A comprehensive deep learning approach to end-to-end language identification

Trung Ngo Trong

Master’s Thesis

ITÄ-SUOMEN YLIOPISTO

Faculty of Science and Forestry
School of Computer Science
January 2017
Abstract:

A new machine learning paradigm, called deep learning, has accelerated the development of state-of-the-art systems in various research domains. Deep learning leverages a sophisticated network of non-linear units and their connections to learn multiple levels of abstraction from the data. Each of these units is inspired by a model of organic brain cell which is known as a neuron. A deep neural network (DNN), which contains millions of neurons with their own parameters, allows the model freely optimizes its feature representations for particular tasks. This capability has been proven to be an universal function approximators which properly benefits the processing of complex signals, for instance, the voice signal. These architectures and algorithms have been the core of the recent groundbreaking approaches to automatic speech processing which includes automatic speech recognition (ASR), speaker recognition (SR) and language identification (LID) tasks. In particular, a combination of both deep learning and acoustic modeling has brought about breakthroughs in ASR, especially, an end-to-end SR system can interpret speech directly from its most primitive spectral form. The end-to-end design enhances completeness and is more flexible adapting to wide range of voice signals without the requirement of excessive hand-engineering features, moreover, it also increase the reliability by reducing the stacked deficiency of multiple components. However, similar exploration in LID is quite limited by the attention of research community, regardless the connection to speech technology of all three research fields.

This Master Thesis is motivated to unify the most recent advances in deep learning for speech processing, in order to solve end-to-end LID tasks. The work aims for multi-perspective approaches in which the investigation are different combination of the recent state-of-the-art deep networks’ architectures and conventional LID algorithms. The approaches are evaluated on Language Recognition Evaluation 2015 (LRE’15) from National Institute of Standards and Technology (NIST), the corpus are recorded audio in heterogeneous environments and languages which are considered the newest challenging LID dataset.

Keywords:

language identification, end-to-end, deep learning, recurrent neural network, convolu-
tional neural network, batch normalization, imbalanced dataset
Foreword

This Master Thesis summarizes the work carried out during the last year of my Master studies with the Speech and Image Processing Unit (SIPU) at University of Eastern Finland. The thesis contains most of the research published in [104], and I thank to Odyssey 2016 program committee for accepting and publishing our contributions to the conference.

I would like to dedicate this thesis to my loving family. I am truly grateful for every moment and words of encouragement from my mom, my dad and my grandmother. They have always been there, more than 6000 kilometers away, for me.

I also specially thank to enormous support and guidance from my supervisor Dr. Ville Hautamäki. He has always been great teacher with open mind, his enthusiasm and knowledge has inspired me to persuade the study in language identification and speech technology. I would like to express profound appreciation to my colleagues at UEF, especially from Ivan Kukanov for his great sense of humor and wise advices.

In addition, I want to express my great gratitude to Prof. Kristiina Jokinen. Her collaboration and advices are always enormous support. Additionally, I want to thank for financial support from DigiSami Project (Academy of Finland Project grant nro. 270082) during my work in this thesis.
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR</td>
<td>Automatic Speech Recognition</td>
</tr>
<tr>
<td>SR</td>
<td>Speaker Recognition</td>
</tr>
<tr>
<td>LID</td>
<td>Language Identification</td>
</tr>
<tr>
<td>BNF</td>
<td>Bottleneck network features</td>
</tr>
<tr>
<td>LRE</td>
<td>Language recognition evaluation</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>LLR</td>
<td>Log likelihood ratio</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Networks</td>
</tr>
<tr>
<td>MLPs</td>
<td>Multi-layer-perceptrons</td>
</tr>
<tr>
<td>DCN</td>
<td>Densely Connected Network</td>
</tr>
<tr>
<td>FNN</td>
<td>Feedfoward Neural Network</td>
</tr>
<tr>
<td>HMMs</td>
<td>Hidden Markov models</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Networks</td>
</tr>
<tr>
<td>RNNs</td>
<td>Recurrent neural networks</td>
</tr>
<tr>
<td>BPTT</td>
<td>Backpropagation through time</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long-short term memory</td>
</tr>
<tr>
<td>GRU</td>
<td>Gated recurrent neural network</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier transform</td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier transform</td>
</tr>
<tr>
<td>MFCCs</td>
<td>Mel-frequency cepstral coefficients</td>
</tr>
<tr>
<td>DCT</td>
<td>discrete cosine transform</td>
</tr>
<tr>
<td>GL</td>
<td>Generalization loss</td>
</tr>
<tr>
<td>MCLR</td>
<td>Multi-class logistic regression</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified linear unit</td>
</tr>
<tr>
<td>VAD</td>
<td>Voice activity detection</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean squared error</td>
</tr>
<tr>
<td>HD</td>
<td>Hellinger Distance</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.5.3 Gated recurrent neural network (GRU)</td>
<td>40</td>
</tr>
<tr>
<td>3.5.4 Addressing gradient vanishing with LSTM and GRU</td>
<td>40</td>
</tr>
<tr>
<td>3.5.5 RNN and Markov models</td>
<td>42</td>
</tr>
<tr>
<td>3.6 Dropout</td>
<td>43</td>
</tr>
<tr>
<td>3.7 Batch Normalization</td>
<td>44</td>
</tr>
<tr>
<td>3.7.1 CNNs and batch normalization</td>
<td>45</td>
</tr>
<tr>
<td>3.7.2 RNNs and batch normalization</td>
<td>45</td>
</tr>
<tr>
<td>4 Deep learning approaches to language identification</td>
<td>47</td>
</tr>
<tr>
<td>4.1 The &quot;indirect&quot; approach</td>
<td>47</td>
</tr>
<tr>
<td>4.2 The &quot;direct&quot; approach: an end-to-end system</td>
<td>49</td>
</tr>
<tr>
<td>5 Experiments in networks design</td>
<td>51</td>
</tr>
<tr>
<td>5.1 Speech corpus for LID</td>
<td>51</td>
</tr>
<tr>
<td>5.2 Setup</td>
<td>53</td>
</tr>
<tr>
<td>5.2.1 Initializing networks’ parameters</td>
<td>54</td>
</tr>
<tr>
<td>5.2.2 Optimization of deep networks</td>
<td>56</td>
</tr>
<tr>
<td>5.3 Baseline system</td>
<td>57</td>
</tr>
<tr>
<td>5.4 Results and analysis</td>
<td>58</td>
</tr>
<tr>
<td>5.4.1 Feature selection</td>
<td>58</td>
</tr>
<tr>
<td>5.4.2 The power of depth network</td>
<td>59</td>
</tr>
<tr>
<td>5.4.3 RNN variants</td>
<td>60</td>
</tr>
<tr>
<td>5.4.4 Multiple architecture design</td>
<td>61</td>
</tr>
<tr>
<td>5.4.5 Recurrent pooling in time</td>
<td>62</td>
</tr>
<tr>
<td>5.4.6 Optimizing the convolutional architecture</td>
<td>63</td>
</tr>
<tr>
<td>5.4.7 Deep language network</td>
<td>64</td>
</tr>
<tr>
<td>6 Tackling imbalanced dataset for end-to-end networks</td>
<td>69</td>
</tr>
<tr>
<td>6.1 Cost-sensitive objective function</td>
<td>69</td>
</tr>
<tr>
<td>6.2 Batch normalization</td>
<td>70</td>
</tr>
<tr>
<td>6.3 Data sampling</td>
<td>70</td>
</tr>
<tr>
<td>6.4 Score calibration</td>
<td>72</td>
</tr>
<tr>
<td>6.5 Discussion</td>
<td>73</td>
</tr>
<tr>
<td>7 Conclusions</td>
<td>75</td>
</tr>
</tbody>
</table>
This chapter provides a gentle introduction to the main topics of this master thesis, automatic language identification (LID). Since keyboard and mouse have been the basics of human-computer interaction for decades, voice user interface is considered the latest challenge to bridge the gap of natural communication between human and computer. This hands-free and eyes-free design are capable of extending the usability and productivity of traditional systems. As a result, speech signal processing is one of the crucial components to create "the machine that does understand". As the problem has been studied for decades, the research community has made tremendous progress in improving machine understanding of speech signal. There are three major processing tasks: 1) automatic speech recognition (ASR) [3, 23, 87]. 2) speaker recognition (SR) [47, 63, 87], 3) language identification (LID) [34, 56, 71]. Regardless different objectives, they share some properties in common which can be leveraged to enhance the performance of a particular task.

In this thesis, our work concentrates on improving the LID system. Spoken language identification [71] is the process of identifying the language spoken in a test utterance. In most of the tasks, the speech utterances are assumed to contain only one language, however, a more challenging task is the language diarization [74] in which the utterances contain segments from multiple languages. The recording environment is variated on a wide range of ambient noises, recording equipment and speaker accent, which requires the adaptation for channel differences among utterances.

In real world application, spoken language and accent contribute significant effect on the performance of speech processing systems [71], as a result, the task attracts increasing attention in the speech community because of its practical potential [71]. One typical application
of LID is language-oriented user interaction, the system acts as a gateway to the service, recognizes users’ language preference and intelligently customizes the interface [28]. Introducing LID component to speech processing system not only enhances the performance of other processing tasks include SR and ASR, but also advances the development of universal communication system which can classify and process multilingual audio data [71].

1.1 A classical approach to language identification

In recent years, the state-of-the-art LID systems have been based on spectral-based approaches [65, 86], which relies on purely acoustic of sequences of short-term spectral feature vectors. These vectors are assumed having statistical characteristics that differ from one language to another [25, 71].

Alternatively, the phonotactic approaches use a phonetic recognizer, together with a language model to characterize each language by the statistics of possible phones’ combinations [41]. Phone is one of the smallest units of sound which aims to specify pronunciation of any words [13], hence, it is considered a universal presentation of languages instead of syllables [13].

In general, building LID systems involves three main steps shown in Fig 1.1, which requires building a pipeline of handcrafted feature extraction and applicable classifier [86, 104]. This pipeline results in a complicated process and requires the adjustment of many hyperparameters for each block separately. As a result, researchers have introduced advanced machine learning algorithms to improve performance of each individual component [25, 57, 96], this is known as both drawback and merit of classical LID. One drawback of this design is that the feature representation might not be optimized for the classification objective. On the other hand, advances in machine learning can directly improve LID system in many aspects because of this modular design. As a result, deep learning is one of the techniques has been widely applied in the LID pipeline [25, 56, 86].

![Figure 1.1: General procedure for building LID system, inspired by [65]](image-url)
1.2 Motivation

The impressive gains in performance obtained using deep neural networks (DNNs) for automatic speech recognition [23, 50, 51, 92] have motivated the application of DNNs to other speech tasks such as speaker recognition and language identification [86]. Based on general approach to design LID system illustrated in Fig 1.1, two methods of applying DNN’s to the SR and LID tasks have been shown to be effective [86]. The “indirect” approach uses a DNN as a feature extractor, this network is trained for an ASR purpose to extract more robust and phonetically-meaningful representation. These features are then used to train a secondary classifier for LID task. On the other hand, the “direct” approach leverages an end-to-end DNN network as both a feature extraction and a classifier. As a result, the learned feature representation is optimized to directly benefit the recognition task.

In this study, we present the first large-scale analysis of various DNN architectures for the LID tasks. We hypothesize that “the combination of various Deep Neural Network architectures in an end-to-end design can be used to learn advanced language representation and benefits the language identification task”. Moreover, recent developments in deep learning [23, 52, 66, 91, 104] have suggested many designs and algorithms to address recognized drawbacks of DNN (e.g. training process [52], computational cost [23], regularization [98], and so on), which encourages us to search through the most reasonable combinations for the LID tasks. Inspired by the work in [91], we conduct a series of experiments using NIST LRE’15 corpus for a systematic study of the best network design for the LID tasks. We show that an end-to-end deep learning system can be used to recognize language from speech utterances with various lengths. Our results show that a combination of three deep architectures: feed-forward network, convolutional network and recurrent network can achieve the best performance compared to other network designs.

Additionally, our investigation shows that unequally distribution of training classes in NIST LRE’15 has a strong negative impact on the network performance [48, 104]. Specifically, the degradation can often be traced to the fact that the majority classes dominate training error and drive the network to sub-optimal solutions [32, 104]. Thus, we are also motivated to tackle the data imbalance challenge to solve the LID tasks using an end-to-end deep neural network. We conduct a series of experiments to search for the robust deep architecture that handles class-imbalance. Furthermore, we propose a framework to minimize the effect of dominant classes when training the end-to-end network and also post-processing
the output distribution for LID.

We compare our network performance to state-of-the-art BNF-based i-vector system [96] on NIST 2015 Language Recognition Evaluation corpus. The key to our approach is that we effectively address computational and regularization issues into the network structure to build deeper architecture compared to any previous DNN approaches to language recognition task.

### 1.3 Outline of the Thesis

The thesis is organized as follows:

- Chapter 1: a gentle introduction about language recognition and recent advances in the field of deep learning, and how the fusion of these two fields can advance further speech processing technologies.

- Chapter 2 summarizes the traditional approaches to speech processing for language recognition.

- Chapter 3 presents core concepts behind deep learning, and the most recent advances in the field which can benefit LID tasks.

- Chapter 4 describes the setup and configuration to validate our hypothesis, as well as detailing and analyzing the achieved results.

- Chapter 5 proposes a framework of integrated solutions for end-to-end networks dealing with imbalanced dataset.

- Chapter 6 draws the main conclusions, and proposing further direction for improving language identification system.
CHAPTER 2

Speech processing for language identification task

Like speech recognition or speaker recognition, LID shares the same kind of input data involved in speech processing. Speech is the vocalized form of communication which has been continuously evolved along with human history for millions of years [17]. The unique ability to develop an extreme complex syntactic combination of lexical and names has allowed us to advance our civilization and society. There are approximately 6500 spoken languages in the world today, each spoken word is created out of the phonetic combination of a limited set of vowel and consonant speech sound units [17]. Moreover, there exist acoustic differences in the way each individual pronouncing phones which make speech one of the most diverse and complex signals.

As a result, speech processing task requires the extraction of relevant features, which often involves transform of raw signals into time-frequency domain which enhances the details and preserves critical characteristics of speech [37]. However, the final goal of a LID system is different from, for instance, a speaker recognition one. The former tries to minimize speaker variation and emphasize the discriminative representation among languages during preprocessing [58], on the contrary, a speaker recognition system wants to recognize the distinction of each user by minimizing the within-speaker variability and normalizing irrelevant source of differences among speakers (e.g. language-normalized, source-normalized, and channel-normalized) [61]. In this chapter, we present most the applicable techniques for preprocessing audio segments in LID.
2.1 Speech processing

Speech signals is a digitalized continuous sound pressure wave, speech processing is the study of these signals (Fig. 2.1). The signals are always represented as a discrete set of values at a point in time. Sampling is periodically measuring the amplitude of signal after every time-step $t$, for example, 16000(Hz) means 16000 samples per second. As human speech is located below than 10000 Hz [20], we only need the maximum of 20000 samples per second to record all characteristics of human speech. Since representing amplitude in continuous values is confusing and resource-consuming, the quantization process is introduced to convert real values to integers, for example, a 16-bit PCM will be able to detail magnitude from -32768 to 32767.

