around 2006 that the poor local minima problem was solved and deep learning was revived, mainly due to the work of Geoffry Hinton and his group. They showed that very deep nets could be conditioned towards a good local minima by unsupervised pre-training of one hidden layer at a time [2]. Hinton used Restricted Boltzmann Machines trained in an unsupervised fashion for the pre-training of the parameters in each layer, after which supervised training of the network converged much faster to a good local minima. From that year on, also thanks to larger computational power through the use of GPUs, deep learning has produced state-of-the-art results on many machine learning tasks, from computer vision to speech recognition and natural language processing.

Interestingly, when applied for vision and object recognition tasks, it was found that the features learned in deep architectures resemble those observed in the first two layers of the mammalian visual cortex (areas V1 and V2) [43], and that they become increasingly invariant to factors of variation (such as camera movement) in higher layers [44].

Another important finding was that the use of rectifier non-linearities in the hidden units, also called reLUs (Rectifier Linear Units), speeds up the learning of DNNs and can achieve good results even without pre-training [45].

Also the introduction of Dropout [46] and other advanced regularization techniques to prevent overfitting contributed to the success of DNNs on many real-world datasets with limited amount of training data.

3.2 Mathematical formulation

Informed by these recent findings, we present now the architecture and techniques we used to train a DNN for speaker identification.

One powerful property of an ANN is that it can solve a multiclass classification problem with a single discriminatively trained model. In particular the network can be trained to directly estimate the Bayesian *a posteriori* probability for each class, namely $p\left(C_i|X\right)$. Following the minimum-error bayesian framework, classification is performed by assigning the input to the class with the highest Bayesian posterior probability. Also interpretation of network outputs as Bayesian probabilities allows outputs from multiple networks to be combined for higher level decision making, simplifies creation of rejection thresholds and makes it possible to compensate for differences between pattern class probabilities in training and test data [47].

The architecture we decided to use is therefore a multiclass DNN with two/three hidden layers. The hidden units have reLU activation function and the architecture is fully connected. The output layer has as many units as classes (the target speakers) and uses the softmax function to estimate posterior probabilities.

Let's define some mathematical notation in order to better define these concepts. We refer to the input units as x_i with $i = 1, ..., L_F$. At each time frame the network receives one MFCC feature vector as input and feedforwards it through the network. Hidden unit j in layer l will be referred to as $y_j^{(l)}$ with $j = 1, ..., D_l$. The matrix of connections modelling the synapses between neurons of layer $l-1$ and l is $W^{(l)}$. Its entries are $w_{j,i}^{(l)}$, the synaptic weight between neuron i in layer $l-1$ and neuron j in the following layer l. The output units are z_k for $k = 1, ..., N_S$ and they have to be as close as possible to the real target of the network o_k, which is 1 for the target class and 0 for all the other speakers. Equivalently the notation can be written using vectors, for example $\boldsymbol{o}$ is a one-hot vector[1] of target posterior probabilities. The input to each hidden (or output) unit j, for all layers from 2 to the number of layers L, is a linear combination of all the units in the previous layer plus a bias term, and we will refer to it as net_j:

$$net_j^{(2)} = \sum_{i=1}^{D_1} w_{j,i}^{(2)} x_i + b_j^{(2)} \tag{3.1}$$

$$net_j^{(l)} = \sum_{i=1}^{D_{l-1}} w_{j,i}^{(l)} y_i^{l-1} + b_j^{(l)} \qquad l = 2, ..., L. \tag{3.2}$$

Given this notation, the activation of hidden neuron j in layer l is:

$$y_j^{(l)} = \max\left(0, net_j^{(l)}\right) \qquad l = 2, ..., L-1. \tag{3.3}$$

[1]In one-hot vectors all the components are 0 except the one corresponding to the target class, which takes the value of 1.

where we applied the reLU non-linearity $\max(0, x)$. reLUs are more plausible activation functions for modelling the firing of a neuron, being unbounded for $x \to \infty$ and also having true zero activation when $x \leq 0$. They naturally lead to sparse representations in the hidden layers of the network, which again is more in-line with what is observed in the brain, where only 1% to 4% of neurons fire at the same time [48]. Sparsity is also an advantage from a computational and mathematical perspective, as explained in [45]. In particular computation is linear in the subset of active neurons, since the gradient is always 1 for active neurons there is no vanishing gradient problem during backpropagation (while the gradient is 0 for saturating neurons using traditional activation functions like sigmoid or tanh). We chose reLUs also for their faster training time and ability to find good local minima even without pre-training.

The softmax output function for output unit j is defined as:

$$z_j = \frac{e^{net_j}}{\sum_{k=1}^{N_S} e^{net_k}} \tag{3.4}$$

Where we omitted the superscript L, which indicate that net_j refers to the last layer L.

This output function is a generalization of the sigmoid function to many outputs and guarantees that the outputs are bounded in $[0, 1]$ and sum up to 1, as is supposed to be for the posterior probabilities of all target speakers.

With these network architecture we have a powerful non-linear predictor that outputs the posterior probability $p(s_k|\boldsymbol{x_j})$ for each speaker s_k, given the input feature vector $\boldsymbol{x_j}$. In order to obtain a single score for utterance u_i we have to aggregate the scores coming from all its frames X_i. Assuming independent frames over time we can compute the following

$$\log p(s_k|X_i) = \sum_{j=1}^{N_F(u_i)} \log p(s_k|\boldsymbol{x_j}) \tag{3.5}$$

The utterances have different length, thus to achieve comparable scores between different utterances we decided to normalize the scores dividing by the number of frames $N_F(u_i)$ in the utterance. This leads to a score which is the average log-likelihood of the speaker over the utterance frames.

The classification decision for speaker identification is then taken with the following rule:

$$\hat{s} = \arg\max_k \frac{\log p(s_k|X_i)}{N_F(u_i)} \qquad k = 1, ..., N_S \tag{3.6}$$

And since the score does not depend on the length of the utterance, we can use the trained network also for speaker verification, by setting a speaker-independent threshold θ and using it to reject or accept an utterance looking at only one output unit k of the network (one speaker).

$$\frac{\log p\left(s_k|X_i\right)}{N_F(u_i)} = \begin{cases} \geq \theta, & \text{accept } H_0 \\ < \theta, & \text{reject } H_0 \end{cases} \tag{3.7}$$

Where H_0 is the hypothesis that u_i is spoken by s_k.

3.3 Learning with backpropagation and related tricks

In order to train the ANN we have to update all its parameters, which is updating all the matrices of weights $W^{(l)}$ and biases $b_j^{(l)}$ for $l = 2, ..., L$.

To do this we use an optimization approach. Given a training set $\left\{(\boldsymbol{x}^{(i)}, \boldsymbol{o}^{(i)}) : i = 1 : N_F\right\}$ composed of N_F feature vectors with the corresponding speaker target, we define the negative log-likelihood cost function over the training set as:

$$L\left(\boldsymbol{W}\right) = -\frac{1}{N_F}\sum_{i=1}^{N_F} \boldsymbol{o}^{(i)} \log \boldsymbol{z}^{(i)} \tag{3.8}$$

Where we highlighted the dependency of the cost from the whole set of weights $\boldsymbol{W}$, which are the free parameters of the network we can update during training. We use the notation $\boldsymbol{W}$ to refer to all the free parameters, both the weights $w_{j,i}$ and the biases b_j.

For every training frame $\boldsymbol{x}^{(i)}$ the cost is given by the estimated negative log-likelihood of the target speaker. In the ideal case the output is 1 for the target speaker leading to a cost of 0, but the cost increases exponentially as the estimated posterior probability gets smaller than 1.

Our goal is to minimize this cost function by updating $\boldsymbol{W}$. Direct minimization cannot be performed, but through the iterative optimization process of gradient descent we can get closer and closer to a local minima. The gradient descent equation is:

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta \frac{\partial L}{\partial \boldsymbol{W}_{t-1}} \tag{3.9}$$

where η is the learning rate, an hyperparameter that must be set a priori and controls the speed of the updates of $\boldsymbol{W}$. If η is set too high the update can become too big and the loss can start to oscillate inside a local minima or even diverge. On the other hand, if η is too small, the descent in the space spanned by $\boldsymbol{W}$ will take a very long time, with $\boldsymbol{W}$ changing just a little at every update and moving very slowly towards a local minima.

3.3.1 Stochastic gradient descent

The gradient for a particular training vector is computed using the backpropagation algorithm [41], where the training vector is forward-propagated through the ANN and then the error is back-propagated from the output towards the input layer, in order to compute the gradient of L with respect to all the free parameters $w_{j,i}$ and b_j. For a good tutorial and mathematical explanation of the backpropagation algorithm we refer to chapter 2 of [49].

If we want to optimize the learning over the whole training set, we need to combine the gradients from all training examples. This leads to an estimate of the gradient $\frac{\partial L}{\partial \boldsymbol{W}}$ after a full sweep through the whole training data. This method is called full batch gradient descent and ensures that the cost is monotonically decreasing with every update. The opposite strategy is to update $\boldsymbol{W}$ as often as possible, using a single training example to estimate $\frac{\partial L}{\partial \boldsymbol{W}}$. This approach is called online learning, and leads to much more zig-zag on the loss surface due to many updates with poorly estimated gradients that optimize only for a single training example.

