Contrastive Predictive Coding Based Feature for Automatic Speaker Verification

by

Cheng-I Jeff Lai

A thesis submitted to The Johns Hopkins University

in conformity with the requirements for the degree of

Bachelor of Science

Baltimore, Maryland

December, 2018

Abstract

This thesis describes our ongoing work on Contrastive Predictive Coding (CPC) features for speaker verification. CPC is a recently proposed representation learning framework based on predictive coding and noise contrastive estimation. We focus on incorporating CPC features into the standard automatic speaker verification systems, and we present our methods, experiments, and analysis. This thesis also details necessary background knowledge in past and recent work on automatic speaker verification systems, conventional speech features, and the motivation and techniques behind CPC.

Acknowledgments

I have the privilege to be advised by one of the best in the field of speech processing and surrounded by talented and motivated individuals who encourages me to make strides for the speech community. I want to give my sincere gratitude to my advisor at JHU, Professor Najim Dehak, who introduced me to speech processing and pushed me to become better every day. There were times that I could not make good progress and felt like giving up, and it was Najim who supported me through those difficult moments. I could never forget the trust Najim has given to me and he is the best advisor I can ask for as an undergraduate. Thanks to Dr. Jesús Villalba, who has infinite patience for me in the past two years. I came in to the field with little knowledge in machine learning and coding, and it was Jesús who guided me step by step and taught me to be persistent in research. Thanks to Professor Simon King, who hosted me at University of Edinburgh and gave me the resources, guidance, and research environment to work on anti-spoofing. I have the best summer ever at Edinburgh without a doubt. Thanks to Professor Hynek Hermansky, who advised me the importance of the basics and scrutiny of conducting good research. Thansk to Professor Korin Richmond, Professor Junichi Jamagishi, and Professor Alberto Abad, who patiently answered

several questions I have on anti-spoofing during the several meetings we had. I would also like to thank Laure moro, for the Parkinson's disease project and his help on improving my presentation skills, Phani Nidadavolu, for the bandwidth extension project and the good practices he taught me in conducting experiments, and Nanxin Chen, for helping me to learn coding, debug, discover sources for new research paper, and inspiring me to do creative work.

I would like to extend my gratitude to other members in the CLSP group at JHU and the CSTR group at University of Edinburgh, especially Professor Daniel Povey, Professor Shinji Watanabe, Professor Alan Yuille, Professor Colin Wilson, Professor Mounya Elhilali, Paola Garcia-Perera, Dimitra Emmanouilidou, Debmalya Chakrabarty, Arun Nair, Matthew Wiesner, David Synder, Lucas Ondel, Aswin Subramanian, Ruizhi Li, ChuCheng Lin, Raghavendra Reddy, JaeJin Cho, Saurabh Bhati, Peter Frederiksen, Saurabh Kataria, Xutai Ma, Xiaofei Wang, Kelly Marchisio, Sray Chen, Cassia Valentini, Catherine Lai, Joanna Równicka, Julie-Anne Meaney, Mark Sinclair, Felipe Espic, and Pacco.

I met a lot of brilliant people during college. Thanks to William Zhang, James Shamul, Justin Chua, Bijan Varjavand, Aurik Sarker, Harrison Nguyen, Eric Tsai, Kiki Chang, Vladimir Postnikov, Esther Tien, William Shyr, Jeff Peng, Kevin Chen, Kevin Ma, Chris Hong, Chin-Fu Liu, Ray Cheng, Tom Shen, Max Novick, Adriana Donis, Jillian Ho, Richard Oh, Alejandro Contreras, Cindy Yuan, Allen Ren, Linh Tran, Charlie Wang, Weiwei Lai, Michael Chan, and Emily Sun, for their love and support. Their kindness and presence have made all the differences in my life. I want to give special thanks to Daniel

Hsu, who took good care of me when I was suffering from a herniated disc and sciatica, and for being an awesome roommate and friend. Finally, I want to thank my family members in Taiwan for supporting me emotionally and financially. I am especially grateful for my mom, who has always encouraged me to venture to a bigger world and do greater things.

Table of Contents

List of Tables

List of Figures

Chapter 1

Automatic Speaker Verification

1.1 Introduction

[1] Speech is the main medium we use to communicate with the others, and therefore it contains rich information of our interests. Upon hearing a speech, in addition to identify what its content, it is natural for us to ask: Who is the speaker? What is the nationality of the speaker? What is his/her emotion?

Speaker Recognition is the collection of techniques to either identifies or verifies the speaker-related information of segments of speech utterances, and Automatic Speaker Recognition is speaker recognition performed by machines. Figure 1.1 is an overview of the speaker information in speech. Speaker information is embedding in speech, but it is often corrupted by channel effects to some degree. Channel effects can be environment noises, and more often recording noises since automatic speaker recognition is performed on speech recordings. There are some speaker-related information we are also interested in, such as age, emotion and language.

[1] The organization of this Chapter is inspired from Nanxin Chen's Center for Language and Speech Processing Seminar Talk "Advances in speech representation for speaker recognition".

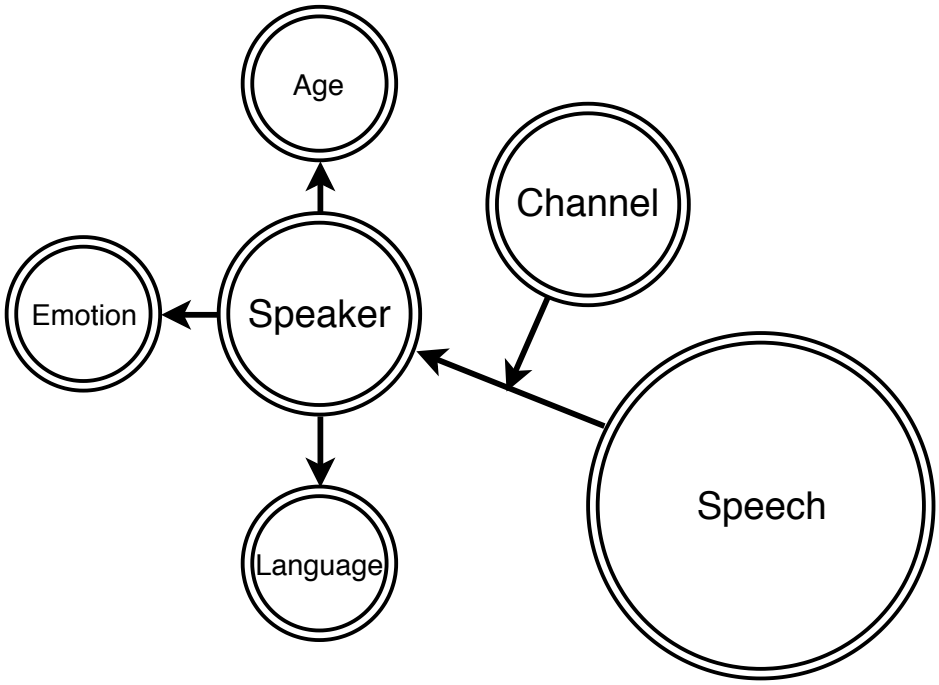
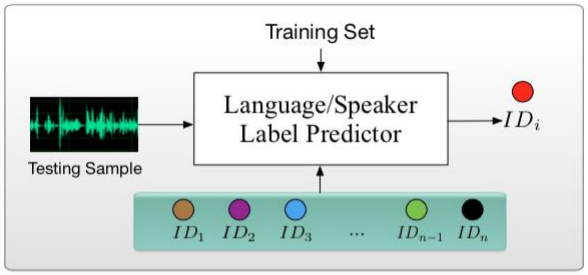


Figure 1.1: An overview of speaker information in speech. Speaker information is embedded in speech and it is often disrupted by channel noises. From the speaker information, age, emotion, language, etc. of the speech can be inferred.

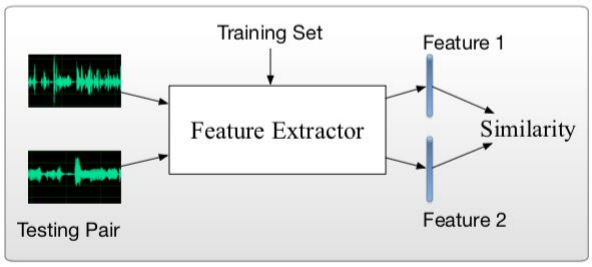
This Chapter first gives a overview of Automatic Speaker Verification. Then several major speaker verification techniques, from the earlier Gaussian Mixture Models to the recent neural models, are presented subsequently.

1.1.1 Speaker Identification v.s. Verification

Speaker Recognition concerns with speaker-related information. Automatic Speaker Recognition is therefore the machines that perform speaker recognition for humans. Speaker Recognition can be categorized into Speaker Identification and Speaker Verification, by the testing protocol (Figure 1.2). As with any machine learning models, Automatic Speaker Recognition requires training data and testing data. Speaker Identification is to identify whether



(a) Closed-set identification



(b) Open-set verification

Figure 1.2: Speaker identification v.s. Speaker verification (cai2018exploring). Speaker identification can be framed as a closed-set problem, while verification can be framed as an open-set problem.

the speaker of a testing utterance matches any training utterances, and hence it is a closed-set problem. On the other hand, Speaker Verification is to verify weather the speakers of a pair of utterances match. The pair is consisted of an enrollment utterance and a testing utterance, which may not be presented beforehand, and hence it is a more challenging open-set problem.

This thesis work focuses on Automatic Speaker Verification.

1.1.2 General Processing Pipeline

Figure 1.3 describes the four main stages of Automatic Speaker Recognition (thus includes Verification). Most systems have these four aspects in their system design. Feature Processing is to get low-level feature descriptors from

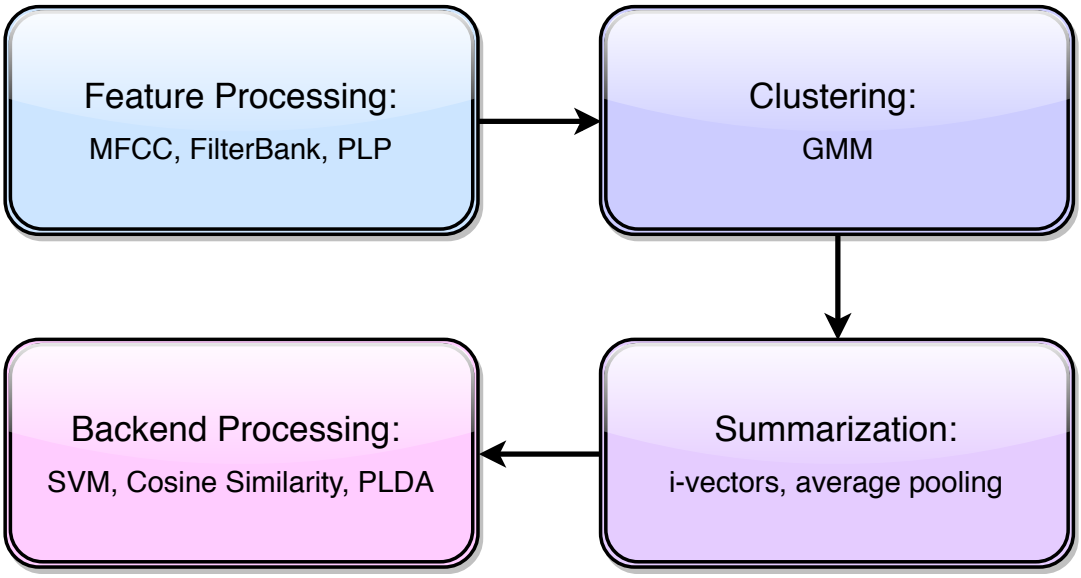


Figure 1.3: Four stages of speaker recognition: Feature Processing, Clustering, Summarization, and Backend Processing. Feature Processing is to get low-level features from speech utterances, such as MFCC, FilterBank, and PLP. Clustering is the process to differentiate different acoustic units and process them separately, such as GMM. Summarization is the conversion from variable-length frame-level features to a fixed-length utterance-level feature, such as the i-vectors. Backend Processing is for scoring and making decisions, such as SVM, Cosine Similarity and PLDA.

the speech waveforms, such as Mel-Frequency Cepstral Coefficients (MFCC), FilterBank, Perceptual Linear Predictive (PLP) Analysis, or bottleneck features. Clustering is the process to differentiate different acoustic units and process them separately, and it is commonly adopted in speaker recognition, such as Gaussian Mixture Model (GMM). Summarization is the conversion from variable-length frame-level features to a fixed-length utterance-level feature, such as the i-vectors or average pooling. Backend Processing is for scoring and making decisions, such as Support Vector Machine (SVM), Cosine Similarity or Probablistic Linear Discriminant Analysis (PLDA).