![Figure 2.1: Three basics steps of sampling speech signals, inspired by [15]](image)

In practice, researchers usually avoid modeling raw audio signals because it ticks so quickly, typically 16000 samples per second (i.e. 16000 Hz) or more [105]. Furthermore, the signals are also different among recording devices, hence, a careful normalization and calibration are often needed [77]. As a result, the signals are transformed into time–frequency domains which are also known as spectral representation [37].

Fig. 2.2 describes the overall process of extracting spectrogram for time-frequency analysis. First, the speech signal is enhanced using pre-emphasis (Eq. 2.1) to boost the energy in the high frequencies because the voiced segments have more energy at lower frequencies than higher frequencies (i.e. spectral tilt [59]). Increasing high-frequency energy introduces more information to the acoustic model [14]

$$y_t = \alpha x_t + (1 - \alpha)x_{t-1},$$ (2.1)

where $x_t$ is sampled signal, and $\alpha$ is the coefficient of pre-emphasis filter which is chose
CHAPTER 2. SPEECH PROCESSING FOR LANGUAGE IDENTIFICATION TASK

![Image of speech processing workflow]

Figure 2.2: Step-by-step for extracting audio spectrogram in time-frequency analysis.

from 0.9 to 1.0, the lower the value, the more weights accumulated from previous time steps.

2.1.1 Segmentation and windowing

Moreover, speech is not a stationary signal [18, 27], because speech signal has many frequency contents and all of these contents change continuously over time. As a result, processing the whole audio files would introduce uncontrollable noise and result in unstable spectrogram representation of speech [59]. Hence, we divide the signal into small enough frames that speech can be characterized in the spectral information, typically, 10-25 milliseconds. We also want to minimize the information leakage [21] during partition process, hence, we only shift 5-10 milliseconds between successive frames. We use windowing to segment long audio into segments. Simply cut the files into multiple chunks can cause its spectrum segments develop non-zero values at irrelevant frequencies (spectral leakage [21]). One of the reason is that fast Fourier transform (FFT) [29] is a convolution operator which spreads the amplitude of true frequency to the frequency bins around it. As a result, we use Hamming windows [21], invented by Richard W. Hamming. The windows were commonly used in narrowband applications (e.g. telephone calls) which is the data used in NIST LRE’15 corpus. The algorithm reduces the long-distance spreading and is suited for frequency-selective analysis, for example, human speech [21].
CHAPTER 2. SPEECH PROCESSING FOR LANGUAGE IDENTIFICATION TASK

2.1.2 Time-frequency analysis

The FFT is performed on each frame to compute their discrete Fourier transform (DFT) [29]. Fourier analysis decomposes the signal into a combination of simple waves, as a result, Fourier transform represents signals in terms of the frequencies of the different waves made up that signals. Since the sound is a propagated vibration as a wave of pressure and displacement, it is synthesis of complex physical waves which is relevant for Fourier analysis [37]

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

transform a sequence of \( N \) complex numbers \( x_0, x_1, \ldots, x_{N-1} \) into an \( N \)-periodic sequence, \( k \) runs from \([0, N - 1]\), \( i \) is the imagine part of the complex number, and \( \pi \) is pi number.
≈ 3.14159. A number of FFT components, abbreviated as $N$, will be chosen in advance which represents $N$ discrete frequency bands. The process results in a complex number $X[k]$ representing magnitude and phase for each frequency component in the original signal. Since the variances of voiced sounds mostly encode in the differences of magnitude among frequency banks, we take the magnitude of all components by calculating the sum of squared real and imagine parts. As a result, we achieve spectrogram features at this steps which are already usable in many systems.

On the other hand, human perception of sound frequency is non-linear [99], we are less sensitive to high frequencies sound which are larger than 1000 Hz [99]. In 1937, Stevens, Volkmann, and Newman in 1937 introduced a non-linear, perceptual scale of pitches, named Mel-scale which comes from the word “melody”. The scale is defined as follow [99]

$$m = 2595 \log_{10} \left( 1 + \frac{f}{700} \right).$$  \hspace{1cm} (2.3)

Additionally, the Mel-filter bank showed Fig. 2.2 is the mapping from obtained spectrogram onto the Mel-scale, using triangular overlapping windows, which procedure uniformly distance within 1 kHz bins and logarithmic spaces after 1 kHz.

The logarithm of Mel-filter bank features are widely used in deep learning system for speech recognition [43, 44, 91], since its properties directly reflect human auditory system’s responses. The logarithm makes estimations less sensitive to minor variations in input (power variation due to the speaker), besides, human response to signal level is logarithmic [99], i.e. we are less sensitive to the changes in high amplitudes.

However, Mel-frequency cepstral coefficients (MFCCs) [110] is sometimes preferable because they are compact and robust to speaker variation. MFCCs are cepstrum coefficients (i.e. a nonlinear "spectrum-of-a-spectrum"), the core idea behind cepstrum is separating the source (F0 and details of the glottal pulse) and filter (articulators position) [110]. In general, speech is generated by a glottal source waveform passed through a vocal tract which because of its shape has a particular filtering characteristic [17]. Hence, the analysis of discriminative spoken languages only focuses on the diversity of the filter.

The process of extracting MFCCs is as following [110]:

1. Start from logarithm Mel-filter banks spectrogram.

2. Since we are going from the frequency domain back to time domain, we need to apply
inverse DFT to get the spectrum of log spectrum. We can achieve this by applying
discrete cosine transform (DCT).

3. The coefficients are the amplitudes of the resulting spectrum.

DCT produces highly uncorrelated features which make MFCCs very popular in audio com-
pression. In practice, we only select the first 13 coefficients since the later ones contain
unnecessary information about F0 spikes [59].

\[
\begin{align*}
\text{MFCC} & \quad 13 \\
\text{Log mel-filter banks} & \quad 40 \\
\text{Spectrogram} & \quad 129
\end{align*}
\]

Figure 2.4: Comparison between MFCC, log-mel filter banks, and spectrogram.

The comparison of 3 different feature types for LID is illustrated in Fig. 2.4. We can see
that spectrogram contains a lot of features (129 frequency components) but most of their in-
tensities are close to zero which are represented by black color. Log-mel filter banks feature
still encapsulates a wide range of details in speech, but the features also put more empha-
size on speech and make the non-voiced segments more distinguishable (i.e. the blue area).
MFCC features have the lowest amount of coefficients, however, it removes most of the re-
dundant information about non-speech and noise. As a result, MFCCs have higher diversity
and variation in colors which indicates richer representation.

### 2.1.3 Delta features

In many cases, the changes in features over time also contains critical patterns, especially for
dynamic signals. Hence, we often concatenate delta (speed) and double delta (acceleration)
to model the changes of spectrum over time. The derivatives are calculated along features
axis as follows [37]:

\[
d_t = \frac{\sum_{n=1}^{N} n(x_{t+n} - x_{t-n})}{2 \sum_{n=1}^{N} n^2},
\]

(2.4)
CHAPTER 2. SPEECH PROCESSING FOR LANGUAGE IDENTIFICATION TASK

Log mel-filter banks

Delta features

Double delta features

Figure 2.5: Calculating delta and double delta features for log Mel-filter banks features.

where $d_i$ is delta coefficients, $N$ is number of frames over which to compute the delta features, typically, 9 frames. From Fig. 2.5, we can see that introducing delta and delta-delta features enhanced the details of spectrogram over time. As a result, we calculate delta and double delta for all of our features.

2.2 Acoustic approaches to LID

Acoustic systems are low-level systems since they directly use the spectral information (or raw waveform [90]) as representative features for recognition or classification tasks. However, as detailed in the study [64], acoustic features contain the differences in languages and encapsulates distinctive linguistic structures with individual preferences (i.e. speaker variabilities). Moreover, speech isn’t static pictures of the spectrogram, it changes over time, and the same speech provided by the same person is always different [64]. Hence, a conventional acoustic approach involves many additional steps to transform spectrogram into rich representative space that benefits the tasks [34, 65, 86]. Fig. 2.6 illustrates the most popular acoustic approach, an i-vector approach, which contributes to many state-of-the-art LID recently [25, 33, 65, 86].

i-vector is low-dimensional hidden variable vector in the total variability space [25, 33]. Classical joint factor analysis (JFA) [33] models two distinct spaces for languages and channels which correspondingly encapsulate the language and channel variabilities [25]. Any language is assumed to be representable by the affine combination of two independent vectors from these two spaces. However, this assumption does not hold in many situations, hence,
CHAPTER 2. SPEECH PROCESSING FOR LANGUAGE IDENTIFICATION TASK

Figure 2.6: Simplified block diagram of conventional i-vector extraction followed by classifying or scoring backend, inspired from [65, 86].

The presentation of a combined total variability space in [33]. In this space, the Gaussian mixture model (GMM) [8] supervector of each language given an utterance (M) is represented as [33]

\[ M = m + Tw, \]  

(2.5)

where \( m \) is the utterance-independent component (the universal background model (UBM) supervector), \( T \) represents the inter-utterances variability which is a rectangular low-rank matrix [25], and \( w \) (total factors [33]) is an independent random vector of distribution \( \mathcal{N}(0, I) \). The extracted i-vector is then the mean of the posterior distribution of \( w \). In this modeling, \( M \) is assumed to belong to the distribution \( \mathcal{N}(m, TT^T) \), and the i-vector is the estimation of the residual variability, not captured by the total variability matrix \( T \), in our case, the inter-languages variabilities [80].

The overall process of deploying i-vector system is detailed in [26, 80, 86], and can be summarized as follow:

1. Each audio segment is processed according to Sec. 2.1. For LID, the system typically uses MFCCs or shifted delta cepstra (SDC) [103] which is actually stacking the static features with delta cepstra computed across multiple speech frames.

2. Training language-independent GMM (the universal background model - UBM), in order to model the essential of all languages.

3. GMM-posterior and the feature vectors in the segment are used to accumulate zeroth (index of Gaussian component), first (mean), and second (the variance) order sufficient statistics (SS). Then, these SSs are used to estimate low-dimensional i-vector representation using a total variability matrix, \( T \).

4. The i-vector is whitened to zero mean and unit length, using global mean, and the inverse square root of a global covariance matrix [86]. Additionally, within-class covariance normalization (WCCN) [26] is applied to compensate unwanted intra-class
variations in the total variability space.

5. From this point, there exist different strategies to incorporate i-vector features and the labeled data for training a LID backend (i.e. the algorithm matching each utterance to a language). The simplest approach is calculating the similarity score between a model representing a language and test i-vector [86]. On the other hand, a classifier can be trained to use i-vector for estimating the languages likelihood given an utterance [65].

During the process, the UBM $m, T$ are hyper-parameters of the system and often evaluated by a validation set. Moreover, they are trained in unsupervised manner, in order to encapsulate feature distributions and the total variances, hence, a similar unlabeled dataset can be used to estimate them.

### 2.3 Phonotactic approaches to LID

Since the primary goal of speech is linguistic communication, identifying languages based on the linguistic differences is a feasible approach. However, each language is constructed of distinctive structures include letters, words, grammar, and syntax rules. Recognizing millions of these compositions from many different languages is intractable. As a result, a smaller unit is defined as the phone [13], it is a unit of sounds, which distinguishes one word from another in all languages. Using a phone recognizer can transcribe any speech into sequence phones, then decoded into words, and sentences [83]. Then, the systems model the phonetic distribution and use it as the morphological rules to determine how close a transcription to a language.

![Figure 2.7: Block diagram of general design for phonotactic LID systems, inspired from [83].](image)
As illustrated in Fig. 2.7, phonotactic system also requires language model to identify each language from the linguistic point of view. The model is an \textit{n-gram} model \cite{83}, which represents the languages as a sequence of concurrence phones and their statistics. For instance, if \( n = 2 \), we have bigrams model as follow \cite{83}

\[
\hat{P}(w_1, \ldots, w_n) = \prod_{i=1}^{m=2} \hat{P}(w_i|w_{i-1}).
\]  

(2.6)

In the final stages, the utterance phones distribution is compared to the distribution of each language model, a score represented similarity of the utterance to each language is computed for making the final decision. These components are combined together for a phone recognition language modeling (PRLM) approach \cite{83}.

### 2.4 Acoustic versus phonotactic approaches

Most of the state-of-the-art LID systems use acoustic features \cite{25, 56, 65, 97}. Drawbacks of phonotactic recognizers can be summarized into three points:

- Building this system requires external data with information about the alignment of phones within each audio segment. In “closed conditions” of NIST LRE’15, only the given corpus and \textit{Switchboard-1} \cite{2} corpus are allowed \cite{78}, but the data only contains English transcription, hence, we lack of the sufficient statistics for phones distribution of other languages.

- In order to form reliable language models, the system must relies on phonetic recognizer that converts speech segments into sequences of phones. Since the number of phones and its acoustic diversity become exponentially complex as the number of languages increase, we introduce additional error and bias to the system.

- Training the system involves creating a \textit{n-grams} language model, which represents the probability of a phone given \( n \) previous ones. Thus, the task requires collecting a large enough corpus for calculating reliable statistics of \textit{n-grams} model. Due to a large amount of combinations of phones, repeating the process for each language is time-consuming and also resource-consuming.

- The thesis is concentrated on validating the performance of an end-to-end approach to LID, we can see from Fig. 2.6 that an acoustic system can be “shortcuted” and
significantly eliminates the burden of hand-crafted features. On the other hand, the phonotactic system is a multi-modal system, hence, the approach requires multiple inputs with different characteristics and multiple outputs for different purposes which make the task of end-to-end it complicated and unsound.

As a result, we follow the acoustic approach to pre-process audio files into the spectrum which encapsulates most of the relevant detail for speech characterization.
Artificial neural networks (ANNs) [95] are computational models which are inspired by biological nervous systems of animal brains. These methods provide a powerful framework to estimate or approximate unknown functions that can depend on a large number of inputs and parameters [54]. The evolution of neural network started from the 1940s and has significantly accelerated at the end of the 20th century [107], which results in many breakthroughs in artificial intelligence in the last decade [69]. A modest illustration of this overall process is showed in Fig. 3.1.

Modern deep learning techniques can learn multiple levels of abstraction from input features, and form very complicated representations that are important for specialized objective and suppress irrelevant variations [69]. The key aspect of deep learning concept is that the learned features are not handcrafted by human engineers: they are optimized from data using a general-purpose learning procedure [69].

This chapter describes the core parametric function approximation technology that is behind nearly all practical applications of deep learning to speech processing. We begin by describing the feedforward deep network model that is used to represent these functions. Next, we present more specialized architectures for scaling these models to large inputs such as high-resolution images or long temporal sequences. We introduce the convolutional network for scaling to large images and the recurrent neural network for processing temporal sequences. Finally, we present general guidelines for the practical methodology involved in designing, building, and configuring a LID system involving deep learning, and review some of the approaches.
CHAPTER 3. DEEP LEARNING

1940s
The early idea
McCulloch and Pitts: bridging logical calculus and nervous activity

1950s
The dawn of ANN
Frank Rosenblatt and his linear threshold perceptron, the perceptron is trained by simple logic rule.