The best strategy, which combines frequent updates with better gradient estimates, is called stochastic gradient descent, in which we update $\boldsymbol{W}$ after estimating the gradient on a mini-batch of N_B randomly selected training examples. Each training example is used only once until all of them have been seen and one epoch of learning has been completed, after which the process is repeated. We can summarize the three learning paradigms as follows:

- *Full-Batch*: only one update per epoch. Monotonically decreasing cost, but slow learning.
- *Online*: N_F updates per epoch. Cost oscillates a lot as a function of time, but updates are very frequent.
- *Stochastic/Mini-Bacth*: N_F/N_B updates per epoch. Good trade-off between learning speed and accuracy in decreasing the cost.

Stochastic gradient descent is by far the most used training method for deep learning. Especially when the training set is large and contains redundant training examples, it is hugely convenient to use mini-batches to speed up the learning of the ANN.

3.3.2 Random weights initialization

The outcome of gradient descent depends on the initialization of $\boldsymbol{W}$. If all the weights were initialized equally the optimization would fail, because all the neurons would be learning the same feature. The biases b_j can all be set to zero, but it is necessary to have a random initialization of the weights $w_{j,i}$ to break the symmetry of the ANN.

It is important to stress that not all random initializations are equally good: the gradients and the attraction towards a good local minima depend strongly on the initial position of $\boldsymbol{W}$. Gradients can therefore become unstable if $\boldsymbol{W}$ is not properly initialized. In particular we would like the activations to have the same variance in each layer as a training vector is forward-propagated and the back-propagated gradients should also have the same variance in each layer, so that each layer can learn at the same speed. With a purely random initialization the gradient can start exploding or vanishing from one layer to the other, making the training difficult. In particular some activations can saturate when the activation is very high and the derivative becomes zero. Saturated neurons have pretty much stopped learning, so we need to initialize $\boldsymbol{W}$ such that training vectors and gradients flow well through the network, without saturated neurons from the beginning.

Glorot has shown in [50] that poor weight initialization was one of the main reasons why DNNs failed to be trained successfully, with the vanishing gradient due to saturated neurons as one of the main reasons. He introduced a successful weight initialization that solves the problem for tanh hidden neurons by ensuring that the magnitude and variance of activations and gradients is conserved by layer-to-layer propagation. The formula is commonly known as the Xavier initialization method:

$$W^{(l)} \sim U\left[-\frac{\sqrt{6}}{\sqrt{D_l + D_{l-1}}}, \frac{\sqrt{6}}{\sqrt{D_l + D_{l-1}}}\right] \tag{3.10}$$

Networks with reLU activation have been shown to be easier to train. reLUs are not prone to gradient vanishing: their linear part does never saturate, so that a neuron is always learning in its positive part or completely shut off and not learning in its negative part. In [51] He introduced a new initialization designed especially for reLU neurons, with the goal of better and faster convergence that does not stagnate even with very deep networks:

$$W^{(l)} \sim \mathcal{N}\left[0\,,\,\sqrt{\frac{2}{D_{l-1}}}\right] \tag{3.11}$$

We used He's initialization method for our DNN with reLU activations. For the biases we decided to set them to the small constant value of 0.1, to push more neurons to be in the active linear state at the beginning of training.

3.3.3 Nesterov's Accelerated Gradient

Many improvements over simple gradient descent have been introduced to speed up and improve the training of ANNs. One very successful technique is that of momentum, a way of smoothing the movement of $\boldsymbol{W}$ on the cost surface [52]. The core idea is to accumulate the updates in a velocity vector which points in the direction of persistent reduction of the cost. The momentum update tends to stagnate less in the valleys of the cost function and also gets rid of oscillations when going down a ravine with high curvature. This leads to faster convergence to a local minima. The equations for standard momentum update are:

$$\boldsymbol{v}_t = \mu \boldsymbol{v}_{t-1} - \eta \nabla L\left(\boldsymbol{W}_{t-1}\right) \tag{3.12}$$

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} + \boldsymbol{v}_t \tag{3.13}$$

where $\mu \in [0, 1]$ is an hyperparameter called the momentum constant, which controls how much contribution the past updates have on the velocity vector $\boldsymbol{v}$. Typical values for μ are 0.9 or 0.95, while setting $\mu = 0$ leads to the standard steepest descent update.

Nesterov's Accelerated Gradient (NAG) is a first-order optimization method proven to achieve a better convergence rate with gradient descent for convex functions [53]. Recently Ilya Sutskever derived a new formulation of the NAG equations [54], showing that it implements an update rule which is very similar to standard momentum, but with a key difference that makes it perform better in many occasions. The NAG update equations are:

$$\boldsymbol{v}_t = \mu \boldsymbol{v}_{t-1} - \eta \nabla L \left(\boldsymbol{W}_{t-1} + \mu_{t-1} \boldsymbol{v}_{t-1} \right) \tag{3.14}$$

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} + \boldsymbol{v}_t \tag{3.15}$$

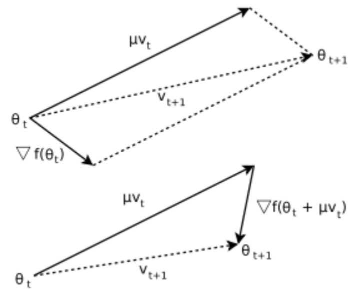
The only difference is the position at which the gradient is evaluated at each time step. This is best explained graphically: in fig 3.2a we can see that, at a particular time step t, standard momentum computes the gradient at the starting position, accumulates it in $\boldsymbol{v}$ and makes a big jump in the resulting direction. NAG makes the jump first, computes the gradient at the resulting position $\boldsymbol{W}_{t-1} + \mu_{t-1} \boldsymbol{v}_{t-1}$ and uses it to make a correction. In the words of Geoffry Hinton, "it is better to gamble and then make a correction than to make a correction first and then gamble".

Figure 3.2b shows the minimization of a two-dimensional oblong quadratic, where it is evident that NAG changes $\boldsymbol{v}$ in a quicker and more responsive way, achieving higher stability over many iterations, while standard momentum oscillates strongly along the high-curvature vertical direction.

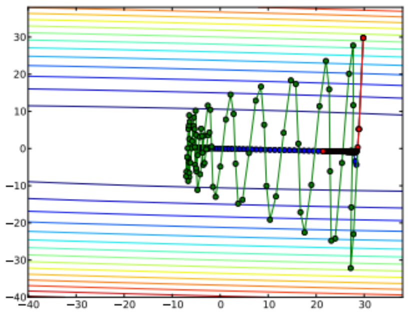
For our implementation of NAG we used the formulation proposed by Yoshua Bengio in [55], which focuses on the "peeked-ahead" parameters $\Theta_{t-1} \equiv \boldsymbol{W}_{t-1} + \mu_{t-1} \boldsymbol{v}_{t-1}$.

$$\begin{aligned} \boldsymbol{v}_t &= \mu_{t-1} \boldsymbol{v}_{t-1} - \eta_{t-1} \nabla L \left(\Theta_{t-1} \right) & (3.16) \\ \Theta_t &= \Theta_{t-1} - \mu_{t-1} \boldsymbol{v}_{t-1} + \mu_t \boldsymbol{v}_t + \boldsymbol{v}_t & (3.17) \\ &= \Theta_{t-1} + \mu_t \mu_{t-1} \boldsymbol{v}_{t-1} - \left(1 + \mu_t\right) \eta_{t-1} \nabla L \left(\Theta_{t-1} \right) & (3.18) \end{aligned}$$

Assuming zero velocities at the beginning and at convergence of optimization, the parameters Θ are equivalent to $\boldsymbol{W}$. The learning rate and the momentum hyperparameters can be functions of time, but in practice we use a constant μ and η. In [55] it is noted that NAG can take advantage of higher values of μ, storing past velocities for a longer time while actually using those velocities more conservatively during the updates.



(a) **(top)** Momentum **(bottom)** Nesterov's accelerated gradient



(b) Nesterov and Momentum on a quadratic

Figure 3.2: a) One step of Nesterov's Accelerated Gradient and of standard momentum. b) The trajectories of GD (red), momentum (green), and Nesterov's accelerated gradient (blue). Both methods had μ set to 0.95. The global minimum of the quadratic is in the center of the figure, at $(0, 0)$. Image taken from [54].

3.3.4 RMS-prop for adaptive learning rates

Another optimization improvement is to change the value of the learning rate over time. A good intuition is that if the learning rate is too big the system has too much kinetic energy and our parameter vector will start to bounce around the cost surface chaotically. This is particularly true for the later stages of learning. In the beginning it might be advantageous to have a big learning rate and get a quick descent, but after a few epochs we might reach a local minima. Thus it is better to anneal the learning rate to try to descend deeper in the local minima without bouncing back up.

Another thing that must be stressed is that gradients vary widely in their magnitudes. Some gradients can be much higher than others, so it is difficult to set a global learning rate. This leads to the idea of per-parameter learning rates, where each weight has a different learning rate which changes over time depending on the values of the partial derivatives of L over time.

We will present here an unpublished method called RMS-prop, which was introduced by Geoffrey Hinton and Tijmen Tieleman and for which we reference the Coursera lecture slides available from the university of Toronto [56]. Its core idea is to keep a moving average of the squared gradient for each weight separately and then use it to normalize the global learning rate η separately for each weight.

$$S(w, t) = \alpha S(w, t-1) + (1-\alpha)\left(\frac{\partial L}{\partial w}(t)\right)^2 \tag{3.19}$$

where α is the integrating constant for the moving average of the squared gradients. Typical values for α are in the range $[0.9, 0.99]$.