1.1.3 Metrics

There are various metrics defining how well a system performs, such as the Decision Cost Functio (DCF) and Equal Error Rate (EER). DCF is defined as:

$$C_{DET}(\theta) = C_{FRR} \times P_{Target} \times P_{FRR}(\theta) + C_{FAR} \times (1 - P_{Target}) \times P_{FAR}(\theta) \quad (1.1)$$

EER is the equilibrium point between False Alarm Rate and False Negative Rate. We adopt EER for this thesis work for its common use in Automatic Speaker Recognition work.

1.1.4 Challenges

Speaker Recognition at its core is to optimize a Sequence-to-One mapping function. From the task perspective, it is supposedly easier than Sequence-to-Sequence tasks since it only outputs once per sequence. However, from the data perspective, it is much harder. Comparing to automatic speech recognition or machine translation, which are Sequence-to-Sequence mappings, there is very little data for automatic speaker recognition. For example, a 100 seconds YouTube video could have more than 100 words spoken but only 1 speaker identity. In addition to data, channel effects have been the major bottleneck for previous research work on speaker recognition (Figure 1.1). Advances in the field has developed techniques that aim to address it, such as the Joint Factor Analysis, but channel effects still play a significant role. This is one reason why the most fundamental task in speech, voice activity detection, still remains as a research problem.

1.1.5 Applications

Automatic Speaker Recognition techniques are transferable to the aforementioned tasks: Language Recognition (**dehak2011language**), Age Estimation (**chen2018measuring**) (**Ghahremani2018**), Emotion Classification (**Cho2018**), and Spoofing Attacks Detection (**lai2018attentive**).

1.2 Adapted Gaussian Mixture Models (GMM-UBM)

In the 1990s, Gaussian Mixture Models (GMM) based systems was the dominant approach to automatic speaker verification. Building on top of GMM, Gaussian Mixture Model-Universal Background Model (GMM-UBM) builds a large speaker-independent GMM, referred to as UBM, and adapts the UBM to specific speaker models via Bayesian adaptation (**reynolds2000speaker**). UBM-GMM is the basis for later work such as the i-vectors, which collects sufficient statistics from a UBM, and UBM-GMM is one of the most important developments for automatic speaker verification.

1.2.1 Likelihood Ratio Detector

The task of speaker verification is to determine whether an test utterance U is spoken by a given speaker S. GMM-UBM defines two models: Background Model (UBM) and Speaker Model (GMM). If the likelihood that U comes from S-dependent GMM is larger than the likelihood that U comes from S-independent UBM, then U is spoken by S, and vice versa. The process above

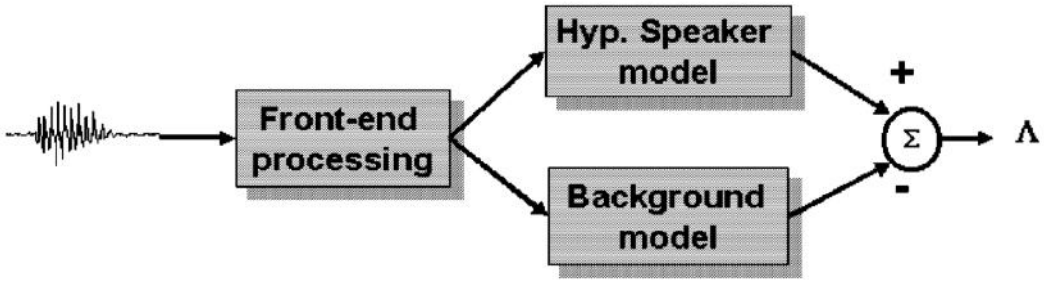


Figure 1.4: Liklihood Ratio Detector for GMM-UBM (reynolds2000speaker).

is defined as likelihood ratio:

$$\delta = \frac{P(U \mid GMM)}{P(U \mid UBM)}, \tag{1.2}$$

where δ is called the likelihood ratio detector. Figure 1.4 is an illustration of δ.

1.2.2 UBM

One basic assumption GMM-UBM assumes is that human speech can be decomposed into speaker-independent and speaker-dependent characteristics. Speaker-independent characteristics are traits that are shared across human speech, and example of which could be pitch and vowels. Speaker-dependent characteristics are traits that are unique to every speaker, and example of which could be accent. GMM-UBM builds upon this assumption. First, speaker-independent characteristics are modeled by a large GMM, a UBM. Since it should capture traits shared across all humans, UBM is trained on large data, usually the whole train dataset. Secondly, speaker-dependent characteristics, which is usually presented in the enrollment data, is obtained by adapting the UBM. UBM is trained by the EM algorithm, and the speaker model adaptation is done via MAP estimation.

Another motivation to split speaker modeling into two steps is that there is often very little enrollment data. For example, setting up smartphones with finger printer readers usually only takes a couple seconds. The enrollment data that is collected is too little to build a powerful model. On the other hand, there are tons of unlabelled data available for training but it does not come from the user. GMM-UBM is one solution that takes advantage of large unlabelled data to build a speaker-specific model by adaptation.

1.2.3 MAP Estimation

MAP estimation is illustrated in Figure 1.5. Given the sufficient statistics of UBM (mixture weights w, mixture means m, mixture variances v) and some enrollment data, MAP estimation linearly adapts w, m and v. In (**reynolds2000speaker**), all w, m and v are adapted although it is common to only adapt the mixture means, and keep the weights and variances fixed.

1.3 Joint Factor Analysis (JFA)

Joint Factor Analysis is proposed to compensate the shortcomings of GMM-UBM. Refer to Figure 1.5, UBM is adapted via MAP to speaker-dependent GMM. If we consider only mean adaptation, we can put the mean vectors m of each Gaussian mixture into a huge vector, which is termed the "Supervector". Let $m \in \mathbb{R}^F$, where F is the feature dimension, and assume there are C number of mixtures in the UBM. Then, the supervector $\mathbf{m} \in \mathbb{R}^{F \times C}$. Let us further denote the real speaker mean supervector as $\mathbf{M}$, then MAP estimation is essentially a high-dimensional mapping from $\mathbf{m}$ to $\mathbf{M}$. This is not ideal since

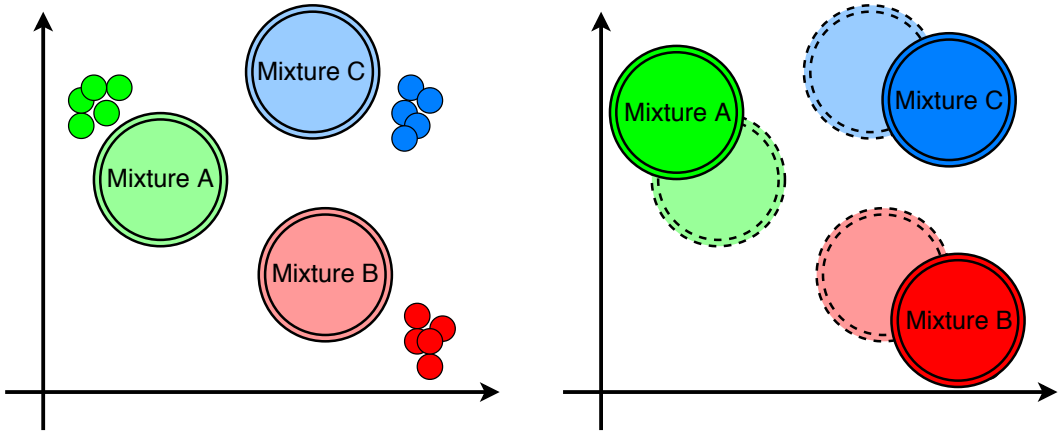


Figure 1.5: Speaker Adaptation illustration of GMM-UBM with threee mixtures. (Left) Universal Background Model with three mixtures and some training data. (Right) Adaptation of speaker model with maximum a posteriori estimation using enrollment data. Note that in this case, only the mixture means are adapted, and mixture variance is fixed.

MAP not only adapts speaker-specific information but also the channel effects (Figure 1.1). Another disadvantage of representing speaker with a mean supervector is that the dimension is too huge. For example, it is common to have F as 39 (with delta and double-deltas), and C as 1024. $F \times C$ will end up with a almost 40,000 dimension supervector.

JFA proposed to address the problem by splitting the supervector $\mathbf{M}$ into speaker independent, speaker dependent, channel dependent, and residual subspaces (**lei2011joint**), with each subspace represented by a low-dimensional vector. JFA is formulated as follows:

$$\mathbf{M} = \mathbf{m} + Vy + Ux + Dz, \tag{1.3}$$

where V, U, D are low rank matrices for speaker-dependent, channel-dependent, and residual subspaces respectively. With JFA, a low dimensional speaker

vector y is extracted. Compare y to GMM-UBM's M, y is of much lower dimension (300 v.s. 40,000) and does not have channel effects.

1.4 Front-End Factor Analysis (i-vectors)

One empirical finding suggested that the channel vector x in JFA also contains speaker information, and a subsequently modification of JFA is proposed and has been one of the most dominant speech representaiton in the last decade: the i-vectors (**dehak2011front**). The modified formula is:

$$\mathbf{M} = \mathbf{m} + Tw, \tag{1.4}$$

where T is the total variability matrix (also low rank), and w is the i-vectors. Compare this to Equation 1.3, there is only one low-rank matrix which models both speaker and channel variabilities. Figure 1.6 is a simple illustration of how JFA and i-vectors converts the supervectors to a low-dimensional embedding.

After w is extracted, it is used to represent the speaker. In Figure 1.3, we refer to i-vectors as a summarization step since it reduces the variable-length supervector to a fixed-length vector. In (**dehak2011front**), SVM and cosine similarity are used for backend processing. However, i-vector PLDA was a more popular combination.