1960s
The Golden Age
Ivakhnenko and Lapa applied thin but deep network with polynomial activation, the network was layer-by-layer trained with least square cost.

1970s
The quiet years
Minsky and Papert proved that ANN suffers from the same flaw as the perceptron; namely the inability to compute certain problem such as XOR

1980s
The renaissance age
Hopfield network was introduced, which is important design for modern recurrent neural network

1990s
Advances and improvements
Backpropagation is widely applied for training multilayer neural networks

2000s
Revolution and breakthrough
Exploring the issues of training deep networks, gradients exploding and vanishing, saddle point and local minima

The research focus on reducing overfitting, improving and stabilizing training speed. Unsupervised learning and generative network are also received attention.

Figure 3.1: Evolution timeline of artificial neural network, a study from [75, 76, 88, 95]

3.1 Feedforward neural network

Feedforward network (FNN), known as multi-layer-perceptrons (MLPs) or densely connected network (DCN), are the basic architecture of deep learning. The goal of a feedforward network is to estimate or approximate unknown function $f^*$. A multilayer neural network has been proven to be a universal approximator under a series assumptions for an accurate
estimation [40, 60, 67], these include: enough number of parameters, optimization result a
global minimum, sufficient training examples, and the priori class distribution of training set
must be representative of the whole data set. As a result, the model is a powerful framework
for supervised paradigm which maps an input vector $x$ to a category variable $y$. By approx-
imating $y^*$, a feed-forward network defines a mapping $y = f(x; \theta)$ and adjusts the value of
parameters $\theta$ that are optimized for certain objective tied to a supervised task.

![Figure 3.2: Perceptron, simplest version of feedfoward network with only one neuron](image)

FNN is called *network* because it is typically built by composing together many different
computation units. Each of these units is called “Neuron”, the simplest version of the network
contains only one neuron which is also called *perceptron* [88] illustrated in Fig 3.2. The input
to a neuron is a multi-dimensions vector $x = (x_1, x_2, \ldots, x_n)$, each dimension is weighted by
appropriate parameter (e.g $w_1, w_2, \ldots, w_n$) which is represented the connection from input to
neuron. These adjustable parameters are real numbers that can be seen as “knobs” controlling
the network outcome.

A neuron is the essence of the neural network, it intuitively transforms inputs into useful
information. The general structure of a neuron is the combination of 2 components: an
algorithm to combine weighted inputs, and an activation function. Most of the neurons
use linear affine transform for all weighted inputs together with a bias unit. An activation
function or “squashing” function is used to transform the output into the desired domain.
The function can be a linear or non-linear function, and one of the most common is *sigmoid
function*. It is often used to represent probability value because of the $(0, 1)$ output domain.
More details about activation functions will be presented in Sec. ??.

A more sophisticated model associates neurons into a directed acyclic graph. This graph
has the hierarchical architecture which is composed of layers. As a result, the first and the last
layer of a network is input and output layers, respectively, the middle layers are hidden layers. Each layer contains one or more neurons, since the layer try to expand the representation of input into multi-dimensional space. Fig 3.3 illustrates a densely connected network of three

![Feedforward neural network diagram](image)

Figure 3.3: Feedforward neural network

layers (i.e two hidden layers and one output layer). For instance, the network approximates the mapping function \( y = f^*(x) \) by performing a series of transformation

\[
y \approx f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))),
\]

where \( f^{(1)}(.) \) is the output of the first layer taking in the original input, \( f^{(2)}(.) \) is the output of the second layer taking the results from previous (first) layer as its input, and so on.

This chain structure forms a flexible and general-purpose learning procedure that can be extended to discover the intricate pattern in high-dimensional data [69]. During the optimization process, each layer extracts a different level of abstracted representation, and all of these representations are optimized for the same objective which is to amplify the information learned from the input [69, 95]. Therefore, the model removes the burden of handcrafting the feature extraction, so it can be benefit from increasing amount of available computation and data.

In practice, an objective function is used to measure the error between network output
$f(x)$ and the target variable $y$. This objective is differentiable [69, 95], hence, the network can compute the gradients of the parameters with respect to the error of mis-approximation [95, 104]. This process is called backpropagation, and illustrated by horizontal gradient line in the top of Fig 3.3. It is notable that the strength (i.e $L2$-norm value) of the gradient signal at each layer decrease as its relative position to the output layer. Hence, the higher level of abstraction, which directly affects the approximation, is learned at the top layers, and more robust representation of the input is preprocessed at the beginning layers. Overall procedure of computing backpropagation is illustrated on the top of Fig 3.3, the calculation is applied to each layer according to the chain rule in calculus, this process is detailed in the next section.

### 3.1.1 Backpropagation

Backpropagation is gradient-based learning methods [70]. Following the process in Fig. 3.3, for each input $x_i$, we compute an objective function $f_o$ (a differentiable function) between the network output $f(x_i)$ and the target variable $y_i$

$$E_i = f_o(y_i, f(x_i)),$$

where $E_i$ is the measure of discrepancy between the desired output and the actual output of the network. The average cost function,

$$E_{\text{train}} = \frac{1}{n} \sum_{i=0}^{n} E_i,$$

is the mean of all training examples’ error given a set of $n$ input/output pairs [70]. In practice, fitting the whole dataset into memory is nontrivial and impossible in many cases. Furthermore, repeatedly calculating the cost over the whole training set every iteration is very slow. Especially, when the cost surface is non-convex and high dimensional with many local minima, saddle points or flat regions because of non-linear ANN outputs [70], a gradient-based algorithm requires significant amount of iteration in searching for reasonable convergent points. As a result, we define a subset of the dataset, a “mini-batch” ($1 < n_{\text{batch}} < n$), then, we slice the dataset into many mini-batches and iteratively train the network on them. This approach is called mini-batch learning, in contrast to stochastic learning, in which $n_{\text{batch}} = 1$. Since the mini-batch learning approaches are more developed in the field [62, 70, 102, 108], and are more hardware-friendly because it significantly reduces the I/O operation and throughput during training by grouping data points and loading them into the
memory at the same time.

Backpropagation is based on the chain rules of calculus \[5\], let \( F = f \circ g \), or \( F(x) = f(g(x)) \), then we have

\[
F'(x) = f'(g(x)) g'(x), \quad \text{or} \quad \frac{dF}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}. \tag{3.4}
\]

Applying this rule to optimize our network parameters, for a network with \( L \) layers, we have \( X^{l-1} \) is the input to the \( l^{th} \) layer, and \( W^l \) is the weights matrix of the \( l^{th} \) layer. Then, the output of a layer can be represented as

\[
X^l = f^{(l)}(X^{l-1}). \tag{3.5}
\]

Starting from the output layer, since we calculated the cost for each data point \( X_i^0 \), we can directly take the partial derivatives of \( E_i \) with respect to \( W^L \)

\[
G^L_i = \frac{\partial E_i}{\partial W^L}, \tag{3.6}
\]

where \( G^L_i \) is the gradient matrix of \( W^L \) at the \( i^{th} \) data point. For \( W^{L-1} \), we have

\[
X_i^L = f^{(L-1)}(X_i^{L-1}), \tag{3.7}
\]

hence, the gradient of \( W^{L-1} \) become

\[
G_i^{L-1} = \frac{\partial E_i}{\partial X_i^L} \cdot \frac{\partial X_i^L}{\partial W^{L-1}}, \tag{3.8}
\]

Repeating the same computation for the 2\(^{nd} \) layer from the output,

\[
G_i^{L-2} = \frac{\partial E_i}{\partial X_i^L} \cdot \frac{\partial X_i^L}{\partial X_i^{L-1}} \cdot \frac{\partial X_i^{L-1}}{\partial W^{L-2}}, \tag{3.9}
\]

and recursively applying this rule, we can achieve a more general equation for the gradient of the \( l^{th} \) \((l < L)\)

\[
G_i^l = \frac{\partial E_i}{\partial X_i^L} \cdot \prod_{j=1}^{L-l} \frac{\partial X_i^{L-j}}{\partial X_i^{L-(j+1)}} \cdot \frac{\partial X_i^{L+1}}{\partial W^l}. \tag{3.10}
\]

After getting the gradient values of all parameters, the simplest learning procedure to minimize the cost value is gradient descent algorithm \[70\], the algorithm iteratively updates
each weights matrix by the following rule

$$W^l(t) = W^l(t - 1) - \eta \cdot \frac{1}{n_{\text{batch}}} \sum_{i=0}^{n_{\text{batch}}} \frac{\partial E_i}{\partial W^l(t - 1)},$$

(3.11)

where $W^l(t - 1)$ if current parameters of the $l^{th}$ layer, $W^l(t)$ is the new parameters, and $\eta$ is the learning rate which defines the learning speed of our network. Since $\eta$ is a hyper-parameters, it is good practice to select a low-value $\eta$ then slightly increase the learning rate and check the convergence of our network (i.e. the validating cost on the validation set is decreasing). If $\eta$ is too big, the network will fail to convergent and the cost value will fluctuate since it cannot reach a reasonable minimum [70]. In fact, it is suggested to have different learning rate for each parameter [62, 70, 102, 108], the strategy has been empirically proved to significantly speed up the training process [62, 70, 102, 108], it also remove the burden of selecting appropriate learning rate by an adaptive $\eta$, and slightly boost the overall performance in some cases.

### 3.1.2 Training a neural network

A general procedure of training a neural network using gradient-based methods is specified in Alg. 1. The algorithm iterates over the whole dataset for a fixed number of the epochs. During inference process (i.e. making the prediction), only the forward pass is performed and none of the parameters is updated.

It should be emphasized that we are more interested in the generalized ability to new data which have never been observed in the training set. In order to evaluate the overall performance, we use test set which is totally disjointed from the training set, and none of the network parameters or hyper-parameters should have any connection to the test set. On the other hand, training a neural network involves optimizing series of parameters and hyper-parameters, since the backpropagation algorithm only optimize the objective with respect to parameters (weights), the hyper-parameters must be selected by heuristic search and trial-error method. Fig. 3.4 details the training process from data preparation to network training and evaluation.

Moreover, learning rate imperatively contributes to the final result of neural network, its effect is viewed in Fig. 3.5(a). As we want the algorithm to perform well on unseen data, we want to maximize the performance on the validation set, since our assumption is that all three
Algorithm 1 General learning procedure of neural network

Require: initialize all weights $W(0)$ (sufficient small values is important [70])

for $1$ to $n_{\text{epoch}}$ do
  shuffle-training-set # suggested in [70]
  for mini-batch to training-batches do
    # Forward pass
    mini-batch = normalize-data(mini-batch) # suggested in [70]
    prediction = network-output(mini-batch | $W(t - 1)$)
    error = objective-function(target, prediction)
  end for
  # Backward pass
  gradients = $\frac{\partial error}{\partial W(t - 1)}$
  gradients = apply-constraint(gradients) # prevent grad. vanishing, exploding [101]
  $W(t) = \text{update-algorithm}(W(t - 1), \eta, \text{gradients})$
end for

# validating can be in the middle or in the end of an epoch
if need-validation then
  for mini-batch to validating-batches do
    # only forward pass
    prediction = network-output(mini-batch | $W(t)$)
    $score_{\text{batch}} = \text{scoring-function}(\text{target, prediction})$
  end for
  if is-generalization-lost($\text{mean}(score_{\text{batch}})$) then
    if no-more-patience then
      early-stop-training
    else
      decrease-the-patience-value
    end if
  end if
end if

# evaluating model using test set (inference process)
for mini-batch to test-batches do
  # only forward pass
  prediction = network-output(mini-batch | $W(\text{best})$)
  $score_{\text{batch}} = \text{scoring-function}(\text{target, prediction})$
end for
if is-the-best-score($\text{mean}(score_{\text{batch}})$) then
  pick-the-model
else
  reject-the-model
end if
sets (training, validating and test set) are homogeneous and come from the same distribution. Fig. 3.5(b) highlights the negative impact of under-training and over-training by selecting too small or too large the number of the epochs. In underfitting scenarios, the model is trained for an insufficient time period, hence, it hasn’t learned representative patterns in the training which results in poor performance on validation set (i.e. low generalizability) [70]. On the contrary, overfitting is the phenomenon that the model “learns by heart” everything in the training set, included noise and irrelevant patterns, as a result, the validating cost start going up as we train further [70].

![Training process of neural network.](image)

**Figure 3.4:** Training process of neural network.

![Choosing a reasonable learning rate (left) and comparing the effect of underfitting and overfitting (right).](image)

**Figure 3.5:** Choosing a reasonable learning rate (left) and comparing the effect of underfitting and overfitting (right).

### 3.2 Activation function

Activation function, also known as transfer function or squashed function, aims for creating non-linear decision boundary which enables the network modeling more complicated data
Table 3.1: Four most popular activation functions which are used in our networks.

<table>
<thead>
<tr>
<th>Description</th>
<th>Equation</th>
<th>Gradient Derivative</th>
<th>Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sigmoid:</strong></td>
<td>$\sigma(x) = \frac{1}{1 + e^{-x}}$</td>
<td>$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$</td>
<td>![Sigmoid graph]</td>
</tr>
<tr>
<td>Input domain: $(-\infty, +\infty)$</td>
<td>Output domain: $(0, 1)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Softmax:</strong></td>
<td>$\sigma(x)<em>i = \frac{e^{x_i}}{\sum</em>{k=1}^{K} e^{x_k}}$</td>
<td>$\frac{d\sigma(x)_i}{dx_j} = \frac{\sigma(x)_i \cdot (1 - \sigma(x)<em>j) \cdot \delta</em>{ij}}{\sigma(x)_j}$ if $i \neq j$</td>
<td>![Softmax graph]</td>
</tr>
<tr>
<td>Input domain: $(-\infty, +\infty)$</td>
<td>Output domain: $(0, 1)$</td>
<td></td>
<td>Same as sigmoid for each class.</td>
</tr>
<tr>
<td><strong>Hyperbolic tangent:</strong></td>
<td>$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$</td>
<td>$\frac{d\tanh(x)}{dx} = 1 - \tanh^2(x)$</td>
<td>![Tanh graph]</td>
</tr>
<tr>
<td>Input domain: $(-\infty, +\infty)$</td>
<td>Output domain: $(-1, 1)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rectifier (or relu):</strong></td>
<td>$relu(x) = \max(0, x)$</td>
<td>$\frac{drelu(x)}{dx} = \begin{cases} 0 &amp; \text{if } x \leq 0 \ 1 &amp; \text{if } x &gt; 0 \end{cases}$</td>
<td>![Rectifier graph]</td>
</tr>
<tr>
<td>Input domain: $(-\infty, +\infty)$</td>
<td>Output domain: $[0, +\infty)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is also used to transform the output of layer to a specific domain that matches prior knowledge defined by developer [19]. Tab. 3.1 list all four activation used in our model, includes sigmoid, softmax, hyperbolic tangent (tanh) and rectifier [106]. Since sigmoid and softmax output values from $(0, 1)$, they are used to model probability values. $tanh$ function is used when we need both negative and positive values which is in range $(-1, 1)$, this function is used to activate cell memory of long-short term memory network [53]. All of these three functions suffer from gradient vanishing issue during training a deep network [82], since its absolute value is always smaller than 1, and its gradient is the multiplication of small numbers which leads to a close-to-zero gradients.
On the other hand, rectifier function, introduced in [106], doesn’t suffer from gradient vanishing. We can see its gradients are fixed at 1 when \( x > \varepsilon \), this not only stabilizes the training process but also significantly speeds up the gradients calculation (backward pass) [106]. It has been shown that the networks with rectifier activation can remove the burden of pre-training in deep networks [106].

