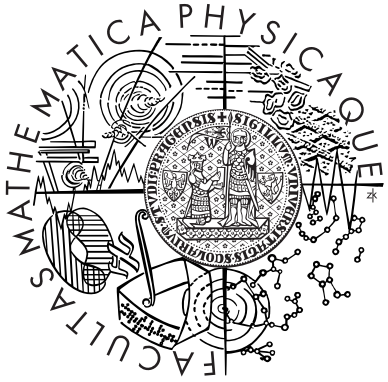
Charles University in Prague
Faculty of Mathematics and Physics

University of Groningen
Faculty of Arts

MASTER THESIS



Bich Ngoc Do

Neural Networks for Automatic Speaker, Language and Sex Identification

Supervisors: Ing. Mgr. Filip Jurčíček, Ph.D.
Dr. Marco Wiering

Master of Computer Science
Mathematical Linguistics

Master of Arts
Linguistics

Prague 2015

Title: Neural networks for automatic speaker, language, and sex identification

Author: Bich-Ngoc Do

Department: Institute of Formal and Applied Linguistics, Faculty of Mathematics Physics, Charles University in Prague; Department of Linguistics, Faculty of Arts, University of Groningen

Supervisor: Ing. Mgr. Filip Jurčíček, Ph.D., Institute of Formal and Applied Linguistics, Charles University in Prague and Dr. Marco Wiering, Institute of Artificial Intelligence and Cognitive Engineering, Faculty of Mathematics and Natural Sciences, University of Groningen

Abstract: Speaker recognition is a challenging task and has applications in many areas, such as access control or forensic science. Moreover, in recent years, the deep learning paradigm and its branch, deep neural networks have emerged as powerful machine learning techniques and achieved state-of-the-art performance in many fields of natural language processing and speech technology. Therefore, the aim of this work is to explore the capability of a deep neural network model, recurrent neural networks, in speaker recognition. Our proposed systems are evaluated on the TIMIT corpus using speaker identification tasks. In comparison with other systems in the same test conditions, our systems could not surpass reference ones due to the sparsity of validation data. In general, our experiments show that the best system configuration is a combination of MFCCs with their dynamic features and a recurrent neural network model. We also experiment recurrent neural networks and convolutional neural networks in a simpler task, sex identification, on the same TIMIT data.

Keywords: speaker identification, sex identification, deep neural network, recurrent neural network, convolution neural network, MFCC, TIMIT

Contents

Chapter 1

Introduction

Communication is an essential need of humans, and speaking is one of the most natural forms of communication besides facial expressions, eye contact and body language. The study of speech dates back even before the digital era, with legends about mechanical devices which were able to imitate human voices in the 13th century [5]. However, the development of speech processing did not progress rapidly until 1930s after two inventions about speech analysis and synthesis at Bell Laboratories. Those events are often considered to be the beginning of the modern speech technology era [14].

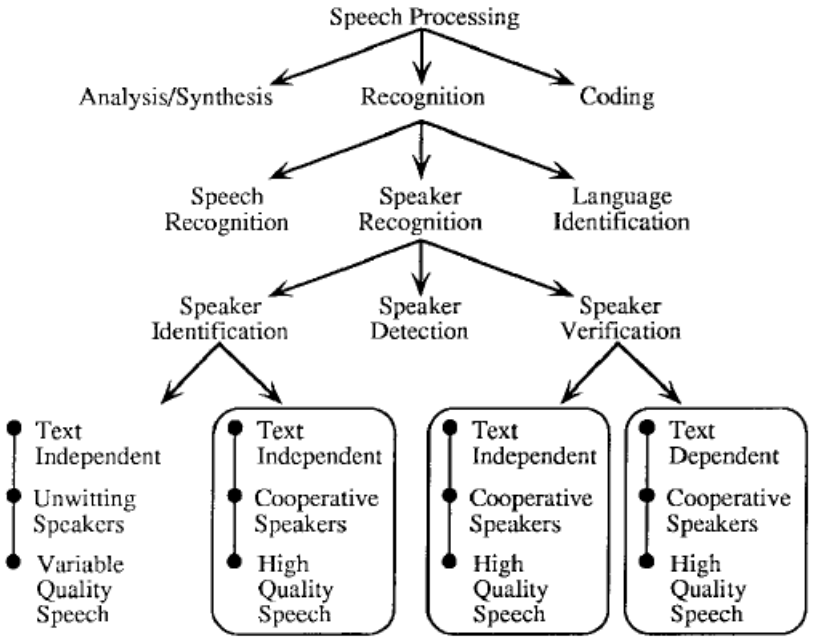


Figure 1.1: Some areas in speech processing (adapted from [9])

There is not a unique way to classify subfields in speech processing, but in general, it can be divided into some main components: analysis, coding/synthesis and recognition [14]. Among those, *recognition* area directly deals with basic information that speech delivers, for instance, its message of words (*speech recognition*), language (*language identification*) and information about the speaker such as his or her gender, emotion (*speaker recognition*) (see figure 1.1).

In other words, beside transmitting a message as other means of communication do, speech also reveals the identity of its speaker. Together with other biometrics such as face recognition, DNA, fingerprint, ..., speaker recognition plays an important role in many fields, from forensics to security control. The first attempts at this field were made in the 1960s [20]; since then its approaches have ranged from simple template matching to advanced statistical modeling like hidden Markov models or artificial neural networks.

In our work, we would like to use one of the most effective statistical models today to solve speaker recognition problems, which is deep neural networks. Hence, the aim of this thesis is to apply deep neural network models to identify speakers, to show if this approach is promising and to prove its efficiency by comparing its results to other techniques. Our evaluation is conducted on TIMIT data released in the year 1990.

1.1 Problem Definition

Speaker recognition is the task of recognizing a speaker's identity from his or her voice, and is different from *speech recognition* of which purpose is to recognize the content of the speech. It is also referred as *voice recognition*, but this term is not encouraged since it has been used with the meaning of speech recognition for a long time [4]. The area of speaker recognition involves two major tasks: verification and identification (figure 1.1). Their basic structures are shown in figure 1.2.

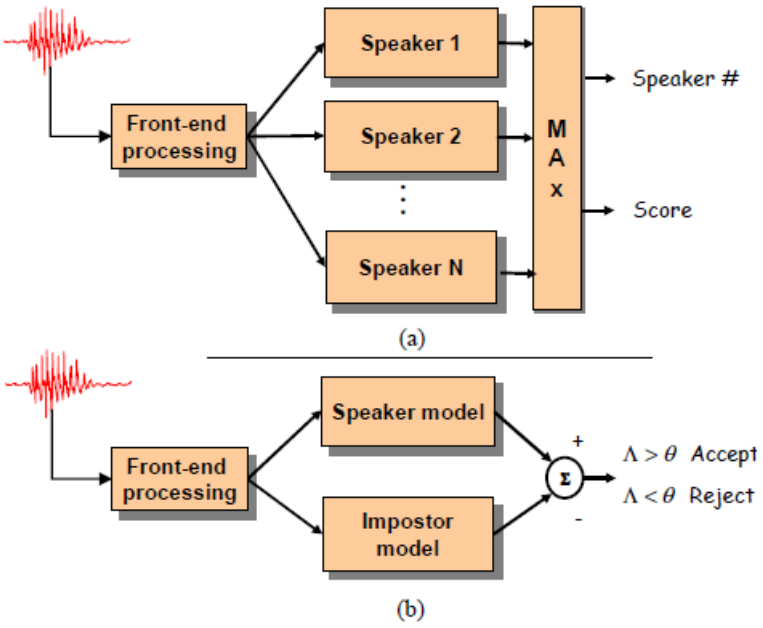


Figure 1.2: Structures of (a) speech identification and (b) speech verification (adapted from [55])

Speaker verification is the task of authenticating a speaker's identity, i.e., to check whether the speaker is the one he or she claims to be (yes or no decision). The speaker who claims the identity is known as the *test speaker*; the signal is

then compared against the model of the *claimant*, i.e. the speaker whose identity the system knows about. Other speakers except the claimant are called *impostors*. A verification system is trained using not only the claimant's signal but also data from other speakers, called *background speakers*. In the evaluation phase, the system compares the likelihood ratio Δ (between the score corresponding to the claimant's model to that of the background speakers' model) with a threshold θ. If $\Delta \geq \theta$, the speaker is accepted, otherwise he or she is rejected. Since the system usually does not know about the test speaker identity, this task is an *open-set* problem.

Speaker identification, on the other hand, determines who the speaker is among known voices registered in the system. Given an unknown speaker, the system must compare his or her voice to a set of available models, thus makes this task a one-vs-all classification problem. The type of identification can be *closed-set* or *open-set* depending on its assumption. If the test speaker is guaranteed to come from the set of registered speakers, its type is closed-set, and the system returns the most probable model ID. In case its type is open-set, there is a chance that the test speaker's identity is unknown, and the system should make a rejection in this situation.

Speaker detection is another subtask of speaker recognition, which aims at detecting one or more specific speakers in a stream of audio [4]. It can be viewed as a combination of segmentation together with speaker verification and/or identification. Depending on a specific situation, this problem can be formulated as a speech recognition problem, a verification problem or both of them. For instance, one way to combine both tasks is to perform identification first, and then use returned the ID for the verification session.

Based on the restriction of texts used in speech, speech recognition can be further categorized as *text-dependent* and *text-independent* [54]. In *text-dependent speech recognition*, all speakers say the same words or phrases during both training and testing phases. This modality is more likely to be used in speaker verification than other branches [4]. In *text-independent speech recognition*, there is no constraint placed on training and testing texts; therefore, it is more flexible and can be used in all branches of speech recognition.

1.2 Components of a Speaker Recognition System

Figure 1.2 illustrates a basic structure of a speaker identification and a speaker verification system. In both systems, first, the audio signal is directed to the *front-end processing*, where features that represent the speaker information are extracted. The heart of the front-end is undoubtedly a feature extraction module, which transforms a signal into a vector of features. The *short-time spectral* is the most frequently used type of features in speech processing, but types of feature may range from short-time spectral (e.g. spectrum) to prosodic and auditory features (e.g. pitch, loudness, rhythm, ...) and even high level features such as phones or accent. The front-end may also include pre/post processing modules as well, such as *voice activity detection* to remove silence from the input, or a *channel compensation* module to normalize the effect of the recording channel

[55].

In a speaker recognition system, a vector of features acquired from the previous step is compared against a set of *speaker models*. The identity of the test speaker is associated with the ID of the highest scoring model. A speaker model is a statistical model that represents speaker-dependent information, and can be used to predict new data. Generally, any modeling techniques can be used, but the most popular techniques are: clustering, hidden Markov model, artificial neural network and Gaussian mixture model.

A speaker verification system has an extra *impostor model* which stands for non-speaker probability. An impostor model can apply any technique in speaker models, but there are two main approaches for impostor modeling [55]. The first approach is to use a *cohort*, also known as a likelihood set, a background set, which is a set of background speaker models. The impostor likelihood is computed as a function of all match scores of background speakers. The second approach uses a single model trained on a large amount of speakers to represent general speech patterns. It is known as *general*, *world* or *universal background model*.

1.3 Thesis Outline

This thesis is organized into 6 chapters, of which contents are described as follows:

Chapter 1 The current chapter provides general information about our research interest, speaker identification, and its related problems.

Chapter 2 This chapter revises the theory of speech signal processing that becomes the foundation of extracting speech features. Important topics are frequency analysis, short-term processing and cepstrum.

Chapter 3 This chapter presents common techniques in speaker identification, including the baseline system Gaussian mixture models and the state-of-the-art technique i-vector.

Chapter 4 In this chapter, the method that inspires this project, deep neural networks, is inspected closely.

Chapter 5 This chapter presents the data that are used to evaluate our approach and details about our experimental systems. Experiment results are compared with reference systems and analyzed.

Chapter 6 This chapter serves as a summary of our work and presents some future directions.

Chapter 2

Speech Signal Processing

In this chapter, we characterize speech as a *signal*. All speech processing techniques are based on signal processing; therefore, we revise the most fundamental definition in signal processing such as signals and systems, signal representation and frequency analysis. After that, *short-term analysis* is introduced as an effective set of techniques to analyze speech signals despite our limited knowledge about them. Finally, the history and idea of *cepstrum* is discussed briefly.

2.1 Speech Signals and Systems

In signal processing, a *signal* is an observed measurement of some phenomenon [4]. The velocity of a car or the price of a stock are both examples of signals in different domains. Normally, a signal is modeled as a function of some independent variable. Usually, this variable is time, and we can denote that signal as $f(t)$. However, a signal does not need to be a function of a single variable. For instance, an image is a signal $f(x, y)$ which denotes the color at point (x, y).

2.1.1 Analog and digital signals

If the range and the domain of a signal are *continuous* (i.e. the independent variables and the value of the signal can take arbitrary values), it is an *analog signal*. Although analog signals have the advantage of being analyzed by calculus methods, they are hard to be stored on computers where most signal processing takes place today. In fact, they need to be converted into *digital signals*, of which domains and ranges are discrete.

2.1.2 Sampling and quantization

The machine which *digitizes* an analog signal is called an *analog-to-digital* (A/D) or *continuous-to-discrete* (C/D) converter. First, we have to measure the signal's value at specific points of interest. This process is known as *sampling*. Let $x_a(t)$ be an analog signal as a function of time t. If we sample x_a with a sampling period T, the output digital of this process is $x[n] = x_a(nT)$. The *sampling frequency* F_s is defined as the inverse of the sampling period, $F_s = 1/T$, and its unit is **hertz** (Hz). Figure 2.1 shows some sampling processes of a sinusoidal signal. From this

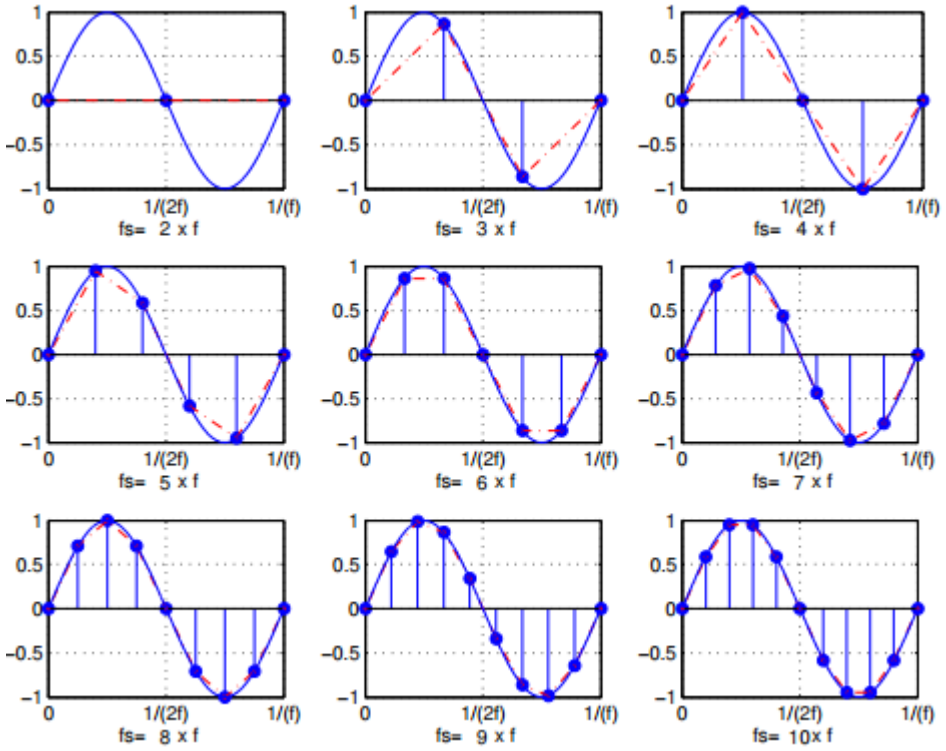


Figure 2.1: Sampling a sinusoidal signal at different sampling rates; f - signal frequency, f_s - sampling frequency (adapted from [4])

point, analog or continuous-time signals will use parentheses such as $x(t)$, while digital or discrete-time signals will be represented by square brackets such as $x[n]$.

After sampling, acquired values of the signal must be converted into some discrete set of values. This process is called *quantization*. In audio signals, the quantization level is normally given as the number of bits needed to represent the range of the signal. For example, values of a 16-bit signal may range from -32768 to 32767. Figure 2.2 illustrates an analog signal which is quantized at different levels.

The processes of sampling and quantization cause losses in information of a signal, thus they introduce noise and errors to the output. While the sampling frequency needs to be fast enough in order to effectively reconstruct the original signal, in case of quantization, the main problem is a trade-off between the output signal quality and its size.

2.1.3 Digital systems

In general, a system is some structure that receives information from signals and performs some tasks. A digital system is defined as a transformation of an input signal into an output signal:

$$y[n] = T\{x[n]\} \tag{2.1}$$

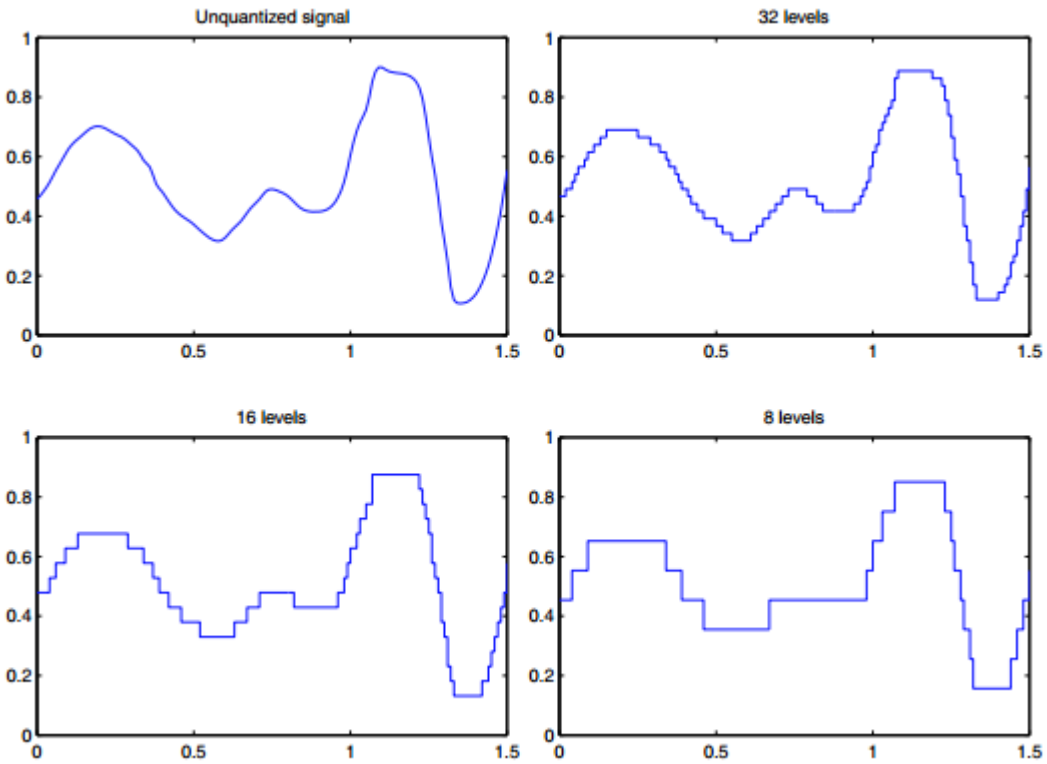


Figure 2.2: Quantized versions of an analog signal at different levels (adapted from [10])

2.2 Signal Representation: Time Domain and Frequency Domain

Speech sounds are produced by vibrations of vocal cords. The output of this process is *sound pressure*, which is changes in air pressure caused by sound wave. The measurement of sound pressure is called *amplitude*. A *speech waveform* is a representation of sound in the *time domain*. It displays changes of amplitude through time. Figure 2.3a is the plot of a speech waveform. The waveform shape tells us in an intuitive way about the periodicity of the speech signal, i.e. its repetition over a time period (figure 2.4). Formally, an analog signal $x_a(t)$ is *periodic* with period T if and only if:

$$x_a(t+T) = x_a(t) \quad \forall t \tag{2.2}$$

Similary, a digital signal $x[n]$ is *periodic* with period N if and only if:

$$x[n+N] = x[n] \quad \forall n \tag{2.3}$$

In contrast, a signal that does not satisfy 2.2 (if it is analog) or 2.3 (if it is digital) is *nonperiodic* or *aperiodic*.