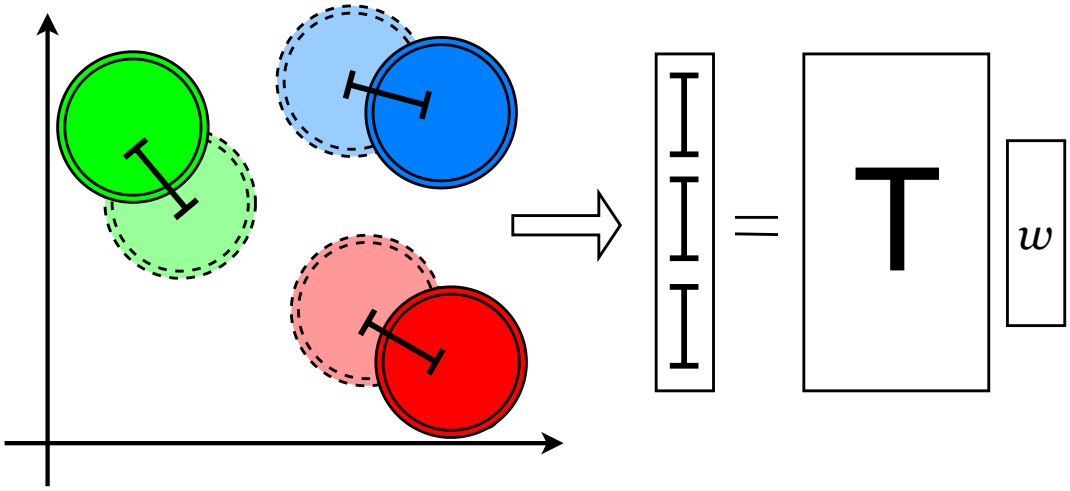


Figure 1.6: Speaker Adaptation illustration of GMM-UBM with threee mixtures. Supervectors

1.5 Robust DNN Embeddings (x-vectors)

i-vectors systems have produced several state-of-the-art results on speaker-related tasks. However, as with any statistical systems, an i-vector system is composed of several independent (unsupervised) subsystems trained with different objectives: an UBM for collecting sufficient statistics, an i-vector extrator for extracting i-vectors, and a scoring backend (usually PLDA). x-vectors systems is a supervised DNN-based speaker recognition system that was aimed to combine the clustering and summarization steps in Figure 1.3 into one (**snyder2017deep**)(**snyder2018x**). The DNN is based on Network-In-Network (**lin2013network**), and trained to classify different speakers (Figure 1.7). The layer outputs after the statistical pooling layer can be used as the speaker embeddings, or the x-vectors. Since x-vectors is based on DNN, which requires lots of data, x-vectors systems also utilize data-augmentation by adding noises and reverberations to increase the total amount of data.

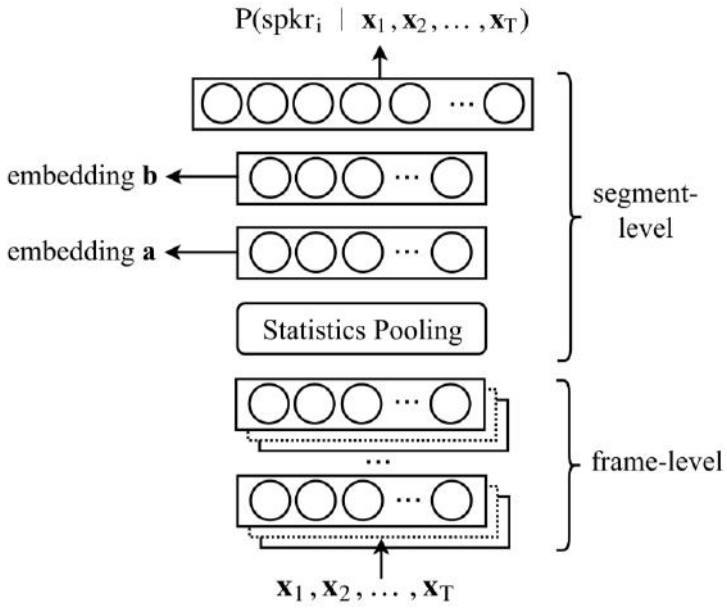


Figure 1.7: x-vectors (snyder2017deep).

x-vectors do not necessarily outperform i-vectors on speaker recognition, especially if data and computational resources are limited.

1.6 Learnable Dictionary Encoding (LDE)

The x-vectors framework is not truly end-to-end since it uses a separately trained PLDA for scoring. An elegant end-to-end framework, Learnable Dictionary Encoding, explores a few pooling layers and loss functions (**cai2018exploring**), and showed that it is possible to combine the clustering, summarization, and backend processing steps in Figure 1.3.

Instead of using a feed-forward deep neural network, LDE employs ResNet34 (**he2016deep**) in its framework. In addition, contrary to the x-vectors DNN in Figure 1.7 where there are few layers after the pooling layer, LDE only has

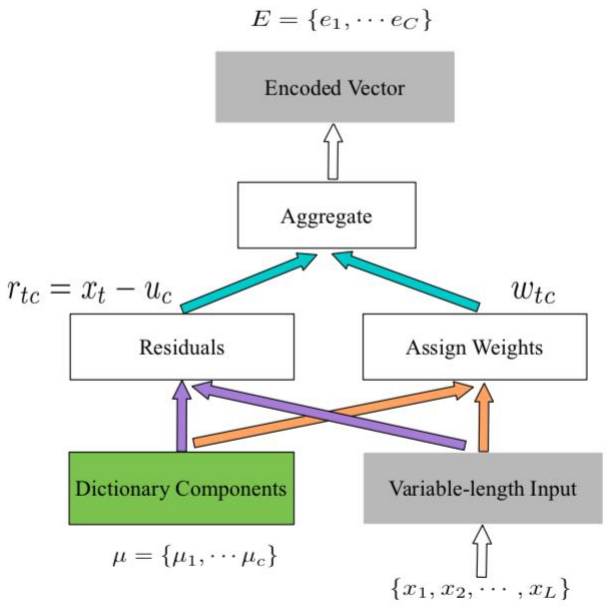


Figure 1.8: Learnable Dictionary Encoding layer (cai2018exploring). LDE layer is inspired from the dictionary-learning procedure of GMM, where a set of dictionary means and weights are learned and aggregated for calculating the fixed-dimensional representation (speaker representation).

a fully-connected layer (for classification) after its pooling layer. LDE uses a LDE layer for pooling (or summarization) in Figure 1.8.

i-vectors and x-vectors systems requires a separately trained backend (PLDA) for scoring, and LDE showed that with Angular Softmax Losses (**liu2017sphereface**), a separate backend is not necessary and hence the whole-framework is end-to-end.

Chapter 2

Conventional Speech Features

2.1 Introduction

The Feature Processing step in 1.3 extracts low-level feature descriptors from raw waveform, and several earlier work showed that Fourier analysis based transforms can effectively capture information of speech signals. Conventional low-level speech features include Log-spectrogram, Log-Filterbank, Mel-Frequency Cepstral Coefficients (MFCC), and Peceptual Linera Predictive (PLP) Analysis. DNN-based speech recognition systems (**hinton2012deep**), GMM-UBM systems (**reynolds2000speaker**) and i-vectors systems (**dehak2011front**) are based on MFCC; x-vectors systems (**snyder2018x**) and LDE (**cai2018exploring**) are based on Log-Filterbank; Attentive Filtering Network (**lai2018attentive**) is based on Log-Spectrogram. We established our baseline on MFCC, and this chapter will introduce MFCC and the MFCC configuration used in our experiments in Chapter 4.

2.2 Mel-Frequency Cepstral Coefficients (MFCC)

MFCC is one of the most standard and common low-level feature in automatic speaker recognition systems. The procedure of MFCC extraction is followed:

1. Take Short-Term Fourier Transform (STFT) on the waveform. This step will give us a Spectrogram.
2. Apply Mel-scale Filters. This step will give us a Filterbank.
3. Take the logarithm on the powers in all Mel-bins. Logarithm is taken also for Log-Spectrogram and Log-Filterbank.
4. Apply Discrete Consine Transform (DCT), and keep several cepstral coefficients. This step decorrelates and reduces the dimensionality.

A visual comparison of Log-Spectrogram, Log-Filterbank, and MFCC is 2.1. We can see that there are more structures in Log-Spectrogram and Log-Filterbank, and MFCC has less dimensions than the former two.

2.3 MFCC Details

Our experiments (see Chapter 4 for more details) are conducted on the LibriSpeech Corpus (**panayotov2015librispeech**), in which speech utterances are recorded in 16k Hz. We used the standard 25 ms frame-length and 10 ms frame-shift for STFT computation, 40 Mel filters, and took 24 cepstral coefficients after DCT. The first and second order derivatives (deltas and double-deltas) are computed during UBM training. Details of our MFCC configuration is in Table 2.1.

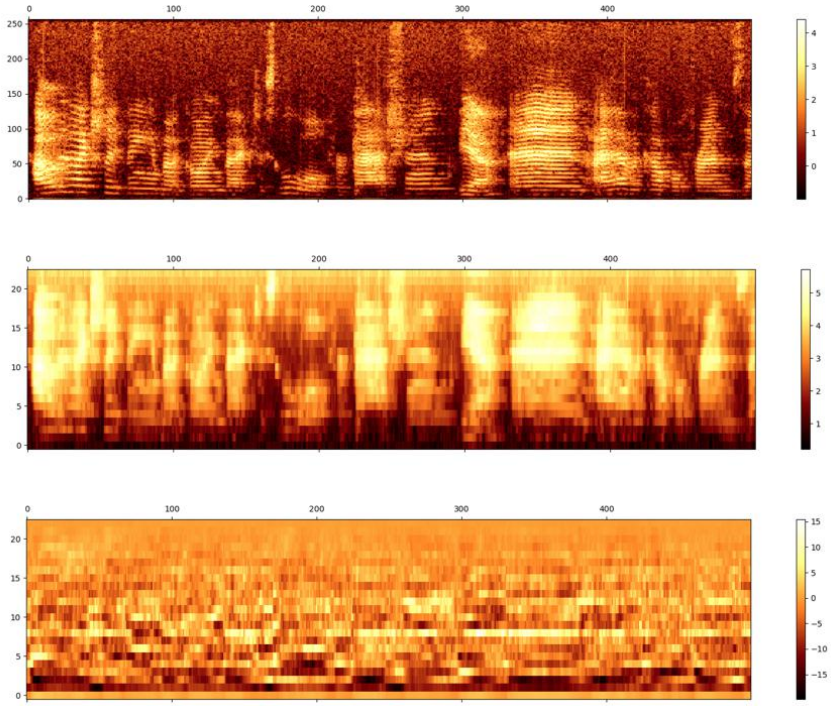


Figure 2.1: An Visual Comparison of (top) Log-Spectrogram, (middle) Log-Filterbank, and (bottom) MFCC.

MFCC Details	
Sampling Frequency	16000 Hz
Frame Length for STFT	25 ms
Frame Shift for STFT	10 ms
High Frequency Cutoff for Mel Bins	7600 Hz
Low Frequency Cutoff for Mel Bins	20 Hz
Number of Mel Bins	40
Number of Cepstral Coefficients after DCT	24

Table 2.1: Our MFCC Configuration. The configuration is mostly based on the Kaldi toolkit (povey2011kaldi).