### 3.3 Objective function

The purpose of the objective function is creating a measurable and optimizable quantity which represents the difference between network estimation and true values. In our task, we have to measure the divergence between two discrete distributions (i.e. categorical distribution [4]). The target variable is an integer number which represents the index of each language and is encoded into a one-hot vector [12]. The output of the network is a vector of probability values from softmax [16], which represents the confidence value of the network for each language. Consequently, our objective must be appropriate for measuring the distance between an integer sparse vector and a continuous probability vector. The following objectives have been showed to be suitable for the task and are applied in our experiments.

**Definition 3.3.1 (Categorical cross-entropy).** [7] For a batch of data \( X \), and each pair \((x_i, y_i)\) is the input and true value of output to the network,

\[
L(\theta| (X, y)) = -\frac{1}{n} \sum_{i=1}^{n} y_i \cdot \log(f(x_i, \theta)),
\]

where \( f(x_i, \theta) \) is the prediction given the training example and current parameters, and \( n \) is the size of mini-batch.

Since \( y \) is one-hot-encoded [12] vector, the term \( L(\theta| (X, y)) \) only maximizes the log-likelihood of true class label and ignore the rest of output information. In the case of an imbalanced dataset (i.e. skew distribution for languages), the dominant class backpropagates strongest gradient signal and drives the network parameters to the sub-optimal region, thus, degrades the generalizability.

In [32], it was suggested that a modified version of cross-entropy which takes into account the prior distribution of the training set, and scales the loss value appropriately for each class.
CHAPTER 3. DEEP LEARNING

Definition 3.3.2 (Bayesian categorical cross-entropy). for $K$ classes, and $p(y_i)$ is the probability of class $y_i$ given a batch of data.

$$L(\theta|(X, y)) = -\frac{1}{Kn} \sum_{i=1}^{n} y_i \log(f(x_i, \theta)) \cdot \frac{\log(f(x_i, \theta))}{p(y_i)}.$$ (3.12)

Alternatively, density estimation metrics, which often used to measure the distance between two continuous variables, can be used. One of the most popular metrics is mean squared error (MSE) [11]

Definition 3.3.3 (Mean squared error). The distance between network’s output vector and true output vector is calculated as

$$D(\theta|(X, y)) = \frac{1}{n} \sum_{i=1}^{n} (f(x_i, \theta) - y_i)^2.$$ (3.13)

The smaller this distance, the better our model fitted to training distribution. Additionally, Hellinger Distance (HD) [39] measures the distance between probability values which is independent of the dominating parameters, the discrete case is

Definition 3.3.4 (Hellinger Distance). for two discrete distribution $y$ and $f(x, \theta)$, the distance is defined as

$$H(\theta|(X, y)) = \sqrt{\frac{\sum_{i=1}^{n}(\sqrt{y_i} - \sqrt{f(x_i, \theta)})^2}{2}}.$$ (3.14)

Unlike cross entropy cost, MSE and HD measure the distance between all variable of prediction and target vector, and this distance is always positive. As a result, these two functions are less sensitive the skewness of training distribution since it equally takes into account the differences of all classes.

3.4 Convolutional neural network

Convolutional Neural Networks (CNN) [6] are variants of MLPs that are inspired by biological visual cortex [31]. Its ability to extract invariant feature representations in different aspects has greatly benefited many recognition tasks included visual and aural signal [69, 95].
In this chapter, we first investigate the key idea behind the development of this architecture, *convolution operator*. Then, we go into the detail of CNN and its application in learning more sophisticated representation.

### 3.4.1 Convolution operator

*Affine transformations* are the “Swiss knife” of neural networks to manipulate the input vectors. The network’s output is produced by the dot product of the network’s weights and its input, then, a bias vector is added to the output to allow the shift of activated values. Subsequently, a non-linear function is applied to project output to the desired space with certain properties. The same is performed for any types of input, which can be images, audio signal, or text. A feed-forward network flattens any multi-dimensions input tensor (i.e. the multi-dimensional array) into a 2-D matrix and learns richer representation.

Regardless the distinguish contents of different kind of signals, they share three essential properties that are critical to design a representation learner [35]:

- They are multi-dimensional tensors (e.g. time-frequency domain for audio signal, color-width-height for images, time-embedding for text).
- The ordering of one or more axes do matter (e.g. spatial axes of images, time axis for text and audio signal).
- There exist multiple views of the signal within the features representation (e.g. colors-channel of images which view the same geometrical object in different color, frequency axis of audio spectrogram represent the sound magnitude at different levels).

Unfortunately, ordinary affine transforms treat all feature dimensions equally regardless their order (because all inputs are flattened into 2D matrices), as a result, topological information is not exploited by the feed-forward “densely connected” network. Modeling the implicit structure of data is critical for many pattern recognition tasks, for instance, speech recognition, image classification and segmentation, hence, the application of discrete convolutions [35].

Convolution operator is the integral of the point-wise multiplication between two func-
CHAPTER 3. DEEP LEARNING

Consider functions \( f \) and \( g \):

\[
(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t - \tau) d\tau
\]

\[
= \int_{-\infty}^{+\infty} f(t - \tau) g(\tau) d\tau.
\]

(3.15)

A discrete convolution can be defined for functions on the discrete unit of time \( t \). The operator can be described as a weighted average of the function \( f(\tau) \) parametrized by \( g(t - \tau) \) at the moment \( t \). As \( t \) is sliding through \( f \), the weighting function emphasizes different regions of the input function. However, the symbol \( t \) is not necessary representing the time domain, but also the ordering in the input signal [6].

![Figure 3.6: Illustration of discrete convolution operator.](Image)

In a discrete configuration, \( g \) represents a sparse tensor (i.e. only a few weight units contribute to a given output unit). In other words, the same parameters are applied multiple times in a different location of the input. Fig. 3.6 provides a step-by-step calculation of convolution operator, the input grid is also called the input feature map, we have only one feature map in this example. A kernel (light blue area) is a parameter matrix which is sliding through the feature map during the computation. At each location, the product between each element of the kernel and the input element it overlaps is computed and the results are summed up to obtain the output in the current location.

The procedure can be repeated using different kernels to learn multiple output feature maps (Fig. 3.7) which represent diverse aspects of the signal. If the input has multiple feature maps (e.g. RGB channels of images), distinct kernels are equivalently convolved on each one of the feature maps, and the final representation will be the sum of all intermediate feature map created. As the kernel sliding over input feature maps, there are two parameters that control the characteristic of the output [35]:

<table>
<thead>
<tr>
<th>Input (( f ))</th>
<th>Kernel (( g ))</th>
<th>Output (( f * g ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 8 6</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>7 7 0</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>4 8 6</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
<td></td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
<tr>
<td>6 0 2 4</td>
<td>6 0 2 4</td>
<td>5 8 6 7 7 0 4 8 6</td>
</tr>
</tbody>
</table>

\[ = 6 \times 5 + 0 \times 8 + 2 \times 7 + 4 \times 7 = 72 \]

\[ = 6 \times 8 + 6 \times 0 + 7 \times 2 + 0 \times 4 = 62 \]

\[ = 6 \times 7 + 0 \times 7 + 2 \times 4 + 4 \times 8 = 82 \]

\[ = 6 \times 7 + 0 \times 0 + 2 \times 8 + 4 \times 6 = 82 \]
1. stride: distance between two consecutive locations of the moving kernel along each axis (when \( \text{stride} \geq 2 \) also known as sub-sampling the input signal).

2. padding: number of zeros added at the beginning and at the end of each axis before applying convolution. There are three common convolution modes: full - zeros are padded so the convolution starts at the first values, the signals do not overlap completely; valid - the convolution product is only performed where the signals overlap completely; same - padding so the output signal reversed the same shape as the input signal.

Fig. 3.8 illustrate the effect of strides and padding on convolution. With \( \text{padding} \geq 1 \), the convolution introduces boundary effects which smoothen the boundary of the output signal. On the other hand, \( \text{stride} \geq 2 \) sub-sampled the signal, and if \( \text{stride} \geq \text{size}_{\text{kernel}} \), we can lose information from input signal during the convolution.

### 3.4.2 Pooling

*Pooling* [35] is sub-sampling operation which are repeated on each feature map. Pooling also uses a sliding window to feed patch of the signal to a pooling function (e.g. a function taking the maximum of a sub-region). Pooling ideally works like a convolution operator,
Figure 3.8: The window (dash line) moving on input feature map with different strides and padding configuration.

but without parametric combination with the kernel, a simpler function is applied on feature maps to simplify the input and extract invariant structure. Fig. 3.9 illustrates a max pooling function that reduces the dimension of its input but selecting the maximum values within each patch of input.

### 3.4.3 Convolutional neural network

Convolutional neural network leverages the idea from convolutional operator and pooling to propose three architectural ideas for modeling correlation invariance within the signal [68]:

- Sparse connectivity: unlike fully connected neural network, CNNs only implement local connectivity pattern between neurons of adjacent layers. In other words, CNNs shifts a small computational window across all subregions to calculate activation val-
users for next layers, as illustrated in Fig 3.10. The strategy emphasize the importance of local correlation within the subregion of the signal.

- Shared weights: additionally, the same parameters (weights) are used for all subregion and form a feature map. Weights of the same color (Fig 3.10) are identical because of shared constraints. Replicating units in this way allows for features to be detected regardless of their position in the input. Moreover, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learned. The constraints on the model enable CNNs to achieve better generalization on many complex problems [69].

- (Optional) Spatial and temporal sub-sampling: purposely reduces the dimensions of each feature map. As a result, it outputs a lower-resolution, more compact and noise-reduced images which are less sensitive to translation and distortions [68].
While DCN uses multiple processing layers for extracting hierarchical representations that benefit the discriminative objective, CNN has the ability to extract local invariant features from a different aspect of input [89]. As a result, it has been applied to speech processing with many recent successes [80, 89, 91]. Moreover, the speech example is a sequence of time-aligned frames of an utterance, $X \in \mathbb{R}^{t\times f}$, where $t$ and $f$ are time and frequency dimensions of the speech features respectively. Typical convolutional network, Eq. (3.16), convolves its shared weight matrix $W \in \mathbb{R}^{(w\times h)\times n}$ with the full input $X$. A small patch of size $w \times h$ are spanned across time-frequency regions of signal to extract local correlations [80],

$$h_{1 \leq k \leq n} = f(\text{convolve}(W_k, X) + b_k).$$

The process repeats for $n$ feature maps to form a richer representation of the data.

On the other hand, CNN architecture heavily relies on data, learned convolutional weights are widely used for features extractor and transfer learning given similar tasks [79, 109]. As a result, we leverage its robustness to form invariant features during training. Our architecture does both temporal and frequency convolutions, as it was emphasized in [23, 89] that 2D-convolutions improve performance substantially on noisy data. Additionally, we notice that using stride parameters (i.e. shifting distance for each convolving operator) larger than 1 during convolution is more efficient dimension-reduction strategy than pooling. With sufficiently large filter size (i.e. at least double the stride value), striding forces CNN to learn more compact representation with a significant amount of computation reduced.

### 3.4.4 Backpropagation for convolutional neural network

During the forward pass, computing the convolutional operator involves the dot product of many small input patches with a same size weights matrix Fig. 3.6, and the process is repeated for each feature map. We can think of the whole process as repetition of small matrices multiplication, hence, the computational complexity of this operator is addition of dot product computation, which is $O(N_{rep} \cdot w \cdot h)$ ($N_{rep}$ if the number of repetition, $w$ is kernel width and $h$ is kernel height, $w$ and $h$ is independent of images size $(W, H)$). Conversely, the computation of DCN involves dot product of two big matrices, $O(W \cdot H \cdot N_{hidden})$. As a result, when the size of input is expanded (i.e. $W$ and $H$ are increased), CNN linearly scales the computation as $N_{rep}$ is higher for bigger input, however, the computation of DCN is exponentially elevated. Consequently, CNN can scale to larger input with lower computation in the forward pass.
CNN backward pass is the inverse of Fig. 3.10. We have $c_{in}$ number of input channels and $c_{out}$ number of output channels, our kernel tensor is $W_{ij}$ where $i$, $j$ is the index of input and output channels. The input image is segmented into patches, and $x_k^i$ represent the $k^{th}$ patch at the $i^{th}$ channel of the input image. Then, the output for the $k^{th}$ patch at the $j^{th}$ output channel is

$$y_k^j = f_a \left( \frac{1}{c_{in}} \sum_{i=1}^{c_{in}} x_k^i \cdot W_{ij} \right),$$

where $f_a$ is activation function. Subsequently, the gradients of $W_{ij}$ is calculated as following

$$\frac{\partial y_k^j}{\partial W_{ij}} = \frac{1}{c_{in}} f'_a \left( \sum_{i=1}^{c_{in}} x_k^i \cdot W_{ij} \right) \cdot \sum_{i=1}^{c_{in}} x_k^i.$$

In other words, the gradient of a kernel with respect to an image patch is an average of all the gradients from input channels. This process is repeated the same as the forward pass, which brings similar advantages in scaling to larger inputs.

### 3.5 Recurrent neural network

Feed forward neural networks and convolutional neural networks rely on the assumption of independence among the examples, and the entire states of the network are reset after each processed data point. If data points are related in time (e.g. segments from audio, frames from video, words from text sentences), the assumption fails and the network cannot model the critical structure of signal over time. Additionally, feed-forward networks can only process fixed length vectors, except that the convolutional neural network can convolve arbitrary input but will provide arbitrary output size also. Thus a more powerful sequential learning tools are desirable in many domains. Recurrent neural networks (RNNs) are connection-

![Figure 3.11: RNN applied to different sequential learning tasks.](image-url)

35
CHAPTER 3. DEEP LEARNING

ist models where connections between units form a directed cycle. Unlike standard neural network, RNN has its own internal states which are updated at each time-step. This property allows the network to selectively store information across a sequence of steps, which exhibits the dynamic temporal behavior of signal [72]. Unlike convolutional neural network, we can control number the output from RNN for specific sequential learning tasks (from left to right Fig. 3.11):

1. Sequence to one learning: is classification task which takes a sequence of input and output a category for each sample. One the important example is language identification from speech signals [104].