The *frequency domain* is another point of view to look at a signal besides the time domain. A very famous example of the frequency domain is the experiment of directing white light through a prism. Newton showed in his experiment [68] that a prism could break white light up into a band of colors, or *spectrum*, and

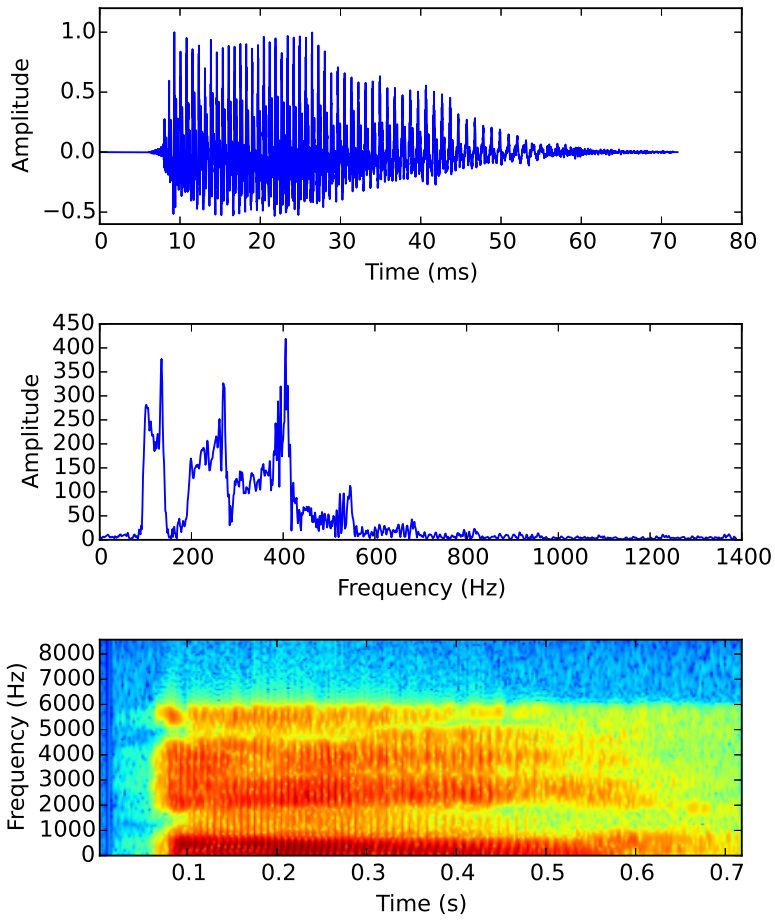


Figure 2.3: An adult male voice saying [a:] sampled at 44100 Hz: (a) waveform (b) spectrum limited to 1400 Hz (c) spectrogram limited from 0 Hz to 8000 Hz

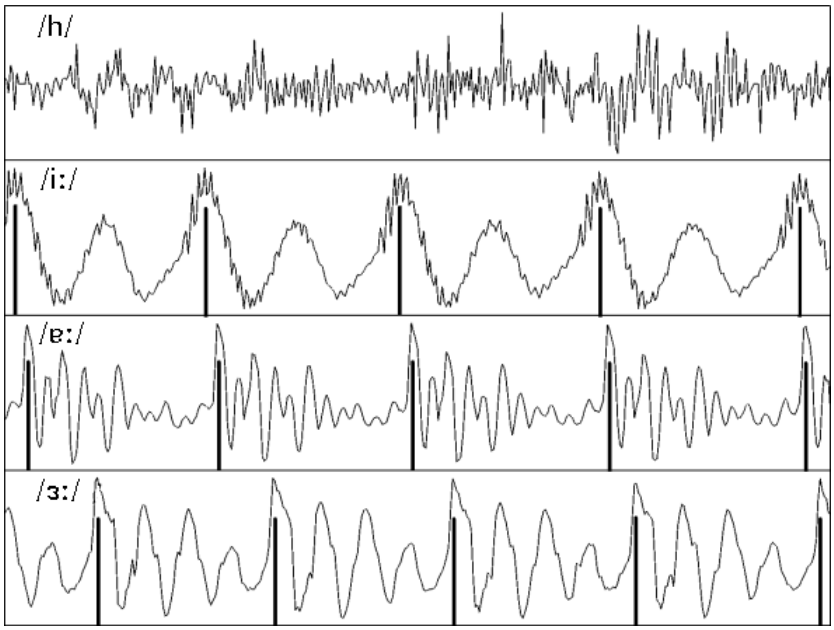


Figure 2.4: Periodic and aperiodic speech signals (adapted from [43]). The waveform of voiceless fricative [h] is aperiodic while the waveforms of three vowels are periodic.

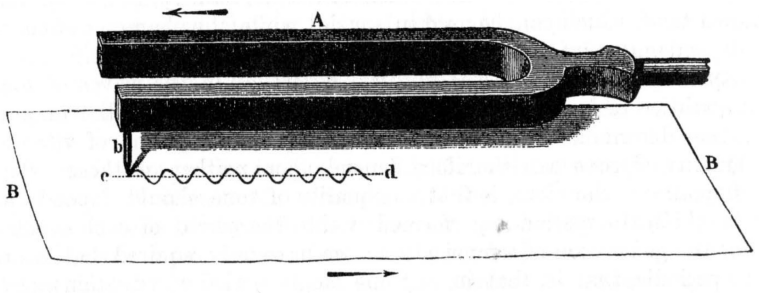


Figure 2.5: Illustration of the Helmholtz's experiment (adapted from [24])

furthermore, these color rays could be reconstituted into white light using the second prism. Therefore, white light can be *analyzed* into color components. We also know that each primary color corresponds to a range of frequencies. Hence, decomposing white light into colors is a form of *frequency analysis*.

In digital processing, the sine wave or *sinusoid* is a very important type of signal:

$$x_a(t) = A\cos(\omega t + \phi) \quad -\infty < t < \infty \tag{2.4}$$

where A is the *amplitude* of the signal, ω is the *angular frequency* in radians per second, and ϕ is the *phase* in radians. The *frequency* f of the signal in hertzs is related to the angular frequency by:

$$\omega = 2\pi f \tag{2.5}$$

Clearly, the sinusoid is periodic with period $T = 1/f$ from equation 2.2. Its digital version has the form:

$$x[n] = A\cos(\omega n + \phi) \quad -\infty < n < \infty \tag{2.6}$$

However, from equation 2.3, $x[n]$ is periodic with period N if and only if $\omega = 2\pi/N$ or its frequency $f = \omega/2\pi$ is a rational number. Therefore, the digital signal in equation 2.6 is not periodic for all values of ω.

A sinusoid with a specific frequency in speech processing is known as a *pure tone*. In the 19th century, Helmholtz discovered the connection between pitches and frequencies using a tuning fork and a pen attached to one of its tines [67] (figure 2.5). While the tuning fork was vibrating as a specific pitch, the pen was drawing the waveform across a piece of paper. It turned out that each pure tone is related to a frequency.

Hence, frequency analysis of a speech signal can be seen as decomposing it as sums of sinusoids. An example of speech signal decomposition is illustrated in figure 2.6. The process of changing a signal from time domain to frequency domain is called *frequency transformation*.

A *spectrum* is a representation of sound in frequency domain as it plots the amplitude at each corresponding frequency (see figure 2.3b). On the other hand, a *spectrogram* (see figure 2.3c) is a three dimension representation of spectral information. As usual, the horizontal axis displays time and the vertical axis displays frequencies. The shade at each time-frequency point represents the amplitude level. The higher the amplitude, the darker (or hotter if using colors) the shade. Spectrograms are effective visual cues to study the acoustics of speech.

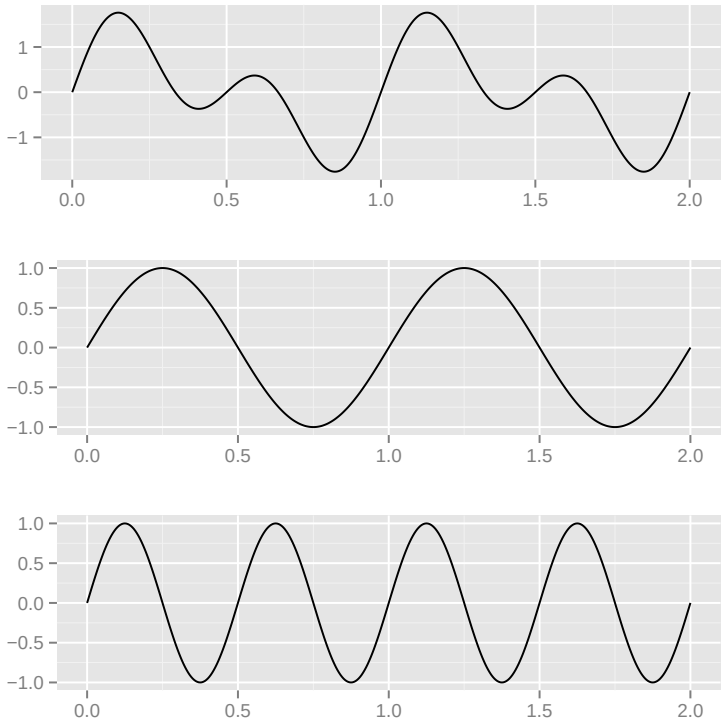


Figure 2.6: Decomposing a speech signal into sinusoids

Time domain properties	Periodic	Aperiodic	
Continuous	Fourier Series (FS)	Fourier Transform (FT)	Aperiodic
Discrete	Discrete Fourier Transform (DFT)	Discrete-Time Fourier Transform (DTFT)	Periodic
	Discrete	Continuous	Frequency domain properties

Table 2.1: Summary of Fourier analysis techniques (reproduced from [10])

2.3 Frequency Analysis

Fourier analysis techniques are mathematical tools which are usually used to transform a signal into the frequency domain. Which type of those techniques is chosen depends on whether a signal is analog or digital, and its periodicity. In summary, four types of Fourier analysis techniques are summarized in table 2.1. Each technique consists of a pair of transformations.

The *Fourier Series* (FS) of a continuous periodic signal $x(t)$ with period T is defined as:

$$c_k = \frac{1}{T} \int_T x(t) e^{-j2\pi kt/T} dt \tag{2.7}$$

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi kt/T} \tag{2.8}$$

The *Fourier Transform* (FT) of a continuous aperiodic signal $x(t)$ is defined as:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \tag{2.9}$$

$$x(t) = \int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega \tag{2.10}$$

The *Discrete Fourier Transform* (DFT) of a discrete periodic signal $x[n]$ with period N is defined as:

$$c_k = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \tag{2.11}$$

$$x[n] = \sum_{k=0}^{N-1} c_k e^{j2\pi kn/N} \tag{2.12}$$

The *Discrete-Time Fourier Transform* (DTFT) of a discrete aperiodic signal $x[n]$ is defined as:

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{2.13}$$

$$x[n] = \frac{1}{2\pi}\int_{2\pi} X(\omega)e^{j\omega n}d\omega \tag{2.14}$$

2.4 Short-Term Processing of Speech

Speech signals are *non-stationary*, which means their statistical parameters (intensity, variance, ...) change over time [4]. They may be periodic in a small interval, but no longer have that characteristic when longer segments are considered. Therefore, we cannot analyze them using Fourier transformations since it requires the knowledge of signals for infinite time. This problem led to a set of techniques called *short-time analysis*. Their ideas are splitting a signal into short segments or *frames*, assuming that the signal is *stationary* and periodic in one segment and analyzing each frame separately. The essence of those techniques is that each region needs to be short enough in order to satisfy the assumption, in practice, 10 to 20 ms. The spectrogram as discussed in section 2.2 is an example of short-time analysis. DTFT (section 2.3) is applied in each frame resulting a representation of spectra over time.

Given a speech signal $x[n]$, the *short-time signal* $x_m[n]$ of frame m is defined as:

$$x_m[n] = x[n]w_m[n] \tag{2.15}$$

with $w_m[n]$ is a *window function* that is zero outside a specific region. In general, we want $w_m[n]$ to be the same for all frames. Therefore, we can simplify it as:

$$w_m[n] = w[m-n] \tag{2.16}$$

$$w[n] = \begin{cases} \hat{w}[n] & |n| \leq \frac{N}{2} \\ 0 & |n| > \frac{N}{2} \end{cases} \tag{2.17}$$

where N is the length of the window.

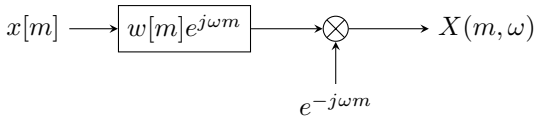


Figure 2.7: Block diagram of filter bank view of short-time DTFT

2.4.1 Short-time Fourier analysis

Considering Fourier analysis, given signal $x[n]$, from 2.13 the DTFT of frame $x_m[n]$ is:

$$\begin{aligned} X(m,\omega) = X_m(\omega) &= \sum_{n=-\infty}^{\infty} x_m[n]e^{-j\omega n} \\ &= \sum_{n=-\infty}^{\infty} x[n]w[m-n]e^{-j\omega n} \end{aligned} \tag{2.18}$$

Equation 2.18 is *short-time DTFT* of signal $x[n]$. It can be interpreted in two ways [52]:

- Fourier transform view: Short-time DTFT is considered as a set of DTFT at each time segment m, or
- Filter bank view: We rewrite 2.18 using a *convolution*[1] operator as:

  $$X(m,\omega) = e^{-j\omega m}(x[m] * w[m]e^{j\omega m}) \tag{2.19}$$

  Equation 2.19 is equivalent to passing $x[m]$ through a bank of bandpass filters centered around each frequency ω (figure 2.7).

2.4.2 Spectrograms

The magnitude of a spectrogram is computed as:

$$S(\omega,t) = \left|X(\omega,t)^2\right| \tag{2.20}$$

There are two kinds of spectrograms: *narrow-band* and *wide-band* (figure 2.8). Wide-band spectrograms use a short window length ($<$ 10 ms) which leads to filters with wide bandwidth ($>$ 200 Hz). In contrast, narrow-band spectrograms use longer window ($>$ 20 ms) which corresponds to narrow bandwidth ($<$ 100 Hz). The difference in window duration between two types of spectrograms results in time and frequency representation: while wide-band spectrograms give a good view of time resolution such as pitches, they are less useful with harmonics (i.e. component frequencies). Narrow-band spectrograms have a better resolution with frequencies but smear periodic changes over time. In general, wide-band spectrograms are more preferred in phonetic study.

[1]The convolution of f and g is defined as:

$$f[n] * g[n] = \sum_{k=-\infty}^{\infty} f[k]g[n-k]$$

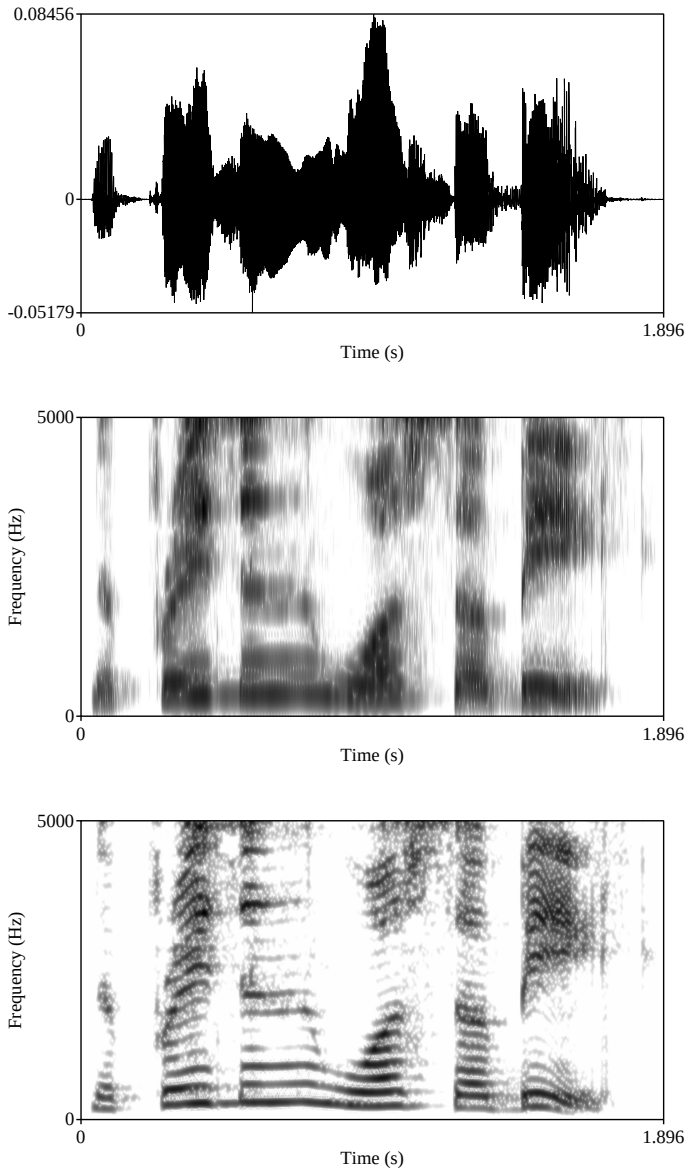


Figure 2.8: Two types of spectrograms: (a) original sound wave (b) wide-band spectrogram using 5 ms Hanning windows (c) narrow-band spectrogram using 23 ms Hanning windows

Spectral domain	Cepstral domain
Frequency	Quefrency
Spectrum	Cepstrum
Phase	Saphe
Amplitude	Gamnitude
Filter	Lifter
Harmonic	Rahmonic
Period	Repiod

Table 2.2: Corresponding terminology in spectral and cepstral domain (reproduced from [4])

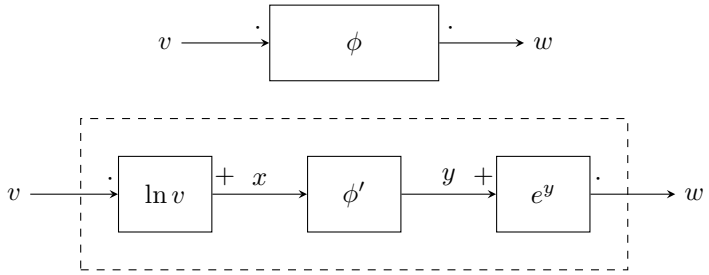


Figure 2.9: A homomorphic system with multiplication as input and output operation with two equivalent representations (adapted from [48])

2.5 Cepstral Analysis

The term *cepstrum* was first defined by Bogert et al. [8] as the inverse Fourier transform of the log magnitude spectrum of a signal. The transformation was used to separate a signal with simple echo into two components: a function of the original signal and a periodic function of which frequency is the echo delay. The independent variable of the transformation was not frequency, it was time but not the original time domain. Thus, Bogert et al. referred this new domain as *quefrency domain*, and the result of this process was called *cepstrum*. Both terms are anagrams of analog terms in spectral domain (frequency and spectrum) by flipping the first four letters. The authors also invented other terms in quefrency domain using the same scheme (table 2.2), however only some of them are used today.