Chapter 3

Contrastive Predictive Coding

3.1 Introduction

Predictive coding is a well-motivated and developed research area in neuroscience. The central idea of predictive coding is that the current and past states of a system contain relevant information of its future states. On the other hand, one long-standing research question in speech processing has been to extract global information from noisy speech recordings. In speech recognition, this can be related to as retrieving phone labels from the recordings. In speaker recognition, the same research question could be framed as sentiment analysis of the recordings. Could we harness the concept of predictive coding to design a model which extracts representations that are invariant to noise? Contrastive Predictive Coding (CPC) connects the idea of predictive coding and representation learning. This Chapter will give a background overview of predictive coding in neuroscience 3.2, a background of CPC 3.3 and CPC models 3.3. Lastly, the application of CPC for speaker verification is presented 3.5.

3.2 Predictive Coding in Neuroscience

In a famous study by (**hubel1968receptive**), the visual Receptive Field (RF) in the monkey striate cortex is studied. Macaque monkey is presented with line stimuli of different orientations while RF responses in the striate cortex are recorded. The experiment showed that cells responded optimally (with high firing rates) to particular line orientations, illustrated in Figure 3.1. The interesting question to ask here is: why don't neurons always respond in proportion to the stimulus magnitude?

Predictive coding is one prominent theory that aims to provide a possible explanation. Predictive coding states that human brain can be modeled by a framework that is constantly generating hypotheses and fixing its internal states through an error feedback loop. Since neighboring neurons are likely to be correlated, predictive coding implies that the RF response of a neuron can be predicted by those RF responses of its surroundings, and therefore a strong stimulus does not always correspond to a strong RF response. The first hierarchical model with several levels of predictive coding is proposed for visual processing in (**rao1999predictive**). Each level receives a prediction from the previous level and calculates the residual error between prediction and the reality. To achieve efficient coding, only the residual error is propagated forward to the next level, while the next prediction for the current level is made, illustrated in Figure 3.2.

The study of (**rao1999predictive**) suggested the importance of feedback connection in addition to feedforward information transmission for visual processing. However, the key insight of how predictive coding is connected

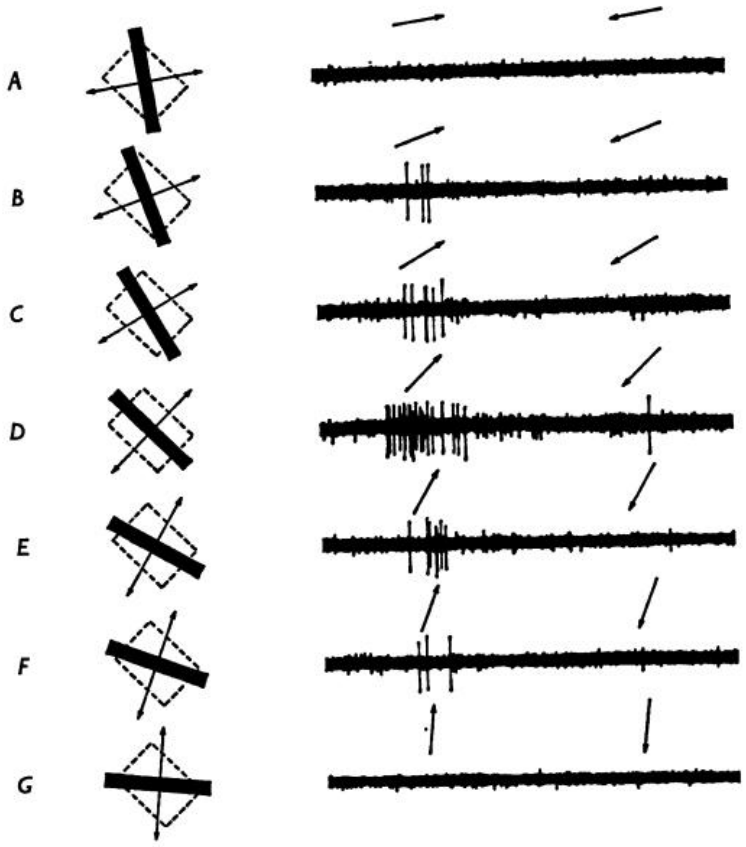


Figure 3.1: RF responses to line stimuli Illustration of the RF firing responses to the same line segment but different line orientations from a cell in the monkey striate cortex (**hubel1968receptive**)

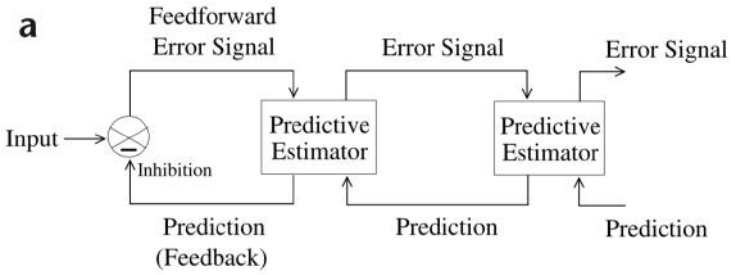


Figure 3.2: Hierarchical model of predictive coding Illustration of how residual error is propagated and how prediction is made in the hierarchical model in (**rao1999predictive**)

to representation learning is that by learning to predict, the model should implicitly retain properties or structures of the input.

3.3 Contrastive Predictive Coding (CPC)

3.3.1 Connection to Predictive Coding

Contrastive Predictive Coding (CPC) is proposed in (**oord2018representation**) as a new unsupervised representation learning framework. One challenging aspect of representation learning within high dimensional signal is noise. The primary goal of CPC is to extract high-level representation, or the slow-varying features (**wiskott2002slow**), from a sensory signal full of low-level noises. On the other hand, predictive coding retains properties or structures of the input 3.2. By predicting the future, the model has to infer global properties or structures from the past, and therefore has to separate global information from noises. One example is TV show series. After watching several episodes of a TV series, most people could generally predict some plots in the next few episodes. But only a few who know the entire series and its history very well can make plot predictions beyond five episodes. These few people has "mastered" the TV series such that they can tell the important plot development from those that are minor in comparison. CPC leverage this idea and therefore could be powerful for separating high-level representation from noises.

However, how do we quantify high-level representation and monitor how well the model is learning? To quantify high-level representation, CPC calculates the mutual information $I(x; C)$ between the sensory signal x and

global information C. Let us refer back to the TV series example. The correct prediction of the plots in future episodes are often hidden as several key points in previous episodes. If we put it in terms of mutual information, the sensory signal x is the future episode plots, and global information C is the several key points, such as an important plot twist or character development. 3.3.2 gives a background of mutual information theory.

What metric should we use to train the predictive coding model? Figure 3.2 is the original hierarchical model of predictive coding proposed for visual processing, and from the figure we can see that the residual error is calculated during the feedforward pass. An straightforward implementation of residual error could be the L1 loss 3.3.1 or Mean Squared Error (MSE) 3.3.1 between prediction $D(H)$ and actual value A, where H is some learnable latent representation and D is a mapping from the latent space to input space. In fact, this implementation can be dated back to the 1960s where MSE is used for training the predictive coding model for speech coding (**atal1970adaptive**). Predictive Coding Network, another predictive coding based unsupervised learning framework, is trained with L1 loss (**lotter2016deep**). However, either L1 loss or MSE loss requires a mapping function, namely a decoder D, that computes $p(x \mid C)$. In our TV series example, $p(x \mid C)$ is saying, "tell me all the details x of future plots given the several key points C. Intuitively, this is a hard task and unnecessary for our purpose since we are interested in high-level representations. To get around this issue, CPC models the mutual information directly with the noise contrastive estimation technique, which is introduced in 3.3.3.

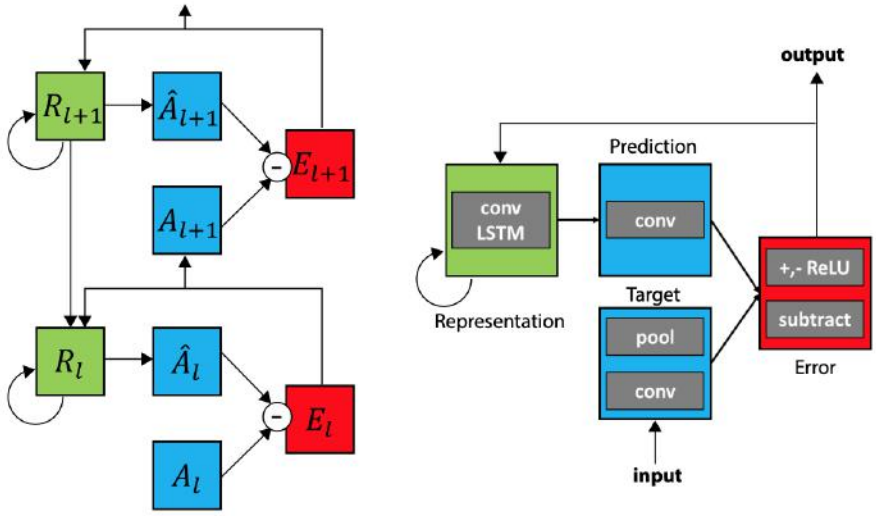


Figure 3.3: Predictive Coding Network (PredNet) Illustration of information flow in PredNet, which is trained with the L1 loss between $\hat{A_{l+1}}$ and A_{l+1}. (**lotter2016deep**)

$$L1 = \sum_{i=1}^{N} (D(h_i) - a_i)^2 \tag{3.1}$$

.

$$MSE = \sum_{i=1}^{N} \mid D(h_i) - a_i \mid . \tag{3.2}$$

3.3.2 Mutual Information

Mutual information denotes the amount of information shared between the two variables. Given two random variable X and Y, mutual information

$I(X;Y)$ is defined as,

$$I(X;Y) = H(X) - H(X \mid Y), \tag{3.3}$$

where $H(X)$ is the entropy of X and $H(X \mid Y)$ is the conditional entropy of Y given X. $H(X)$ is defined as,

$$H(X) = -\sum_{i=1}^{n} P(X = x_i) \log P(X = x_i), \tag{3.4}$$

and $H(X \mid Y)$ is defined as,

$$H(X \mid Y) = -\sum_{i=1}^{n} P(X = x_i \mid Y) \log P(X = x_i \mid Y). \tag{3.5}$$

With the above definitions, we can subsequently show the following:

$$I(X;Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i, y_j) \log \frac{p(x_i \mid y_j)}{p(x_i)} \tag{3.6}$$

Proof. First we expand 3.3.2 as:

$$H(X \mid Y) = -\sum_{i=1}^{n} P(X = x_i \mid Y) \log P(X = x_i \mid Y) \tag{3.7}$$

$$= -\sum_{i=1}^{n} \sum_{j=1}^{m} P(X = x_i \mid Y = y_j) P(Y = y_j) \log P(X = x_i \mid Y = y_j) \tag{3.8}$$

$$= -\sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i \mid y_j) p(y_j) \log p(x_i \mid y_j) \tag{3.9}$$

Then by substitution and Baye's rule,

$$I(X;Y) = H(X) - H(X \mid Y) \tag{3.10}$$

$$= -\sum_{i=1}^{n} p(x_i) \log p(x_i) + \sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i \mid y_j) p(y_j) \log p(x_i \mid y_j) \tag{3.11}$$

$$= -\sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i, y_j) \log p(x_i) + \sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i, y_j) \log \frac{p(x_i, y_j)}{p(y_j)} \tag{3.12}$$

$$= -\sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i, y_j) \log \frac{p(x_i)p(y_j)}{p(x_i, y_j)} \tag{3.13}$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i, y_j) \log \frac{p(x_i \mid y_j)}{p(x_i)} \tag{3.14}$$

□

We can also easily show that if X and Y are independent, their mutual information is zero:

Proof. Given X and Y are independent, $P(X \mid Y) = P(X)$. By definition, we can rewrite $H(X \mid Y)$ as:

$$H(X \mid Y) = -\sum_{i=1}^{n} P(X = x_i \mid Y) \log P(X = x_i \mid Y) \tag{3.15}$$

$$= -\sum_{i=1}^{n} P(X = x_i) \log P(X = x_i) \tag{3.16}$$

$$= H(X), \tag{3.17}$$

and therefore, we have:

$$I(X;Y) = H(X) - H(X \mid Y) \tag{3.18}$$

$$= H(X) - H(X) \tag{3.19}$$

$$= 0 \tag{3.20}$$

□

In the context of representation learning, mutual information gives us a quantitative measure of how well a model learns the global information. Let us look back at the TV series example again. If a person only has limited memory and has successfully observed the key developments, denoted as C_1, over the past episodes, those developments are likely to be highly relevant to the upcoming episodes, denoted as X. We can say that their mutual information $I(X;C_1)$ is high. Hoewver, given the limited amount of memory everyone has, if the person only remembered the minor plot developments, denoted as C_2, the mutual information $I(X;C_2)$ is most likely to be low.