The learning rate of parameter w as a function of time is then obtained as:

$$\varepsilon(w,t) = \frac{\eta}{\sqrt{S(w,t)}} \tag{3.20}$$

and the gradient descent update for parameter w becomes:

$$w_t = w_{t-1} - \varepsilon(w,t-1)\frac{\partial L}{\partial w_{t-1}} \tag{3.21}$$

The method is called RMS-prop because the normalizing factor for the gradients is the root mean square of each partial derivative.

The effectiveness of these optimization methods for minimizing the cost function was confirmed in our experiments. Fig 3.3 shows the results of training a DNN with three hidden layers of size 500 for a speaker identification task with 80 speakers. Comparison is made between training with standard Stochastic Gradient Descent (SGD), NAG and RMS-prop. The convergence over the 20 epochs of training is fastest when combining RMS-prop with NAG, but even alone the two methods achieve a great training speed up compared to standard SGD.

Also the use RMS-prop makes it easier to set the initial learning rate η. Since it is adapted automatically for every w during training, the outcome depends less on its initial value, while for simple gradient descent it must be fine-tuned to achieve a reasonable learning speed.

3.3.5 Regularization

The aim of regularization for training a machine learning model is to prevent it from overfitting the training data, which would lead to poor generalization and bad classification accuracy on real-world examples. DNNs are very powerful models with lots of parameters, and they can thus overfit easily, especially if the training set is small. Alternatively we can say regularization adresses the bias/variance dilemma: it tries to find an optimal trade-off between the high bias of a very constrained model and the high variance of a model with too much freedom [57].

The classic approach to regularize a model is to introduce a penalty in the cost function, which forces the weights to have smaller weights and thus avoid spikes and strong non-linearities which could lead to an overfitted model. This class of methods is called weight decay and is a special case of Tikhonov regularizaion [58]. Two of these methods which are widely used in the training of ANNs are L2 and L1 regularization.

Another approach is that of early-stopping [59]. The idea is to stop the training when the network starts to overfit, by monitoring the loss or the classification error on both the training data and an independent validation set. There are many heuristics to decide when exactly to stop and a good review of them can be found in [60]. The basic idea is that when the performance on the validation set stops increasing and starts to drop the model is overfitting. A simple graphical representation of this is shown in fig 3.4. After a large enough number of epochs any performance gain over the training set does not generalize

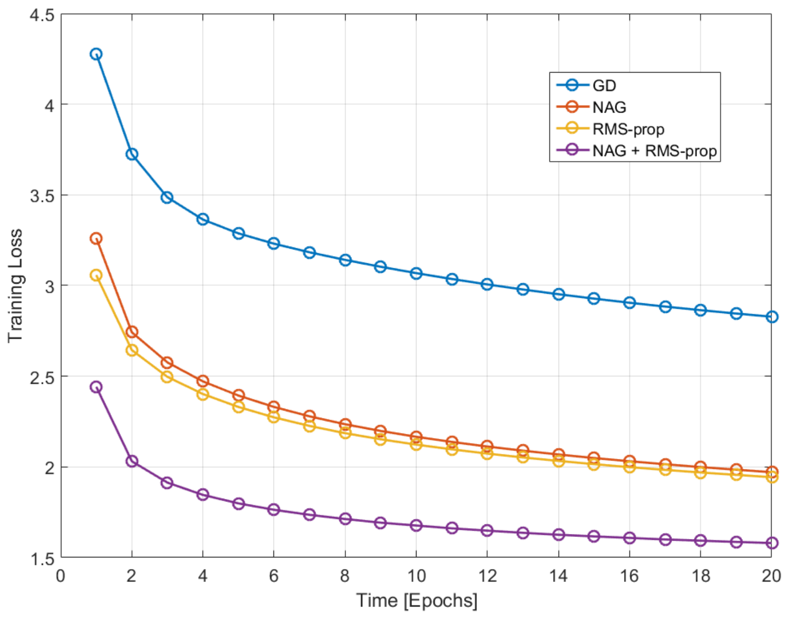


Figure 3.3: Training loss as a function of epochs of training for standard SGD (blue), NAG (red), RMS-prop (yellow) and NAG+RMS-prop (purple). The global learning rate η was 0.003 for the first two methods and 0.0001 for the curves using RMS-prop. The hyperparameters were $\mu = 0.95$ and $\alpha = 0.99$.

to real test data, but just overfits the training set. The goal is to stop training at the optimal point where the validation error (or loss) is minimal.

Finally, a new powerful regularization method for DNNs, called dropout, has been introduced recently by Hinton [46, 61]. Its core idea, inspired by how genes are transmitted in sexual reproduction, is to shut down randomly a subset of hidden units for each training example. The most common implementation shuts down each unit with probability 0.5. This approach forces each unit to learn more useful features by decoupling it from the other hidden units and thus prevent complex co-adaptations of feature detectors. Dropout has been shown to be a very powerful and effective regularizer, having also properties that link it to the technique of bagging, where a large number of models are trained on a subset and then averaged together. We can think of dropout as sampling a different network architecture for every training example it is presented with, and then averaging them all together by forcing them to share the same weights. In this sense each model is strongly regularized by all the other models. At test time all hidden units are used, but their outgoing weights are halved. Hinton proved that, for a softmax network with a single hidden layer, this computes the geometric mean of the predictions of all possible sub-models. The number of sub-models grows exponentially with the number of hidden units, so dropout is a really extreme form of bagging: averaging an exponential number of models trained on a single example.

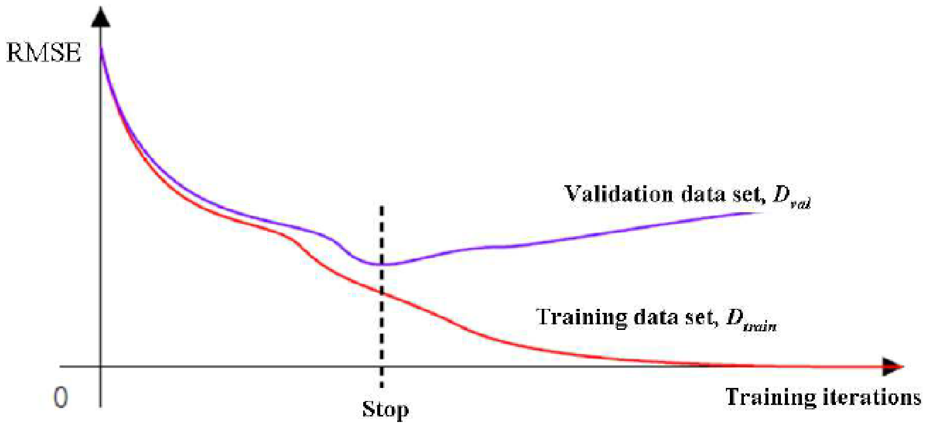


Figure 3.4: Early stopping of training to prevent overfitting of the model

A DNN which is overfitting will generalize better to the test set when trained with dropout. Hinton suggests to use a large enough model that can overfit the training set easily and then train it with dropout, which can be rephrased also as: "there should be many more synapses than experiences, therefore many more parameters than training examples". This seems to work really well when we have enough time to train the model with dropout and make it converge, much better than using early-stopping. The problem is this kind of training process usually takes a very long time.

In our case, given the big training set and our need to train the models in a reasonable time on a CPU to perform experiments, we decided to use classic weight penalty or early stopping for regularization.

For L2 regularization, the cost function in equation 3.8 is modified as:

$$L\left(\boldsymbol{W}\right) = -\frac{1}{N_F}\sum_{i=1}^{N_F} \boldsymbol{o}^{(i)} \log \boldsymbol{z}^{(i)} + \lambda \sum_{w} w^2 \tag{3.22}$$

While for L1 regularization we have:

$$L\left(\boldsymbol{W}\right) = -\frac{1}{N_F}\sum_{i=1}^{N_F} \boldsymbol{o}^{(i)} \log \boldsymbol{z}^{(i)} + \lambda \sum_{w} |w| \tag{3.23}$$

L2 and L1 refer to the fact that we use the L2 norm or the L1 norm of $\boldsymbol{W}$ to penalize models with large weights. The hyperparameter λ must be tuned to achieve the desired bias/variance tradeoff, where $\lambda = 0$ is equivalent to no regularization and a large λ would lead to a very constrained model. It is common to look for the best λ by searching in the set of values $[0.01, 0.03, 0.001, 0.003, 0.0001, ...]$.

A good explanation of the difference between the two methods is given by Nielsen in [49]. "When a particular weight has a large magnitude $|w|$, L1 regularization shrinks the weight much less than L2 regularization does. By contrast, when $|w|$ is small, L1 regularization shrinks the weight much more than L2 regularization. The net result is that L1 regularization tends to concentrate the weight of the network in a relatively small number of high-importance connections, while the other weights are driven toward zero". L1 thus is said to enforce more sparsity in the model, which can be advantageous for many tasks, for example compressing singals. In [45] Glorot uses L1 penalty to enforce sparse representations in DNNs with rectifier non linearities. His experiments suggest that sparsity is a very desirable property which improves classification generalization up to a certain percentage, as shown in fig 3.5.