2. Sequence to sequence learning: is the task of training a mapping from sequence to sequence. An ideal example for this task is machine translation [69], which translate text from one language to other languages.

3. Sequence generation: is the modeling of input distribution, which then can be used for synthesizing new data.

In this section, we review different architectural designs of RNN, their merits, drawbacks and application in modeling sequential data especially speech signal.

3.5.1 Standard recurrent neural network

The most classical architecture for RNN is illustrated in Fig. 3.12 which is based on Eq. 3.18. Recurrent neural network recursively applies the same set of weights over a set

![Figure 3.12: Recurrent neural network unfolded.](image-url)
of time dependent examples. In some sense, RNN is the same as the feed-forward neural network if the maximum number of time-step is set to one. The structure of RNN can be characterized by following equation [54]

\[ h_t = F(x_t, h_{t-1}, \theta), \]  

(3.17)

intuitively, the input to the network is the input features and its own hidden states. Thus, RNN has a consistent hidden states along time axis, which allows it to remember and model temporal pattern. Conversely, DCN and CNN resets its states after each sample, this behavior forces them to exploit internal structure of signal and miss the important correlation between data points.

In practice, time signal arrives with separated parameters for time index, and the critical patterns can be correlated with indices \((t; t-1), (t; t-2)\) or lagged very far behind \((t; t-n)\). In order to cope the diverse of temporal dependency, RNN combines the input vector \(x_t\) with their state vector \(h_{t-1}\) to produce a next state vector \(h_t\) by using a learnable function with parameters \(\theta\). This strategy is repeated for every time-step \(t\) as following [54]

\[ h_t = f_a(\theta_x \cdot x_t + \theta_h \cdot h_{t-1} + b), \]  

(3.18)

the equation is the simplest form of RNN [100] with two weights matrix \(\theta_x\) and \(\theta_h\) in order to project input and hidden states into representative latent space, the bias \(b\) is also added to the model. \(\theta_x\) will be optimized by backpropagation through time (BPTT) [72]. The algorithm is the generalized version of backpropagation on time axis, it unrolls the recursive connection into a deep feed-forward network and backpropagates gradients through this structure.

As speech is the continuous vibration of the vocal cords (Chap. 2), its strong temporal structure is indisputable. Chap. 2 also reveals the complication of the speech signal which is composed of many frequency components simultaneously changed over time. As RNN structure reflects the strong characteristic of the speech signal (i.e. time dependency), it has been widely introduced into speech recognition fields with state-of-the-art performance [23, 44, 87].
3.5.2 Long-short term memory neural network (LSTM)

Ordinary RNN has convergence issues [82]. In practice, training the network often confronts the problem of vanishing gradient and exploding gradient problems as described in [82].

As a result, several architectures were proposed to address these issues. One of the most popular variants uses Gate units to control information flow into or out from the internal state. This is known as long-short term memory (LSTM) recurrent network [46]. The key to LSTM is memory cell which is regulated by gating units to update its state over time. Thus, LSTM network is capable of learning long-term dependencies and was proven to work tremendously well on a large variety of tasks [30, 46, 93, 101].

![LSTM architecture](image)

**Figure 3.13:** A LSTM architecture, as a flow of information through memory block which is controlled by input gate $i_t$, forget gate $f_t$ and output gate $o_t$

The modern architecture of LSTM is illustrated in Figure 3.13. The whole process can be interpreted as a flow of information vectors from left to right, which includes:

- $X_t$: input vector at time-step $t$.
- $h_{t-1}$: vector represents previous hidden state at the time-step $t - 1$.
- $c_{t-1}$: previous memory cell state from time-step $t - 1$ encoded as a vector.

The cell state acts like a “conveyor belt”, it runs straight down the entire information chain to create precise timing signal, also known as peephole [46]. The three vectors form 3 gating units as a “throttle” of information, these units regulate data vectors allowing modification of cell state to capture long-term temporal patterns. The modification includes: store
(i.e input gate $i_t$), remove (i.e forget gate $f_t$) and response (i.e output gate $o_t$). Unlike classical RNN, LSTM decouples the hidden states $h_t$ and the memory cell $c_t$ which doubles the memory capacity and allows the network to learn longer temporal pattern by created a dedicated memory to learn and forget subsequently. Modern architecture of LSTM, which is used in [30], is defined by the following system

\[
\begin{align*}
    i_t &= \sigma_i(x_t W_{xi} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i), \\
    f_t &= \sigma_f(x_t W_{xf} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f), \\
    \tilde{c}_t &= \tanh(x_t W_{xc} + h_{t-1} W_{hc} + b_c), \\
    c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
    o_t &= \sigma_o(x_t W_{xo} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o), \\
    h_t &= o_t \odot \tanh(c_t),
\end{align*}
\]

where $\odot$ represents element-wise multiplication operator, and $W_{-}$ denotes weight matrices (e.g $W_{xi}$ is the matrix of parameters mapping input $x_t$ to input gate dimension). The $b_{-}$ term denotes bias vectors, sigmoid is used to activate gate units and $\tanh$ is for cell memory activations. The idea behinds this system of equation can be intuitively explained as following:

1. The input gate $i_t$ and forget gate $f_t$ formulate learnable functions from new input $x_t$, and learned experience $h_{t-1}$, $c_{t-1}$. These gates are then activated into probability values using the sigmoid function.

2. The fourth equation highlights the brilliant idea behind LSTM. The input gate regulated new formed memory ($\tilde{c}_t$ - $i_t \odot \tilde{c}_t$), and the forget gate is used to select old memory $f_t \odot c_{t-1}$. The combination of these two terms not only form long-term memory but also combat the challenge of gradient vanishing, which will be explained in detail Sec. 3.5.4.

3. The network also learns a function to map from long-term memory $c_t$ to working memory $h_t$ which determines further action, the output gate is responsible for filtering appropriate information from cell memory that can benefit our prediction task, and encode it into hidden state $h_t$. 


3.5.3 Gated recurrent neural network (GRU)

According to [46], many variants of LSTM have been proposed since its inception in 1995. Each with their own merits and drawbacks performs differently in various tasks, however, the most known variant is the gated recurrent unit (GRU) [30]. GRU simplifies the LSTM architecture by coupling the input and the forget gate into update gate ($u_t$), together with reset gate ($r_t$) to schedule the update of hidden state, illustrated in Fig. 3.14. The performance of GRU can be comparable to LSTM [30], however, its design significantly reduces the number of parameters, as can be seen from following equation [30]

\[
\begin{align*}
    r_t &= \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r), \\
    u_t &= \sigma(x_t W_{xu} + h_{t-1} W_{hu} + b_u), \\
    \tilde{h}_t &= \tanh(x_t W_{xc} + r_t \odot (h_{t-1} W_{hc}) + b_c), \\
    h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t.
\end{align*}
\] (3.20)

3.5.4 Addressing gradient vanishing with LSTM and GRU

From Fig. 3.12, we can see that RNN is a very deep feed-forward neural network along the time axis. According to Eq. 3.18, the forward pass of RNN is a recursive process of applying the same function on its inputs and hidden states using the same parameters. Given a sequence of input with length $T$, at each time steps, the network will provide one output $y_t$ coordinated to its input $x_t$ ($1 \leq t \leq T$), hence, the error is calculated at each time steps is
Then, the overall objective for training RNN is

\[ E = \sum_{t=1}^{T} E_t \]

Using this objective, the gradient of \( \theta \) is calculated as following [82]

\[ \frac{\partial E}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial \theta} \]  \hspace{1cm} (3.21)

However, the recursive structure of RNN makes the computation of \( \frac{\partial E_t}{\partial \theta} \) involves multiplication of many terms, for instance, with \( t = 3 \) [82]

\[
h_3 = F(x_3, F(x_2, F(x_1, h_0, \theta), \theta), \theta)
\]

\[
\Rightarrow \frac{\partial E_3}{\partial \theta} = \sum_{k=1}^{3} \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \left( \prod_{i=k+1}^{3} \frac{\partial h_i}{\partial h_{i-1}} \right).
\]  \hspace{1cm} (3.22)

A more general equation from [82] is

\[
\frac{\partial E_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \left( \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right)
\]  \hspace{1cm} (3.23)

As the sequence length \( T \rightarrow +\infty \), we see two issues emerge [82]:

- The term \( \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \) may result a “close-to-zero” number if we use the \textit{sigmoid} or the \textit{tanh} activation, since it is the multiplication of many smaller-than-1.0 numbers.

- If the activation function provides gradient which is greater than 1, the number of summation in the gradients is exponentially increased as the sequence length rise. Hence, the final gradients can be huge and unpredictable.

Both of this points had been analyzed in [82].

Conversely, LSTM and GRU combat the issue by simple summation in their hidden update equation (i.e. the fourth equation of LSTM and GRU). For instance, in GRU, we have

\[
h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t,
\]

\[ \text{Where } u_t, \tilde{h}_t \text{ are the update gate and candidate hidden state respectively.} \]
then, the gradient of current states with respect to previous states is

$$\frac{\partial h_t}{\partial h_{t-1}} = (1 - u_t) \cdot \frac{\partial u_t}{\partial h_{t-1}} + \left( \frac{\partial u_t}{\partial h_{t-1}} \odot \tilde{h}_t + u_t \odot \frac{\partial \tilde{h}_t}{\partial h_{t-1}} \right). \quad (3.24)$$

Since $0 \leq u_t \leq 1$ and $0 \leq 1 - u_t \leq 1$, the gradient is converted to the summation of small numbers. Furthermore, the amount of information updated to new hidden states is regulated by two opposite learnable functions (the reset and the update gate), hence, it is rarely the case that both of them are significantly increased and causing gradients exploding.

### 3.5.5 RNN and Markov models

*Markov chains* [10], named after the mathematician Andrey Markov in 1906, is a stochastic process to model the transitions between states, and make predictions for the future of the process based on a subset of most recent states. However, the conventional Markov chains heavily rely on the assumption of a fully observable states space, which is unsound in many cases. As a result, *hidden Markov models* (HMMs) [9] have been widely used as a replacement in sequence learning (especially in speech recognition [38]). HMMs assume the process has unobserved (hidden) states, and try to model an observed sequence as probabilistically dependent upon a sequence of unobserved states [72].

In practice, Markov model stores its discrete state space $S$, which leads to a table of size $|S|^2$ for states transitions. Training Markov model involves updating the probability of transitions which will scale in time $O(|S|^2)$ [72]. This is a significant burden as the number of states rise unpredictable in many tasks. Furthermore, each hidden state has the only temporal dependency on the known numbers of previous states, and as the size of context increases, the size of models (i.e. states table and transition table) grows exponentially as well. As a result, Markov models are computationally impractical for modeling long-range dependencies [45].

RNNs, on the other hand, are capable of modeling long-range time dependencies [45], because any state of the network depends on not only the current input but also the internal states. Moreover, the hidden state does not store "hard" information related to previous states, but it encodes the temporal pattern of arbitrarily long context window using a learnable function. This is feasible because the hidden state is a continuous vector which can represent an infinite number of states, and as the number of nodes grows exponentially, its capability is also exponentially grown as well [72].
As a result, we investigate the use of both LSTM and GRU to select the best architecture for language identification task. Moreover, Feedfoward neural network (FNN), convolutional neural network (CNN), and recurrent neural network (RNN) are complementary in their modeling capabilities to capture different patterns. While FNN using multiple processing layers is able to extract hierarchical representations that benefit the discriminative objective, CNN has ability to extract local invariant features in both time and frequency domain. Conversely, RNN combines the input vector $x_t$ (i.e t-th frames of utterances) with their internal state vector to exhibit dynamic temporal pattern in signal. As sequence-training is critical for speech processing, conventional FNN approaches have been proven its inefficiency in both language and speaker identification task [47, 73]. Our observation shows that LRE’15 dataset contains long conversation with continual silence between each talk, hence, the frames-level features extracted by FNN introduce extra biases and noises to the network, as shown in Ch. 6.

### 3.6 Dropout

Deep learning [69] leverages multiple non-linear layers to learn distributed representation of data with high level of abstraction. However, its strong nonlinearity has a serious drawback on handling small or imbalanced dataset, the issue is also known as overfitting (illustrated in Subsection 3.1.2) [98]. The idea of ensemble multiple models for more powerful prediction has been applied since the early day of machine learning. However, a large network is a heavy computation model, which makes it difficult to train multiple predictors. As a result, dropout has been introduced in 2014 [98] as an architectural “hack” addressing this issue.

The key idea is to randomly drop activation values of some a selected set of layers during training, which also disconnects the gradients of dropped units. This prevents the units in the same layer from co-adapting to a fixed, easily formulated pattern [98]. Consequently, the dropout at network inputs results in a training process on an exponential number of augmented datasets, and dropout at layers is like sampling from exponential number of different "thinned" networks [98]. At the test time, the predictions are the average values of all these augmented datasets and thinned networks. The results from [98] showed that dropout surpasses other regularization methods (e.g. elastic net regularization, weights norm regulariza-
tion, ...) and significantly reduces overfitting.

\[
\text{mask}_i \sim \text{Bernoulli}(p, \text{shape} = (a_i)_{\text{shape}})
\]
\[
a_i = a_i \times \text{mask}_i
\]  
(3.25)

In practice, implementing dropout only involves random sampling a mask with the same shape of activations from a Bernoulli distribution as showed in Eq. 3.25, where \( p \) is the probability of dropped values (range from 0.0 to 1.0) and \( a_i \) is the activated values after one layer or inputs to the network. Then, the activations are re-scaled using retain probability which equals to \((1.0 - p)\)

\[
a_i = \frac{a_i}{1.0 - p}.
\]  
(3.26)

During the test time, all activation values are kept.

### 3.7 Batch Normalization

**Batch Normalization** (BN) has been introduced in [55] as a technique to accelerate and stabilize training process of deep networks. As the number of layers grow, the optimizer must simultaneously adapt to the change of all layers’ inputs during training. This phenomenon is known as “internal covariate shift” [55] which slows the convergent speed of training process. One intuitive solution to this issue is normalizing each layer’s inputs in order to stabilize their distributions and eliminate abnormal high (or low) input values which are the cause of saturating gradients [82]. As a result, each training mini-batch is normalized to zero mean and unit variance

\[
x_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \gamma + \beta,
\]  
(3.27)

where \( \gamma \) is the scale parameter which is often initialized at 1.0, and \( \beta \) is the shift parameter which is initialized to 0.0. This 2 parameters can be learned to better adapt the normalized feature space. The crucial part is that the mean (\( \mu \)) and variance (\( \sigma^2 \)) are computed across the batch dimension, i.e., over mini-batch of examples, not per example. During training, \( \mu \) and \( \sigma^2 \) are the current input mini-batch statistics, and during evaluation, they are replaced with running average statistics over the trained data. Consequently, the network is less sensitive to weight initialization, and we can train the network with higher learning rates [55].