3.3.3 Noise-Contrastive Estimation (NCE)

Noise-Contrastive Estimation (NCE) is an estimation technique for estimating the parameters of parametric density functions (**gutmann2012noise**). Let us consider a set of observations $X = (\vec{x_1}, \vec{x_2}, \vec{x_3}, ..., \vec{x_N})$, where $\vec{x_i} \in \mathbb{R}^n$. In real world examples, n is often of high dimension, and the goal of all machine learning models is to find, or give an accurate estimate of, the underlying data distribution, the probability density function (pdf) P_D, from the observable

set X. NCE makes an assumption that P_D comes from a parameterized family of functions:

$$P_D \in \{P_M(;\boldsymbol{\theta})\}, \tag{3.21}$$

where $\boldsymbol{\theta}$ is a set of parameters. Put it another way, there exists some $\theta^\star$ such that the following is true,

$$P_D = P_M(;\theta^\star). \tag{3.22}$$

Now, let us denote any estimate of $\theta^\star$ as $\bar{\theta}$. Then, the following must hold for any pdf $P_M(;\bar{\theta})$:

$$P_M(;\bar{\theta}) \geq 0 \tag{3.23}$$

$$\int P_M(\vec{x};\bar{\theta})d\vec{x} = 1 \tag{3.24}$$

If these two constraints are satisfied for all $\theta \in \boldsymbol{\theta}$, then we say P_D is normalized; otherwise, P_D is unnormalized. It is common for models to be unnormazlied, such as the Gibbs distribution. Let us further give these unnormalized parametric models a name, $P^0_M(;\alpha)$. To normalize $P^0_M(;\alpha)$, we would need to calculate the partition function $Z(\alpha)$:

$$Z(\alpha) = \int P^0_M(\vec{x};\alpha)d\vec{x}, \tag{3.25}$$

and $P^0_M(;\alpha)$ can be normalized by $\frac{P^0_M(;\alpha)}{Z(\alpha)}$.

Everything so far is reasonable, except that in real word examples, $Z(\alpha)$ is certainly intractable for high-dimensional data (curse of dimensionality),

and thus $P_M^0(;\alpha)$ is still unnormalized. One simple solution NCE proposed is, why not make $Z(\alpha)$ an additional parameter (**gutmann2012noise**)? Let us define the new pdf $P_M(;\vec{\theta})$ accordingly:

$$\ln P_M(;\vec{\theta}) := \ln P_M^0(;\alpha) + c, \tag{3.26}$$

where $c = \frac{1}{Z(\alpha)}$, and $\vec{\theta} = (\alpha, c)$. The estimate $\bar{\theta} = (\bar{\alpha}, \bar{c})$ now is not subject to the two constraints above since $\bar{c}$ provides a scaling factor. The intuition here is that instead of calculating $Z(\alpha)$ to normalize $P_M^0(;\alpha)$ for all α, only $P_M^0(;\bar{\alpha})$ is normalized.

However, Maximum Likelihood Estimation only works for normalized pdf, and $P_M^0(;\alpha)$ is not normalized for all α. NCE is therefore proposed for estimating **unnormalized** parametric pdfs.

3.3.3.1 Density Estimation in a Supervised Setting

The goal of density estimation is to give an accurate description of the underlying probablistic density distribution of an observable data set X with unknown density P_D. The intuition of NCE is that by comparing X against a known set Y, which has a known density P_N, we can get a good grasp of what P_D looks like. Put it more concretely, by drawing samples from $Y = (\vec{y_1}, \vec{y_2}, \vec{y_3}, ..., \vec{y}_{T_y})$ with a known pdf P_N, and samples from $X = (\vec{x_1}, \vec{x_2}, \vec{x_3}, ..., \vec{x}_{T_x})$, we can estimate the density ratio $\frac{P_D}{P_N}$. With $\frac{P_D}{P_N}$ and P_N, we have the target density P_D.

By classifying samples X from noise Y with a simple classifier, in this case logistic regression, we show NCE gets a estimate of the probability density ratio $\frac{P_D}{P_N}$.

Let X and Y be two observable sets containing data $X = (\vec{x_1}, \vec{x_2}, \vec{x_3}, ..., \vec{x}_{T_x})$, $Y = (\vec{y_1}, \vec{y_2}, \vec{y_3}, ..., \vec{y}_{T_y})$, and let U be $X \cup Y$, $U = (\vec{u_1}, \vec{u_2}, \vec{u_3}, ..., \vec{u}_{T_x+T_y})$. X is drawn from an unknown pdf $P_D \in \{P_M(;\boldsymbol{\theta})\}$, and Y is drawn from a known pdf P_N. Since Y is not our target, it is commonly referred to as the "noise". We also assign each datapoint in U a label C_t: $C_t = 1$ if $u_t \in X$ and $C_t = 0$ if $u_t \in Y$. From the above settings, the likelihood distributions are then:

$$P(\vec{u} \mid C = 1) = P_M(\vec{u};\boldsymbol{\theta}) \tag{3.27}$$

$$P(\vec{u} \mid C = 0) = P_N(\vec{u}) \tag{3.28}$$

The prior distributions are:

$$P(C = 1) = \frac{T_x}{T_x + T_y} \tag{3.29}$$

$$P(C = 0) = \frac{T_y}{T_x + T_y} \tag{3.30}$$

The probability of the data $P(\vec{u})$ is thus:

$$P(\vec{u}) = P(C = 0) \times P(\vec{u} \mid C = 0) + P(C = 1) \times P(\vec{u} \mid C = 1) \tag{3.31}$$

$$= \frac{T_y}{T_x + T_y} \times P_N(\vec{u}) + \frac{T_x}{T_x + T_y} \times P_M(\vec{u};\boldsymbol{\theta}) \tag{3.32}$$

With Baye's rule, we can derive the posterior distributions of $P(C = 1 \mid \vec{u})$

and $P(C = 0 \mid \vec{u})$:

$$P(C = 1 \mid \vec{u}) = \frac{P(C = 1) \times P(\vec{u} \mid C = 1)}{P(\vec{u})} \tag{3.33}$$

$$= \frac{\frac{T_x}{T_x+T_y} \times P_M(\vec{u};\boldsymbol{\theta})}{\frac{T_y}{T_x+T_y} \times P_N(\vec{u}) + \frac{T_x}{T_x+T_y} \times P_M(\vec{u};\boldsymbol{\theta})} \tag{3.34}$$

$$= \frac{P_M(\vec{u};\boldsymbol{\theta})}{P_M(\vec{u};\boldsymbol{\theta}) + vP_N(\vec{u})} \tag{3.35}$$

where

$$v = \frac{T_y}{T_x}. \tag{3.36}$$

Similarly, we can get

$$P(C = 0 \mid \vec{u}) = \frac{vP_N(\vec{u})}{P_M(\vec{u};\boldsymbol{\theta}) + vP_N(\vec{u})} \tag{3.37}$$

$P(C = 1 \mid \vec{u})$ can further be expressed as,

$$P(C = 1 \mid \vec{u}) = \frac{P_M(\vec{u};\boldsymbol{\theta})}{P_M(\vec{u};\boldsymbol{\theta}) + vP_N(\vec{u})} \tag{3.38}$$

$$= \left(1 + v\frac{P_N(\vec{u})}{P_M(\vec{u};\boldsymbol{\theta})}\right)^{-1} \tag{3.39}$$

Now, we can denote our target density ratio $\frac{P_N(\vec{u})}{P_M(\vec{u};\boldsymbol{\theta})}$ with a new variable G:

$$G(\vec{u};\boldsymbol{\theta}) = \ln \frac{P_M(\vec{u};\boldsymbol{\theta})}{P_N(\vec{u})} \tag{3.40}$$

$$= \ln P_M(\vec{u};\boldsymbol{\theta}) - \ln P_N(\vec{u}). \tag{3.41}$$

$P(C = 1 \mid \vec{u})$ is then:

$$P(C = 1 \mid \vec{u}) = \text{sigmoid}(G(\vec{u};\boldsymbol{\theta})) \tag{3.42}$$

$$= h(\vec{u};\boldsymbol{\theta}) \tag{3.43}$$

Finally, since C_t is a Bernoulli distribution with value of 0 or 1. We can write the log-likelihood as:

$$l(\boldsymbol{\theta}) = \sum_{t=1}^{T_x+T_y} C_t \ln P(C_t = 1 \mid \vec{u}_t) + (1 - C_t) \ln P(C_t = 0 \mid \vec{u}_t) \tag{3.44}$$

$$= \sum_{t=1}^{T_x} \ln h(\vec{x}_t;\boldsymbol{\theta}) + \sum_{t=1}^{T_y} \ln \Big(1 - h(\vec{y}_t;\boldsymbol{\theta})\Big) \tag{3.45}$$

Optimize $l(\boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}$ will lead to an estimate of $G(\vec{u};\bar{\theta})$, which is the density ratio we want. If we take a step back, we can see that $-l(\boldsymbol{\theta})$ is in fact a cross-entropy loss. In a supervised setting, NCE gives us a density estimation!

3.3.3.2 The NCE Estimator

Let us refer back to 3.3.3. We are now ready to introduce the NCE estimator:

$$J_T(\vec{\theta}) = \frac{1}{T_d}\left(\sum_{t=1}^{T_x} \ln h(\vec{x}_t;\vec{\theta}) + \sum_{t=1}^{T_y} \ln \Big(1 - h(\vec{y}_t;\vec{\theta})\Big)\right), \tag{3.46}$$

which is off by a scaling constant as 3.3.3.1.