In an independent work, Oppenheim was doing his PhD thesis on non-linear signal processing as the concept of *homomorphic system* [48]. In a homomorphic system, first the vector space of input operation was mapped on a vector space under addition, where we could apply linear transformation. Then, the intermediate vector space was mapped on a vector space of output operation. An example of a homomorphic transformation is illustrated in figure 2.9. The application of such systems in signal processing is known as *homomorphic filtering*.

Consider a homomorphic filtering with convolution as input operation. The first component of the system is responsible to map a convolution operation into an addition operation, or *deconvolution*:

$$D(s_1[t] * s_2[t]) = D(s_1[t]) + D(s_2[t]) \tag{2.21}$$

Intuitively, this transformation can be done by cascading Fourier transforms,

logarithms and inverse Fourier transforms as the definition of cepstrum.

The definition of the *complex cepstrum* of a discrete signal is:

$$\hat{x}[n] = \frac{1}{2\pi} \int_{2\pi} \hat{X}(\omega) e^{j\omega n} d\omega \tag{2.22}$$

where $X(\omega)$ is the DTFT of $x[n]$ and:

$$\hat{X}(\omega) = \log[X(\omega)] = \log(|X(\omega)|) + j \arctan(X(\omega)) \tag{2.23}$$

Similarly, the *real cepstrum* is defined as:

$$\hat{x}[n] = \frac{1}{2\pi} \int_{2\pi} \log(|X(\omega)|) e^{j\omega n} d\omega \tag{2.24}$$

Chapter 3

Approaches in Speaker Identification

After a careful review of speech processing theory in chapter 2, this chapter discusses contemporary methods and techniques used in speaker identification. The chapter is divided into three parts. The first part is dedicated to feature extraction or the *front-end* of a speaker identification system, which is firmly based on the theory introduced in chapter 2. Methods to model speakers or the *back-end* are described in the second part. Finally, the state-of-the-art technique in speaker identification, i-vector, is introduced.

3.1 Speaker Feature Extraction

The *short-time analysis* ideas discussed in section 2.4 and *cepstral analysis techniques* in section 2.5 have provided a powerful framework for modern speech analysis. In fact, the *short-time cepstrum* is the most frequently used analysis technique in speech recognition and speaker recognition. In practice, spectrum and cepstrum are computed by DFT as a sampled version of DTFT [53]:

$$X[k] = X(2\pi k/N) \tag{3.1}$$

The complex cepstrum is approximately computed using the following equations:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \tag{3.2}$$

$$\hat{X}[k] = \log(|X[k]|) + j\arctan(X[k]) \tag{3.3}$$

$$\hat{x}[n] = \frac{1}{N}\sum_{k=0}^{N-1} \hat{X}[k]e^{j2\pi kn/N} \tag{3.4}$$

Finally, the short-time spectrum and cepstrum are calculated by replacing a signal with its finite windowed segments $x_m[n]$.

3.1.1 Mel-frequency cepstral coefficients

First introduced in 1980 [12], *Mel-Frequency Cepstral Coeffcients* (MFCCs) are one of the best known parameterizations in speech recognition. MFCCs are different from the conventional cepstrum as they use a non-linear frequency scale

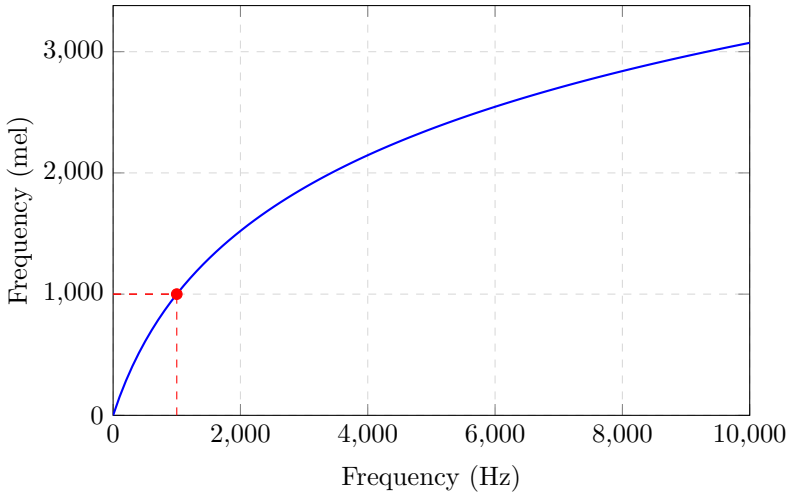


Figure 3.1: Relationship between the frequency scale and mel scale

based on auditory perception. MFCCs are based on *mel scale*. A *mel* is a unit of "measure of perceived *pitch or frequency* of a tone" [14]. In 1940, Stevens and Volkman [63] assigned 1000 mels as 1000 Hz, and asked participants to change the frequency until they perceived the pitch changed some proportions in comparison with the referential tone. The threshold frequencies were marked, resulting a mapping between real frequency scale (in Hz) and perceived frequency scale (in mel). A popular formula to convert from frequency scale to mel scale is:

$$f_{mel} = 1127 \ln\left(1 + \frac{f_{Hz}}{700}\right) \tag{3.5}$$

where f_{mel} is the frequency in mels and f_{Hz} is the normal frequency in Hz. This relationship is plotted in figure 3.1.

MFCCs are often computed using a filter bank of M filters ($m = 0, 1, ..., M - 1$), each one has a triangular shape and is spaced uniformly on the mel scale (figure 3.2). Each filter is defined by:

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \frac{k - f[m-1]}{f[m] - f[m-1]} & f[m-1] < k \leq f[m] \\ \frac{f[m+1] - k}{f[m+1] - f[m]} & f[m] \leq k < f[m+1] \\ 0 & k \geq f[m+1] \end{cases} \tag{3.6}$$

Given the DFT of the input signal in equation 3.2 with N as the sampling size of DFT, let us define f_{min} and f_{max} the lowest and highest frequencies of the filter bank in Hz and F_s the sampling frequency. $M + 2$ boundary points $f[m]$ ($m = -1, 0, ..., M$) are uniformly spaced between f_{min} and f_{max} on mel scale:

$$f[m] = \frac{N}{F_s} B^{-1}\left(B(f_{min}) + m\frac{B(f_{max}) - B(f_{min})}{M + 1}\right) \tag{3.7}$$

where B is the conversion from frequency scale to mel scale given in equation 3.5 and B^{-1} is its inversion:

$$f_{Hz} = 700\left(\exp\left(\frac{f_{mel}}{1125}\right) - 1\right) \tag{3.8}$$

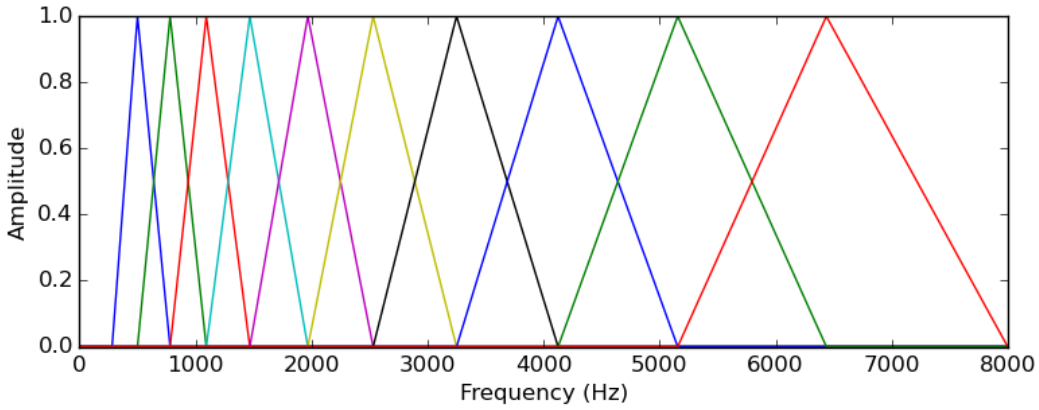


Figure 3.2: A filter bank of 10 filters used in MFCC

The log-energy mel spectrum is calculated as:

$$S[m] = \ln\left[\sum_{k=0}^{N-1} |X[k]|^2 H_m[k]\right] \quad m = 0, 1, ..., M-1 \tag{3.9}$$

with $X[k]$ is the output of DFT in equation 3.2.

Although traditional cepstrum uses inverse discrete Fourier transform (IDFT) as in equation 3.4, mel frequency cepstrum is normally implemented using discrete cosine transform II (DCT-II) since $S[m]$ is even [31]:

$$\hat{x}[n] = \sum_{m=0}^{M-1} S[m] \cos\left[\left(m + \frac{1}{2}\right)\frac{\pi n}{M}\right] \quad n = 0, 1, ..., M-1 \tag{3.10}$$

Typically, the number of filters M ranges from 20 to 40, and the number of kept coefficients is 13. Some research reported that the performance of speech recognition and speaker identification systems reached peak with 32-35 filters [65, 18]. Many speech recognition systems remove the zeroth coefficient from MFCCs because it is the average power of the signal [4].

3.1.2 Linear-frequency cepstral coefficients

Linear-Frequency Cepstral Coefficients (LFCCs) are very similar to MFCCs except that their frequency is not warped by a non-linear frequency scale, but a linear one (figure 3.3). The boundary points of the LFCC filter bank are spaced uniformly in frequency domain, between f_{min} and f_{max}:

$$f[m] = f_{min} + m\frac{f_{max} - f_{min}}{M+1} \tag{3.11}$$

Although MFCCs are more popular as features in speaker recognition, their high frequency range has poor resolution due to the characteristic of mel scale. Some works have proven the effect of frequency resolution on speaker recognition, for instance, Zhou et al. suggested that LFCCs performed better than MFCCs in female trials [70], or Lei and Gonzalo concluded that LFCCs had significant improvement in nasal and non-nasal consonant regions [40]

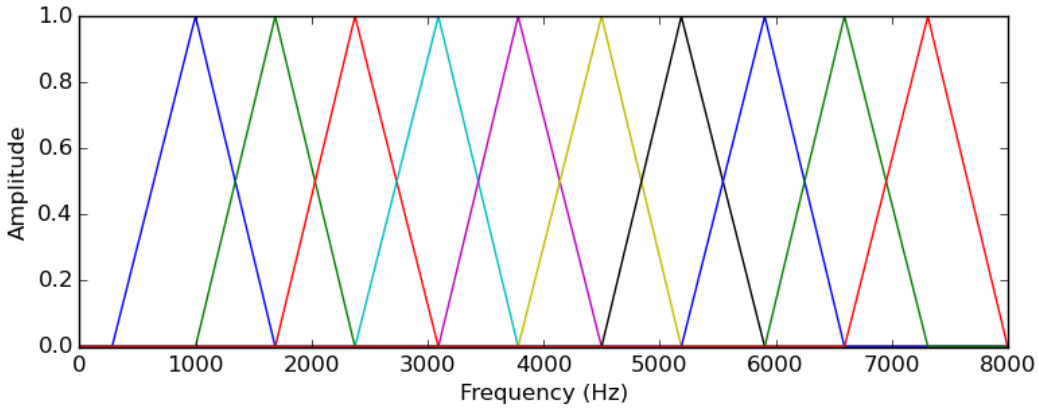


Figure 3.3: A filter bank of 10 filters used in LFCC

3.1.3 Linear predictive coding and linear predictive cepstral coefficients

The basic idea of linear predictive coding (linear predictive analysis) is that we can predict a speech sample by a linear combination of its previous samples [31]. A linear predictor of order p is defined as a system of which the output is:

$$\tilde{x}[n] = \sum_{k=1}^{p} \alpha_k x[n-k] \tag{3.12}$$

$\alpha_1, \alpha_2, ..., \alpha_p$ are called *prediction coefficients*, or *linear prediction coefficients* (LPCs). The prediction coefficients are determined by minimizing the sum of squared differences between the original signal and the predicted one. The prediction error is:

$$e[n] = x[n] - \tilde{x}[n] = x[n] - \sum_{k=1}^{p} \alpha_k x[n-k] \tag{3.13}$$

The *linear predictive cepstral coefficients* (LPCCs) can be computed directly from LPCs using a recursive formula [31]:

$$\sigma^2 = \sum_n e^2[n] \tag{3.14}$$

$$\hat{c}[n] = \begin{cases} 0 & n < 0 \\ \ln \sigma & n = 0 \\ \alpha_n + \sum_{k=1}^{n-1} \left(\frac{k}{n}\right) \hat{c}[k] a_{n-k} & 0 < n \leq p \\ \sum_{k=n-p}^{n-1} \left(\frac{k}{n}\right) \hat{c}[k] a_{n-k} & n > p \end{cases} \tag{3.15}$$

Linear predictive coding is a powerful technique, and is widely used in speech quantization. In speaker recognition, many studies have been conducted on the effectiveness of linear prediction methods. Dhonde and Jagade concluded that LPCs were good as features for speech recognition but not for speaker recognition [16], while according to Atal's study, LPCCs yielded the best accuracy in speaker identification among linear predictor derived parameter representations [3]. Besides MFCCs, LPCCs are one of the most commonly used features in speaker recognition.

3.2 Speaker Modeling Techniques

Given a set of feature vectors, we wish to build a model for each speaker so that a vector from the same speaker has higher probability belonging to that model than any other models. In general, any learning method can be used, but in this section we focus on the most basic approaches in text-independent speaker identification.

3.2.1 k-nearest neighbors

k-Nearest Neighbors (kNN) is a simple, nonparametric learning algorithm used in classification. Each training sample is represented as a vector with a label, and an unknown sample is normally classified into one or more groups according to the labels of the k closest vectors, or its neighbors. An early work using kNN in speaker identification used the following *distance* [26]:

$$d(U,R) = \frac{1}{|U|}\sum_{u_i \in U} \min_{r_j \in R} |u_i - r_j|^2 + \frac{1}{|R|}\sum_{r_j \in R} \min_{u_i \in U} |u_i - r_j|^2 - \frac{1}{|U|}\sum_{u_i \in U} \min_{u_j \in U, j \neq i} |u_i - u_j|^2 - \frac{1}{|R|}\sum_{r_i \in R} \min_{r_j \in R, j \neq i} |r_i - r_j|^2 \quad (3.16)$$

Despite its straightforward approach, classification using kNN is costly and ineffective due to these reasons [34]: (1) it has to store all training samples, thus a large storage is required; (2) all computations are performed in the testing phase; and (3) the case that two groups tie when making decision needs to be further investigated (because the system should classify a sample into only one class). Therefore, in order to apply this method effectively, one has to speed up the conventional approach, for example, using dimension reduction [34], or use kNN as a coarse classifier in combination with other methods [69].

3.2.2 Vector quantization and clustering algorithms

The idea of vector quantization (VQ) is to compress a set of data into a small set of representatives, which reduces the space to store data, but still maintains sufficient information. Therefore, VQ is widely applied in signal quantization, transmitting and speech recognition.

Given a k-dimension vector $a = (a_1, a_2, ..., a_k)^T \in R^k$, after VQ, a is assigned to a vector space S_j:

$$q(a) = S_j \quad (3.17)$$

with $q(\cdot)$ is the quantization operator. The whole vector space is $S = S_1 \cup S_2 \cup ... \cup S_M$, each partition S_j forms a non-overlapping region, and is characterized by its centroid vector z_j. Set $Z = \{z_1, z_2, ..., z_M\}$ is called a *codebook* and z_j is the j-th *codeword*. M is the size or the number of levels of the codebook. The error between a vector and a codeword $d(x, z)$ is called *distortion error*. A vector is always assigned to the region with the smallest distortion:

$$q(a) = S_j \iff j = \text{argmin}_{1 \leq j \leq M}\, d(a, z_j) \quad (3.18)$$

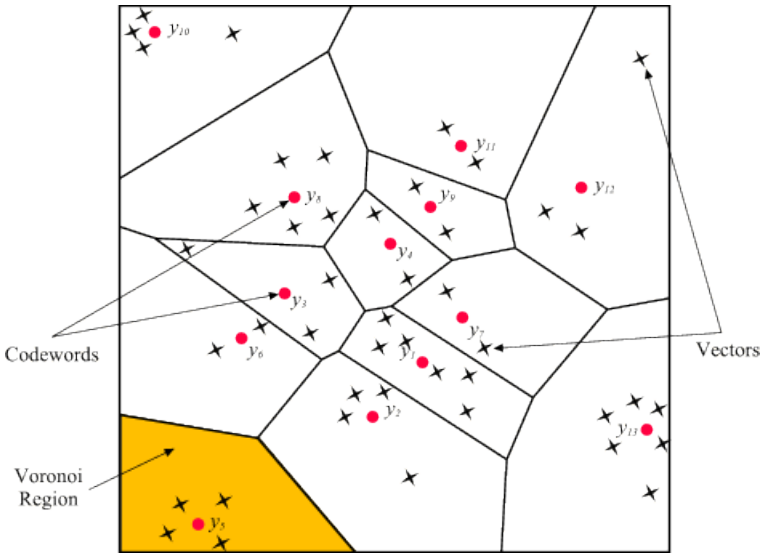


Figure 3.4: A codebook in 2 dimensions. Input vectors are marked with x symbols, codewords are marked with circles (adapted from [51]).

A set of vectors $\{x_1, x_2, ..., x_N\}$ is quantized to a codebook $Z = \{z_1, z_2, ..., z_M\}$ so that the average distortion:

$$D = \frac{1}{N} \sum_{i=1}^{N} \min_{1 \leq j \leq M} d(x_i, z_j) \tag{3.19}$$

is minimized over all input vectors. Figure 3.4 illustrates a codebook in 2 dimensional space. K-means and LBG (Linde-Buzo-Gray) are two popular techniques to design codebooks in VQ.

The K-means algorithm is described as follows [31]:

Step 1 Initialization. Generate M codewords using some random logic or assumptions about clusters.

Step 2 Nearest-neighbor classification. Classify each input vector x_i into region S_j according to equation 3.18.

Step 3 Codebook updating. Re-calculate a centroid using all vectors in that region:

$$\hat{z}_j = \frac{1}{N_j} \sum_{x \in S_j} x \tag{3.20}$$

N_j is the number of vectors in region S_j.

Step 4 Iteration. Repeat step 2 and 3 until the difference between the new distortion and the previous one is below a pre-defined threshold.