44
3.7.1 CNNs and batch normalization

Applying BN to CNNs require minor modification to Eq. 3.27. The idea is applying different normalization for each input channel because the different input feature maps may represent the different aspects of the signal with distinctive statistics, hence, normalizing all of them to a unique space will create an imprecise representation. As a result, we use the following equation to normalize CNN

\[
y = f(\text{BN}_{\gamma, \beta}(h_{1 \leq k \leq n})) + \epsilon \\
= f \left( \frac{(h_{1 \leq k \leq n} - E[h_1])}{\sqrt{\text{Var}(h_{1 \leq k \leq n})}} \ast \gamma + \beta \right) + \epsilon,
\]

where \(\epsilon\) is residual term explained the differences between training and evaluation data, which is represented as Gaussian noise adding to the activation of CNN during training. The function \(f\) is rectifier function [106], and \(\gamma, \beta\) are network parameters updated during training which is then used to scale and translate features to a normalized space during evaluation. Furthermore, normalizing each feature maps over just current batch degrades the performance [23], hence, we store a running average of mean and variance collected during training, and use these values for the evaluation process [55].

3.7.2 RNNs and batch normalization

Since RNNs are optimized using backpropagation through time (BPTT) algorithm, it is an extremely deep neural architecture rolling in time. As a result, training RNNs on imbalanced dataset such as NIST LRE’ 15 need careful regularization. In this work, we also adapt batch normalization for RNN architectures. We specifically design batch normalization for LSTM, however, the same idea can be easily adapted to other design of RNNs.

A natural approach, inspired from [55], would insert the transformation directly before non-linear function for each gate at every time step. We show an input gate with standard batch normalization Eq. 3.27,

\[
i_t = \sigma_i(\text{BN}_{\gamma, \beta}(x_t W_{xt} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1})).
\]

However, this approach introduces additional computation per time step which leads to very complicated backpropagation [23]. Furthermore, each connection has different weights oper-
ating in different spaces with distinctive statistics which leads to no improvements in overall performance [23]. We find more appropriate normalization strategy in [23, 66], which is adapted for LSTM architecture. First, the input-to-hidden connections are normalized over the whole sequence of input $X$ by,

$$X_{\text{norm}}^{(i)} = \text{BN}_{\gamma, \beta}(X W_{xi}),$$
$$X_{\text{norm}}^{(f)} = \text{BN}_{\gamma, \beta}(X W_{xf}), \text{ and}$$
$$X_{\text{norm}}^{(o)} = \text{BN}_{\gamma, \beta}(X W_{xo}).$$

Then, normalized input at each time step are used to compute normal gate activation of the LSTM

$$i_t = \sigma(x_{\text{norm}}^{(i,t)} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i)$$
$$f_t = \sigma(x_{\text{norm}}^{(f,t)} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(x_t W_{xc} + h_{t-1} W_{hc} + b_c)$$
$$o_t = \sigma(x_{\text{norm}}^{(o,t)} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

where $\odot$ represents element-wise operator, and $W$ denotes weights matrices (e.g $W_{X_i}$ is the matrix of parameters mapping input $x_t$ to input gate dimension). The $b$ term denotes bias vectors, $\sigma$ is activation functions which often are $\text{sigmoid}$ for gate units and $\tanh$ for hidden activation.

This normalization approach computes the mean and variance statistics among all time steps over all examples in mini-batch, which results in more stable convergence of LSTM and significant performance improvements in our experiments.
CHAPTER 4

Deep learning approaches to language identification

As introduced in Chapter 1, there are two approaches for applying deep learning to LID tasks. The differences in the pipeline of each approach are illustrated in Fig 4.1. In this chapter,

![Diagram of indirect and direct approach LID systems]

**Indirect approach LID system**

**Direct approach LID system**

Figure 4.1: Comparison between indirect and direct deep learning approaches to LID, inspired by [65].

we investigate the influence of these designs to the LID tasks. As a result, we propose our improvement in the architecture for an end-to-end system.

4.1 The "indirect" approach

The “indirect” system is composed of two main components: the front-end for features extraction, and the back-end for language classification. DNN is mostly used in the front-end, and the network is trained to estimate the phonetic distribution from acoustic features. As a result, the extracted features are more of phonetically-balanced representation [57]. These features are then used to train a secondary classifier in the back-end for the LID task. In terms
of design, there are two different approaches for constructing a deep network in the front-end. The first one, known as the “bottleneck” features [86], proposes a special bottleneck layer which is closed to the output layer [57]. This strategy enforces a compact representation with the high-level of abstraction in the bottleneck as information flow from input to the output layer.

The second method (“DNN-posterior”) uses phones (or senones, i.e. a sequence of 3 phones, also known as tri-phones) posteriors extracted from the output of DNN to accumulate multi-modal statistics [86]. Subsequently, an i-vector classifier is trained using those
features, then, a scoring algorithm is applied to calculate the similarity of each utterance to target languages based on their super-vector features [57, 86]. The approach has been the key to achieve state-of-the-art performance on both SR and LID tasks [86].

4.2 The "direct" approach: an end-to-end system

The “direct” method replaces the middle four blocks or the whole pipeline by a DNN (Fig. 4.1). As an end-to-end system, deep learning removes the burden of hand crafting the feature extraction is the conventional approach in LID task. This versatility is achieved by integrating feature extractor and classifier into a single algorithm, and training the algorithm to learn distributed representations of speech features with multiple levels of abstraction that explicitly benefit the task. This learning strategy allows the network to be optimized to handle a wide range of speech diversity including ambient noise, speaker variation, and channels. In [73], it was found that a deep learning system surpassed i-vector based approaches with a lower number of parameters when a large amount of training data was available. However, the paper only tries to use a single network architecture. Conversely, it was reported in [91] that a combination of many deep architectures outperforms conventional deep learning approach to ASR.

As a result, the main subject of our study is adapting the most recent advances in DNN architectures for LID task. The key to our approach is the recurrent architecture of DNN, a model has recently been shown to outperform the state-of-the-art DNN systems for acoustic modeling in speech domain [23, 92, 93, 44]. The central idea behind recurrent neural network (RNN) is its feedback connection which creates an internal state to model temporal dependency in data which is essential in speech. The difference between traditional DNN approaches and our approaches is the way out network processing the acoustic features, which is illustrated in the middle of Fig. 4.4 (from left to right):

- Our approach is an RNN network which encodes a sequence of speech frames into its hidden states. In this case, temporal pattern of signal is preserved and learnt by the network.

- Multiple frames are stacked into a “super” vector. Then, a feed-forward network or convolutional neural networks is trained using the stacked features. Thus, the approach ignores the time axis and its specific order, hence, learning any temporal dependency
is more difficult.

Figure 4.4: Two different strategies (RNN: left, FNN: right) for applying end-to-end deep learning to language identification.

Finally, a softmax activation function projects hidden states of RNN into interpretable probability vector of the target languages [54],

$$
\varphi(y)_j = \frac{e^{y_j}}{\sum_{k=1}^{K} e^{y_k}},
$$

(4.1)

where $K$ is the total number of classes, vector $y_j$ is the affine transform of activations from the last hidden layer, and $\varphi(y)$ is the posterior distribution of the target languages. A hard decision can be made by selecting the most probable class. Moreover, another approach may transforms the posterior probability into log-likelihood ratio (LLR), which allows more flexible decision making process.
CHAPTER 5

Experiments in networks design

5.1 Speech corpus for LID

Spoken language recognition evaluation (LRE) campaigns, routinely conducted by National Institute of Standards and Technology (NIST), have a major effect on advancing the research in LID [65]. The newest LRE corpus was released by NIST in 2015, which is considered as a great challenge to the community due to its differences in certain key aspects [65].

The corpus contains \( \approx 796 \) hours of speech. The task requires identifying languages in the more ambiguous context of closely related languages in specific languages cluster. There are six different language clusters, which are combined of twenty languages. Fig 5.1 also indicates heterogeneous distribution among languages, clusters and even datasets which emphasizes the importance of prior information in NIST LRE’15. There exists not only the unbalanced distribution among languages but also the mismatch in prior distribution between the development set and the evaluation set. The competitors of NIST LRE’15 also have to deal with diverse audio conditions and qualities, from a phone conversation with ambient noise to a formal interview. Furthermore, our inspection on three random speech utterances from 3 clusters (Fig 5.2) suggests diverse structures in the audio clips, some utterances contain very long silences mixed with noise and the speech activities are occasionally short and meaningless [104]. This observation implies the importance of robustness in learning speech utterances representation, a distributed representation obtained by a deep network can represent many intermediate concepts that are useful to capture the statistical dependencies of input signal and output language.
NIST evaluates system performance on LRE’15 corpus using a closed-set scenario, where the set of non-target languages are limited to other languages in the same cluster [65]. The output from each system is a (20-dimensional) vector of log-likelihood-ratio (LLR) scores for each test segment. The objectives of LRE15 is minimizing the criterion in Eq. (5.1), which applies to each cluster and all of its target/non-target languages pairs \((L_T, L_N)\)

\[
C_{\text{avg}} = \frac{1}{N_{L}} \left\{ C_{\text{miss}} \cdot P_{\text{Target}} \cdot \sum_{L_T} P_{\text{Miss}}(L_T) \right\} + \frac{1}{N_{L} - 1} \left\{ C_{\text{FA}} \cdot (1 - P_{\text{Target}}) \cdot \sum_{L_T} \sum_{L_N} P_{\text{FA}}(L_T, L_N) \right\},
\]

where \(N_{L}\) is the number of language in the cluster, \(C_{\text{miss}}, C_{\text{FA}}\) and \(P_{\text{Target}}\) are application-specific parameters represent the weights of detection miss and false alarm probabilities. For LRE15, the application parameters will be: \(C_{\text{miss}} = C_{\text{FA}} = 1\), and \(P_{\text{Target}} = 0.5\). This objective is used throughout all experiments in this thesis as a criterion to judge the final performance of each system. Furthermore, our intention in this work is constructing the most applicable LID system given a corpus, hence, the training data for any algorithm is only limited to the set illustrated in Fig 5.1. Since the primary cost function of NIST LRE’ 15 is applied separately for each language cluster [78], we decided to train different networks for each cluster, and the final \(C_{\text{avg}}\) is an average of all clusters’ scores.
CHAPTER 5. EXPERIMENTS IN NETWORKS DESIGN

Figure 5.2: Waveform of three randomly sampled audio files, which indicates very long silences between speech.

5.2 Setup

The output of our system is a (20-dimensional) vector of log-likelihood-ratio scores for each test segment. The objectives of LRE15 is minimizing the criterion in Eq. (5.1), which apply for each cluster and its pairs of target/non-target languages ($L_T, L_N$): In practice, we trained our model using cross-entropy loss (Eq. 5.2) to categorize each training frames to its true language label.

\[ L = - \sum_i t_i \log(p_i) \]  

(5.2)

where $t_i$ is one-hot-encoded true class label, and $p_i$ is output vector contained predicted probabilities for each class. Acknowledge the nontrivial difference between the distribution of training and evaluating dataset, we propose a dataset partition strategy specified in Tab. 5.1. Note that all dataset are randomly split and completely non-overlap.

Additionally, we use batch learning [70] because of the huge dataset. We also shuffle the whole dataset after each epoch to accelerate the learning process and prevent the parameters converged to a fixed local minima [70]. Data from each audio files is normalized to be centered at zero with standard deviation one, using Eq. 5.3, where the mean and std are calculated among all samples (i.e. first axis).
Table 5.1: Splitting scheme for training and evaluation corpora

<table>
<thead>
<tr>
<th>Alias</th>
<th>Corpus</th>
<th>Partition</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set01</td>
<td>Dev.</td>
<td>20%</td>
<td>Validating training results on training dataset</td>
</tr>
<tr>
<td>Set02</td>
<td>Dev.</td>
<td>80%</td>
<td>all networks were trained using this data set</td>
</tr>
<tr>
<td>Set03</td>
<td>Eval.</td>
<td>20%</td>
<td>Tuning, features selection, model selection</td>
</tr>
<tr>
<td>Set04</td>
<td>Eval.</td>
<td>80%</td>
<td>Evaluate overall performance of system</td>
</tr>
</tbody>
</table>

\[ X = \frac{X - \text{mean}(X)}{\text{std}(X)} \]  

(5.3)

We also use accuracy score (Eq. 5.4) as our criterion during training, because the measurement is faster to calculate and the value also reflect actual LRE15 evaluation score

\[ \text{accuracy}_{\text{per_sequences}} = \frac{\sum_i \mathbb{1}(t_i = \arg\max(p_i))}{n}, \]  

(5.4)

where: \( \mathbb{1} \) is identity function and \( n \) is the number of samples. It is notable that we compute this score for each example and one utterance can be segmented into many examples with the same label. Further study will average softmax probabilities from all samples of each utterance as the final score, and \( C_{\text{avg}} \) on these scores are used for evaluation as we need a more precise criterion to compare our network to the baseline i-vector approach.

### 5.2.1 Initializing networks’ parameters

All networks are initialized using the same random seed to remove randomness on the results. As suggested in [70], the initialization can have a significant effect on the convergent process of the weights. If the weights are too large or small, the result gradients will be very small which leads to slow convergence and probably saturates the learning process [82]. We use different initializing strategy for different architecture.

For feed-forward (fully connected) layers, we sample the weights from a symmetric uniform distribution which is centered at zero, the parameters of the distribution is based on
the shape of the weights following

\[
a = \mu - \sqrt{3} * \sigma \\
b = \mu + \sqrt{3} * \sigma
\]  

(5.5)

\[W \sim Uniform(low = a, high = b, size = W_{shape}),\]

where \(\mu\) and \(\sigma\) are chosen in advance which are 0 and 0.01 in our case.