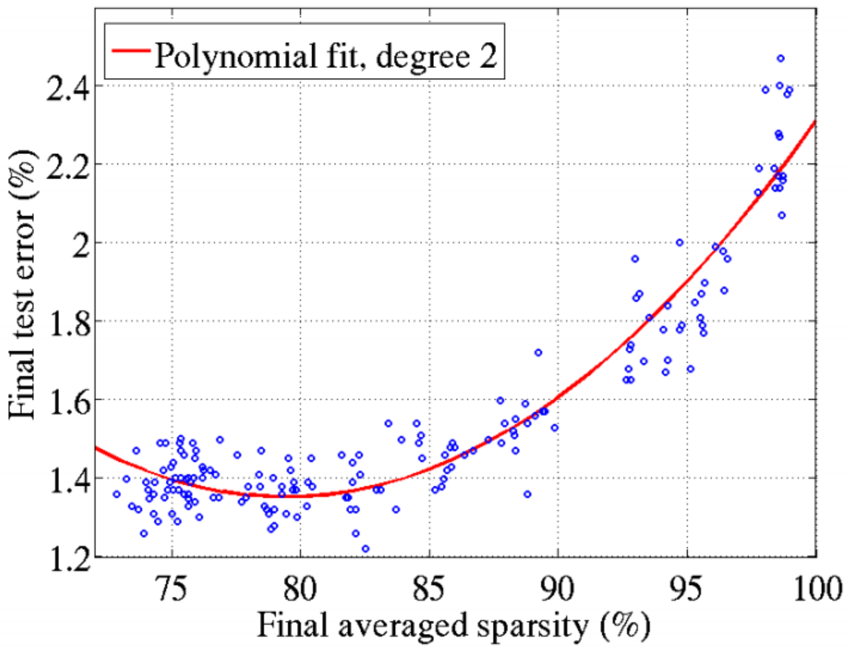


Figure 3.5: Influence of final sparsity on classification accuracy. 200 randomly initialized deep rectifier networks were trained on MNIST with various L1 penalties (from 0 to 0.01) to obtain different sparsity levels. Results show that enforcing sparsity of the activation does not hurt final performance until around 85% of true zeros. Taken from [45]

In practice L2 and L1 perform very similarly for the purpose of regularizing ANNs for most classification problems. In our experiment we tested both methods and found that we achieve equivalent performance by tuning the coefficient λ properly for each method.

3.3.6 Learning with imbalanced classes

A problem we faced in training a DNN on the speakers of the Librispeech corpus is that of imbalanced classes. The amount of training data varies a lot between different speakers: each speaker has a different number of utterances and each utterance has a different number of training frames. Variance over the training set was quite big, with some speakers with as few as 15 training utterances and others with more than 150. The length of utterances varied between roughly 3 to 30 seconds. This meant that after extracting the dataset we had some classes that were under-represented in the training set, while some other speakers had almost 4 times more training data than most other speakers.

This is a typical problem for both binary and multiclass neural networks. If during each epoch of training the network sees a different number of examples for each class it can tend to optimize its parameters in the direction of the dominant classes, while neglecting the smaller classes. This can be diagnosed easily by looking at the precision and recall performance metric: big classes tend to be favoured in classification, achieving high recall but low precision, while small classes are frequently confused for the bigger ones, having thus very low recall.

Much research has been conducted for this type of problem, often referred to as imbalanced and cost-sensitive learning. We refer to [62] and [63] for a review of the topic. Most papers focus on the two class problem where one class has a much higher importance than the other. The typical example is that of cancer detection, where the two classes are very imbalanced in the training set and the cost of misclassifying a positive for a negative is much higher than misclassifying a negative for a positive.

Our case is much different and it makes it difficult to follow the cost-sensitive learning approach. Ideally we would like to give the same importance to each speaker and we don't have a way of setting a priori the cost matrix for misclassifying one speaker for the other. One idea would be to penalize misclassification of male for females and the other way around. Also other criteria could be used, but it would lead to a very ad-hoc technique.

In practice we tried to solve the problem in the easiest way possible. The simplest way to deal with imbalanced classes is to randomly sample the training set. Undersampling is used to shrink the bigger classes and upsampling to give more weight to the smaller classes, thus achieving the same amount of training frames for each speaker. This approach is straightforward to implement and it is worth trying as a first solution to the problem.

As explained in [64], oversampling usually increases the training time and can lead to overfitting the small classes, due to the use of exact copies of the training examples. A possible solution is to upsample in a smarter way, like in the SMOTE algorithm [65], where synthetic examples are created on the lines between the minority class nearest neighbours. On large datasets it is usually preferrable to use undersampling, especially when examples are not removed randomly, but by using a smarter criteria based on the concept of Tomek links. As explained in [66], these techniques try to get rid of borderline and noisy examples, while keeping all examples that better represent the majority class.

In practice we implemented the simplest approach of random undersampling most classes and upsampling the subset of minority classes in order to achieve the same amount of training data for all speakers. This lead to satisfactory results so we did not experiment with more advanced techniques.

We also tried a simplified version of cost-sensitive learning, similar to the one proposed in [67]. The idea is to weigh the per-example cost differently according to the amount of training data available for the example class. The cost function in eq 3.8 is modified as follows:

$$L(\boldsymbol{W}) = -\frac{1}{N_F} \sum_{i=1}^{N_F} cost(C_k^{(i)}) \boldsymbol{o^{(i)}} \log \boldsymbol{z^{(i)}} \tag{3.24}$$

Where $C_k^{(i)}$ is the class of training example i. For each class, we define a cost using the per class amounts of training data N_j.

$$cost(C_k) = \frac{\max_j N_j}{N_k} \quad k = 1, ..., N_S \tag{3.25}$$

In this way the biggest class has a cost of 1, while a class with half its training examples will have a cost of 2, and so on. This linear rescaling of the cost function will give more importance to the examples of the minority classes, therefore moving the classification boundaries in the direction of the majority classes, which are weighted less.

Comparing the two approaches we found they both improve performance for unbalanced training of ANNs, without any of the two being clearly superior in classification accuracy. One nice aspect of the cost sensitive learning is the absence of any resampling, leading to a more automated training on complicated unbalanced datasets. In practice though, we decided to use random reseampling to speed up the training, which is advantageous for faster experimentation on a dataset with lots of classes and training data like Librispeech.

To recap, in this chapter we presented the architecture and techniques we implemented to train a DNN in a purely supervised fashion on the Librispeech database. We explained the tricks of NAG and RMS-prop we used to improve convergence and training time as well as regularization techniques to avoid overfitting and resampling and cost-sensitive learning to address the class imbalance problem.

Using a properly tuned combination of this techniques we were able to significatively improve the training time and accuracy of the DNN, achieving better speaker recognition results than our baseline GMM and i-vector systems. This higher accuracy comes at the cost of scalability, computational complexity and memory allocation: a multi-class neural network is very costly and complicated to train compared to a GMM. It has many more parameters and often requires careful tuning of hyperparameters to perform at its best. The main drawback we identified is that of scalability, and we will address it by introducing a new hybrid approach in chapter 4.

Chapter 4

Scalable Speaker Verification with ANNs and GMM-UBM

4.1 Motivation

Despite the superior performance of our discriminatively trained DNN, we were not satisfied with the lack of scalability of the new system compared to the baseline. In the GMM system we have a separate model for each speaker and the process of enrolling a new speaker is straightforward and independent of the speakers that have already been enrolled. We can also train and score the speakers in parallel on different machines if required. The same holds for the i-vector method, were we can use the same i-vector extractor to enroll speakers in parallel. The big advantage is that once the per-speaker models are trained, we can get rid of their training data if we want and still we will be able to extend the system with new speakers in the future.

On the contrary, the DNN is a single big architecture that requires to see all the training data of all speakers at once to be trained discriminatively. If we want to add a single new speaker in the future, we would have to add a new output neuron to the softmax layer and retrain the network on the whole training set. We have to keep all the training data from all speakers and we cannot take advantage of past training effectively. This problem has not received much attention from the machine learning community., where usually a classification problem is very well defined, with a fixed amount of classes that does never change. For such problems a multiclass ANN is very well suited, but for many real-world problems the number of classes needs to change over time. Think for example if we were building a speaker recognition system for the employees of a big company. The number of employees is constantly changing, with new ones being hired and others being fired or leaving the company. The DNN would be a very inconvenient system in this scenario. Also the DNN is a very big model suited for computing the posteriors over the whole set of target speakers, but in the speaker verification case we are interested in only one target speaker. If we want to perform verification with the DNN we would still perform forward propagation over the whole network and then look at a single output neuron, which is more costly than scoring a single smaller model like a GMM.

4.2 The ANN-UBM method for speaker verification

The above observations suggest that for practical speaker verification having a separate model for each speaker is advantageous. Motivated by this, we came up with a way for using discriminatively trained DNNs, with their advantages in terms of classification performance and representational power, but maintaining the advantage of having one model per speaker for superior scalability.

The idea is very similar to the One-Against-All strategy to decompose a multiclass problem into a set of smaller binary problems. This is still the most popular strategy for many multiclass classifiers, in particular multiclass SVM [68]. The idea is to train as many binary classifiers as classes, with each classifier learning to discriminate one single class against all the other classes pooled together. At test time all classifiers are scored and the class with the highest output is chosen, in a winner-takes-all fashion.

This simple idea seems very suitable for our goal of having one DNN for each speaker, but it doesn't solve the scalability problem because all the training data from all classes needs to be available and pooled together to train each DNN in a discriminative way.

We suggest using the UBM to solve the problem of the adversarial data. By definition the UBM models the distribution of all possible classes and in our case we already use it to model the rejection hypothesis in the GMM system. Our idea is to generate the adversarial training examples using the GMM-UBM, so that each network will learn to discriminate between the target speaker data and the impostor speaker data generated with the UBM. The training is outlined in figure 4.1, where for each speaker k we train a separate ANN with a single sigmoid unit as the output, which estimates the posterior $p(s_k|x_i)$ for each training frame x_i. The target training frames are the MFCC frames extracted from the utterances of target speaker k and have target label $o = 1$. The impostor training data is generated on-the-fly with the GMM-UBM and have target label $o = 0$.