3.4 Representation Learning with CPC

3.4.1 Single Autoregressive Model

As mentioned in the previous sections, mutual information gives the model a good criterion to measure how much global information is preserved. We can explicitly write out the formula for mutual information:

$$I(X;Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} p(x_i, y_j) \log \frac{p(x_i \mid y_j)}{p(x_i)} \tag{3.47}$$

In speech, we can make X the waveform of any utterance, and Y the global information such as speaker label. Therefore, the mutual information we are interested in becomes:

$$I(U;S) = \sum_{i=1}^{n}\sum_{j=1}^{m} p(u_i, s_j) \log \frac{p(u_i \mid s_j)}{p(u_i)} \tag{3.48}$$

where U represents utterance and S represents speaker label. In (**oord2018representation**), NCE objective is introduced for model training, and the term $\frac{p(u_i|s_j)}{p(u_i)}$ is selected as the density ratio to be estimated in NCE. We will prove why $\frac{p(u_i|s_j)}{p(u_i)}$ is selected later. The NCE objective is subsequently named NCE loss.

3.4 is an illustration of the proposed CPC model in (**oord2018representation**). The model takes in raw waveforms U as input and transforms it to some latent space L by an encoder. In the latent space, an Recurrent Neural Network is trained by the NCE loss to learn S.

3.4.1.1 NCE Loss

CPC selects $\frac{p(u_i|s_i)}{p(u_i)}$ as the density ratio to be estimated in the NCE estimator. We can denote it with f_i:

$$f_i(u_i, s_i) = \frac{p(u_i \mid s_i)}{p(u_i)} \tag{3.49}$$

We can see that f_i is unnormalized, and this is the reason why we started off with NCE. In addition, since f_i could not be explicitly computed. An alternative way is to model f_i with log-bilinear model, which signifies how relevant the input is to the context:

$$f_i(u_i, s_i) = \exp(s_i \cdot u_i). \tag{3.50}$$

Refer back to the model 3.4, we can see that s_i is modeled by the context vector C_i of the recurrent neural network, and u_i can be modeled by either the waveform or latent space L_i. Since we would like the model to learn high-level information, it makes more sense to model u_i with L_i. Therefore, f_i becomes:

$$f_i(u_i, s_i) = \exp(C_i \cdot L_i). \tag{3.51}$$

However, the dimension of the context vector C_i and latent space L_i do not always agree. A simple solution is to add a matrix to conform the dimension. Let $C_i \in \mathbb{R}^a$ and $L_i \in \mathbb{R}^b$. We define a matrix $W_i \in \mathbb{R}^{a \times b}$ and 3.4.1.1 becomes:

$$f_i(u_i, s_i) = \exp\left(L_i \cdot (W_i C_i)\right) \tag{3.52}$$

$$= \exp\left(L_i^T (W_i C_i)\right) \tag{3.53}$$

We are now ready to define the NCE loss $\mathcal{L}$ for training the CPC model. Refer to 3.3.3.2, NCE gives an estimate of the density ratio by classifying data samples from noise samples. Given a batch of utterances $B = (b_1, b_2, b_3, ..., b_N)$, which includes 1 data sampels and $N - 1$ noise samples, where the positive sample comes from the data distribution $p(u_i \mid s_i)$ and the noise samples come from noise distributions $p(u_i)$. NCE loss is defined as:

$$\mathcal{L} = -\frac{1}{N} \sum_{B} \left(\log \frac{f_p(u_i, s_i)}{\sum_B f_n(u_i, s_i)} \right) \tag{3.54}$$

$$= -\mathop{\mathbb{E}}_{B} \left[\log \frac{f_p(u_i, s_i)}{\sum_B f_n(u_i, s_i)} \right] \tag{3.55}$$

where u_i is any frame segment from utterance $b_i \forall i$, s_i is the corresponding global context for frame segment u_i, $\frac{f_p}{\sum_B f_n}$ is the prediction of the model, and $\log \frac{f_p}{\sum_B f_n}$ is taking the softmax over B.

However, the current loss $\mathcal{L}$ has nothing to do with predictive coding 3.2, where a prediction of the future is made by the context and the residual error is propagated back to correct the context (**lotter2016deep**). Similarly, CPC model also incorporates future frame predictions. We can modify the $\mathcal{L}$ as:

$$\mathcal{L} = -\mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \left[\log \frac{f_p(u_{i+t}, s_i)}{\sum_B f_n(u_{i+t}, s_i)} \right], \tag{3.56}$$

where instead of computing loss only with the density ratio of current frame $f_i(u_i, s_i)$, we also calculate the density ratio of future frames up to T frames in the future, $f_i(u_{i+t}, s_i)$.

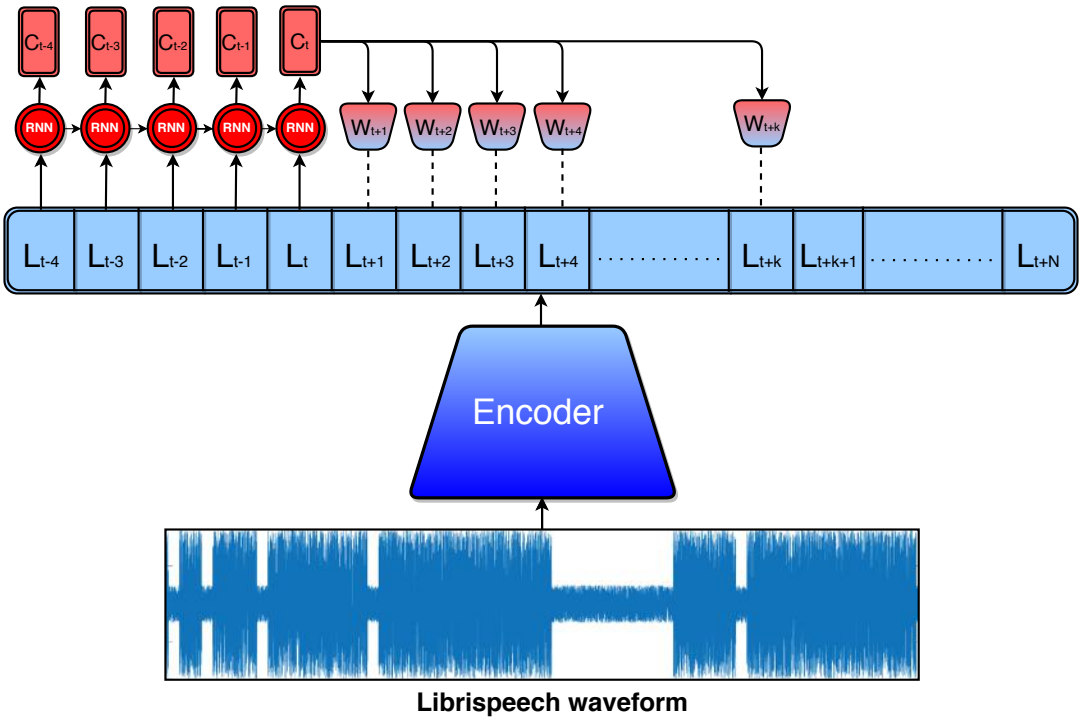


Figure 3.4: CPC Single Autoregressive Model Illustration of the CPC single autoregressive model's training stage. The model takes in raw waveform and transform it to some latent space by an encoder. An recurrent neural network is trained to learn global information in the latent space with NCE loss.

3.4.1.2 Connection to Mutual Information

Why does CPC selects $\frac{p(u_i|s_i)}{p(u_i)}$ as the density ratio to be estimated in the NCE estimator? How does it connect to mutual information?

We will show that minimizing the NCE loss $\mathcal{L}$ will result in maximizing the mutual information. First, we prove that optimizing $\mathcal{L}$ will converge the density ratio $f_i(u_i, s_i)$ to $\frac{p(u_i|s_i)}{p(u_i)}$.

Proof. $f_i(u_i, s_i)$ will converge to $\frac{p(u_i|s_i)}{p(u_i)}$ by optimizing $\mathcal{L}$, where $p(u_i \mid s_i)$ is the data distribution and $p(u_i)$ is the noise distribution.

The prediction of $\mathcal{L}$ is $\frac{f_p}{\sum_B f_n}$. Let us denote the optimal probability of classifying positive samples i correctly as $P(i = \text{positive} \mid U, C)$ (it is correct if it comes from the data distribution, and therefore incorrect if it comes from the noise distribution):

$$P(i = \text{positive} \mid U, C) = \frac{p(u_i \mid C) \prod_{j \neq i} p(u_j)}{\sum_{k=1}^{N} p(u_k \mid C) \prod_{j \neq k} p(u_j)} \tag{3.57}$$

$$= \frac{\frac{p(u_i|C)}{p(u_i)}}{\sum_{k=1}^{N} \frac{p(u_k|C)}{p(u_k)}} \tag{3.58}$$

Compare $\frac{f_p}{\sum_B f_n}$ and $P(i = \text{positive} \mid U, C)$ we have,

$$\frac{f_p}{\sum_B f_n} = \frac{\frac{p(u_i|C)}{p(u_i)}}{\sum_{k=1}^{N} \frac{p(u_k|C)}{p(u_k)}} \tag{3.59}$$

Therefore, f_i will converge to $\frac{p(u_i|s_i)}{p(u_i)}$. □

Now, with the optimal f_i, we can proof mutual information $I(u_{i+t}, s_i) >=$

$\log N - \mathcal{L}^{opt}$, where $\mathcal{L}^{opt}$ is the optimal loss. Minimizing the NCE loss $\mathcal{L}$ will result in maximizing the mutual information $I(u_{i+t}, s_i)$.

Proof. The lower bound for $I(u_{i+t}, s_i)$ is $\log N - \mathcal{L}^{opt}$.

We first rewrite $\mathcal{L}$ by separating the positive sample and negative samples explicitly,

$$\mathcal{L} = -\mathop{\mathbb{E}}_{B}\mathop{\mathbb{E}}_{T}\big[\log \frac{f_p(u_{i+t}, s_i)}{\sum_B f_n(u_{i+t}, s_i)}\big] \tag{3.60}$$

$$= -\mathop{\mathbb{E}}_{B}\mathop{\mathbb{E}}_{T}\big[\log \frac{f_p(u_{i+t}, s_i)}{f_p(u_{i+t}, s_i) + \sum_{B_{negative}} f_n(u_{i+t}, s_i)}\big] \tag{3.61}$$

where $B_{negative}$ is the negative samples in batch B, in which there are N samples. By substituting the optimal density ratio f_i in $\mathcal{L}$, we will get the optimal loss $\mathcal{L}^{opt}$:

$$\mathcal{L}^{opt} = -\mathop{\mathbb{E}}_{B}\mathop{\mathbb{E}}_{T}\big[\log \big(\frac{\frac{p(u_{i+t}|s_i)}{p(u_{i+t})}}{\frac{p(u_{i+t}|s_i)}{p(u_{i+t})} + \sum_{B_{negative}} \frac{p(u_{i+t}|s_i)}{p(u_{i+t})}}\big)\big] \tag{3.62}$$

$$= \mathop{\mathbb{E}}_{B}\mathop{\mathbb{E}}_{T}\big[\log \big(1 + \frac{p(u_{i+t})}{p(u_{i+t} \mid s_i)} \sum_{B_{negative}} \frac{p(u_{i+t} \mid s_i)}{p(u_{i+t})}\big)\big] \tag{3.63}$$

$$\approx \mathop{\mathbb{E}}_{B}\mathop{\mathbb{E}}_{T}\big[\log \big(1 + \frac{p(u_{i+t})}{p(u_{i+t} \mid s_i)}(N-1) \mathop{\mathbb{E}}_{B_{negative}}[\frac{p(u_{i+t} \mid s_i)}{p(u_{i+t})}]\big)\big] \tag{3.64}$$