The LBG algorithm is a wrapper algorithm around unsupervised clustering techniques proposed in 1980 [41]. It is a hierarchical clustering algorithm, which first starts with a 1-level codebook, then uses a splitting method to obtain a 2-level codebook, and continues until a M-level codebook is acquired. The formal procedure of LBG is described as follows [31]:

Step 1 Initialization. Set $M = 1$. Find the centroid of all data according to equation 3.20.

Step 2 Splitting. Split M codeword into $2M$ codewords by splitting each vector z_j into two close vectors:

$$z_j^+ = z_j + \epsilon$$
$$z_j^- = z_j - \epsilon$$

Set $M = 2M$.

Step 3 Clustering. Using a clustering algorithm (e.g., K-means) to reach the best centroids for the new codebook.

Step 4 Termination. If M is the desired codebook size, stop. Otherwise, go to step 2.

In speaker identification, after preprocessing, all speech vectors of a speaker are used to build a M-level codebook of that speaker, resulting in L codebooks of L different speakers [41]. The average distortion with respect to codebook (or speaker) l of a test set $\{x_1, x_2, ..., x_N\}$ corresponding to an unknown speaker is:

$$D^l = \frac{1}{N} \sum_{i=1}^{N} \min_{1 \le j \le M} d(x_i, z_j^l) \tag{3.21}$$

N average distortions are then compared, and the speaker's ID is decided by the minimum distortion:

$$l^* = \text{argmin}_{1 \le l \le L} D^l \tag{3.22}$$

3.2.3 Hidden Markov model

In speech and speaker recognition, we always have to deal with a sequence of objects. Those sequences may be words, phonemes, or feature vectors. In those cases, not only the order of the sequence is important, but also its content. Hidden Markov models (HMMs) are powerful statistical techniques to characterize observed data of a time series.

A HMM is characterized by:

- N: the number of states in the model, the set of states $S = \{s_1, s_2, ..., s_N\}$.
- $A = \{a_{ij}\}$: the transition probability matrix, where a_{ij} is the probability of taking a transition from state s_i to state s_j:

  $$a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$$

  where $Q = \{q_1 q_2 ... q_L\}$ is the (unknown) sequence of states corresponding to the time series.
- $B = b_j(k)$: the observation probabilities, where $b_j(k)$ is the probability of emitting symbol o_k at state j. Let $X = \{X_1 X_2 ... X_L\}$ be a sequence, $b_j(k)$ can be defined as:

  $$b_j(k) = P(X_t = o_k \mid q_t = s_j)$$

- $\pi = \{\pi_i\}$: the initial state distribution where:

$$\pi_i = P(q_1 = s_i)$$

For convenience, we use the compact notation $\lambda = (A, B, \pi)$ as a parameter set of a HMM.

The observation probabilities B can be discrete or continuous. In case it is continuous, $b_j(k)$ can be assumed to follow any continuous distribution, for instance, Gaussian distribution $b_j(k) \sim \mathcal{N}(o_k; \mu_j, \Sigma_j)$, or a mixture of Gaussian components:

$$b_j(k) = \sum_{m=1}^{M} c_{jm} b_{jm}(k) \tag{3.23}$$

$$b_{jm}(k) \sim \mathcal{N}(o_k; \mu_{jm}, \Sigma_{jm}) \tag{3.24}$$

where M is the number of Gaussian mixtures, μ_{jm}, Σ_{jm} are the mean and covariance matrix of the m-th mixture, and c_{jm} is the weight coefficient of the m-th mixture. c_{jm} satisfies:

$$\sum_{m=1}^{M} c_{jm} = 1 \quad 1 \leq j \leq N \tag{3.25}$$

The probability density of each mixture component is:

$$b_{jm}(k) = \frac{1}{\sqrt{(2\pi)^R |\Sigma_{jm}|}} \exp\left[-\frac{1}{2}(o_k - \mu_{jm})^T \Sigma_{jm}^{-1} (o_k - \mu_{jm})\right] \tag{3.26}$$

where R is the dimensionality of the observation space.

There are 3 basic problems with regards to HMMs:

- Evaluation problem: Given a HMM $\lambda = (A, B, \pi)$ and an observation sequence $O = \{o_1 o_2 ... o_L\}$, find the probability that λ generates this sequence $P(O \mid \lambda)$. This problem can be solved by the *forward algorithm* [2, 15.5].

- Optimal state sequence problem: Given a HMM $\lambda = (A, B, \pi)$ and an observation sequence $O = \{o_1 o_2 ... o_L\}$, find the most likely state sequence $Q = \{q_1 q_2 ... q_L\}$ that generates this sequence, namely find Q^* that maximizes $P(Q \mid O, \lambda)$. This problem can be solved by the *Viterbi algorithm* [2, 15.6].

- Estimate problem: Given a training set of observation sequences $\mathcal{X} = \{O^k\}$, we want to learn the model parameters λ^* that maximize the probability of generating $\mathcal{X}$, $P(\mathcal{X} \mid \lambda)$. This problem is also known as the training process of HMMs, and is usually implemented using the *Baum-Welch algorithm* [2, 15.7].

HMMs provide an effective framework to model time sequences, hence they have become popular in speech technology. After their success in speech recognition, this technique was adapted as a text-dependent speaker identification framework. Feature vectors can be used directly with continuous HMMs [50, 64] or in combination with VQ (see section 3.2.2) [46, 1]. A HMM-based speaker

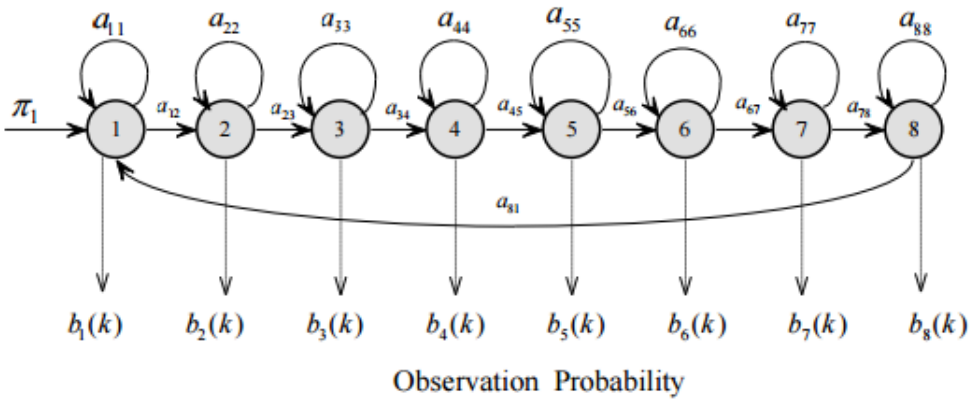


Figure 3.5: A left-to-right HMM model used in speaker identification (adapted from [1]).

identification system builds a HMM for each speaker, and the model that yields the highest probability for a testing sequence gives the final identification.

If using VQ, first a codebook corresponding with each speaker is generated. By using codebooks, the domain of observation probabilities becomes discrete, and the system can use discrete HMMs. However, in some cases, a codebook of a different speaker may be the nearest codebook to the testing sequence, thus the recognition is poor [46]. Continuous HMMs are able to solve this problem, and Matsui and Furui showed that continuous HMMs had much better results than discrete HMMs.

In speaker identification, the most common types of HMM structure are *ergodic* HMMs (i.e., HMMs that have full connection between states) and *left-to-right* HMMs (i.e., HMMs only allow transitions in the same direction, or transitions to the same state). A left-to-right HMM is illustrated in figure 3.5).

3.2.4 Gaussian mixture model: The baseline

Gaussian mixture models (GMMs) are generative approaches in speaker identification that provide a probabilistic model of a speaker's voice. However, unlike the HMM approach in section 3.2.3, it does not involve any Markov process. GMMs are one of the most effective techniques in speaker recognition, and are also considered the *baseline model* in this field.

A Gaussian mixture distribution is a weighted sum of M component densities:

$$p(\vec{x} \mid \lambda) = \sum_{i=1}^{M} p_i b_i(\vec{x}) \tag{3.27}$$

where $\vec{x}$ is a D-dimensional vector, $b_i(x)$ is the i-th component density, and p_i is the weight of the i-th component. The mixture weights satisfy:

$$\sum_{i=1}^{M} p_i = 1$$

Each mixture component is a D-variate Gaussian density function:

$$b_i(\vec{x}) = \frac{1}{\sqrt{(2\pi)^{D/2}\left|\Sigma_i\right|^{1/2}}} \exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu}_i)^T\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i)\right] \tag{3.28}$$

μ_i is the mean vector, and Σ_i is the covariance matrix.

A GMM is characterized by the mean vector, covariance matrix and weight from all components. Thus, we represent it by a compact notation:

$$\lambda = (p_i, \vec{\mu}_i, \Sigma_i) \quad i = 1, 2, ..., M \tag{3.29}$$

In speaker identification, each speaker is characterized by a GMM with its parameters λ. There are many different choices of covariance matrices [56], for example, the model may use one covariance matrix per component, one covariance matrix for all components or one covariance matrix for components in a speaker model. The shape of covariance matrices can be full or diagonal.

Given a set of training samples X, probably, the most popular method to train a GMM is maximum likelihood (ML) estimation. The likelihood of a GMM is:

$$p(X \mid \lambda) = \prod_{t=1}^{T} p(\vec{x}_t \mid \lambda) \tag{3.30}$$

ML parameters are normally estimated using the *expectation maximization* (EM) algorithm [56].

Among a set of speakers characterized by parameters $\lambda_1, \lambda_2, ..., \lambda_n$, a GMM system makes its prediction by returning the speaker that maximizes the *a posteriori* probability given an utterance X:

$$\hat{s} = \text{argmax}_{1\leq k\leq n}\, P(X \mid \lambda_k) = \frac{P(X \mid \lambda_k)P(\lambda_k)}{P(X)} \tag{3.31}$$

If prior probabilities of all speakers are equal, e.g. $P(\lambda_k) = 1/n\ \forall k$, since $P(X)$ is the same for all speakers and logarithm is monotonic, we can rewrite equation 3.31 as:

$$\hat{s} = \text{argmax}_{1\leq k\leq n} \log P(X \mid \lambda_k) \tag{3.32}$$

$$= \text{argmax}_{1\leq k\leq n} \sum_{t=1}^{T} \log p(\vec{x}_t \mid \lambda_k) \tag{3.33}$$

Despite of their power, GMMs still face some disadvantages [66]. Firstly, GMMs have a large number of parameters to train. This fact not only leads to expensive computation, but also requires a sufficient amount of training data. Therefore, the performance of a GMM is unreliable if it is trained on a small dataset. Secondly, as a generative model, GMMs do not work well with unseen data, which easily yield low likelihood scores. Fortunately, these two problems can be overcome by *speaker adaptation*. The main idea of speaker adaptation is building speaker-dependent systems by *adapting* (i.e. modifying) a speaker-independent system constructed using all speaker data. A GMM trained on all speaker identities is the *universal background model* (UBM), of which concept was discussed in section 1.2. The GMM-UBM is then modified into a speaker's model using maximum a posteriori (MAP) adaptation [57].

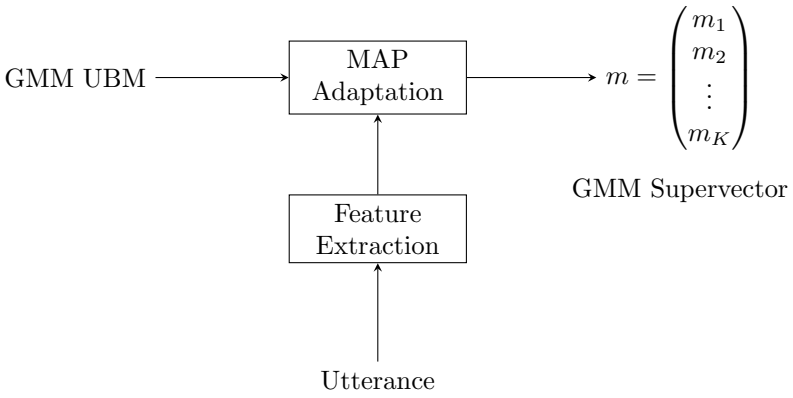


Figure 3.6: Computing GMM supervector of an utterance

3.3 I-Vector: The State-of-the-Art

Given an adapted GMM, by stacking all means of its components, we have a vector called GMM *supervector*. Thus, we can easily obtain a GMM supervector of a speaker through speaker adaptation, as well as a GMM supervector of an arbitrary utterance by adapting a single utterance only. The process of calculating a GMM supervector of an utterance is illustrated in figure 3.6.

In Joint Factor Analysis (JFA) [35], the supervector of a speaker is decomposed into the form:

$$s = m + Vy + Dz \tag{3.34}$$

where m is the speaker-and-channel independent supervector, which is normally generated from the UBM. V and D are *factor loading matrices*, y and z are *common speaker factors* and *special speaker factors* respectively which follow a standard normal density. V represents the speaker subspace, while Dz serves as a residual. The supervector of an utterance is assumed to be synthesized from s:

$$M = s + Ux \tag{3.35}$$

where U is a factor loading matrix that defines a *channel subspace*, x are *common channel factors* having standard normal distributions. In summary:

$$M = m + Ux + Vy + Dz \tag{3.36}$$

In [13], based on an experiment showing that JFA channel factors also contained speaker information, a new single subspace was defined to model both channel and speaker variabilities. The new space was referred as *total variability space*, and the new speaker-and-channel dependent supervector was defined as:

$$M = m + Tw \tag{3.37}$$

T is a low rank rectangular matrix, and w is a random vector with standard normal distribution. The new type of vectors were referred as identity vectors or *i-vectors*. Extracted i-vectors can be used as features for other classification back-end such as support vector machines, or to be used directly using cosine kernel scoring:

$$\text{score}(w_{target}, w_{test}) = \frac{\langle w_{target}, w_{test} \rangle}{\|w_{target}\| \|w_{test}\|} \tag{3.38}$$

The i-vector technique is considered to be an effective way to reduce from high dimensional input data to low dimensional feature vectors. Today, i-vector systems have become the state-of-the-art in speaker recognition [33, 45].

Chapter 4

Deep Neural Networks

It has been more than 70 years since Warren McCulloch and Walter Pitts modeled the first artificial neural network (ANN) that mimicked the way brains work. These day, ANNs have become one of the most powerful tools in machine learning, and their effectiveness have been tested empirically in many real world applications. In combination with the *deep learning* paradigm, ANNs have achieved state-of-the-art results in plenty of areas, especially in natural language processing and speech technology (see [60] for more details).

This chapter serves as reference for ideas and techniques we use directly in our speaker identification systems. First, an overview of ANNs and deep learning is presented, then we review some available applications of ANNs in speaker identification.

4.1 Artifical Neural Networks at a Glance

The concept of ANNs was inspired by the biological nature of the human brain. The brain consists of interconnected cells called *neurons*, which transmit information to each other using electrical and chemical signals. The lines that connect neurons together are called *axons*. If the sum of signals at one neuron is sufficient to *activate* itself, the neuron will transmit this signal along axons to other neurons attached at the other end of axons. In fact, the brain contains about 10^{11} neurons, each connects on average to 10,000 others. The fastest switching time of nerons is 10^{-3} seconds, which is much slower than that of a computer: 10^{-10} seconds [47]. However, in reality, humans are able to make complex decisions such as face detection or speech recognition in surprisingly effective ways.

ANN models are based closely on the biological neural system. In ANNs, the basic processing unit is a *perceptron* (figure 4.1). The inputs of a perceptron may come from the environment, or from other perceptrons' outputs. Each input is associated with a *weight*; therefore, a perceptron combines its input as a weighted sum plus a *bias*. The strength of aggregation is then modified by an *activation function*, yielding the final output of the perceptron. Let x be the input vector, w be the corresponding weight vector, b be the bias and φ be the activation function. The output of a perceptron is formulated as:

$$y = \varphi(w \cdot x + b) \tag{4.1}$$

Common activation functions are *sigmoid*, *tanh* and *rectified linear* (ReL).

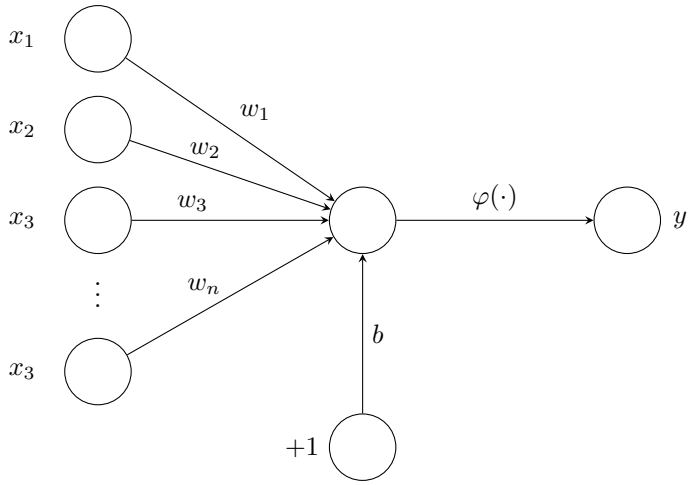


Figure 4.1: A perceptron

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

Tanh

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.3}$$

ReL

$$f(x) = \max(0, x) \tag{4.4}$$

The visual representation of a perceptron is a hyperplane in n-dimensional space, since its ouput is a linear combination of inputs. Thus, a single perceptron is not very interesting. Now let us organize perceptrons into a layer, and cascade these layers into a network. We shall give one more restriction, that is connections between layers follow only one direction. The type of ANNs that we have just defined is called a *feedforward neural network* (FNN), or *multilayer perceptron* (MLP). The layer that receives connections from inputs is the *input layer*, the outermost layer is the *output layer*, and the rest of the layers between the input and output layers are called *hidden layers*. Figure 4.2 illustrates a MLP with three layers. The computation of a MLP can be defined by the following formula:

$$h^{(l)} = \varphi^{(l)}(W^{(l)} \cdot h^{(l-1)} + b^{(l)}) \tag{4.5}$$

where $h^{(l)}$ is the output vector of layer l, $l = 1...L$ where L is the number of layers in the network. $h^{(0)}$ is the input of the network. $W^{(l)}$, $b^{(l)}$ and $\varphi^{(l)}$ in turn are the weight matrix, the bias vector and the activation function of layer l.

The role of activation functions in MLPs is very important, because they give MLPs the ability to compute nonlinear function: if outputs of hidden layers were linear, the network output would be just a linear combination of inputs, which is not very useful. In regression, the activation function used in the output layer is usually linear, while in classification of K classes, it could be a sigmoid ($K = 2$) or *softmax* ($K > 2$) function. Choosing activation functions for hidden layers will be discussed further in section 4.5.

Given a set of samples $\{(x^{(1)}, y^{(1)}), ..., (x^{(M)}, y^{(M)})\}$ and a MLP with initial parameters θ (characterized by weight matrices and bias vectors), we would like to train the MLP so that it can learn the mapping given in our set. If we see the whole network as a function:

$$\hat{y} = F(x; \theta) \tag{4.6}$$

and define some *loss function* $E(x, y, \theta)$, then the goal of training our network becomes minimizing $E(x, y, \theta)$. Luckily, the gradient of E tells us the direction to go in order to increase E:

$$\nabla E(\theta) = \left[\frac{\partial E}{\partial \theta_1}, ..., \frac{\partial E}{\partial \theta_n}\right] \tag{4.7}$$

Since the gradient of E specifies the direction to increase E, at each step parameters will be updated proportionally to the negative of the gradient:

$$\theta_i \leftarrow \theta_i + \Delta\theta_i \tag{4.8}$$

where:

$$\Delta\theta_i = -\eta \frac{\partial E}{\partial \theta_i} \tag{4.9}$$

The training procedure is *gradient descent*, and η is a small positive training parameter called *learning rate*.