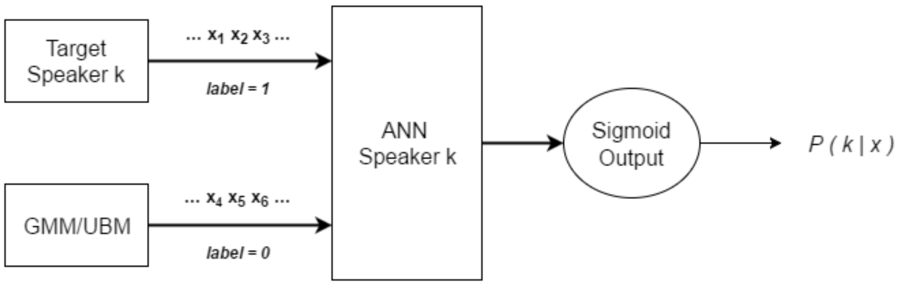


Figure 4.1: The proposed method for modular ANN training using a GMM-UBM. Each speaker s_k is modelled by one ANN with a sigmoid output neuron to model the posterior probability $p(s_k|x_i)$

By using this hybrid generative-discriminative approach we can maintain full scalability and still use discriminatively trained DNNs, which have more representational power and can learn complex invariances that could lead to robustness to noise and other characteristics which are difficult to obtain with a GMM. The system is also more suited for speaker verification, since we can score each speaker independetly of each other. We will refer to this new method as ANN-UBM to distinguish it from the DNN presented in chapter 3.

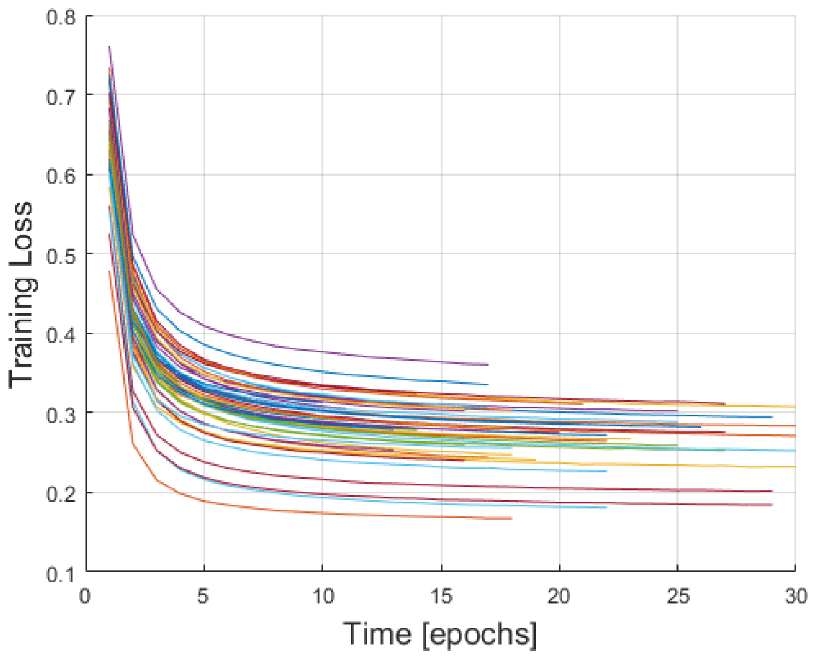
The scoring is straightforward. Since each network outputs postrior probabilites, we can get the log-likelihood over all the frames X_i of utterance u_i via eq (3.5). The classification is then performed using eq (3.6) for identification and eq (3.7) for verification. It is worth noting that the posterior probabilites are over only two classes, since each network is a binary model which discriminates between a target and the generated impostor data. We can qualitatively expect the threshold θ to be around $\log(0.5)$ for speaker verification with this architecture.

The idea presented here is very general and could work with other discriminative classifiers, like SVMs. We decided to use ANNs to have a more direct comparison with the DNN presented in chapter 3. Training a large number of ANNs is challenging because it is not easy to optimize hyperparameters separately for each network. By using RMS-prop with $\alpha = 0.99$ we got rid of the problem of the learning rate, since it adapts automatically during training. We also used NAG with $mu = 0.95$ to speed up convergence. We added L1 regularization with the same parameter λ for all ANNs, optimized with the overall test accuracy as objective. Finally we chose to use early stopping by tracking the loss over validation data taken as 10% of the training set before learning. In this way we can set a common maximum number of epochs for all ANNs. If the validation loss stagnates or increases for two consecutive epochs we stop the training. This is a very simple rule which could be improved as described in [60], but we obtained satisfactory results so we decided to keep our simple early-stop decision.

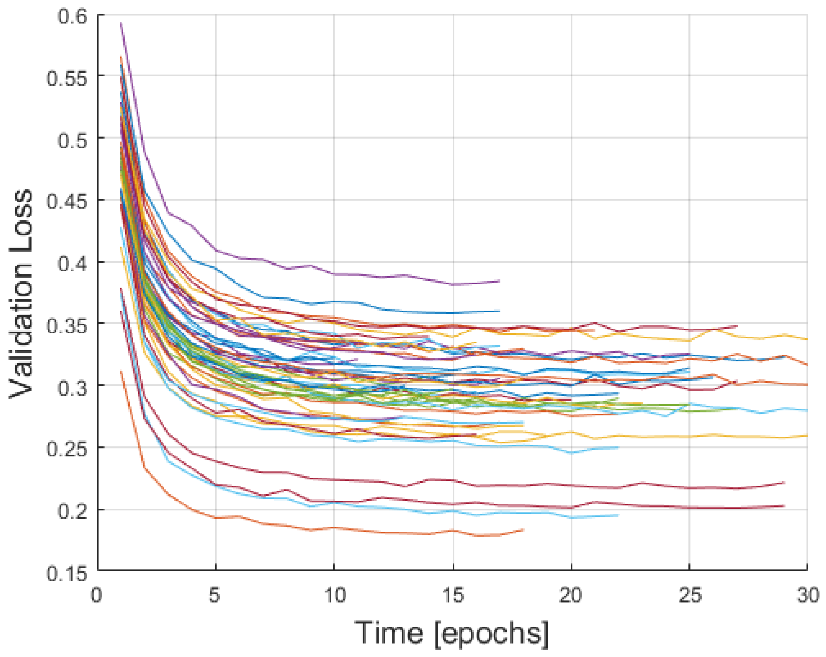
Moreover we performed optimization on the amount of adversarial data to generate with the GMM-UBM. The class modelled by the UBM should in theory be larger than the target speaker, since it models all the possible adversarial speakers. We therefore varied the ratio between the amount of adversarial and target data. We found that the best performance on the test set was achieved by using twice as much data generated with the UBM.

In this way we managed to train all networks without having to worry about hyperparameters and avoiding overfitting. Figures 4.2a and 4.2b show an example of training 50 ANNs with early stopping and with the maximum number of epochs set at 30. Some networks stopped earlier than the others and overall it is clear that none of the networks overfitted. If one network overfitted we would see the characteristic U-shaped curve in the validation loss, while here the training loss seems to have almost converged for most networks and the validation loss also oscillates around a stable final value.

It is also interesting to note that some networks reach a smaller loss value than others, with as much as 100% difference. This could indicate that some voices are much more similar to the average human voice modelled by the GMM-UBM, and thus are difficult to discriminate perfectly. The curves reaching a very low cost probably come from speakers with very peculiar voices which can be easily discriminated from the impostor data.



(a)



(b)

Figure 4.2: Training (a) and validaton (b) loss as function of time for the novel ANN-UBM method

Chapter 5

Experiments and Results

All the experiments were carried out at Ericsson research using MATLAB as the programming environment. We used our own MATLAB implementations of all the algorithms described in the thesis report.

The dataset we chose for our experiments is the Librispeech ASR corpus [6]. It consists of audiobooks read-out-loud by 2483 speakers, 1281 male and 1202 female volunteers who recorded their voices spontaneously. The speech signal is usually clean, but the recoding device and channel conditions vary a lot between different utterances and speakers. This makes it an interesting dataset to test algorithms for practical applications. Every speaker has a different amount of recorded utterances and each utterance has a variable length between 3 and 30 seconds, with the average length of roughly 15 seconds. The Librispeech database is open access and can be freely downloaded for further experiments or comparisons.

We decided to keep 70% of the utterances of each speaker for training, and the reamaining 30% as a fixed test set for evaluation.

To train the UBM we pooled together the MFCC training vectors from 330 speakers, 166 female and 164 male. This amounted to approximately 20 hours of clean speech. We trained a GMM with 1024 diagonal components on the pooled data.

For the speaker dependent GMMs we used 32 components with diagonal covariance matrix, in order to have enough training data per parameter, while for the i-vector system we set the dimension of the i-vectors to be 200.

The hyperparameters and settings for the training of the DNN are shown in table 5.1. We trained for only 5 epochs because of time constraints. Since the network has a huge number of parameters it takes a long time to perform SGD. Thanks to the optimization methods described in chapter 3 the network achieved a good local minima even after only 5 epochs. We found that the best test performance was achieved without the need to add any regularization, which means the DNN did not overfit and generalizes well from training to test data.

For the ANN-UBM method, the parameters and training settings are visible in table 5.2.