Then, simplify the term $\mathbb{E}_{B_{negative}}[\frac{p(u_{i+t}|s_i)}{p(u_{i+t})}]$. Since $\frac{p(u_{i+t}|s_i)}{p(u_{i+t})}$ is the ratio of two continuous probability densities, it is also continuous and thus we can write

the Expectation term in integral:

$$\mathop{\mathbb{E}}_{B}[\frac{p(u \mid s}{p(u)}] = \int_B \frac{p(u \mid s)}{p(u)} p(u) du \tag{3.65}$$

$$= \frac{1}{p(s)} \int_B \frac{p(u,s)}{p(u)} p(u) du \tag{3.66}$$

$$= \frac{1}{p(s)} \int_B p(u,s) du \tag{3.67}$$

$$= \frac{1}{p(s)} p(s) \tag{3.68}$$

$$= 1 \tag{3.69}$$

Substitue $\mathbb{E}_B[\frac{p(u|s}{p(u)}]$ back in $\mathcal{L}^{opt}$ and we get:

$$\mathcal{L}^{opt} = \mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \big[\log \big(1 + \frac{p(u_{i+t})}{p(u_{i+t} \mid s_i)} (N-1)\big)\big] \tag{3.70}$$

In addition, since random variables U and S both are sampled from the sample distribution P_{data}, $P(U) \leq P(U \mid S)$ (the uncertainty of a random variable becomes smaller once another variable is fixed). Therefore we have the following relationship:

$$\mathcal{L}^{opt} \geq \mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \big[\log \big(\frac{p(u_{i+t}}{p(u_{i+t} \mid s_i)} N\big)\big] \tag{3.71}$$

$$= \mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \big[\log \big(\frac{p(u_{i+t}}{p(u_{i+t} \mid s_i)}\big)\big] + \mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \big[\log N\big] \tag{3.72}$$

$$= - \mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \big[\log \big(\frac{p(u_{i+t} \mid s_i)}{p(u_{i+t}}\big)\big] + \mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \big[\log N\big] \tag{3.73}$$

$$= -I(u_{i+t}; s_i) + \mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \big[\log N\big] \tag{3.74}$$

Therefore, the lower bound for $I(u_{i+t}; s_i)$ is:

$$I(u_{i+t}; s_i) \geq \mathop{\mathbb{E}}_{B} \mathop{\mathbb{E}}_{T} \left[\log N \right] - \mathcal{L}^{opt} \tag{3.75}$$

Minimizing the loss $\mathcal{L}$ will lead to maximizing the mutual information I. □

3.4.2 Shared Encoder Approach

The original proposed CPC model contains only one autoregressive model - an unidirectional RNN. The unidirectional RNN context vectors from the first few frames of a speech signal can be inaccuracte since the RNN has only seen a few frames. It is therefore common to have a bidirectional RNN instead, such as for machine translation applications. However, similar to language modeling such as n-gram language model, the CPC model is trained on future frames prediction and birdirectional RNN, which takes in the whole sequence, contradicts our NCE training objective.

we took inspiration from (**peters2018deep**), which have two separate RNNs, one for forward sequence and one for backward sequence. The two RNNs are jointly trained, and the hidden states are later concatenated together for next word prediction. We proposed the shared encoder approach - two autoregressive models in the same latent space, illustrated in Figure 3.5. Compare to the single autoregressive model, the shared encoder approach has an additinoal autoregressive model for the backward sequence. The two autoregressive models do frame predictions separately but are optimized

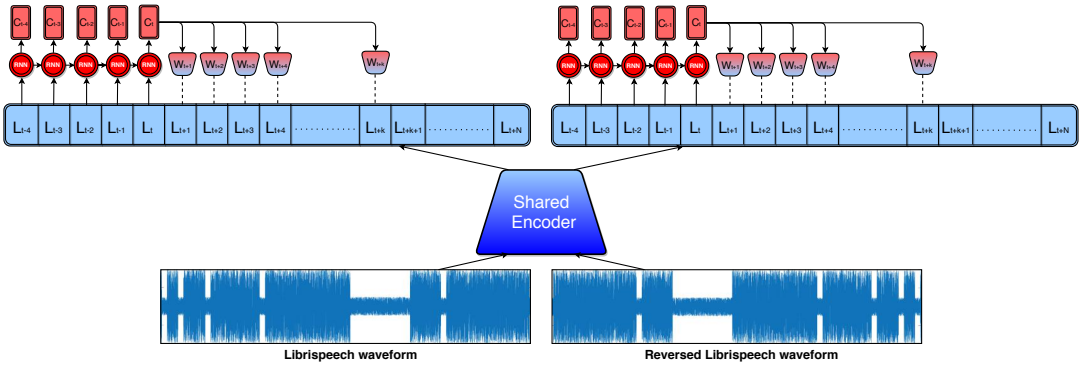


Figure 3.5: CPC Double Autoregressive Model Illustration of the CPC double autoregressive model's training stage. An waveform

jointly with the loss:

$$\mathcal{L}_{joint} = -\frac{1}{2}\mathop{\mathbb{E}}_{B}\mathop{\mathbb{E}}_{T}\left[\log\frac{f_{p1}(u_{i+t}, s_i)}{\sum_B f_{n1}(u_{i+t}, s_i)} + \log\frac{f_{p2}(u_{i+t}, s_i)}{\sum_B f_{n2}(u_{i+t}, s_i)}\right], \tag{3.76}$$

where f_1 is the density ratio from the autoregressive model trained on forward sequence, and f_1 is the density ratio from the second autoregressive model trained on backward sequence. Similar to (**peters2018deep**), we concatenate the context vectors (hidden states) from the two autoregressive models during inference for downstream task (speaker verification).

3.4.3 Detailed Implementation

Most of the CPC model implementation conforms to (**oord2018representation**) with minor modifications. The raw waveform is input to the encoder without being processed with Voice Activity Detection or Mean Variance Normalization. In each training iteration, a segment of 1.28 seconds (or 20480 data points) is randomly extracted from the original waveform for every utterance, before inputting to the encoder. The encoder is a five layers 1-dimensional

CPC model ID	number of GRU(s)	GRU hidden dim	number of GRU layers	CPC feature dim
CDCK2	1	256	1	256
CDCK5	1	40	2	40
CDCK6	2	128	1	256

Table 3.1: CPC Model Summaries

Convolutional Neural Network (CNN) with a 160 downsampling factor. For each of the five layers, the filter (kernel) sizes are $[10, 8, 4, 4, 4]$, the strides are $[5, 4, 2, 2, 2]$, and the zero paddings are $[3, 2, 1, 1, 1]$. All five layers have 512 hidden dimension. In (**oord2018representation**), the autoregressive model is implemented as a GRU with 256 hidden dimension, and the context vector (hidden state) is used as the CPC feature for downstream tasks. However for standard speaker verification systems, 256 input feature dimension would cost weeks to train and therefore it is impractical. We explored three CPC models with different GRU hidden dimension, and a comparison of the three CPC models are detailed in Figure 3.1. CDCK2 and CDCK5 are variants of the single autoregressive model approach, while CDCK6 is based on the shared encoder approach.

To implement the NCE loss $\mathcal{L}$, we draw negative samples from different utterances excluding the current utterance. This can be conveniently implemented by selecting the other samples in the same batch as the negative samples. The advantage of such implementation is that the negative samples can be drawn in one batch of the forward pass. Finally, the timestep k for future frame prediction is set to 12, and the batch size B is set to 64 for all CPC models. Figure 3.6 is a visualization of the details of our CPC model

implementation.

3.5 CPC-based Speaker Verification System

Since CPC feature learns high level information of the given input signal, it could contain relevant speaker information. We are interested in the effectiveness of the CPC feature in speaker verification, and how it fits in a standard speaker verification system. Figure 3.7 describes our CPC-based speaker verification system. The CPC model is trained on the training data, and frame-level representation is extracted by the model. To get a fixed-length utterance-level representation, we either temporally average across all frames for each utterance, or train an additional summarization system, the i-vector extractor. After getting the utterance-level representation, we first mean and length normalize across all representations, and train a Linear Discriminant Analysis to reduce feature dimension per utterance. Lastly, a decision generator, the PLDA model, is trained to get the log-likelihood ratio for each utterance before computing the EER. Figure 3.8 describe the testing pipeline for the CPC-based speaker verification system.

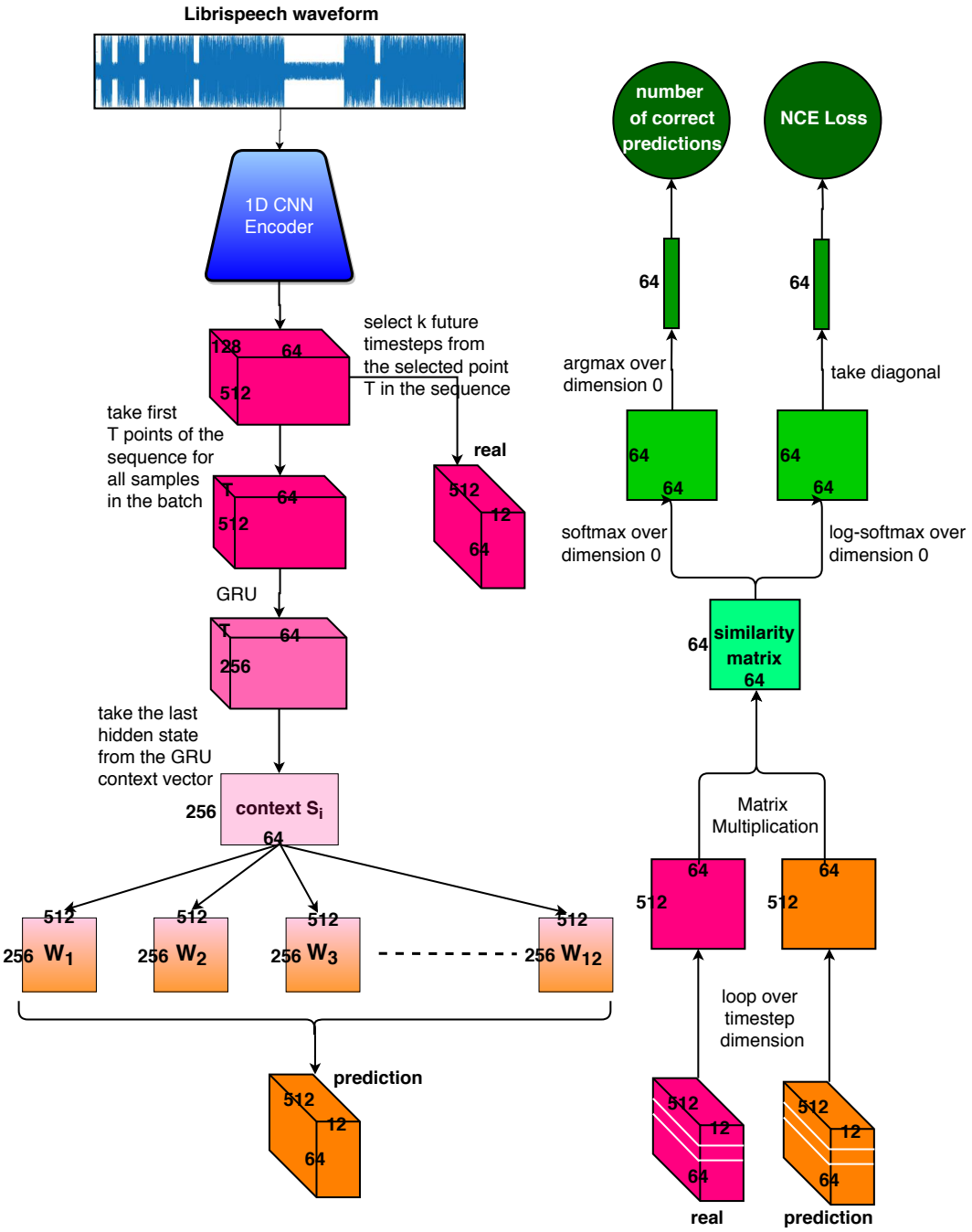


Figure 3.6: Implementation Details of CPC model Illustration of our CPC model implementation.