On the other hand, CNNs are initialized using \textit{Glorot uniform distribution} invented by Xavier Glorot in [42] (Eq. 5.6). \(g\) or \textit{gain} are scaling factor for the weights, which are suggested to be 1.0 for sigmoid or linear activation, and \(\sqrt{2}\) for rectifier activation. \(fan_{in}\) is the number of input units which includes both the input channels and the output channels. \(fan_{out}\) is receptive field sizes which are width and height of the convolutional kernel

\[
a = g \sqrt{\frac{2}{fan_{in} + fan_{out}}},
\]

(5.6)

\[W \sim U[-a, a] \text{ or } W \sim N(0, a),\]

Standard recurrent neural network is actually a feed-forward network rolling in time axis, hence, we initialize its parameters in the same strategy as the FNN. However, LSTM and GRU use gated units to regulate flow of information, each gate has its own connection to both input and previous hidden states. Since the input to hidden connection transforms an input vector to the hidden space, we use \textit{Glorot uniform} distribution for the weights of this connection. On the other hand, the hidden to hidden connection is a squared matrix, and there exists the possibility that these weights will be multiplied with the same values for many time steps, hence, we use orthogonal initialization for this connection [94]. Orthogonal initialization constraints the eigenvalues of matrix had an absolute value 1, which means that, no matter how many times we repeatably multiply this matrix, the outcome matrix doesn’t explode or vanish [94]. The algorithm is described as follow

\[
a \sim N(\mu = 0, \sigma = 1, size = W_{shape}),
\]

\[u, v = SVD(a),
\]

\[W = gain * u,
\]

(5.7)

where \textit{gain} is scaling factor, which is suggested to be 1.0 for linear and sigmoid activation, and \(\sqrt{2}\) for rectifier.
5.2.2 Optimization of deep networks

Unless otherwise indicated, we use 40-dimensional log-Mel filter-banks + delta + double delta coefficients to train our networks. To optimize the network, we use RMSprop optimizer [102] (Eq. 5.8), the algorithm scales the learning rates ($\eta$) by dividing with the moving average of the root mean squared gradients which is controlled by a decay factor ($\rho$). Hence, each parameter has its own learning rate and this rate is adapted to current convergent speed of the network,

$$r_t = \rho r_{t-1} + (1 - \rho) * g^2$$

$$\eta_t = \frac{\eta}{\sqrt{r_t + \epsilon}}, \quad (5.8)$$

where $g$ is gradients’ matrix of network parameters. We choose $\rho = 0.9$ and appropriate learning rate by Eq. 5.9, the equation adapts the number of parameters and the depth of network to choose an initial learning rate that guarantees the network will converge at a reasonable speed

$$\lambda = \text{nparams} \times \sqrt{nlayers},$$

$$\eta = 0.1^{\log_{10} \left( \frac{1}{\lambda^{1/2.03}} \right)}, \quad (5.9)$$

where $\lambda$ is an estimation of network complexity, the value 2.03 is a heuristic value, if this value is greater than 2 the initial learning step increase significantly, however, it increase the risk that the gradients might explode or vanish during training. Our observation showed that a sufficiently small initial learning rate is more important for adaptive optimization algorithm like RMSprop [102], especially for a deep RNN network, as the gradients rolled back in time, they grow very fast for the first few steps. All the networks are first trained to converge in 10 epochs with batch size of 128 and dropout ($p = 0.5$) enabled. Then, the fine tuning process turns Gaussian noise on without dropout to perturb the weights for better generalization.

We use generalization loss (GL) as early stopping criterion [84] and decrease learning rate by 1.5 whenever the network drops its validating score. The criterion is calculated by Eq. 5.10, where $GL(t)$ is the amount of loss in generalization score at epoch $t$. $E$ is objective function, $E_{va}(t)$ is validation error at epoch $t$, and $E_{opt}(t)$ is the lowest obtained validation error until epoch $t$. The strategy is interpreted as: “stop after first $t$ epoch when $GL(t) > \alpha$”, $\alpha$ is an early stopping threshold which is $\alpha = 3$ in our case. After five times violating this criterion, we stop training and only keep the parameters set with the best validation performance.

$$GL(t) = 100 \left( \frac{E_{va}(t)}{E_{opt}(t)} - 1 \right), \quad (5.10)$$
The regularizing effect of $L_2$-norm isn’t clear for a small network. However, we figured out that including L2 regularization for training a big network not only slightly improve the result but also speed up the convergence, since it constraints the weights in an acceptable range and doesn’t allow the optimizing process go too far from the best generalization region.

5.3 Baseline system

We compare our system performance to the bottleneck DNN (BNF) feature [86, 96]. We represent each stream of BNF’s in one utterance by an i-vector. Final classification is performed by multi-class logistic regression (MCLR).

A bottleneck DNN was trained using the 40-dimensional filter bank features with the first and second order derivatives extracted from the switchboard landline data. The features were then applied a global mean and variance normalization followed by a per utterance mean and variance normalization before feeding to the DNN. Random weight initialization is used to start the DNN training. The DNN input contains 21 stacked frames rendering an input layer with 2520 units. Seven hidden layers including one bottleneck layer were trained. Each hidden layer except the bottleneck layer has 1024 hidden units and uses the rectified linear unit (ReLU) activation function. The second to last hidden layer is the bottleneck layer with 64 output units and linear outputs are extracted as the bottleneck features. The output layer has 6111 units corresponding to 6111 senones obtained from the baseline speaker-independent GMM-HMM system trained with 39-dimensional MFCC features (13 static features plus first and second order derivatives) extracted from the switchboard landline data.

The 64-dimensional bottleneck features are used for extracting the i-vectors. An energy-based voice activity detection (VAD) technique was applied to the raw bottleneck features to exclude the silence frames. The voiced frames were then used to train a universal background model with 1024 Gaussians with diagonal covariances. The diagonal UBM was then used as an initial point to train a full-covariance UBM with 1024 Gaussians. The full-covariance UBM is then used to train the total variability matrix and extract the i-vectors.

The MCLR system is based on the multi-class cross-entropy discriminative training in the score vector space. To this end, i-vectors were transformed into log-likelihood score vectors through a set of Gaussian distributions, each representing the distribution of the language class in the i-vector space. As the amount of data is extremely imbalance among classes, with
some languages limited to less than an hour of speech, we trained a global covariance matrix where language-specific covariance could be derived with a smoothing factor of 0.1. Given a test i-vector, a score vector is obtained by concatenating the log-likelihood scores from these Gaussian distribution. Discriminative training is further applied on the score vector.

5.4 Results and analysis

In this section, we investigate a relevant deep learning approach to LID. We start from selecting relevant features configuration, and understand how deep networks interpret the input signal. Then, the optimal design is selected by conducting a series of experiments to augment the network structure.

5.4.1 Feature selection

In all of our experiments, the audio data was preprocessed into 25 ms (millisecond) frames, overlapped by 10 ms. The extracted coefficients were individually normalized using local mean and standard deviation of each utterance to be centered at 0 with variance equal to 1. For feature selection, we separately train a network for each feature configuration, the network has 3 stacked LSTM layers with 250 units per layer.

We initially use 40 filter banks in log-Mel scale feature and propose 4 schemes to process speech utterance into relevant feature vectors. The length of input sequences is critical parameters, as a sufficiently long input should contain all important temporal patterns for RNN. Consequently, frames from each utterance are grouped into fixed-length sequences, and the utterances, which are shorter, are padded with zeros. We also introduce masks (i.e indicator vectors of 0 and 1, 0 - for not used in training frames and 1 - otherwise) for each training examples to carefully leave out these padded frames during recurrent steps. Subsequently, VAD can be used inclusively with mask indices, however, our approach using Signal-to-Noise-Ratio [37] threshold cannot exclude the long silences, so we didn’t introduce VAD in our experiments.

For short input sequences (i.e. 20 frames), we miss important temporal information, and there exist cases that the input only contain long silent signal which biases the network to wrong structure. Furthermore, longer sequences (i.e. 800 frames) also hurt the overall per-
CHAPTER 5. EXPERIMENTS IN NETWORKS DESIGN

Table 5.2: Accuracy (Eq. 5.4) on Set03 using different schemes

<table>
<thead>
<tr>
<th>Length (Frames)</th>
<th>Overlap (Frames)</th>
<th>zho</th>
<th>qsl</th>
<th>spa</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0</td>
<td>0.32</td>
<td>0.68</td>
<td>0.35</td>
<td>0.45</td>
</tr>
<tr>
<td>500</td>
<td>0</td>
<td><strong>0.39</strong></td>
<td><strong>0.74</strong></td>
<td><strong>0.53</strong></td>
<td><strong>0.557</strong></td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>0.38</td>
<td>0.73</td>
<td>0.53</td>
<td>0.547</td>
</tr>
<tr>
<td>800</td>
<td>0</td>
<td>0.32</td>
<td>0.67</td>
<td>0.49</td>
<td>0.493</td>
</tr>
</tbody>
</table>

formance. Consequently, we use the second configuration in Tab. 5.2 for further experiments.

Given that the most popular features to train DNNs and their variants are log-Mel filter bank features, some ASR systems used Mel-frequency cepstral coefficients features and achieved reasonable results on NIST LRE 2009 dataset [51]. In our experiment, the average performance in Tab. 5.3 indicates more advantages of using filter bank features. As we are going to add convolutional layers, log-Mel features are known to be more friendly to the convolution operators [22, 89]. The convolutional layers will reduce the spectral variation and model the correlation among different frequency banks. Hence, for the rest of the paper, we use log-Mel filter banks features rolled into sequences of 500 frames without overlap.

Table 5.3: Accuracy on Set03 using different type of features

<table>
<thead>
<tr>
<th>Features</th>
<th>zho</th>
<th>qsl</th>
<th>spa</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFCC</td>
<td><strong>0.40</strong></td>
<td>0.68</td>
<td>0.51</td>
<td>0.530</td>
</tr>
<tr>
<td>Logmel filter banks</td>
<td>0.39</td>
<td><strong>0.74</strong></td>
<td><strong>0.53</strong></td>
<td><strong>0.557</strong></td>
</tr>
</tbody>
</table>

5.4.2 The power of depth network

As suggested in [81], a thin deep recurrent neural network can significantly outperform shallow version with the same number of parameters. It is also mentioned in [36] that there are simple functions expressible by small 3-layer feed-forward neural networks which cannot be approximated by a 2-layer network. Furthermore, our results in Tab. 5.4 also emphasize the importance of depth when constructing a neural network. The first network with depth one is our baseline LSTM. The next two network designs contain only 250 units per layer, as a result, they have lower number of parameters, but achieved greater performance on most of the clusters.
CHAPTER 5. EXPERIMENTS IN NETWORKS DESIGN

Table 5.4: Accuracy on Set03 with various network depth (units* are the number of units for each layer)

<table>
<thead>
<tr>
<th>Depth × units*</th>
<th>1 × 512</th>
<th>2 × 250</th>
<th>3 × 250</th>
</tr>
</thead>
<tbody>
<tr>
<td># Param.</td>
<td>2.8 × 10^6</td>
<td>1.6 × 10^6</td>
<td>2.1 × 10^6</td>
</tr>
<tr>
<td>ara</td>
<td>0.44</td>
<td>0.48</td>
<td>0.48</td>
</tr>
<tr>
<td>eng</td>
<td>0.43</td>
<td>0.45</td>
<td>0.43</td>
</tr>
<tr>
<td>zho</td>
<td>0.37</td>
<td>0.35</td>
<td>0.39</td>
</tr>
<tr>
<td>qsl</td>
<td>0.67</td>
<td>0.73</td>
<td>0.74</td>
</tr>
<tr>
<td>spa</td>
<td>0.53</td>
<td>0.49</td>
<td>0.53</td>
</tr>
<tr>
<td>Avg.</td>
<td>0.488</td>
<td>0.500</td>
<td>0.514</td>
</tr>
</tbody>
</table>

5.4.3 RNN variants

As proposed in Section 2, we conduct our experiments on three different variants of RNN to select the most appropriate architecture for language recognition task. The three variants are vanilla RNN, LSTM and GRU. We use the network design from the previous section with 3 layers of 250 hidden units for each layer.

Table 5.5: Accuracy on Set03 of 3 RNN variants

<table>
<thead>
<tr>
<th>Variants</th>
<th>RNN</th>
<th>LSTM</th>
<th>GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ara</td>
<td>-</td>
<td>0.48</td>
<td>0.46</td>
</tr>
<tr>
<td>eng</td>
<td>-</td>
<td>0.43</td>
<td>0.43</td>
</tr>
<tr>
<td>zho</td>
<td>-</td>
<td>0.39</td>
<td>0.39</td>
</tr>
<tr>
<td>qsl</td>
<td>-</td>
<td>0.74</td>
<td>0.76</td>
</tr>
<tr>
<td>spa</td>
<td>0.42</td>
<td>0.53</td>
<td>0.50</td>
</tr>
<tr>
<td>Avg.</td>
<td>-</td>
<td>0.514</td>
<td>0.508</td>
</tr>
</tbody>
</table>

The RNN network is difficult to train properly on long sequences, the training took a longer time to converge and we have to use gradient clipping with the maximum norm of 50 [101]. RNN also results in very poor performance on spa clusters, and we skip its training process for other clusters since it showed no improvement compared to previous approaches. Conversely, GRU and LSTM converge without gradient clipping and achieve comparable results. For the overall performance, LSTM outperforms GRU with relative 1.2% improvement, however, GRU is more computationally efficient, since LSTM uses 45.8% more parameters but only results in 0.6% improved accuracy. As a result, we use GRU to build deeper architecture due to this merit.
5.4.4 Multiple architecture design

FNN, CNN, and RNN are complementary in their learning capabilities to capture different patterns. While FNN, with multiple processing layers, is able to extract hierarchical representations that benefit the discriminative objective, CNN has the ability to extract local invariant features in both time and frequency domain [89]. Since the learned representation from CNN is heavily relied on internal structure of data, stacking multiple convolutional layers after the input can capture robust low-level features of the signal, and was reported in [89, 91] to boost the network overall performance. We propose an architecture that leverages the merits of three different variants.

Our first two layers are convolutional layers with 128 feature maps for each layer. The first layer convolves both in time and frequency domain with the filter size of $9 \times 9$ to extract an invariant local representation of spectral information. The second layer use $3 \times 5$ filter with $1 \times 2$ strides to keep time dimension unchanged. A pooling size of 3 on frequency axis was used for the first layer, and no pooling was done in the second layer.

The dimension of the last layer of our CNN is large, because of the increasing in a number of feature maps. As suggested in [91], adding a linear layer to perform dimensional reduction provides more compact representation without any accuracy trade-off. Therefore, we form a linear projection to map output of CNN to 256 dimensions before feeding these features to the next 2 GRU layers of 250 units each.

In the final state, we add 1 fully connected layer of size 512 before the softmax layer. Our network has the depth of 5 layers, as the depth increase we need strong regularization methods. We adopted two well-known techniques dropout [98] and batch normalization [55]. However, implementing batch normalization effectively for RNN is a difficult task as we have a sequence of input, we only introduce batch normalization to the first 2 convolutional layers and compare its result to the dropout version of the network.

<table>
<thead>
<tr>
<th></th>
<th>LSTM</th>
<th>CGFNN</th>
<th>CGFNN*</th>
</tr>
</thead>
<tbody>
<tr>
<td>ara</td>
<td>0.48</td>
<td>0.49</td>
<td>0.49</td>
</tr>
<tr>
<td>eng</td>
<td>0.43</td>
<td>0.44</td>
<td>0.55</td>
</tr>
<tr>
<td>zho</td>
<td>0.39</td>
<td>0.43</td>
<td>0.51</td>
</tr>
<tr>
<td>qsl</td>
<td>0.74</td>
<td>0.78</td>
<td>0.84</td>
</tr>
<tr>
<td>spa</td>
<td>0.53</td>
<td>0.57</td>
<td>0.53</td>
</tr>
<tr>
<td>Avg.</td>
<td>0.514</td>
<td>0.542</td>
<td>0.584</td>
</tr>
</tbody>
</table>
CHAPTER 5. EXPERIMENTS IN NETWORKS DESIGN

CGFNN is our proposed architecture, a combination of CNN, GRU and FNN. CGFNN\(^1\) is the design using dropout for convolution layers, vice versa, CGFNN\(^2\) is same design but using batch normalization. The results indicate improvement by using combined architecture, and batch normalization has proved its indisputable efficiency in regularizing CNN. One of the feasible explanations is that the gradients of convolution layers are averaged over the spatial extent of the feature maps. Since dropout stochastically removes activation (and their gradients), the backpropagation ends up having many correlated terms in the averaged gradient, each with different dropout patterns. As a result, the network converges slower and the learned local patterns become unstable.