DNN parameter	**Value**
Network architecture	[24in 2000 2000 2483out]
Learning rate η	0.0001
NAG μ	0.95
RMS-prop α	0.99
Number of epochs	5
Batch size	15000
Training samples per speaker	10000

Table 5.1: Training parameters for the Deep Neural Network

NN-UBM parameter	**Value**
Network architecture	[24in 400 400 1out]
Learning rate η	0.0001
NAG μ	0.95
RMS-prop α	0.99
L1 regularization λ	0.0001
Ratio UBM/target	2
Number of epochs	30
Batch size	500

Table 5.2: Training parameters for the ANN-UBM method

These choices of architecture and parameters were made to have a reasonable comparison between the different methods. The GMM and i-vector method have roughly 80 training examples per parameter, while the two methods based on ANNs have around 5 training examples per parameter. In practice ANNs can have a very large number of parameters and still be trained effectively with the techniques we described, while for our generative models we preferred to have a larger number of examples per parameter.

We used two scoring scenarios to evaluate the methods:

- Clean test utterances, the ideal and simplest case where there is no mismatch between training and testing conditions.
- Noisy test utterances, where we added cafeteria noise at 15 dB of Signal-to-Noise Ratio (SNR) to test the robustness to additive noise.

The results of each system on both scenarios are shown in table 5.3. It can be seen that the discriminatively trained DNN achieves the best verification performance in the clean test conditions, with only 0.22% EER, which is a 100% reduction in error compared to the GMM-UBM system. This is a statistically significant result, given that more than 200 million verification decisions were taken to compute the score. The DNN is the best also for the speaker identification on the clean test utterances, with 0.72% classification error rate.

The i-vector system performs worse than other methods in both verification and identification of the clean test utterances. This is probably because we used the simplest scoring technique of cosine distance, while a better performance could be reached with the use of LDA and WCCN before computing the cosine distance.

When cafeteria noise is added to the test utterances the performance drops significatively for all methods. It is interesting to note that the i-vector system seems to be the most robust to noise in this case, since it performs better than the others. A possible explanation is the fact that the other methods were too strongly tuned to the clean speech scenario and thus generalized poorly to a mismatched noisy scenario. The i-vector method instead has a more compact representation of each utterance which is less over-fitted to the clean scenario, and thus was affected less by the addition of noise to the test utterances.

We were not satisfied with the performance of the systems in the presence of additive noise, so we decided to use a form of data augmentation before training to improve the robustness of our speaker recognition systems. The idea of data augmentation for machine learning is to expand the training set by adding perturbations to the training examples. This has been shown to be very efficient in many machine learning tasks, including speech recognition and object recognition. In [69] it is shown that training with noise is equivalent to a generalized form of Tikhonov regularization. The idea of noise injection is also linked to the more recent approach of denoising auto-encoders [70], where a set of robust latent variables is learned in an unsupervised manner by training a network to reconstruct its input distorted by randomly sampled noise.

We chose to use a simple form of data augmentation by adding additive noises of various type to the training data before MFCC extraction. In this way we doubled the size of the training set. We decided not to use cafeteria noise in data augmentation to maintain some mismatch between training and test data. Adding the same noise to training and test set would have been an ideal condition that cannot really be achieved in real-life applications. Some types of noise we used are office, fan and car noise. The SNR of the noisy training speech was also set to 15dB.

The results with training data augmentation are shown in table 5.4. The performance in the noisy test scenario is much better than before for all systems and evaluation metrics, confirming that adding noise to the training helps in achieving more robust systems that generalize better to more test scenarios. The cost of data augmentation is a small loss in performance on the clean test utterances, which can be tolerated in order to have a more robust system. In this sense the ANN-UBM seems to really benefit from noise injection, maintaining very good performance on both clean and noisy test utterance. In particular the ANN-UBM method has the best performance on the clean test utterances, with only 0.23% verification EER and 0.36% classification error, which is the best identification performance we could achieve on the Librispeech dataset when training with noisy data augmentation.

A comparison between the different systems in the speaker verification scenario is shown in figure 5.1 and figure 5.2, where we plot the DET curves for the different training and test conditions.

Despite having the best performance on clean test utterances when training with noise augmentation, the ANN-UBM method suffers more in generalizing to noisy test utterances compared to the single DNN. as can be seen in figure 5.1 the ANN-UBM method performs significatively worse, with almost 8% EER. This could be because the adversarial data generated with the UBM is purely clean, so the ANNs don't have noisy adversarial examples available for training.

The performance on noisy test utterances improves drastically with noise-augmented training, as shown in figure 5.2. All methods achieve EERs below 2%, a drastic improvement for the novel ANN-UBM method, which at the same time remains the best system when scoring clean test utterances.

In general the discriminative models performed better on most tasks and training conditions. This is probably due to the higher number of parameters and superior representational power of ANNs compared to GMMs. It is also interesting to compare the performance at different operating points on the DET curves. The i-vector system seems to perform better at points with very low FAR, which is ideal for security systems where we must be sure not to accept impostors. For our target application of query-by-voice and diarization we want to have very low FRR, in order to query successfully all utterances from a target speaker. For this scenario the i-vector system is the worse, while the DNN and the novel ANN-UBM system are the best.

Method	**Verification EER**	**Identification Error Rate**
	Clean test utterances	
GMM-UBM	0.4461 %	0.7662 %
I-vector	0.7041 %	2.3081 %
Single DNN	0.2222 %	0.7193 %
ANN-UBM	**0.1598 %**	**0.3720 %**
	Noisy test utterances	
GMM-UBM	4.8873 %	32.6512 %
I-vector	**3.0970 %**	**21.2187 %**
Single DNN	4.3817 %	36.4753 %
ANN-UBM	7.8997 %	54.1222 %

Table 5.3: Speaker recognition performance on held-out test set, using only clean speech for the training. Performance is reported with EER for verification and classification error rate for identification

Method	**Verification EER**	**Identification Error Rate**
	Clean test utterances	
GMM-UBM	0.6135 %	0.9434 %
I-vector	0.8609 %	3.0250 %
Single DNN	0.2957 %	0.9868 %
ANN-UBM	**0.2262 %**	**0.3555 %**
	Noisy test utterances	
GMM-UBM	1.8212 %	**9.2558 %**
I-vector	1.8868 %	10.0561 %
Single DNN	**1.3622 %**	11.2013 %
ANN-UBM	1.8952 %	13.3780 %

Table 5.4: Speaker recognition performance on held-out test set, using both clean and noisy speech for the training. Performance is reported with EER for verification and classification error rate for identification

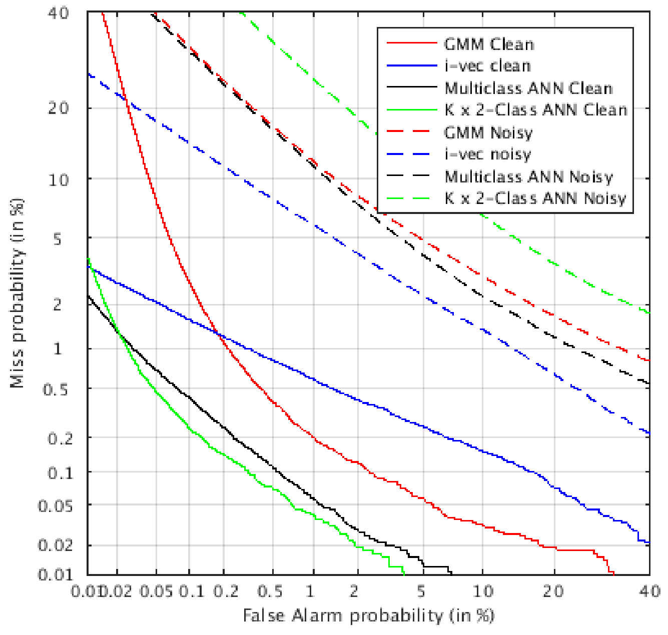


Figure 5.1: DET curves for speaker verification evaluation. Systems trained only on clean speech.

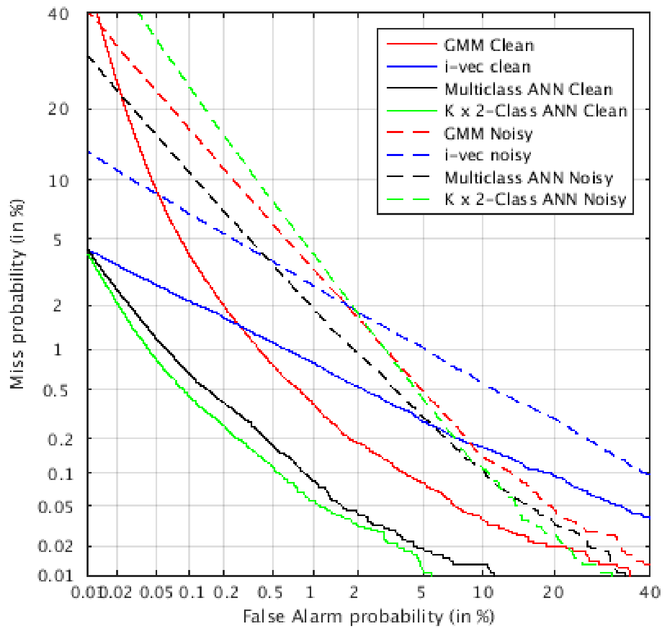


Figure 5.2: DET curves for speaker verification evaluation. Systems trained on both clean and noisy speech.