In our systems, we employ two types of loss functions:

Mean squared error

$$E = \frac{1}{K} \sum_{k=1}^{K} (y_k^{(m)} - \hat{y}_k^{(m)})^2 \tag{4.10}$$

Cross entropy error

$$E = -\frac{1}{K} \sum_{k=1}^{K} y_k^{(m)} \log(\hat{y}_k^{(m)}) \tag{4.11}$$

where m is the index of an arbitrary sample, K is the number of classes. $y_k^{(m)}$ represents the k-th column (corresponding to the probability of class k) of vector $y^{(m)}$.

In conventional systems, the gradient components of the output layer can be computed directly, while they are harder to compute in lower layers. Normally, the current gradient is calculated using the error of the previous step. Since errors are calculated in the reverse direction, this algorithm is known as *backpropagation*.

4.2 Deep Learning and Deep Neural Networks

Until the 1980s, the only applicable structure of ANNs is a shallow structure, which is an ANN with a few hidden layers. The *universal approximation theorem* [11, 30], which states that any function can be approximated by an ANN with three layers with arbitrary accuracy, makes additional hidden layers become unnecessary. Moreover, the backpropagation algorithm did not work well with deep FNNs (see section 4.5), and the computation ability back then was also limited.

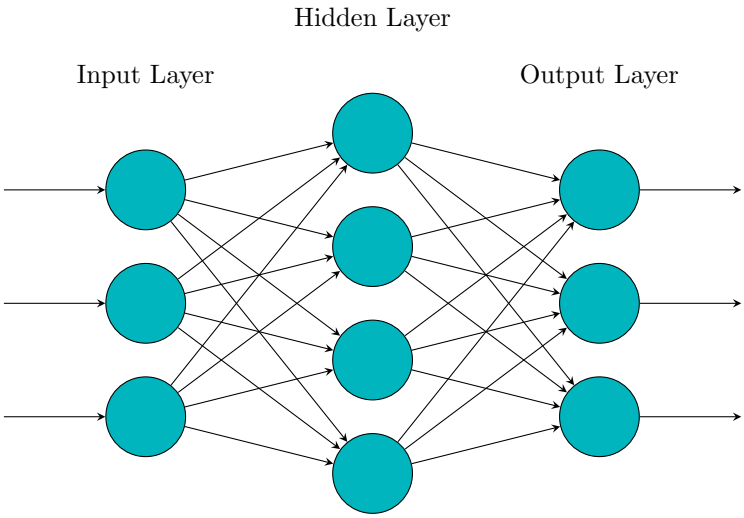


Figure 4.2: A feedforward neural network with one hidden layer

However, the deep structure in human information processing mechanisms suggests the necessity and effectiveness of deep learning algorithms. In 2006, Hinton et al. introduced the deep belief network, a deep neural network (DNN) model composed of Restricted Boltzmann Machines [28]. A deep belief network was trained in unsupervised fashion, one layer at a time from the lowest to the highest layer [28]. Deep feed-forward networks were effectively trained using the same idea by first pre-training each layer as a Restricted Boltzmann Machine, then fine-tuning by backpropagation [27]. Later, deep belief networks achieved low error rates in MNIST handwritten digits, and good results in TIMIT phone recognition [60]. Today, ANNs with deep structures are trained on powerful GPU machines, overcoming both resources and time limits.

Although the history of deep learning originates from ANNs, the term "deep learning" has broader interpretation. There are many definitions of deep learning, but they both mention two key aspects [15]:

1. "models consisting of multiple layers or stages of nonlinear information processing"; and

2. "methods for supervised or unsupervised learning of feature representation at successively higher, more abstract layers"

4.3 Recurrent Neural Networks

A recurrent neural network (RNN) is a model of ANNs used to deal with sequences. It is similar to an ANN except that it allows a self-connected hidden layer that associates with a time delay. Weights of the recurrent layer are shared across time. If we unfold a RNN in time, it becomes a DNN with a layer for each time step.

There are many models of RNNs, but let us consider a simple RNN invented by Elman [17]. The proposed RNN has just three layers, and the hidden layer is self-connected (figure 4.3). The RNN is parameterized by weight matrices

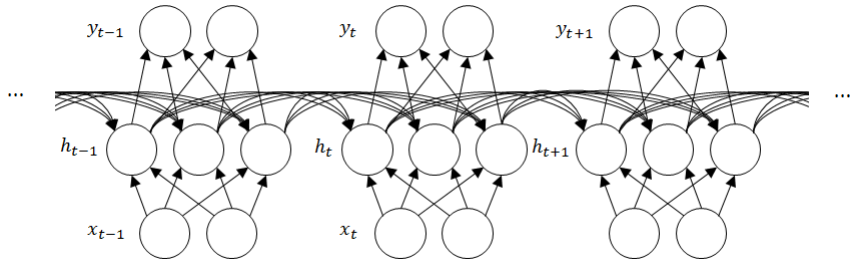


Figure 4.3: A simple recurrent neural network

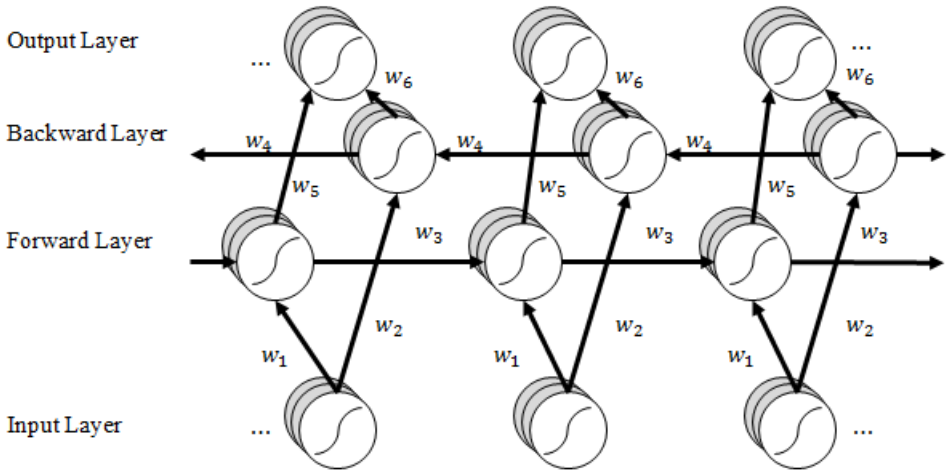


Figure 4.4: A bidirectional recurrent neural network unfolded in time

and bias vectors $[W_{in}, W_h, W_{out}, b_{in}, b_{out}]$. Given input sequence $x_1, x_2, ..., x_T$, the output of the RNN is computed as:

$$h_t = \varphi_z(W_{in} \cdot x_t + W_h \cdot h_{t-1} + b_{in}) \tag{4.12}$$

$$\hat{y}_t = \varphi_o(W_{out} \cdot h_t + b_{out}) \tag{4.13}$$

The simple RNN model is elegant, yet it only captures temporal relations in one direction. Bidirectional RNNs [61] were proposed to overcome this limitation. Instead of using two separate networks for the forward and backward directions, bidirectional RNNs split the old recurrent layer into two distinct layers, one for the positive time direction (forward layer) and one for the negative time direction (backward layer). The output of forward states are not connected to backward states and the other way around (figure 4.4).

4.4 Convolutional Neural Networks

A convolutional neural network (CNN) is similar to an ordinary FNN as the ouput of each layer is a combination of the input, the weight matrix and the bias vector followed by a non-linear transformation. However, what make a CNN different is its *local connectivity*. Rather than having a full connection between a layer and its input, a CNN uses some small filters, slides it across all sub-regions of the input matrix and aggregates all results. In other words, it takes advantages of

the *convolution* operation (see section 2.4.1) between the filters and the input. The inspiration of CNNs is said to be based on the *receptive field* of a neuron, i.e. sub-regions of the visual field that the neuron is sensitive to.

There are several types of layers that make up a CNN:

Convolutional layer A convolutional layer consists of K filters. In general, its input has one or more *feature maps*, e.g., a RGB image has 3 channels red, green and blue. Therefore, the input is a 3-dimensional matrix and its feature maps is considered the depth dimension. Each filter need to have 3-dimensional shape as well with its depth extend to the entire depth of the input (see figure 4.5). The output of the layer is K feature maps, each one is computed as the convolution of the input and a filter k, plus its bias:

$$h_{ijk} = \varphi((W_k * x)_{ij} + b_k) \tag{4.14}$$

where i and j are the row index and the column index, φ is the activation function of the layer and x is its input. Thus the output of a convolutional layer is also a 3-dimensional matrix, and its depth is defined by the number of filters.

Pooling layer A pooling layer is usually inserted between two successive convolutional layers in a CNN. It downsamples the input matrix, thus reducing the space of representation and the number of parameters. The depth dimension remains the same. A pooling layer divides the input into (usually) non-overlapping rectangle regions, of which size defined by the pool shape. Then, it outputs the value of each region using the max, sum or average operator. If a pooling layer uses the max operator, it is called a *max pooling layer*. The pool size is normally set as $(2, 2)$ as larger sizes may lost too much information.

Fully-connected layer One or more fully-connected layers may be placed at the end of a CNN, to refine features learned from convolutional layers, or to return class scores in classification.

The most common architecture of CNNs stacks convoltional layers and pool layers in turn, then ends with fully-connected layers (e.g., LeNet [38]). It is worth considering that a convolutional layer can be substituted by a fully-connected layer of which weight matrix is mostly zero except at some blocks, and the weight of those blocks are equal.

4.5 Difficulties in Training Deep Neural Networks

The reason that the gradient descent algorithm did not work well with DNNs was not fully understood until Hochreiter's diploma thesis in 1991 [29]. In his work, he presented that DNNs suffered from widely known issues by now: the *vanishing* and *exploding gradient*; i.e. in DNNs using the backpropagation algorithm to spread errors, gradients either shrink to zero and disappear, or grow rapidly. In other words, lower layer gradients in DNNs are unstable, they tend to learn with much slower speed, which makes DNNs hard to train.

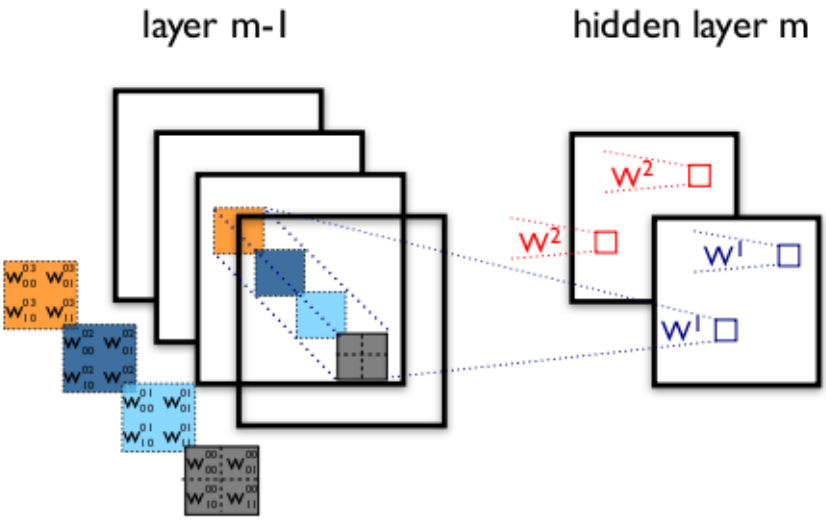


Figure 4.5: An illustration of 3-dimensional convolution (adapted from [38])

The vanishing gradient mainly occurs due to the calculation of local gradients. In the backpropagation algorithm, a local gradient is the aggregate sum of the previous gradients and weights, multiplied by its derivative. Since parameters are usually initialized as small values, their gradients are less than 1; therefore gradients of lower layers are smaller than those of above layers and are easier to reduce to zero. The exploding gradient, on the other hand, normally happens in neural networks with long time dependencies, for instance RNNs, since a large number of components to compute local gradients are prone to explode. In practice, some factors affect the influence of vanishing and exploding gradient problem, which includes the choice of activation functions, the cost function and network initialization [22].

A closer look to the role of activation functions can give us an intuitive understanding of these problems. A sigmoid is a monotonic function that maps its inputs to range $[0, 1]$ (figure 4.6). It was believed to be popular in the past because of the biological inspiration that neurons also followed a sigmoid activation function. A sigmoid function saturates at both tails, at which values remain mostly constant. Thus, gradients at those points is zero, and this phenomenon will be propagated to lower layers, which makes the network hardly learn anything. In consequence, we should pay attention at the initialization phase so that weights are small enough not to fall into saturated regions.

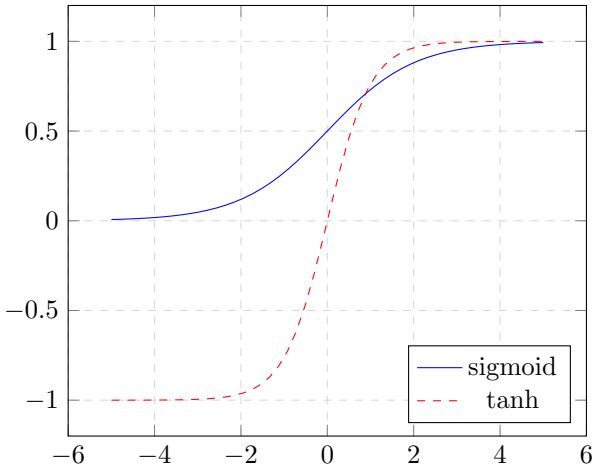


Figure 4.6: Sigmoid and tanh function

The tanh function also has a S-shape like sigmoid, except that it ranges from -1 to 1 instead of 0 to 1. Its characteristics are also the same, but tanh is empirically recommended over sigmoid because it is *zero-centered*. According to LeCun et al., weights should be normalized around 0 to avoid ineffective zigzag updates, which leads to slow convergence [39].

In three types of activation functions, ReL has the cheapest computation and does not suffer from the vanishing gradient along activation units. Many researches reported that ReL improved DNNs in comparison with other activation functions [42]. However, ReL could have problems with 0-gradient case, where a unit never activates during training. This issue may be alleviated by introducing a *leaky* version of ReL:

$$f(x) = \begin{cases} x & x > 0 \\ 0.01x & \text{otherwise} \end{cases} \tag{4.15}$$

A unit with ReL as activation function is called a *rectifier linear unit* (ReLu).

4.6 Neural Network in Speaker Recognition

There are generally two ways to use ANNs in speaker recognition tasks: either as a classifier or a feature extractor. The first usage is referred as a *direct, model-based* method, while the second one is known as an *indirect, feature-based* method.

ANNs has been used to classify speakers since the 90s [59, 19]. However, due to computation limits, neural networks were used as one-vs-all classifiers [19] or pairwise classifiers [59] rather than one large network for all speakers. ANN structures in those days had only one hidden layer for reasons discussed in section 4.2. With one-vs-all classifiers, there are N classifiers to identify N speakers. Each ANN is trained with the corresponding speaker data and anti-speaker data. The target speaker is decided corresponding to the ANN with the highest output probability. On the other hand, with pairwise classifiers, there are $N(N+1)/2$ classifiers. A sample may be passed through all ANNs, and their output are combined by voting. The alternative way is to organize classifiers as in a binary search tree, so that the expected number of comparisons is $O(\log_2 N)$.

In 1998, Konig et al. used features extracted from a *bottleneck* layer of a 5-layer MLP in speaker verification [36]. A bottleneck layer is a layer before the output layer of a neural network that has a reduced number of hidden nodes. This type of features is referred later as *bottleneck features*. In [36], bottleneck features alone performed worse than cepstrum, but reduced error in combination with it. Bottleneck features can be used as input to extract i-vector (section 3.3) as in [58], where bottleneck features and GMM posteriors made the best combination in speaker verification on DAC13 corpora.

Chapter 5

Experiments and Results

In this chapter, our approach to speaker identification is discussed. *Close-set speaker identification* is chosen as the task to assess the efficiency of our systems. The first section reviews available corpora that have been used for evaluation of this task and results of different systems on those data. After that, our choice of database, TIMIT, and the reference systems are introduced. Details about our approach is given next, and finally experiments and their results are presented.

5.1 Corpora for Speaker Identification Evaluation

In the history of speaker recognition, public speech corpora play an important role in research development and evaluation, which allows researchers to compare the performance of different techniques. TIMIT, Switchboard and KING are some of the most commonly used databases in speaker identification. However, since they were not specifically designed for speaker identification, their usages varied among researches, making different evaluation conditions.

5.1.1 TIMIT and its derivatives

The TIMIT database [71] was developed to study phoneme realization and for training and evaluating speech recognition systems. It contains 630 speakers of 8 major dialects of American English; each speaker read 10 different sentences of approximately 3 seconds. However, TIMIT is considered a near-ideal condition since its records were obtained in a single session in a sound booth [54]. Another derivative of TIMIT is NTIMIT, which was collected by playing TIMIT original speeches through an artificial mouth, then recording using a carbon-button telephone handset and transferring via long distance telephone lines [32].

Many systems were evaluated on TIMIT and NTIMIT databases, either on complete databases or their subsets. For instance, Reynolds reported the accuracy of GMM speaker identification system as a function of the population size [54] with accuracy almost 100% in every case of TIMIT original 16 kHz data, while Farrell et al. compared the performance of different classification methods (VQ, kNN, MLP, ...) on a subset of TIMIT of 38 speakers of New England dialect, downsampled to 8 kHz [19] (see table 5.1). The ratio of training to testing data is not identical in these two reports.

Speaker model	5 speakers	10 speakers	20 speakers
FVSQ (128)	100%	98%	96%
TSVQ (64)	100%	94%	88%
MNTN (7 levels)	96%	98%	96%
MLP (16)	96%	90%	90%
ID3	86%	88%	79%
CART	80%	76%	-
C4	92%	84%	73%
BAYES	92%	92%	83%

Table 5.1: Speaker identification accuracy of different algorithms on various sizes of speaker population (reproduced from [19]). Data were selected from 38 speakers of New England subset of TIMIT corpus. FSVQ (128): full-search VQ with codebook size of 128; TSVQ (64): tree-structured VQ with cookbook size of 64; MNTN (7 levels): modified neural tree network pruned to 7 levels; ID3, CART, C4, BAYES: different decision tree algorithms.

Speaker model	60 second	30 second	10 second
GMM [56]	95%	-	94%
kNN [26]	96%	-	-
Robust Segmental Method [21]	100% (Top40Seg)	99% (Top20Seg)	99% (TopSeg2to7)

Table 5.2: Speaker identification accuracy of different algorithms on the SWBDTEST subset of Switchboard corpus

5.1.2 Switchboard

The Switchboard corpus is one of the largest public collections of telephone conversations. It contains data recorded in multiple sessions using different handsets. Conversations were automatically collected under computer supervision [23]. There are two Switchboard corpora, Switchboard-I and Switchboard-II. Switchboard-I has about 2400 two-sided conversations from 534 participants in the United States.