5.4.5 Recurrent pooling in time

As GRU returns a full sequence of 250 frames rolling in time (sub-sampled from 500 to 250 by CNN), the dimension is enormous (i.e \(250 \times 250 = 62500\)). A projection from RNN output to feed-forward layer can have \(32 \times 10^6\) parameters which consume a huge amount of memory and computational resources.

![Figure 5.3: A pooling over time dimension of RNN.](image)

In [24], the authors argue that such high dimensional representation is overly precise and contain much redundant information, and they suggest a pooling over time strategy which is illustrated in Fig. 5.3. The pooling modules can be “max” or “average” pooling, however, they have different interpretations when we apply pooling along the time axis. Therefore, we perform an empirical comparison of two approaches on overall performance. The tests are performed on 3 clusters: zho, qsl and spa because of the computational cost.

Tab. 5.7 shows a promising result of using average pooling over max pooling. A rea-
Table 5.7: Accuracy on Set03 of different pooling function

<table>
<thead>
<tr>
<th></th>
<th>GRU</th>
<th>avr. pooling</th>
<th>max pooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZHO</td>
<td>0.39</td>
<td>0.43</td>
<td>0.31</td>
</tr>
<tr>
<td>QSL</td>
<td>0.76</td>
<td>0.76</td>
<td>0.62</td>
</tr>
<tr>
<td>SPA</td>
<td>0.50</td>
<td>0.49</td>
<td>0.49</td>
</tr>
</tbody>
</table>

Figure 5.4: Transposed convolutional reconstruction network

A reasonable explanation is that obtaining an invariant representation by selecting maximum between 2 high-dimensional vectors of 250 dimensions is difficult, hence, the network drops its temporal information after every pooling step. Since this issue doesn’t appear to average pooling, it is good practice to add time pooling layer before we project RNN output to fully connected layers. In summary, the final architecture is a combination of CNN, RNN, FNN with the addition of batch normalization for CNN and time pooling for RNN.

5.4.6 Optimizing the convolutional architecture

Our experiments reveal nontrivial gap in performance between the different configuration of CNN filters. Understanding the underlying mechanism of these filters is critical information to build more robust extractor that captures stable features of an imbalanced dataset. In order to observe the convolution behavior on the signal, we propose using transposed convolutional decoder [85] to reconstruct original signal from the output of convolutional layers. The weights are tied between convolving and deconvolving layers force the network to learn the most informative filters for the spectrogram input, as shown in Fig. 5.4.

Fig. 5.5 shows the differences in the reconstructed spectrograms, Conv(64, (9, 9)) means 64 feature maps with filter size 9 × 9, stride steps of (2, 2) is also applied to the first convolution layer. Small filter size (i.e. 3 × 3 in this case) create grid-effect and cut the signal into small pieces, conversely, large filter (i.e. 9 × 13) blurs the signal since it correlates noises
into important patterns. We can see that $9 \times 9$ is the most appropriate filter in this case.

5.4.7 Deep language network

As the number of GRU layers increase, placing time pooling after each layer becomes inefficient, we confront performance degradation as we continue training. Since pooling is a dimension reduction technique, this means an amount of information along the time axis is diminished after each GRU layer. To address this issue, we only do pooling in time for the last two GRU layers. Since the first two layers are responsible for learning more primitive and robust representation, dropping frames at higher layers forces the network to learn more compact and abstract features. As a result, this strategy reduces the number of projection parameters by 4 times without performance trade-off.

Regardless a small amount of parameters added by introducing convolutional layers, the size of the tensor during convolutional computation is multiplied by the number of feature maps which consume a significant amount of memory. Hence, we decided to keep 128 feature maps for each CNN layer which leads to our final architecture illustrated in Fig. 5.6.

Tab. 5.9 summarizes the results of the two systems on validation set. We highlight two major results. First, the proposed architecture outperformed BNF i-vector approach on the
Figure 5.6: Deep Language: deep neural network for language recognition

tuning set which has a similar distribution as training data. Second, The distribution of $C_{avg}$ is significantly different between the two approaches, which indicates that the two algorithms
CHAPTER 5. EXPERIMENTS IN NETWORKS DESIGN

exploiting different discriminative information which can benefit complementary tasks.

Table 5.9: $C_{avg}$ on Set01, BNF i-vector baseline, Deep Language

<table>
<thead>
<tr>
<th></th>
<th>BNF i-vector</th>
<th>DeepLang</th>
</tr>
</thead>
<tbody>
<tr>
<td>ara</td>
<td>1.23</td>
<td>4.36</td>
</tr>
<tr>
<td>eng</td>
<td>1.46</td>
<td>0.26</td>
</tr>
<tr>
<td>zho</td>
<td>3.56</td>
<td>2.28</td>
</tr>
<tr>
<td>qsl</td>
<td>1.74</td>
<td>1.81</td>
</tr>
<tr>
<td>spa</td>
<td>11.01</td>
<td>6.28</td>
</tr>
<tr>
<td>Avg.</td>
<td>3.798</td>
<td>2.998</td>
</tr>
</tbody>
</table>

Conversely, our network shows its drawback in generalizing to different data distribution. The performance rapidly drops according to the divergence between the two distribution. Tab. 5.10 emphasizes the weakness of strong nonlinear model compared to the BNF i-vector approach.

Table 5.10: $C_{avg}$ on Set04, i-vector baseline, Deep Language

<table>
<thead>
<tr>
<th></th>
<th>BNF i-vector</th>
<th>CGFNN</th>
<th>DeepLang</th>
</tr>
</thead>
<tbody>
<tr>
<td>ara</td>
<td>22.08</td>
<td>30.16</td>
<td>30.51</td>
</tr>
<tr>
<td>eng</td>
<td>11.52</td>
<td>32.84</td>
<td>32.95</td>
</tr>
<tr>
<td>zho</td>
<td>16.91</td>
<td>29.87</td>
<td>29.39</td>
</tr>
<tr>
<td>qsl</td>
<td>6.46</td>
<td>19.23</td>
<td>18.95</td>
</tr>
<tr>
<td>spa</td>
<td>22.27</td>
<td>39.72</td>
<td>36.38</td>
</tr>
<tr>
<td>Avg.</td>
<td>15.848</td>
<td>30.364</td>
<td>29.636</td>
</tr>
</tbody>
</table>

Figure 5.7: Confusion matrix of our prediction on zho Set01 (left) and Set04 (right)

Our further analysis shows that the degradation effect is mostly created by the dominant classes, Fig. 5.7. In the case of zho cluster, zho-cmn language has $\approx 87\%$ of training data,
hence, most of the gradients will be backpropagated by examples from this class and the network is optimized to predict the given language with > 90% accuracy. Contradictory, the side-effect of the dominant classes is that they prevent over-fitting of the other classes, with a lower amount of data, \textit{zho-cdo}, \textit{zho-wuu} and \textit{zho-yue} are thus better generalized.
CHAPTER 6

Tackling imbalanced dataset for end-to-end networks

A crucial issue in deep learning classification problems is the skewness in the prior probabilities of the individual classes [67]. In recent years, the research community has spent a tremendous amount of efforts to advance the neural architectures and optimization algorithm for the deep neural network [62, 69, 108], however, the issue concerned an imbalanced training set remains open. Our investigation from Chap. 5 shows that unequally distributed training classes have strong negative impact on network performance as shown in [48, 104]. Specifically, the degradation can often be traced to the fact that the majority classes dominate training error and drive the network to sub-optimal solutions [32, 104].

In this chapter, we conduct a series of experiments to test out various techniques for handling class-imbalance. Furthermore, we propose a framework to minimize the effect of dominant classes when training the end-to-end network and also post-processing the output distribution for LID. To validate our hypothesis, we apply the strategies to Iberian cluster. The cluster has the highest level of skewness, which also is the worst performed cluster for both deep language system and the baseline i-vector approach (Chap. 5).

6.1 Cost-sensitive objective function

In this section, we setup up the experiments the validate the efficiency of different objectives in dealing with the imbalanced training set. The objectives have been introduced in Sec. 3.3, and the network design is the same as the deep language system specified in Sec. 5.4.7. The results of training the same network using different objectives are specified in Tab. 6.1. The
table indicates the advantage of using Bayesian cross-entropy over density estimation loss for the imbalanced dataset. MSE and HD treat the correct and incorrect outputs equally, hence, they backpropagate too much of the gradients to the incorrect ones since the loss controlled by majority classes will dominate other classes as well.

<table>
<thead>
<tr>
<th>$C_{avg}(%)$</th>
<th>Cross-entropy</th>
<th>Bayes Cross-entropy</th>
<th>MSE</th>
<th>Hellinger distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>36.38</td>
<td>31.58</td>
<td>46.49</td>
<td>48.16</td>
<td></td>
</tr>
</tbody>
</table>

### 6.2 Batch normalization

In this experiment, all LSTM layers have 250 hidden units and FNN layers have 2048 units with rectifier activation, and we used Bayesian cross-entropy objective for training. Tab. 6.2 highlights improvement in using batch normalization, without the normalization the network marked with * fails to converge and results in totally biased estimation as shown in the comparison of confusion matrix Fig. 6.1. Regardless the fact that spa-car language has the highest amount of training examples, we can see the network performs worst on this language, it might be the issue of non-representative training examples which introduce extra noise to the network as the amount of data increased. We also emphasize that the FNN layers result very poor performance on this task.

<table>
<thead>
<tr>
<th>$C_{avg}(%)$</th>
<th>(3;LSTM)</th>
<th>CNN (3;FNN)*</th>
<th>CNN (2;LSTM)</th>
<th>CNN (2;LSTM)*</th>
</tr>
</thead>
<tbody>
<tr>
<td>38.59</td>
<td>50</td>
<td>31.58</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

### 6.3 Data sampling

The main issue involved the imbalanced dataset is the different occurrences of samples from each class in mini-batch, hence, the dominant classes with more data introduce more errors and backpropagates stronger gradients signals than other classes. One of the simple solutions is control sampling process during training. There are two main strategies for balancing the data distribution (Fig. 6.2):
Without Batch Normalization  Batch Normalization

Figure 6.1: Confusion matrix of validation examples, between the networks without and with batch normalization

- Oversampling: duplicating the samples of all classes until the number of samples reach the maximum samples of one classes.

- Undersampling: randomly ignore samples from all classes until all of them have the same number of samples as the smallest class.

Figure 6.2: Oversampling (left) and undersampling (right).

Using the best network from previous section, the results of different data sampling when training on the same network is shown in Tab. 6.3. Since undersampling throws away a lot of informative examples in training set, oversampling does not introduce new information but re-scheduling the gradients so the network doesn’t move too fast toward suboptimal region driven by the dominating classes. Tab. 6.3 supports our interpretation. It’s also notable that density estimation objective (i.e. MSE) perform better only with balanced data.
CHAPTER 6. TACKLING IMBALANCED DATASET FOR END-TO-END NETWORKS

Table 6.3: Different objective function together with different sampling strategy on training set

<table>
<thead>
<tr>
<th></th>
<th>Cross-entropy Oversampling</th>
<th></th>
<th>Bayes Cross-entropy Oversampling</th>
<th>Bayes Cross-entropy Undersampling</th>
<th>MSE Upsampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{avg}}$ (%)</td>
<td>31.65</td>
<td></td>
<td>30.25</td>
<td>37.20</td>
<td>30.30</td>
</tr>
</tbody>
</table>

6.4 Score calibration

As our network showed strong confusion between spa-car and spa-lac on the validation set, there exist biased estimation in the final score created by a dominant amount of spa-car examples. For score calibration, we first normalize the log likelihood-ratio output from network to $[-1, 1]$ by

$$llr_{\text{normalized}} = -1 + 2 \times \frac{llr - \min(llr)}{\max(llr) - \min(llr)}.$$  \hspace{1cm} (6.1)

Then, our idea is training a neural network as a “swap function” (Fig. 6.3) to move the wrong LLR value based on index given the true identification on the validation set, for this task we use two layers fully connected network with 128 units in each layer and \texttt{tanh} activation as output.

![LLR scores table](image)

Figure 6.3: Calibration using a small neural network to swap wrong scores to right places.

Table 6.4: Comparison with baseline systems

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{avg}}$ (%)</td>
<td>22.27</td>
<td>36.38</td>
<td>28.59</td>
</tr>
</tbody>
</table>

The calibration shows 1.66% improvement in Tab. 6.4. Our system is significantly im-

72
proved in comparison to former deep learning approach in [104], however, BNF i-vector is still the state-of-the-art approach. One of the reasons is that the BNF is trained on Switchboard corpus [96] and our system uses only LRE’15 development data.

6.5 Discussion

In this chapter, a series of experiments were conducted to understand the impact of class imbalance on end-to-end deep learning approach to language identification task. We present a comprehensive framework to tackle the challenge. Our results shows that deep neural network, as a nonlinear model, can be trained on imbalanced dataset with regularization and appropriate objective function. Furthermore, the mismatch between training distribution and predicted distribution, and the improvement after score calibration suggest that many examples of spa-car are non-representative and noisy which introduces bias to the training process. Hence, a good training set selection techniques can remove non-representative examples and improve the system.
CHAPTER 6. TACKLING IMBALANCED DATASET FOR END-TO-END NETWORKS
In this work, we investigate a comprehensive deep learning approach to end-to-end automatic language identification (LID). Motivated by the recent success of DNN to speech recognition task, we explored a combination of the most advanced network architectures including: CNN, RNN, and FNN to replace the pipeline of handcrafted features with BNF and i-vector. Our architecture has taken into account the computational issues and regularization effect to construct deeper network in order to address large-scale LID task. Additionally, we present an integrated framework to effectively tackle the challenge of class imbalance in dataset.

Our results show that deep neural network, as a nonlinear model, can be trained on imbalanced dataset and achieves competitive results for the LID tasks. Even though our proposed architecture hasn’t surpassed the recent state-of-the-art BNF i-vector system, the trained model shows promising results when combining multiple architectures for LID. Our Deep Language system can outperform the shallow and single architecture approach. The network also is iteratively improved by including batch normalization, Bayesian cross-entropy objective, and careful calibration of final scores. An initial good result on the validation set with a moderate performance on evaluation data suggests that the network was able to capture long-term temporal dependency of speech utterances.

On the other hand, the degradation of the system on evaluation corpus leaves plenty of room for further improvement. The mismatch between training distribution and predicted distribution suggests that many examples of spa-car are non-representative and noisy. Hence, a good training set selection techniques can remove non-representative examples and improve the system. It is also notable that BNF of baseline approach was trained using external dataset (i.e Switchboard corpus). Subsequently, we plan to investigate ways to enhance net-
work performance by pre-training it with an external corpus to be able to compete against the BNF system more fairly. [49] also suggests leveraging a large amount of evaluation data by proportionally fitting the model on pseudo-labeled test data.
Bibliography