Chapter 6

Discussion and Future Work

In our experiments discriminative ANNs perform better than GMMs. This could be expected, given the higher representational power of ANNs. We are evaluating on a dataset with 2483 speakers, so we can assume that the feature space is very "crowded", with plenty of superposition between similar voices. This is probably why the use of more powerful non-linear models like ANN leads to significatively better performance. The GMM system could be improved by increasing the number of components, but this would lead to higher computational costs and moreover it is problematic to fit GMMs with many components when training data is scarce.

Regarding the i-vector method, we believe many improvements could be made to our implementation, leading to higher performance. In our experiments i-vectors performed worse than the three other methods. We identified some reasons for this and propose a possible direction for improvement:

- The size and training of the GMM-UBM. We used utterances from only 330 speakers for a total of roughly 20 hours of speech. While this is adequate for many systems, it is probably sub-optimal for our experiments, since our goal was to classify a dataset with 2483 speakers. We believe this affected the performance of the i-vector system, where the extraction of the supervectors depends only on the UBM. We propose to train a larger UBM on the whole Librispeech database, using 1024 or 2048 gaussian componenets. Also the use of full-covariance guassians could be advantageous, as noted in [71].

- The fact that we used simple cosine distance to score the utterances limited the performance of our i-vector system. We should repeat the evaluation with the scoring proposed in [5] and described in 2, using LDA followed by WCCN and length normalization.

- Also other scoring methods have been investigated to boost i-vector system performance. Our experiments with PLDA did not lead to significatively better results on the Librispeech database. LDA followed by heavy-tailed PLDA, where gaussian distributions are replaced by student-t distributions, was shown to perform better than all other methods in [71].

We also want to highlight some other advantages of using ANNs.

- They can easily handle high-dimensional inputs, since computationally they just need to perform linear matrix transformations and a non-linearity. Also, with proper regularization and training hyperparameters, DNNs can be trained effectively with a small number of training examples per parameter. On the contrary, it is difficult to train GMMs in high-dimensional spaces, since the number of required training samples for reliable density estimation grows exponentially with the number of features.

- When using more than one hidden layer, ANNs can learn hierarchical representations of the input data. This is key to achieve properties like robustness to noise and invariance to confounding input variation and has been proven to be one of the reasons why deep learning performs so well in tasks like object recognition. In the case of speech, a DNN could learn invariance to background noise and channel variability.

- Since they can handle larger inputs, it is possible to train them with longer time windows as input. This could be used to learn long-term speech variability which is completely ignored in current speaker recognition systems, as described later in the chapter.

Regarding the comparisons of our systems, we can make some remarks also on the complexity and memory requirements.

- *Memory requirements:* the systems based on ANNs have larger memory requirements, since the networks have lots of parameters to be stored. The GMM system is the one with less parameters, but the most convenient system for storing speaker models is the i-vector one. Only the T and Σ matrices must be stored for the extraction of the i-vectors and each speaker and utterance is modelled by a single low-dimensional vector.

- *Scalability:* as already mentioned, the DNN system is not scalable, while the GMM, i-vector and ANN-UBM systems are fully scalable. We can enroll or remove speakers easily and perform training and scoring in parallel on different machines.

- *Scoring speed:* the i-vector system again is the most convenient, since scoring is performed with a simple distance measure between two low-dimensional vectors. For speaker identification, the single DNN has good scoring speed since it computes all the outputs with a single feedforward pass, but it is not convenient for speaker verification.

- *Robustness to noise and channel variability:* in our experiments GMMs are the most robust to additive noise in the test utterances. But the problem for ANN is easily solved by adding training noise to the training set. In this scenario the DNN performs better than all other methods. We believe data augmentation is really important to train a robust system. Unfortunately the ANN-UBM method benefits less than the other three from data augmentation, probably because the adversarial data generated with the UBM models clean speech and is difficult to inject noise related information in the UBM.

Given all these considerations we believe that, given enough computational power and memory, the DNN and ANN-UBM methods are superior to the other systems and have the potential for achieving state-of-the-art results in the future. Also for practical applications they can perform better on clean speech (ANN-UBM system) and also on noisy speech when trained with noise augmentation (DNN system). It would be important in the future to repeat our experiments on a benchamark dataset, like the one used in the NIST speaker recognition evaluation. This would allow us to make a better comparison with the state-of-the-art i-vector systems.

Another possible direction for future work is a different use of the discriminatively trained DNN. The idea, already explored in [23], is to use the DNN as a feature extractor, by accumulating and normalizing the activations in the deepest hidden layer of the network we can obtain a fixed size representation of an utterance, which can be used and scored exactly like an i-vector. In [23] they called this representation d-vector and compared it with an i-vector system. The performance of their particular system was a little worse, but we believe the idea is promising and with some modifications could lead to very good results.

Also we could make a modification to our ANN-UBM system, by using the same idea, but with the deep features learned by the larger single DNN. As can be noted in figure 4.1, the ANN-UBM method fundamentally performs logistic regression on a feature representation learned independelty for each speaker. We could instead use the deep feature representation learned by the single DNN to represent the target data and the impostor data from the UBM. It would then be easy to train a logistic regression classifier in this feature space for each speaker independently. This needs further investigation, but it could be advantageous to keep a shared feature representation for all speakers, learned with the DNN trained with noisy data augmentation on the whole dataset.

We conclude the thesis with some considerations on the future of speaker recognition. One recent trend is trying to incorporate long-time information into the decision process. It is believed that using only frame-based cepstral features is not the optimal solution, because long-term variations contain important information to discriminate between different speakers. Prosody, speed and articulation are all speaker-dependent factors that are ignored by purely frame-based classification, but that we humans use to recognize the identity of a particular voice.

There is a strong belief that prosodic and spectro-temporal features (duration, rhythm, pitch and energy variations) and high-level features (phones, accent, pronunciation) are salient speaker cues [7].

The question is how to capture this temporal and high-level information for the goal of speaker recognition, without necessarily using computationally-intensive speech recognizers to extract semantics from the utterance. We believe that it could be automatically learned from the data by using a model that analyses long-time windows or that keeps an internal state which evolves over time.

An example of a model which could become state-of-the-art in the future is the Recurrent Neural Network (RNN). RNNs are powerful models for sequential data, thanks to their feedback loops between hidden units, creating an internal state which evolves over time. In particular the Long Short-term Memory

RNN [72] is able to learn long-time dependencies in sequential data, making it the most powerful RNN so far, achieving state-of-the-art results in many cursive handwriting recognition tasks and other complex time-sequence analysis. RNNs have historically been very hard to train, but recently many improvements have been made in their optimization and much research is being conducted in their use for speech recognition. For example in [73] they used a deep Long Short-term Memory RNN to achieve state-of-the-art results on the TIMIT phoneme recognition benchmark. Of much interest is also the research direction outlined in [74], where they propose a RNN architecture to achieve end-to-end speech recognition, without the need of a hidden Markov model or an intermediate phonetic representation. All these research suggests that in the future RNNs could become the standard for state-of-the-art speech recognition, outperforming current approaches based on deep feed-forward networks and hidden Markov models. Our proposal is that similar techniques could also be explored for the speaker recognition task, by setting a RNN architecture that could capture both the spectral and the time-varying information which is critical to discriminate between different speakers.

Another interesting deep learning component is the Convolutional Neural Network (CNN), a type of feedforward neural network in which the connectivity structure between neurons is inspired by the ones found in the mammalian visual or auditory cortex. Developed in the 1980s for temporal signals [75, 76], they have been improved by Yan LeCun's group for the task of image classification, with a famous paper 1998 paper on handwritten digit recognition [77]. A nice property of CNN is that they are biologically inspired and designed to use minimal amounts of pre-processing, trying to learn their own hidden feature representations from the raw data. They are by far the most successful deep learning model for image and video classification and are also used in the field of unsupervised representation learning [78]. Despite being so popular in the computer vision community, they have not been tested extensively on speech signals. A nice direction for future work would be to use CNNs on the speech waveform or on its spectrogram to try to learn powerful features for speech and speaker recognition. The idea has been explored in [79], with promising results. Their network architecture is outlined in figure 6.1, where they used 1-D convolutions along the frequency bands of the spectrogram.

This line of research is interesting and it has not been explored thoroughly. Various modifications in the connectivity patterns could be made, with convolutional layers across both frequency and time, or 1-D convolutions across time. In general the use of longer-time inputs to a CNN should be beneficial for the speaker recognition task and is part of the experiments we would like to conduct in the future, both on the speech spectrogram or directly on the speech waveform.

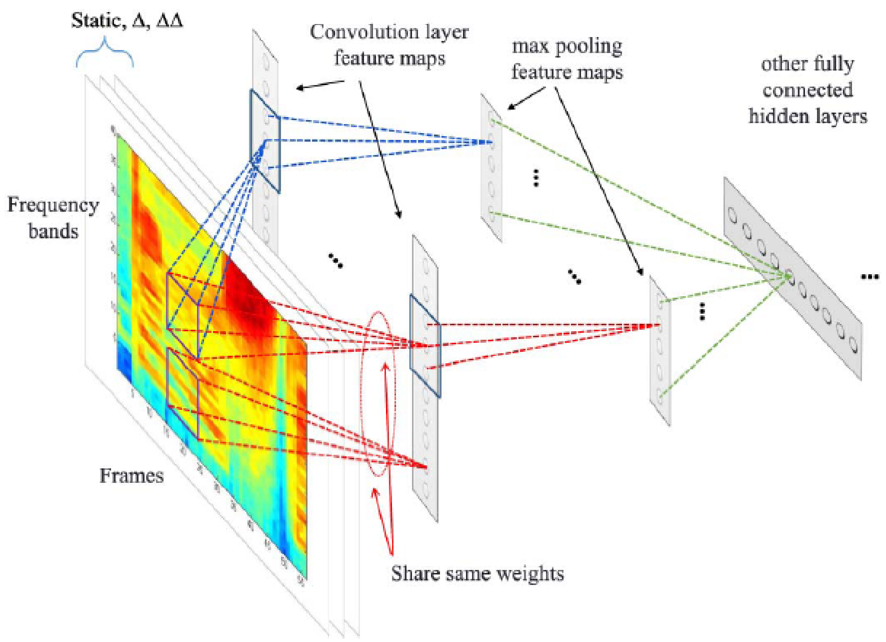


Figure 6.1: An illustration of the regular CNN that uses so-called full weight sharing. Here, a 1-D convolution is applied along frequency bands. Image taken from [79]

To conclude, deep learning approaches hold very big potential for speech analysis and speaker recognition. The hope is that they will be explored much more in the future, leading to big breakthroughs and better audio and speech analysis systems. This research could also lead to interesting links between biology, neuroscience and artificial intelligence, giving some insights into how exactly sound is transformed and processed by the brain. This would help us understand what happens when we are listening to music, understanding speech or just identifying people by the sound of their voices.