Due to its hugeness, many researchers wanted to evaluate their systems on a part of the Switchboard corpus. An important subset of Switchboard-I is SPIDRE, *SPeaker IDentification REsarch*, which was specially planned for close or open-set speaker identification and verification. SPIDRE includes 45 target speakers, 4 conversations per target and 100 calls from non-targets.

Gish and Schmidt achieved identification accuracy of 92% on the SPIDRE 30 second test using robust scoring algorithms [21]. Besides, some systems were tested on a subset of 24 speakers of Switchboard (which was referred as SWBDTEST in [21]), with accuracy higher than 90% [54, 21, 26] (table 5.2).

5.1.3 KING corpus

The KING corpus was designed for closed-set speaker identification and verification experiments. It contains 51 male speakers divided into two groups (25 and 26 speakers), each group was recorded at different locations. Each speaker has

Speaker model	Accuracy (5 second test) (%)
GMM-nv	94.5 ± 1.8
VQ-100	92.9 ± 2.0
GMM-gv	89.5 ± 2.4
VQ-50	90.7 ± 2.3
RBF	87.2 ± 2.6
TGMM	80.1 ± 3.1
GC	67.1 ± 3.7

Table 5.3: Speaker identification accuracy of different algorithms on a subset of King corpus (reproduced from [56]). VQ-50 and VQ-100: VQ with codebook size of 50 and 100; GMM-nv: GMM with nodal variances; GMM-gv: GMM with a single grand variance per model; RBF: radial basis function networks; TGMM: tied GMM; GC: Gaussian classifier.

Dialect	No.	#Male		#Female		Total	
New England	1	31	(63%)	18	(27%)	49	(8%)
Northern	2	71	(70%)	31	(30%)	102	(16%)
North Midland	3	79	(67%)	23	(23%)	102	(16%)
South Midland	4	69	(69%)	31	(31%)	100	(16%)
Southern	5	62	(63%)	36	(37%)	98	(16%)
New York City	6	30	(65%)	16	(35%)	46	(7%)
Western	7	74	(74%)	26	(26%)	100	(16%)
Army Brat	8	22	(67%)	11	(33%)	33	(5%)
Total	8	438	(70%)	192	(30%)	630	(100%)

Table 5.4: TIMIT distribution of speakers over dialects (reproduced from [71])

10 conversations corresponding to 10 sessions. There are two different versions of data: the telephone handset version and the high quality microphone one. Reynolds and Rose used a KING corpus subset of 16 speakers in telephone line to compare the accuracy of GMM to other speaker models [56]. The first three sessions were used as training data, and testing data was extracted from session four and five. Performance was compared using 5 second tests. Results of those models are summarized in table 5.3.

5.2 Database Overview

Although TIMIT does not represent the real speaker recognition condition, we decided to evaluate our systems on it since TIMIT is the only database we possess at the moment, which has been widely used for speaker identification evaluation. After a brief review in section 5.1.1, this section provides more details about the sentence distribution in the TIMIT corpus.

TIMIT contains 6300 sentences spoken by 630 speakers, which were divided into a training set and a test set for speech recognition evaluation. Selected speakers came from 8 major dialect regions of the United States, and the distribution of speakers in different dialects as well as the gender ratio was unbalanced (table 5.4).

Sentence type	#Sentences	#Speaker/ sentence	Total	#Sentence/ speaker
Dialect (SA)	2	630	1260	2
Compact (SX)	450	7	3150	5
Diverse (SI)	1890	1	1890	3
Total	2342		6300	10

Table 5.5: The distribution of speech materials in TIMIT (reproduced from [71])

There are three types of sentences in the corpus:

SA sentences The dialect sentences designed at SRI. There are 2 sentences of this type, and every speaker read both of these sentences.

SX sentences The phonetically-compact sentences designed at MIT. Each speaker read 5 of these sentences, and each sentence was recorded by 7 different people.

SI sentences The phonetically-diverse sentences selected from the Brown corpus and the Playwrights Dialog. Each speaker read 3 of these sentences, and each sentence was read by only one speaker.

Table 5.5 summarizes the distribution of sentences to speakers. Because of the composition of TIMIT, different division of data into training set and test set should affect performance of testing systems. Let 10 sentences of one speaker in TIMIT be named SA_{1-2}, SI_{1-3} and SX_{1-5}, where SA, SI and SX are sentence types, and index n of each sentence indicates the relative order within all sentences spoken by one person. To strictly make TIMIT text-independent, in [54] the last two SX sentences were used as test data and the remaining were training data, while in [37], SA_{1-2}, SI_{1-2} and SX_{1-2} were used for training, SI_3 and SX_3 were used for validation, and SX_{4-5} were used for testing.

5.3 Reference Systems

While some researches achieved almost perfect accuracy on the TIMIT database (99.5% on all 630 speakers [54] and 100% on a subet of 162 speakers [37]), original data are not very suitable for investigating the capability of our systems. Instead, we downsampled data from 16 to 8 kHz (TIMIT-8k) and chose approaches described in [19] as our references systems.

Close-set speaker identification in [19] was performed on population size of 5, 10 and 20. Speakers were selected from a subset of 38 speakers of New England dialect of TIMIT (38 speakers in dialect region 1 in the training set). All data were downsampled to 8 kHz, and 5 sentences were chosen randomly and concenated to use as training data. The remaining 5 sentences were used separately as test data. As a result, the duration of training data of each speaker ranged from 7 to 13 seconds, and each test lasted 0.7 to 3.2 seconds.

After removing silence and pre-emphasizing, speech data were processed using a 30 ms Hamming window applied every 10 ms. Then 12th-order linear predictive coding (section 3.1.3) was performed for each frame, and 12 LPCCs extracted

from that were used as features. Several techniques were compared in the speaker identification task, including:

Full-search VQ VQ technique described in section 3.2.2

Tree-structured VQ VQ technique except that codebooks are organized in a tree structure which is efficient for searching the closest one in the identification phase. Note that the searching algorithm is non-optimal.

MLP A MLP with one hidden layer is constructed for each speaker. The input of the MLP is a feature vector, and the output is the label of that vector, as 1 if it is from the same speaker of the MLP, and 0 otherwise. In the identification phase, all test vectors of an utterance are passed through each MLP, and the outputs of each MLP are accumulated. The speaker is decided as the corresponding MLP with the highest accumulated output.

Decision tree All training data are used to train a binary decision tree for each speaker with identical input and output manner as in MLP method. The probability of classification using decision trees is used to determine the target speaker. Pruning is applied after training to avoid overfitting. Various decision tree algorithms were considered, including C4, ID3, CART and a Bayesian decision tree.

Neural tree network A neural tree network has a tree structure as in decision trees, but each non-leaf node is a single layer of perceptrons. In the enrollment phase, the single layer perceptron at each node is trained to classify data into subsets. The architect of neural tree networks is determined during training rather than pre-defined as in MLPs.

Modified neural tree network A modified neural tree network is different from a neural tree network as it uses the confidence measure at each leaf besides class labels. Confidence measure helps to improve significantly in pruning in comparison to neural tree networks [19].

The best performance of each method is summarized in table 5.1.

5.4 Experimental Framework Description

In this project, we would like to investigate the efficiency of DNN, or more specifically, RNN (see section 4.3) in text-independent speaker identification. Our model was inspired by the RNN model proposed by Hannun et al., which outperformed state-of-the-art systems in speech recognition [25]. As in general speaker identification systems, we divided our framework into two main components: a front-end which transform a speech signal into features, and a back-end as a speaker classifier.

5.4.1 Preprocessing

The original data of TIMIT are in Sphere format, so first they need to be converted into WAV format before using. Because each file in TIMIT is a clean single

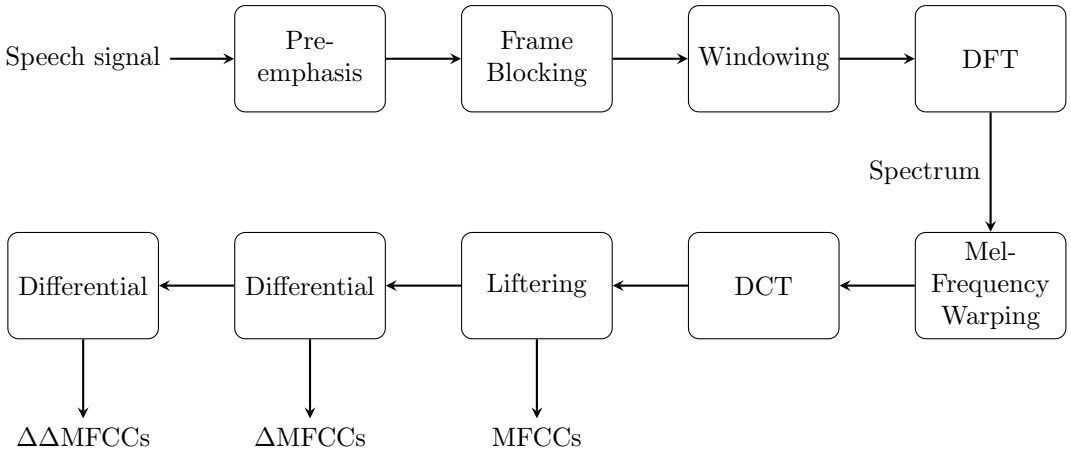


Figure 5.1: The process to convert speech signals into MFCC and its derivatives

sentence, silence is negligible. Therefore, voice activity detection is omitted since it may remove low-energy speech sound and lead to decrease in the performance [37]. We do not use chanel equalization either for the same reason [54].

5.4.2 Front-end

We employed two different types of features in our framework: MFCCs (section 3.1.1) and LFCCs (section 3.1.2). The computation of MFCCs is described in figure 5.1. LFCCs are acquired by the same process as MFCCs except that they are warped by a linear frequency band rather than mel-frequency warping. Details of each step are:

Pre-emphasis Pre-emphasis refers to the process of increasing the magnitude of higher frequencies with respect to that of lower frequencies. Since speech sound contains more energy in low frequencies, it helps to flatten the signal and to remove some glottal effects from the vocal tract parameters. On the other hand, pre-emphasis may increase noise in the high frequency range. Perhaps one of the most frequently used form of pre-emphasis is the first-order differentiator (single-zero filter):

$$\tilde{x}[n] = x[n] - \alpha x[n-1] \tag{5.1}$$

where α is usually ranged from 0.95 to 0.97. In our framework, we use $\alpha = 0.97$.

Frame Blocking As we use a short-time analysis technique to process speech (section 2.4), in this step, the speech signal is blocked into frames, each frame contains N samples and advances M samples from its previous frame ($M < N$). As a result, adjacent frames overlap $N - M$ samples. The signal is processed until all samples are in one or more frames, and the last frame is padded with 0 to have the length of exact N samples. Typically, N ranges from 20 to 30 ms, and M is about half of N.

Windowing As frame blocking breaks the continuity at the beginning and the end of each frame, they are multiplied by a window function to reduce differences, providing smooth transitions between frames. A window function

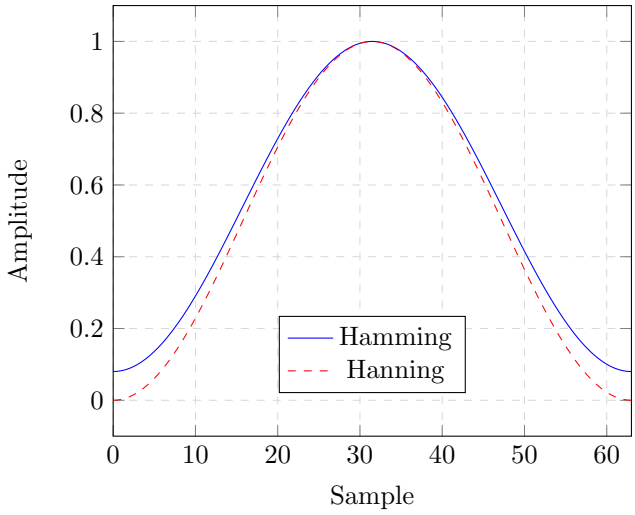


Figure 5.2: Hamming and Hanning windows of length 64

is defined as a mathematical function that is zero outside a specific region (section 2.4) and its simplest form is a rectangular window:

$$w[n] = \begin{cases} 1 & 0 \leq n \leq N-1 \\ 0 & \text{otherwises} \end{cases} \tag{5.2}$$

where N is the length of the window. However, the rectangular window does not have the effect to cancel boundary differences. Instead, bell-shaped windows are more preferred, such as Hamming windows:

$$w[n] = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) & 0 \leq n \leq N-1 \\ 0 & \text{otherwises} \end{cases} \tag{5.3}$$

or Hanning windows:

$$w[n] = \begin{cases} 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) & 0 \leq n \leq N-1 \\ 0 & \text{otherwises} \end{cases} \tag{5.4}$$

Again, N is the length of the window. Figure 5.2 illustrates these two types of window functions.

DFT Speech frames are transformed from time domain to frequency domain as discussed in section 2.3 using DFT as a sampled version of DTFT (section 3.1). The DFT of the m-th frame of the signal is defined as:

$$X_m[k] = \sum_{n=0}^{N-1} x_m[n] e^{-j2\pi kn/N} \tag{5.5}$$

After this step, we compute $|X_m[k]|^2$ for all frames, resulting in a short-time spectrum of the original signal.

Mel-frequency warping The spectrum of each frame is warped by a band of B filters (equation 3.6) to obtain the mel-frequency spectrum:

$$S_m[b] = \sum_{k=0}^{N-1} |X_m[k]|^2 H_b[k] \quad b = 0, 1, ..., B-1 \tag{5.6}$$

DCT Finally, the mel cepstrum is acquired from the mel spectrum using DCT:

$$\hat{x}_m[n] = \sum_{b=0}^{B-1} \ln(S_m[b]) \cos\left[\left(b + \frac{1}{2}\right)\frac{\pi n}{B}\right] \quad n = 0, 1, ..., B-1 \tag{5.7}$$

In our framework, we discard the first coefficient and keep the first K coefficients (except the first one) of the cepstrum as MFCCs.

Liftering Again, a filter is used to balance energies between coefficients of MFCCs, and is called a *lifter* in cepstral domain. Let c_i be the i-th coefficient, it is then liftered as:

$$\hat{c}_i = \left[1 + \frac{L}{2}\sin\left(\frac{\pi n}{L}\right)\right] c_i \quad n = 1, 2, ..., K < L \tag{5.8}$$

Here, L is the lifter coefficient, and its default value is $L = 22$ in our framework. From this point, we referred MFCCs as the liftered version to use as speaker identification features.

Differential The MFCCs are often referred as static features since they only contain information of their current frame. In order to capture temporal relations, cepstral coefficients are modified by calculating their first and second order derivatives. The first order derivative is called delta coefficients, and the second order is called delta-delta coefficients. Delta coefficients are computed from cepstral coefficients as follow:

$$\Delta c_t = \frac{\sum_{\tau=1}^{D} \tau(c_{t+\tau} - c_{t-\tau})}{2\sum_{\tau=1}^{D} \tau^2} \tag{5.9}$$

D is the size of delta window, and is normally chosen as 1 or 2. Consequently, delta-delta coefficients are computed as derivatives of delta-coefficients. While delta coefficients provide information about rate of speech, delta-delta coefficients disclose knowledge about acceleration.

In practice, delta and delta-delta coefficients are added to cepstral coefficients to extend them with dynamic features. Moreover, the energy (sum of spectral values in one frame) and its derivatives are also incorporated as features. For example, 38 dimension MFCC vector was used as features for a speaker recognition system, which included 12 MFCCs, 12 delta MFCCs, 12 delta-delta MFCCs, the log energy, the log energy derivative (delta energy) and the second log energy derivative (delta-delta energy) [44].

Moreover, feature vectors of multiple frames can be concatenated to be used as the input of the next step. A context of size C is added to a current frame by appending features of its C frames each side along with it. We summarized parameters of our front-end in table 5.6.

5.4.3 Back-end

Lying in the heart of our speaker identification framework is a DNN with a bidirectional recurrent layer inspired by the model in [25]. Hannun et al.'s model was composed of 5 hidden layers, but in our model, the number of hidden layers is

Parameter	Meaning
`frame_length`	Number of samples in one frame
`frame_step`	Number of samples advanced between frames
`window`	Type of window
`no_ffts`	Number of bins in DFT
`no_fbs`	Number of filters in mel/linear filterbank
`min_freq`	The minimum frequency in the filterbank
`max_freq`	The maximum frequency in the filterbank
`no_ceps`	Number of kept cepstral coefficients
`preemphasis_coefficient`	Pre-emphasis coefficent
`lifter_coefficient`	Lifter coefficent
`type`	Type of features, e.g., MFCC, MFCC+Δ, ...
`context_size`	Number of feature vectors at each side to be added as context

Table 5.6: Parameters of the front-end and their meanings

left as a parameter. For all terms regarding ANNs, we refer the reader to section 4.1.

Let L be the number of layers in our model, where layer 0 is the input and layer L is the output layer. Then, the bidirectional recurrent layer is placed at position $L-2$. Figure 5.3 illustrates the structure of our model. The input of the DNN is a sequence of speech features of length T, $x = x_0, x_1, ..., x_{T-1}$, where x_t denotes the feature vector at frame t, $t = 0...T-1$.

For the first $L-3$ layers, the output of the l-th layer at time t is computed as:

$$h_t^{(l)} = \varphi(W^{(l)} \cdot h_t^{(l-1)} + b^{(l)}) \tag{5.10}$$

where $W^{(l)}$ and $b^{(l)}$ are the weight matrix and the bias vector of layer l. φ is the activation function, and is selected between sigmoid, tanh and ReL.

The recurrent layer is decomposed into two separate layers for the forward and the backward processes (figure 5.4):

$$h_t^{(f)} = \varphi(W^{(L-2)} \cdot h_t^{(l-1)} + W^{(f)} \cdot h_{t-1}^{(f)} + b^{(f)}) \tag{5.11}$$

$$h_t^{(b)} = \varphi(W^{(L-2)} \cdot h_t^{(l-1)} + W^{(b)} \cdot h_{t+1}^{(b)} + b^{(b)}) \tag{5.12}$$

Note that $h_t^{(f)}$ must be computed in order $t = 0, ..., T-1$ while $h_t^{(b)}$ must be computed in reverse order $t = T-1, ..., 0$. The output of this layer is simply the linear combination of both the forward and the backward output:

$$h_t^{(L-2)} = h_t^{(f)} + h_t^{(b)} \tag{5.13}$$

Layer $L-1$ again is a normal layer:

$$h_t^{(L-1)} = \varphi(W^{(L-1)} \cdot h_t^{(L-2)} + b^{(L-1)}) \tag{5.14}$$

and the output layer is the softmax layer of which each neuron output predicts the probability of a speaker to be the target in the training set:

$$h_t^{(L)} = \text{softmax}(W^{(L)} \cdot h_t^{(L)} + b^{(L)}) \tag{5.15}$$

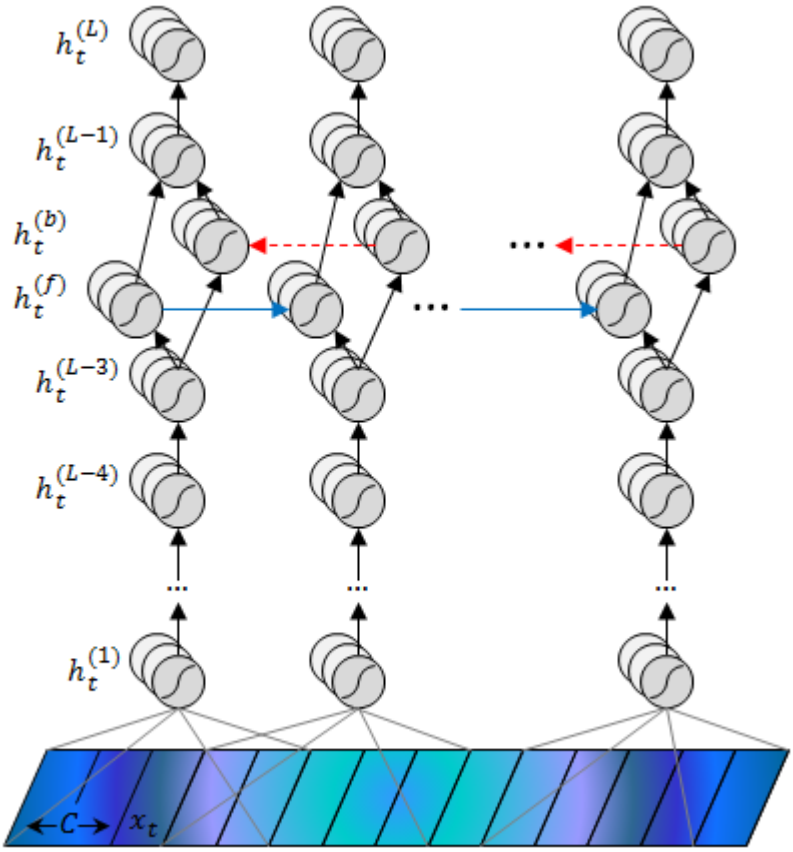


Figure 5.3: The structure of our DNN model

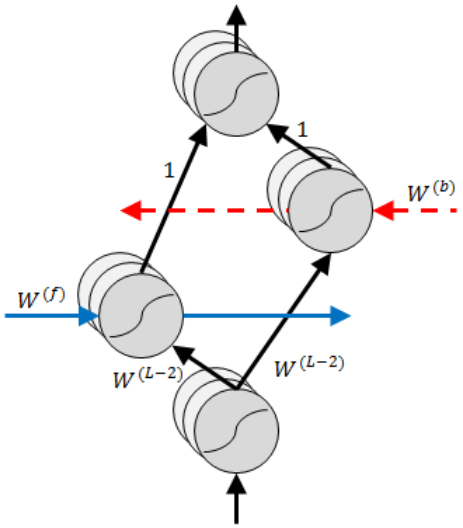


Figure 5.4: A closer look at the recurrent layer

where:

$$\text{softmax}(z)_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)} \tag{5.16}$$

Here z_j represents the j-th column of vector z.

Given a training dataset of S speakers, the DNN simply classifies the target speaker of frame t as the one that maximizes the conditional probability:

$$\hat{s}_t = \text{argmax}_{1 \leq s \leq S} P(s \mid x_t) = \hat{y}_{t,s} = h_{t,s}^{(L)} \tag{5.17}$$

The predicted speaker for the whole speech sequence x is defined by simple voting as summing the accuracy of all frames and normalizing.

The DNN model is trained using the backpropagation algorithm to minimize mean squared or cross entropy error. Parameter update is performed in batch, where each batch contains about 500 frames. We implemented three different update methods:

Gradient descent

$$\theta_{t+1} = \theta_t - \eta \frac{\partial E}{\partial \theta_t} \tag{5.18}$$

Momentum [49]

$$v_{t+1} = \mu v_t - \eta \frac{\partial E}{\partial \theta_t} \tag{5.19}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{5.20}$$

Nesterov's accelerated momentum [6]

$$v_{t+1} = \mu v_t - \eta \frac{\partial E}{\partial (\theta_t + \mu v_t)} \tag{5.21}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{5.22}$$

θ_t is a parameter value at time t, E is the loss function and η is the learning rate. In momentum methods, μ is the momentum and v_t represents the velocity of update at time t.

The learning rate is usually initialized with small value (e.g, 10^{-5} to 10^{-3}) and is reduced by some constant after some predefined number of epochs. Besides, RMSProp, an adaptive learning rate method, was also employed to tune the learning rate locally for each parameter. The update rule of RMSprop is:

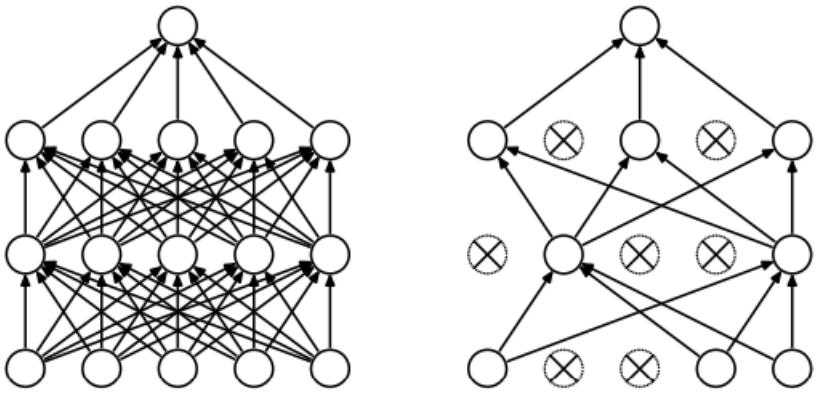
$$g_{t+1} = k g_t + (1 - k) \left(\frac{\partial E}{\partial \theta_t} \right)^2 \tag{5.23}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\partial E}{\partial \theta_t} \cdot \frac{1}{\sqrt{g_{t+1} + 10^{-8}}} \tag{5.24}$$

k is the decay rate, and its typical values are 0.9, 0.99 or 0.999.

To avoid overfitting, during training we use L2 regularization and/or *dropout* [62], which drops some neurons (i.e., set their value to zeros) at some probability p. Figure 5.5 illustrates the idea of dropout. Dropout is performed in all layers except at the input and the recurrent layer.

For reference, all hyperparameters of the back-end are summarized in table 5.7.



(a) Standard Neural Net (b) After applying dropout.

Figure 5.5: The visualization of dropout (adapted from [62])

Type	Hyperparameter	Options	Parameters
Model	Structure		
	Size of each layers		
	Activation function	Sigmoid, tanh, ReL	
Training	Cost function	Mean squared, cross entropy	
	Learning rate		
	Update rule	Gradient descent	
		Momentum	momentum
		Nesterov's accelerated momentum	momentum
		RMSprop	decay rate
	Regularization	L2 regularization	regularization parameter
		Dropout	dropout rate

Table 5.7: Hyperparameters of the back-end and their choices

5.4.4 Configuration file

Our framework was implemented in Python using NumPy [1] and Theano libraries. Theano is a math compiler that supports symbolic mathematical functions. Therefore, gradient expressions are derived automatically, and computation graphs can be optimized, which later can be deployed on CPU or GPU without changing users' code [7]. The system and training parameters are defined using a configuration file in YAML format [2]. An example of configuration files is presented in figure 5.6.

[1] `www.numpy.org`
[2] `yaml.org`

```
!!python/object:sre.train.Train
name: 'template'
system: !!python/object/new:sre.sresystem.SRESystem
  kwds:
    frontend: !!python/object:sre.frontend.Frontend
      preemphasis_coefficient: 0.97
      frame_length: 320
      frame_overlap: 160
      max_freq: 6000
      min_freq: 300
      no_fbs: 26
      no_ffts: 512
      window_func: !!python/name:numpy.lib.function_base.hanning ''
      no_ceps: 13
      lifter_coefficient: 22
      context_size: 0
      type: 'mfcc'
    backend: !!python/object/new:sre.backend.Backend
      kwds:
        no_hiddens: [50, 50, 50]
        no_input: 13
        no_output: 20
        activation: !!python/name:sre.backend.rel ''
algorithm: !!python/object/new:nn.training_algorithms.TrainingAlgorithm
  kwds:
    rate: 0.001
    cost_function: !!python/name:nn.costs.cross_entropy_error ''
    update_rule: !!python/object/new:nn.updates.RMSprop
      kwds:
        decay_rate: 0.99
train_path: '/home/timit/timit-train.list'
trial_path: '/home/timit/timit-trial.list'
batch_size: 500
max_epochs: 1000
eval_epoch: 20
```

Figure 5.6: An example of initializing and training a speaker identification system. Here the front-end returns 13 MFCCs and the back-end has 3 hidden layers with 50 neurons at each level.

5.5 Experiments and Results

5.5.1 Experiment 1: Performance on small size populations

In this experiment, our framework's performance is assessed on a small set of speakers. All tests are evaluated on TIMIT-8k data with the same conditions described in the reference paper (section 5.3). However, since training ANNs

Front-end	Description
MFCC23	23 MFCCs
LFCC23	23 LFCCs
MFCCΔ38	19 MFCCs and 19 delta MFCCs
MFCC$\Delta\Delta$38	12 MFCCs, 12 delta MFCCs, 12 delta-delta MFCCs, delta energy and delta-delta energy

Table 5.8: Description of testing front-ends

Back-end	5 speakers	10 speakers	20 speakers
RNN-structured	100	200	400
FNN-structured	300	600	600

Table 5.9: Hidden layer size with regard to population size

require a validation set as the stop condition, we decided to use 2 files as validation data. The number of files in training data remains the same as 5 files since training duration should have great influence on identification performance. As a result, for each speaker, 5 sentences are used as training data, 2 sentences are used as validation data and the remaining 3 sentences are used as 3 evaluation tests. The reason that we use a validation dataset is that other stop condition types (pre-defined number of epochs, minimum loss values) varies between different population sizes, making it hard to choose a good stopping value.

In the reference paper, it seems that the authors used only one evaluation set for each population size, but we do not know exactly which speakers and which files they used in each part. Therefore, in order to compare our results with their performance, for each population size, both speaker selection and data (training, validation, test) division are performed randomly three times, resulting in three different evaluation sets. Each testing configuration is trained and evaluated on all three sets, and the reporting accuracy is the mean of all test cases.

Each speech utterance is processed using a 20 ms Hanning window every 10 ms. Then DFT spectrum is warped by a Mel/linear filterbank of 26 filters with limiting frequency range 300-3140 Hz. Delta and delta-delta cepstral coefficients are computed around a window size of 2 frames.

We select four types of front-ends to extract features from speech data, of which details are described in table 5.8. In addition to the proposed RNN framework, we also implement a FNN back-end in order to compare their power in speaker identification task. Their combinations result in 6 testing systems. All systems have 3 hidden layer structure and use ReL as activation functions. Since the number of testing speakers affects the size of our neural network models, the size of systems in each case is chosen according to table 5.9.

Each system is trained using RMSprop with cross entropy error and L2 regularization parameter is 0.001. During training, a dropout rate of 5% is applied. RNN-structured systems are trained with learning rate 10^{-4}, while the MLP system is trained with learning rate 10^{-3}. The best system is chosen based on its accuracy on the validation set, and the training process stops if the accuracy does not improve after 20 epochs or the number of training cycles reaches 1000.

Experiment results with a single frame as input (context size is 0) are summarized in table 5.10. In comparison with reference systems (table 5.1), all systems

System	5 speakers	10 speakers	20 speakers
RNN+MFCC23	84.4%	82.2%	86.7%
RNN+LFCC23	**95.5%**	86.7%	85.6%
RNN+MFCCΔ38	88.9%	90.0%	83.3%
RNN+MFCCΔΔ38	91.1%	88.9%	83.9%
FNN+MFCC23	93.3%	94.4%	88.3%
FNN+MFCCΔΔ38	**95.5%**	**97.8%**	**90.5%**

Table 5.10: Identification accuracy of different testing systems

of ours only yield higher accuracy than reference decision tree algorithms ID3, C4 and CART. However, the reason of poor results is due to the difference between validation accuracy and evaluation accuracy. As using only two files to validate, in all cases, our systems easily achieve high accuracy on the validation set (above 95%), and in more than half of them, they get 100% correct on validation data and the training process stops. Using more data on validation and evaluation may alleviate this problem. This is definitely a disadvantage of ANNs in comparison with other methods since a part of data (normally from the training set) is needed for validation.

Our best system is FNN+MFCCΔΔ38. Interestingly, RNN-structured systems do not show their superiority against FNN-structured systems in this task. Although FNN systems have more free parameters than RNN ones, it only takes 45 minutes to train a FNN on 1000 epochs in comparison with 9 hours of a RNN with data of 20 speakers. In some cases, the accuracy of 5 speaker test condition is lower than that of 10 speaker test condition, since it is too easy to obtain 100% correct on the validation dataset of 5 speakers. Otherwise, the accuracy drops when the number of speakers is 20.

We also investigate the influence of context size on the identification result. Four systems are tested with context size 1, 2 and 3, corresponding to 3, 5 and 7 consecutive frames as input. The results are presented in table 5.11. With two RNN-structured systems, the best results are achieved when using longer contexts. The explanation for this phenomenon is that the longer the input vector, the finer the parameters are tuned in order to identify correctly. Therefore, it is not as easy as before to get high accuracy on the validation set, so that the gap between validation accuracy and test accuracy is reduced. However, there is an opposite trend within two FNN-structured systems, as their performance reduces while increasing the size of context, especially when the number of speakers is 20. Since we keep the same configuration while increasing input context, this may be a sign of overfitting as our models could achieve high accuracy on the validation set but are not very good at generalization.

5.5.2 Experiment 2: Performance with regard to training duration

The purpose of this experiment is to examine our framework performance on various training duration. Testing systems are evaluated on a population of 20 speakers with training data ranged from 1 to 5 files. The length of each file is about 3 seconds. Again, we use TIMIT-8k data with the same test conditions

System	Context	5 speakers	10 speakers	20 speakers
RNN+MFCC23	0	84.4%	82.2%	86.7%
RNN+MFCC23	1	82.2%	**92.2%**	89.4%
RNN+MFCC23	2	86.7%	86.7%	**90.6%**
RNN+MFCC23	3	**91.1%**	**92.2%**	90.0%
RNN+MFCCΔΔ38	0	91.1%	88.9%	83.9%
RNN+MFCCΔΔ38	1	86.7%	86.7%	88.9%
RNN+MFCCΔΔ38	2	88.9%	91.1%	90.0%
RNN+MFCCΔΔ38	3	**95.5%**	**96.7%**	**92.8%**
FNN+MFCC23	0	**93.3%**	94.4%	**88.3%**
FNN+MFCC23	1	**93.3%**	94.5%	81.1%
FNN+MFCC23	2	91.1%	**97.8%**	76.1%
FNN+MFCC23	3	**93.3%**	93.3%	85.6%
FNN+MFCCΔΔ38	0	95.5%	97.8%	**90.5%**
FNN+MFCCΔΔ38	1	86.7%	**98.9%**	85.6%
FNN+MFCCΔΔ38	2	**97.8%**	94.5%	85.6%
FNN+MFCCΔΔ38	3	**97.8%**	97.8%	79.4%

Table 5.11: Identification accuracy with different input context sizes. The best context size in each case is marked as bold.

described in section 5.5.1, and each test condition is repeated three times.

Of four systems chosen in this experiment, two RNN-structured systems use a context of size 3, and two FNN-structured systems are evaluated without context. The summary of our four systems' performances are illustrated in figure 5.7a. Having the same type of front-end, RNN systems tend to identify better than FNN systems. On the other hand, MFCCs with dynamic features (MFCCΔΔ38) outperform normal MFCCs in both types of back-ends.

Moreover, the performance of our two best systems, RNN+MFCCΔΔ38 and FNN+MFCCΔΔ38 is compared with two best systems in [19], full-search VQ and modified neural tree network in the same test conditions (figure 5.7b). There is only small difference between full-search VQ, RNN+MFCCΔΔ38 and FNN+ MFCCΔΔ38 when the number of training files is between 2 and 4. Otherwise, full-search VQ is still the most efficient system if using 1 file as training data with about 72% accuracy. The gap between all systems reduces as using more training files.

5.5.3 Experiment 3: Performance on large populations

Our last experiment is dedicated to illustrate the effect of population size on identification accuracy. Intuitively, accuracy tends to decrease as the population grows larger, but this rate is not equal in different systems. We also use the same data division as in the previous experiments, and gradually increase the number of speakers to 100. Data are not restricted to 38 speakers of New England dialect as before. Four systems participating in this experiment use input from a single frame (without context), and their size and learning rate are adjusted according to table 5.12 and 5.13 respectively. Although the number of perceptrons per layer is very large, the accuracy only slightly changes by 2-3% when those numbers

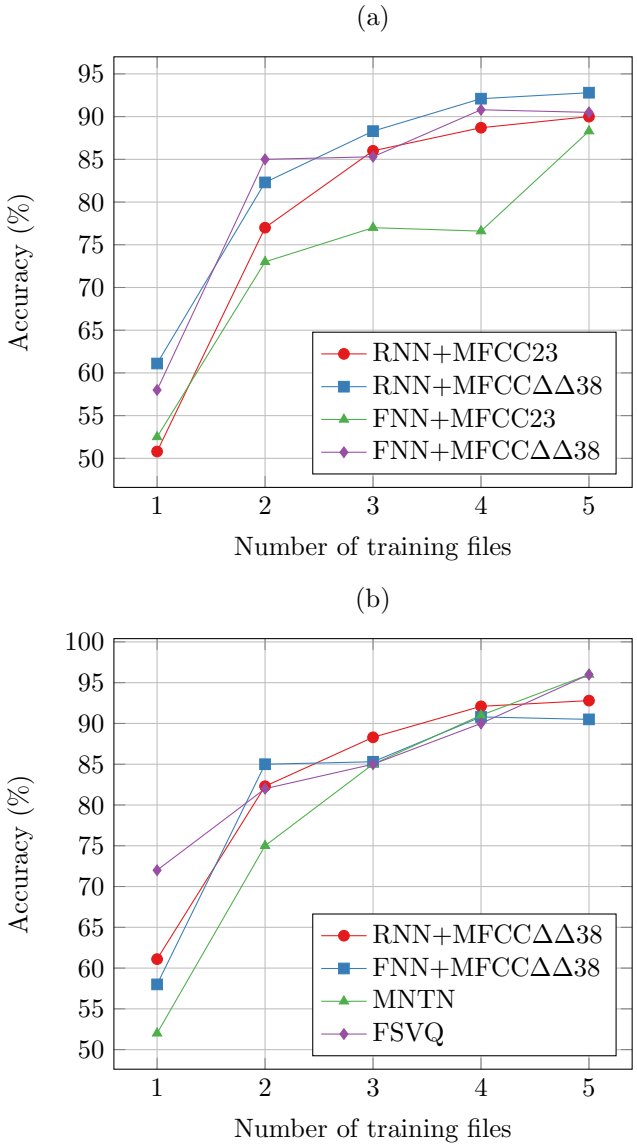


Figure 5.7: Identification accuracy of our systems and two best systems from [19] (MNTN: modified neural tree network, FSVQ: full-search VQ) as a function of training duration

Back-end	40 speakers	60 speakers	80 speakers	100 speakers
RNN-structured	800	1200	800	1000
FNN-structured	2400	3600	4800	6000

Table 5.12: Hidden layer size with regard to population size

Back-end	40 speakers	60 speakers	80 speakers	100 speakers
RNN-structured	10^{-4}	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$
FNN-structured	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	10^{-5}	$5 \cdot 10^{-5}$

Table 5.13: Learning rate with regard to population size

double or reduce by half, so we decide to keep these values to evaluation.