Bibliography

[1] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

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

[3] C.-S. Lee, M.-H. Wang, S.-J. Yen, T.-H. Wei, I. Wu, P.-C. Chou, C.-H. Chou, M.-W. Wang, T.-H. Yang, *et al.*, "Human vs. computer go: Review and prospect," *arXiv preprint arXiv:1606.02032*, 2016.

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

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

[6] 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)*, pp. 5206–5210, IEEE, 2015.

[7] T. Kinnunen and H. Li, "An overview of text-independent speaker recognition: From features to supervectors," *Speech communication*, vol. 52, no. 1, pp. 12–40, 2010.

[8] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[9] A. Martin, G. Doddington, T. Kamm, M. Ordowski, and M. Przybocki, "The DET curve in assessment of detection task performance," tech. rep., DTIC Document, 1997.

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

[11] J. A. Bilmes *et al.*, "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov

models," *International Computer Science Institute*, vol. 4, no. 510, p. 126, 1998.

[12] W. M. Campbell, J. P. Campbell, D. A. Reynolds, D. A. Jones, and T. R. Leek, "Phonetic speaker recognition with support vector machines," in *Advances in neural information processing systems*, 2003.

[13] L. Ferrer, E. Shriberg, S. Kajarekar, and K. Sonrnez, "Parameterization of prosodic feature distributions for SVM modeling in speaker recognition," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, pp. IV–233, IEEE, 2007.

[14] W. M. Campbell, J. P. Campbell, D. A. Reynolds, E. Singer, and P. A. Torres-Carrasquillo, "Support vector machines for speaker and language recognition," *Computer Speech & Language*, vol. 20, no. 2, pp. 210–229, 2006.

[15] 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 *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1, pp. I–I, IEEE, 2006.

[16] A. Solomonoff, W. M. Campbell, and I. Boardman, "Advances in channel compensation for SVM speaker recognition.," in *ICASSP (1)*, pp. 629–632, 2005.

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

[18] P. Kenny, P. Ouellet, N. Dehak, V. Gupta, and P. Dumouchel, "A study of interspeaker variability in speaker verification," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 5, pp. 980–988, 2008.

[19] S. J. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, IEEE, 2007.

[20] L. P. Heck, Y. Konig, M. K. Sönmez, and M. Weintraub, "Robustness to telephone handset distortion in speaker recognition by discriminative feature design," *Speech Communication*, vol. 31, no. 2, pp. 181–192, 2000.

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

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

[23] E. Variani, X. Lei, E. McDermott, I. Lopez Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 4052–4056, IEEE, 2014.

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

[25] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[26] D. A. Reynolds, "Experimental evaluation of features for robust speaker identification," *Speech and Audio Processing, IEEE Transactions on*, vol. 2, no. 4, pp. 639–643, 1994.

[27] S. J. Young and S. Young, *The HTK hidden Markov model toolkit: Design and philosophy.* University of Cambridge, Department of Engineering, 1993.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[29] Y. Chen and M. R. Gupta, "EM demystified: An expectation-maximization tutorial," in *Electrical Engineering*, Citeseer, 2010.

[30] R. Auckenthaler, M. Carey, and H. Lloyd-Thomas, "Score normalization for text-independent speaker verification systems," *Digital Signal Processing*, vol. 10, no. 1, pp. 42–54, 2000.

[31] Y. Zhang and M. S. Scordilis, "Optimization of GMM training for speaker verification," in *ODYSSEY04-The Speaker and Language Recognition Workshop*, 2004.

[32] D. Reynolds, "Gaussian mixture models," *Encyclopedia of biometrics*, pp. 827–832, 2015.

[33] P. Kenny, G. Boulianne, and P. Dumouchel, "Eigenvoice modeling with sparse training data," *IEEE transactions on speech and audio processing*, vol. 13, no. 3, pp. 345–354, 2005.

[34] P. Kenny, "Bayesian speaker verification with heavy-tailed priors.," in *Odyssey*, p. 14, 2010.

[35] H. Lei, "Joint Factor Analysis (JFA) and i-vector tutorial," *ICSI. Web. 02 Oct*, 2011.

[36] P. Kenny, "A small footprint i-vector extractor.," in *Odyssey*, pp. 1–6, 2012.

[37] P.-M. Bousquet, A. Larcher, D. Matrouf, J.-F. Bonastre, and O. Plchot, "Variance-spectra based normalization for i-vector standard and probabilistic linear discriminant analysis.," in *Odyssey*, pp. 157–164, 2012.

[38] S. Biswas, J. Rohdin, and K. Shinoda, "Autonomous selection of i-vectors for PLDA modelling in speaker verification," *Speech Communication*, vol. 72, pp. 32–46, 2015.

[39] O. Glembek, L. Burget, P. Matějka, M. Karafiát, and P. Kenny, "Simplification and optimization of i-vector extraction," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4516–4519, IEEE, 2011.

[40] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[41] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," 1974.

[42] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[43] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area V2," in *Advances in neural information processing systems*, pp. 873–880, 2008.

[44] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng, "Measuring invariances in deep networks," in *Advances in neural information processing systems*, pp. 646–654, 2009.

[45] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks.," in *Aistats*, vol. 15, p. 275, 2011.

[46] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[47] M. D. Richard and R. P. Lippmann, "Neural network classifiers estimate bayesian a posteriori probabilities," *Neural computation*, vol. 3, no. 4, pp. 461–483, 1991.

[48] P. Lennie, "The cost of cortical computation," *Current biology*, vol. 13, no. 6, pp. 493–497, 2003.

[49] M. A. Nielsen, "Neural networks and deep learning," *URL: http://neuralnetworksanddeeplearning. com/.(visited: 01.11. 2014)*, 2015.

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

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

[52] D. C. Plaut *et al.*, "Experiments on learning by back propagation.," 1986.

[53] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence O (1/k2)," in *Doklady an SSSR*, vol. 269, pp. 543–547, 1983.

[54] I. Sutskever, *Training recurrent neural networks.* PhD thesis, University of Toronto, 2013.

[55] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8624–8628, IEEE, 2013.

[56] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, 2012.

[57] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[58] C. W. Groetsch, *The theory of Tikhonov regularization for Fredholm equations of the first kind*, vol. 105. Pitman Advanced Publishing Program, 1984.

[59] R. C. S. L. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, vol. 13, p. 402, MIT Press, 2001.

[60] L. Prechelt, "Early stopping-but when?," in *Neural Networks: Tricks of the trade*, pp. 55–69, Springer, 1998.

[61] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[62] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[63] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, *et al.*, "Handling imbalanced datasets: A review," *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25–36, 2006.

[64] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, 2006.

[65] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[66] M. Kubat, S. Matwin, *et al.*, "Addressing the curse of imbalanced training sets: one-sided selection," in *ICML*, vol. 97, pp. 179–186, Nashville, USA, 1997.

[67] M. Kukar, I. Kononenko, *et al.*, "Cost-sensitive learning with neural networks.," in *ECAI*, pp. 445–449, 1998.

[68] C.-W. Hsu and C.-J. Lin, “A comparison of methods for multiclass support vector machines,” *IEEE transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[69] C. M. Bishop, “Training with noise is equivalent to Tikhonov regularization,” *Neural computation*, vol. 7, no. 1, pp. 108–116, 1995.

[70] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008.

[71] P. Matějka, O. Glembek, F. Castaldo, M. J. Alam, O. Plchot, P. Kenny, L. Burget, and J. Černocky, “Full-covariance UBM and heavy-tailed PLDA in i-vector speaker verification,” in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4828–4831, IEEE, 2011.

[72] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[73] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649, IEEE, 2013.

[74] A. Graves and N. Jaitly, “Towards end-to-end speech recognition with recurrent neural networks.,” in *ICML*, vol. 14, pp. 1764–1772, 2014.

[75] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” tech. rep., DTIC Document, 1985.

[76] L. E. Atlas, T. Homma, and R. J. Marks II, “An artificial neural network for spatio-temporal bipolar patterns: Application to phoneme classification,” in *Proc. Neural Information Processing Systems (NIPS)*, p. 31, 1988.

[77] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[78] Q. V. Le, “Building high-level features using large scale unsupervised learning,” in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 8595–8598, IEEE, 2013.

[79] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, “Convolutional neural networks for speech recognition,” *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

Notes

TRITA EE 2016:151
ISSN 1653-5146