The experiment results are presented in figure 5.8. It is noteworthy that most

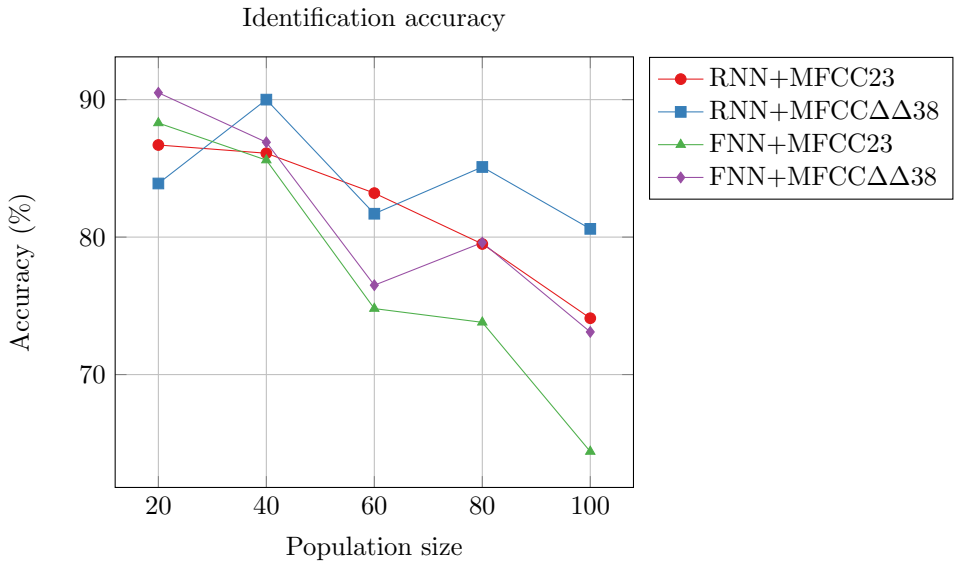


Figure 5.8: Identification accuracy as a function of population size

systems' accuracy drops sharply when the number of testing speakers is 60, which might be due to data division. Otherwise, the results in this experiment agree with those in our second experiment (section 5.5.2) as RNN systems yield higher accuracy than FNN systems, and using dynamics features improves systems' performance. The accuracy of FNN+MFCC23 decreases at the highest rate.

5.5.4 Experiment 4: Sex identification

As we have achieved accuracy of above 90% on a small population of the TIMIT corpus in speaker identification task (section 5.5.1), one could guess that the accuracy in sex identification must be higher. The reason of this prediction bases on the fact that there are only two classes in sex identification, male and female, thus we have more data to train and validate. In this section, we would like to show the capability of some neural network models to identify speakers' gender.

There is only one test condition for this task. The test condition includes 20 speakers in the training set, 10 speakers in the validation set and 40 speakers in the test set. The number of male speakers and that of female speakers in each set are equal, and all sets are mutually disjoint. The number of files per speaker in the training, validation and test set are 3, 2 and 2 respectively. Using random selection from the whole corpus, we produce 3 datasets in total.

We only use one type of frontend, MFCC+ΔΔ38, in this experiment. For the backend, two types of models are considered:

- RNN: The same type of the RNN model we use in previous experiments. Its hidden layers have 20 perceptrons.
- CNN: A CNN that consists of:
  - A convolutional layer containing 6 feature maps of size $(5, 5)$
  - A max pooling layer with shape $(2, 2)$
  - A convolutional layer containing 12 feature maps of size $(4, 4)$

- A max pooling layer with shape $(2, 2)$
- An output layer with softmax activation function

The input of the CNN model has shape $(15, 38)$ by stacking 15 frames of MFCC+$\Delta\Delta$38.

Both systems are trained by RMSprop algorithm with rate 10^{-4}. In the end, RNN system achieves accuracy of 97.9%, and CNN system achieves accuracy of 98.3%. Both systems reach 100% correct on the validation set in less than 100 epochs.

5.5.5 Epilogue: Language identification

The same structure of DNNs could be applied to recognize the language of an utterance. However, since the TIMIT corpus is in English, we could not use those data in the language identification task. Unfortunately, at this moment, we do not have access to any multilingual speech database. Therefore, the task is left for future research.

Chapter 6

Conclusion and Future Work

Close-set text-independent speaker identification is a hard problem since its complexity depends on the number of enrolled speakers. Researches in past decades focused on small populations with telephone quality that resembled real condition. Some approaches achieved good results (section 5.1) with accuracy over 90%. Past research works were reported on different corpora, some of them were private database while others were not specifically designed for speaker identification. Moreover, some research used only a subset of a large corpus, making it impossible to compare its performance without knowing the exact subset. Unfortunately, the latest techniques in speaker recognition usually evaluate their performance on speaker verification rather than speaker identification, so there is no recent corpus for speaker identification. Therefore, in order to assess a new technique capability, it is essential to choose a widely used corpus and reproduce test conditions as close as possible. For that reason, despite of its unrealistic characteristics, TIMIT was chosen as our evaluation data.

ANNs and DNNs (chapter 4) has been proven as powerful techniques in speech processing. Furthermore, a RNN is a DNN which is capable of capturing long distance relations. Thus, it is especially suitable for sequential data.

Based on a successful RNN model in speech recognition (section 5.4.3), we proposed our experimental model to solve the speaker identification task. MFCCs, LFCCs (section 3.1.1, 3.1.2) in combination with their dynamic features (section 5.4.2) were extracted from speech signals as input to the backend. Our systems were implemented in Python using the Theano library, and training was initialized through a configuration file.

Several systems in [19] were chosen as references in our experiments. From experiment results (section 5.5), our systems were less advanced than reference systems when they were tested in small population due to limited amount of validation data. FNN systems yielded better results than RNN systems in small population evaluation, but in most cases, as the number of enrolled speakers increased, the RNN structure had demonstrated its advances over the FNN structure. Moreover, using MFCCs with dynamic features yielded the best identification results.

There are some highlights in using ANNs in speaker identification:

Pros:

- The ANN paradigm is powerful, yet simple and elegant, and ANNs can be trained easily using gradient descent algorithms.

- With GPU-support scientific computing library like Theano, ANNs are implemented effectively without spending too much effort (gradients are computed directly rather than by propagating errors).

Cons:

- The most effective technique to train ANNs requires a validation dataset, which is usually a part of training data.
- When using one large ANN for all speakers, the optimal size of each layer in the network can only be determined by trial. Thus, when the number of speakers increases, several configurations must be tried in order to find a good size for the network.
- ANNs in general have too many hyperparameters: learning rate, update rule, regularization, ... that could affect their performance. Therefore, there are a huge amount of configurations that can be considered, and the selected configuration may not be the optimal one.
- RNNs has shown its effectiveness over FNNs, but they are time-consuming. It took a model for 100 speakers 5-6 days to finish 1000 epochs.

Further research directions can be explored to improve our understanding about the capability of ANNs in speaker identification:

- Trying more configurations: A searching tactic through configuration space (hyperparameter training) should help finding better configurations of the system.
- Voting scheme: The target speaker is determined simply by summing and normalizing probabilities of all frames. Other kinds of score combinations or a second classifier may help improve the accuracy.
- Exploring other types of DNNs: Long short-term memory and deep belief network are DNN structures that have more effective training schemes than normal RNNs.
- Exploring DNN bottleneck features (section 4.6)

Bibliography

[1] Sayed Jaafer Abdallah, Izzeldin Mohamed Osman, and Mohamed Elhafiz Mustafa. Text-independent speaker identification using hidden Markov model. *World of Computer Science and Information Technology Journal (WCSIT)*, 2(6), 2012.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. 2nd edition, 12 2009. ISBN 9780262012430.

[3] Bishnu S. Atal. Effectiveness of linear prediction characteristics of the speech wave for automatic speaker identification and verification. *The Journal of the Acoustical Society of America*, 55(6):1304–1312, 1974.

[4] Homayoon Beigi. *Fundamentals of Speaker Recognition*. Springer, 2011 edition, 2011.

[5] J. Benesty, M. M. Sondhi, and Y. Huang. Introduction to speech processing. In Jacob Benesty, M. Mohan Sondhi, and Yiteng (Arden) Huang, editors, *Springer Handbook of Speech Processing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. ISBN 3540491252.

[6] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE, 2013.

[7] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010. Oral Presentation.

[8] B. Bogert, M. Healy, and J. Tukey. The quefrency alanysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking. In *Proceedings of the Symposium on Time Series Analysis*, pages 209—-243. John Wiley and Sons, Inc., 1963.

[9] J.P. Campbell, Jr. Speaker recognition: A tutorial. *Proceedings of the IEEE*, 85(9):1437–1462, 9 1997. ISSN 0018-9219. doi: 10.1109/5.628714.

[10] Paul Cuff. ELE 201: Information signals - course notes, 2015. URL `http://www.princeton.edu/~cuff/ele201/kulkarni.html`.

[11] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[12] Steven B. Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(4):357–366, 8 1980. ISSN 0096-3518. doi: 10.1109/TASSP.1980.1163420.

[13] Najim Dehak, Patrick Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-end factor analysis for speaker verification. *Audio, Speech, and Language Processing, IEEE Transactions on*, 19(4):788–798, 2011.

[14] John R. Deller, Jr., John G. Proakis, and John H. Hansen. *Discrete Time Processing of Speech Signals*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1993. ISBN 0023283017.

[15] Li Deng and Dong Yu. *Deep learning: Methods and applications*. Now Publisher Inc, 2014. ISBN 1601988141.

[16] S. B. Dhonde and S. M. Jagade. Feature extraction techniques in speaker recognition: A review. *International Journal on Recent Technologies in Mechanical and Electrical Engineering (IJRMEE)*, 2(5):104–106, 5 2015. ISSN 2349-7947.

[17] Jeffrey L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[18] Zheng Fang, Zhang Guoliang, and Song Zhanjiang. Comparison of different implementations of MFCC. *Journal of Computer Science and Technology*, 16(6):582–589, 11 2001. ISSN 1000-9000.

[19] Kevin R. Farrell, Richard J. Mammone, and Khaled T. Assaleh. Speaker recognition using neural networks and conventional classifiers. *Speech and Audio Processing, IEEE Transactions on*, 2(1):194–205, 1994.

[20] Sadaoki Furui. *40 Years of Progress in Automatic Speaker Recognition*, volume 5558 of *Lecture Notes in Computer Science*, pages 1050–1059. 6 2009.

[21] Herbert Gish and Michael Schmidt. Text-independent speaker identification. *Signal Processing Magazine, IEEE*, 11(4):18–32, 1994.

[22] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[23] John J. Godfrey, Edward C. Holliman, and Jane McDaniel. SWITCHBOARD: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1, pages 517–520. IEEE, 1992.

[24] Rachel Hall. The math of waves, 2014. URL `http://thesoundofnumbers.com/category/lectures/`.

[25] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deepspeech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[26] A. L. Higgins, L. G. Bahler, and J. E. Porter. Voice identification using nearest-neighbor distance measure. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, volume 2, pages 375–378. IEEE, 1993.

[27] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[28] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[29] S. Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*. Diploma thesis, Institut f. Informatik, Technische Univ. Munich, 1991.

[30] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[31] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, 1st edition, 2001. ISBN 0130226165.

[32] Charles Jankowski, Ashok Kalyanswamy, Sara Basson, and Judith Spitz. NTIMIT: A phonetically balanced, continuous speech, telephone bandwidth speech database. In *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*, pages 109–112. IEEE, 1990.

[33] Ye Jiang, Kong-Aik Lee, Zhenmin Tang, Bin Ma, Anthony Larcher, and Haizhou Li. PLDA modeling in i-vector and supervector space for speaker verification. In *INTERSPEECH*, 2012.

[34] J. Kacur, R. Vargic, and P. Mulinka. Speaker identification by k-nearest neighbors: Application of PCA and LDA prior to KNN. In *Systems, Signals and Image Processing (IWSSIP), 2011 18th International Conference on*, pages 1–4, 6 2011.

[35] Patrick Kenny, Gilles Boulianne, Pierre Ouellet, and Pierre Dumouchel. Speaker and session variability in GMM-based speaker verification. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(4):1448–1460, 2007.

[36] Yochai Konig, Larry Heck, Mitch Weintraub, and Kemal Sonmez. Nonlinear discriminant feature extraction for robust text-independent speaker recognition. In *Proc. RLA2C, ESCA workshop on Speaker Recognition and its Commercial and Forensic Applications*, pages 72–75, 1998.

[37] Jacques Koreman, Dalei Wu, and Andrew C. Morris. Enhancing speaker discrimination at the feature level. In *Speaker Classification I*, pages 260–277. Springer, 2007.

[38] LISA lab. Convolutional neural networks (LeNet), 2015. URL `http://deeplearning.net/tutorial/lenet.html`.

[39] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[40] Howard Lei and Eduardo López Gonzalo. Mel, linear, and antimel frequency cepstral coefficients in broad phonetic regions for telephone speaker recognition. In *INTERSPEECH*, pages 2323–2326, 2009.

[41] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *Communications, IEEE Transactions on*, 28(1):84–95, 1 1980. ISSN 0090-6778. doi: 10.1109/TCOM.1980.1094577.

[42] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[43] Macquarie University. Speech waveforms. URL `http://clas.mq.edu.au/speech/acoustics/waveforms/speech_waveforms.html`.

[44] Jorge Martinez, Hector Perez, Enrique Escamilla, and Masahisa Mabo Suzuki. Speaker recognition using Mel frequency cepstral coefficients (MFCC) and vector quantization (VQ) techniques. In *CONIELECOMP 2012, 22nd International Conference on Electrical Communications and Computers*, 2012.

[45] Pavel Matějka, Ondřej Glembek, Fabio Castaldo, Md Jahangir Alam, Oldřich Plchot, Patrick Kenny, Lukáš Burget, and Jan Honza Černocky. Full-covariance ubm and heavy-tailed PLDA in i-vector speaker verification. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 4828–4831. IEEE, 2011.

[46] Tomoko Matsui and Sadaoki Furui. Comparison of text-independent speaker recognition methods using VQ-distortion and discrete/continuous HMM's. *Speech and Audio Processing, IEEE Transactions on*, 2(3):456–459, 1994.

[47] Tom M. Mitchell. *Machine Learning*, chapter 4. McGraw-Hill Education, 3 1997. ISBN 0070428077.

[48] Alan V. Oppenheim. *Superposition in a Class of Nonlinear Systems*. PhD thesis, MIT, 1965.

[49] Boris Teodorovich Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[50] Alan B. Poritz. Linear predictive hidden Markov models and the speech signal. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.*, volume 7, pages 1291–1294. IEEE, 1982.

[51] Mohamed Qasem. Vector quantization. URL `http://www.mqasem.net/vectorquantization/vq.html`.

[52] Thomas F. Quatieri. *Discrete-Time Speech Signal Processing: Principles and Practice*. Prentice Hall, 2001.

[53] Lawrence R. Rabiner and Ronald W. Schafer. *Introduction to Digital Speech Processing*. Now Publishers Inc, 2007.

[54] Douglas A. Reynolds. Speaker identification and verification using gaussian mixture speaker models. *The Lincoln Laboratory Journal*, 8(2):173–192, 1995.

[55] Douglas A. Reynolds. Automatic speaker recognition: Current approaches and future trends. *Speaker Verification: From Research to Reality*, pages 14–15, 2001.

[56] Douglas A. Reynolds and Richard C. Rose. Robust text-independent speaker identification using Gaussian mixture speaker models. *Speech and Audio Processing, IEEE Transactions on*, 3(1):72–83, 1995.

[57] Douglas A. Reynolds, Thomas F. Quatieri, and Robert B. Dunn. Speaker verification using adapted gaussian mixture models. *Digital signal processing*, 10(1):19–41, 2000.

[58] Fred Richardson, Douglas Reynolds, and Najim Dehak. Deep neural network approaches to speaker and language recognition. 2015.

[59] Laszlo Rudasi and Stephen A Zahorian. Text-independent talker identification with neural networks. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 389–392. IEEE, 1991.

[60] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[61] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

[62] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[63] S. S. Stevens and J. Volkmann. The relation of pitch to frequency: A revised scale. *The American Journal of Psychology*, 53(3):329–353, 1940.

[64] Naftali Z. Tisby. On the application of mixture AR hidden Markov models to text independent speaker recognition. *Signal Processing, IEEE Transactions on*, 39(3):563–570, 1991.

[65] Vibha Tiwari. MFCC and its applications in speaker recognition. *International Journal on Emerging Technologies*, 1(1):19–22, 2010.

[66] R. Togneri and D. Pullella. An overview of speaker identification: Accuracy and robustness issues. *Circuits and Systems Magazine, IEEE*, 11(2):23–61, 2011. ISSN 1531-636X.

[67] James S. Walker and Gary W. Don. *Mathematics and Music: Composition, Perception, and Performance*. Chapman and Hall/CRC, 2013.

[68] WebExhibits. Newton and the color spectrum. URL `http://www.webexhibits.org/colorart/bh.html`.

[69] H. Zeinali, H. Sameti, and B. BabaAli. A fast speaker identification method using nearest neighbor distance. In *Signal Processing (ICSP), 2012 IEEE 11th International Conference on*, volume 3, pages 2159–2162, 10 2012. doi: 10.1109/ICoSP.2012.6492008.

[70] Xinhui Zhou, Daniel Garcia-Romero, Ramani Duraiswami, Carol Espy-Wilson, and Shihab Shamma. Linear versus mel frequency cepstral coefficients for speaker recognition. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 559–564. IEEE, 2011.

[71] Victor Zue, Stephanie Seneff, and James Glass. Speech database development at MIT: TIMIT and beyond. *Speech Communication*, 9(4):351–356, 1990.

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial neural network
CNN	Convolutional neural network
DCT-II	Discrete cosine transform II
DFT	Discrete Fourier Transform
DNN	Deep neural network
DTFT	Discrete-Time Fourier Transform
EM	Expectation maximization
FNN	Feedforward neural network
FS	Fourier Series
FT	Fourier Transform
GMM	Gaussian mixture model
HMM	Hidden Markov model
IDFT	Inverse discrete Fourier transform
JFA	Joint factor analysis
kNN	k nearest neighbors
LPC	Linear prediction coefficients
LPCC	Linear predictive cepstral coefficients
MAP	Maximum a posteriori
MFCC	Mel-frequency cepstral coefficients
ML	Maximum likelihood
MLP	Multilayer perceptron
ReL	Rectified linear
ReLu	Rectified linear unit
RNN	Recurrent neural network

UBM	Universal background model
VQ	Vector quantization