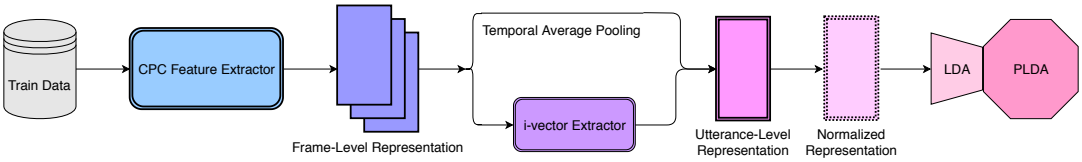


Figure 3.7: CPC-based Speaker Verification System - Training Pipeline Illustration of the training pipeline for CPC-based speaker verification system.

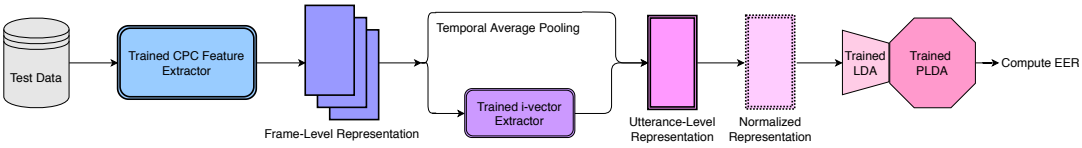


Figure 3.8: CPC-based Speaker Verification System - Testing Pipeline Illustration of the testing pipeline for CPC-based speaker verification system.

Chapter 4

Experiments and Results

4.1 LirbiSpeech

We tested our CPC-model on the LibriSpeech corpus. LibriSpeech Corpus is an 1000-hour speech data set based on LibriVox's audio books (**panayotov2015librispeech**), and it consists of male and female speakers reading segments of book chapters. For example, 1320-122612-0000 means 'Segment 0000 of Chapter 122612 read by Speaker 1320.' The speech data is recorded at 16k Hz. LibriSpeech Corpus is partitioned into 7 subsets, and the description of each subset is summarized in Figure 4.1. In our experiments, we used train-clean-100, train-clean-360, and train-clean-500 subsets for training. Dev-other and dev-test are used as validation and CPC model selection. Finally, we report our speaker verification results on test-clean.

4.2 Speaker Verification Trial List

Since LibriSpeech is originally created for speech recognition, we have to manually create the speaker verification trial list. The trial list contains two

subset	hours	per-spk minutes	female spkrs	male spkrs	total spkrs
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-other-500	496.7	30	564	602	1166

Figure 4.1: LibriSpeech Corpus Summary - number of hours and number of speakers (panayotov2015librispeech)

three columns: enrollment ID, test ID and target/nontarget. The enrollment ID column contains the speech recordings that are enrolled, the test recordings are those tested against the enrollment recordings, and the target/nontarget indicates whether the speaker of the given test recording matches the speaker of the given enrollment recording. Table 4.1 contains three example trials.

enrollment ID	test ID	target/nontarget
908-157963-0027	4970-29095-0029	nontarget
908-157963-0027	908-157963-0028	target
1320-122612-0007	4446-2275-0017	nontarget

Table 4.1: Example of Speaker Verification Trials

We prepared our trial list in two different ways. The first trial list is created by randomly selecting half of the LibriSpeech recordings as enrollment and the other half as test. There are a total of 1716019 trials in the first trial list. The second trial list is also created in the same manner but we made sure that there is no overlap in chapters spoken by the same speaker. For example, the trial '1320-122617-0000 1320-122617-0025 target' is allowed in the first trial list but not in the second trial list. The two trial lists we described above are

CPC model ID	number of epoch	model size	dev NCE loss	dev accuracy
CDCK2	60	7.42M	1.6427	26.42
CDCK5	60	5.58M	1.7818	22.48
CDCK6	30	7.33M	1.6484	28.24

Table 4.2: CPC Model Training Summaries

available for download: first trial list[1] and second trial list[2].

4.3 Speaker Verification EER

We presented the model training results and speaker verification error rate of the three CPC models we implemented in Table 4.2. CDCK2 and CDCK5 are trained for 60 iterations, and CDCK6 is trained for 30 iterations due to time limitation. CDCK5 has around 1.8 million less model parameters than CDCK2 and CDCK6 because its GRU hidden dimension is 40, which is significantly smaller. Expectedly, due to the larger model size, CDCK2 and CDCK6 has smaller NCE losses $\mathcal{L}$ and higher positive sample prediction accuracies than CDCK5. Furthermore, CDCK6 attains higher prediction accuracies with half the training iterations, which suggests that the shared encoder approach is more powerful than the single autoregressive model approach.

Figures 4.2, 4.3, 4.4 are the future frame positive sample prediction accuracies for CDCK2, CDCK5, and CDCK6 respectively. Figures 4.5, 4.6, 4.7 are the NCE losses for CDCK2, CDCK5, and CDCK6 respectively. The reported loss and accuracy are performed on the dev set, and we can see that the losses decrease while the prediction accuracies increase over training iterations. Note

[1] https://drive.google.com/open?id=10h9GH_vi-BRBT_L_xmSM1ZumQ__jRBmx
[2] https://drive.google.com/open?id=1FDOU1iNSdGT-IMCQnuuJCWV421168x4H

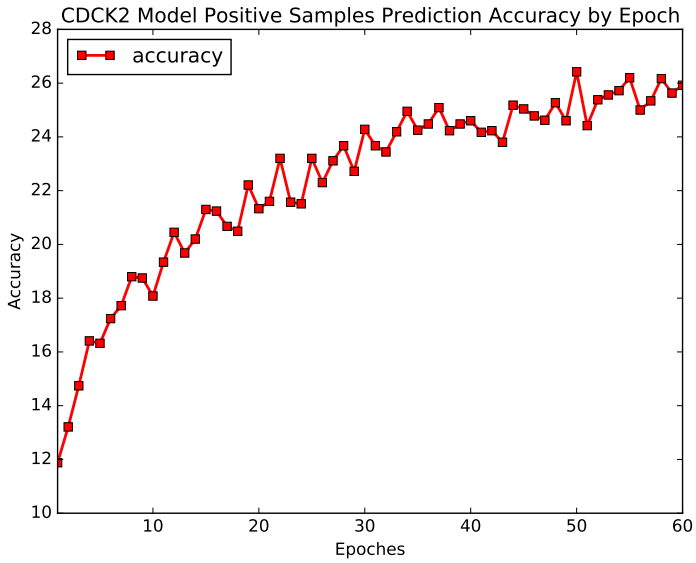


Figure 4.2: Positive Samples Prediction Accuracy of CDCK2 on development set over training iterations

that the NCE loss $\mathcal{L}$ is averaged over all future prediction timesteps $1, 2, .., k$, and the prediction accuracy is calculated only on the last timestep k. In our implementation, k is set to 12. Therefore, $\mathcal{L}$ is averaged over 12 timesteps, but the positive sample prediction accuracy is on the 12^{th} timestep only.

After the CPC models are trained, the context vectors (hidden states) of the models are extracted as the CPC features. These features are used as the input feature for speaker verification. We explored two approaches to summarization in the speaker verification system described in Figure 3.7: temporal average pooling and i-vectors. In the first approach, temporal average pooling, frame-level features are averaged across frames to get a fixed-length utterance-level feature for each utterance. The speaker verification results of the CPC features and the baseline MFCC features with temporal average pooling is summarized in Table 4.3. We can first see that the speaker verification EER of

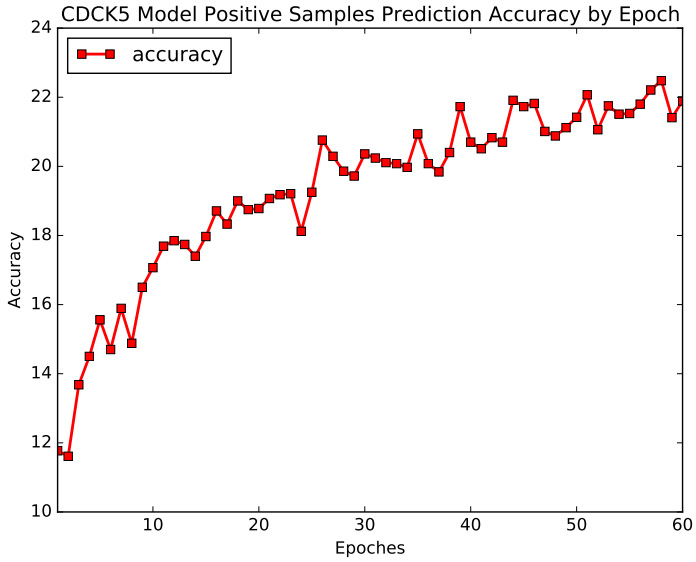


Figure 4.3: Positive Samples Prediction Accuracy of CDCK5 on development set over training iterations

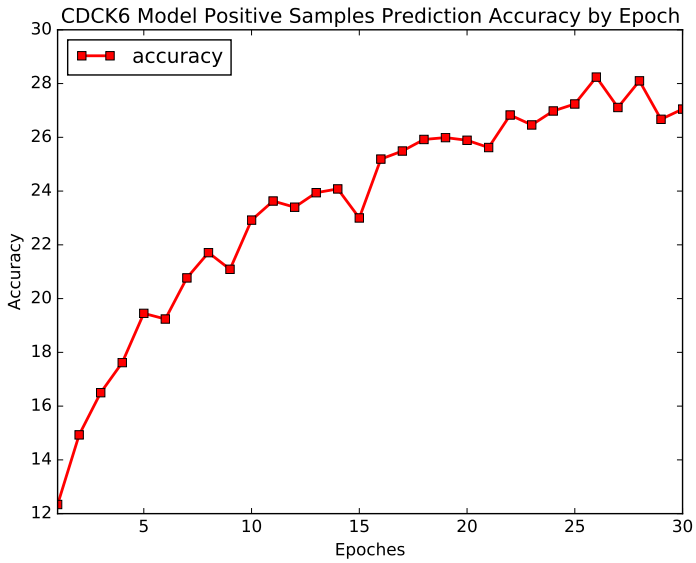


Figure 4.4: Positive Samples Prediction Accuracy of CDCK6 on development set over training iterations

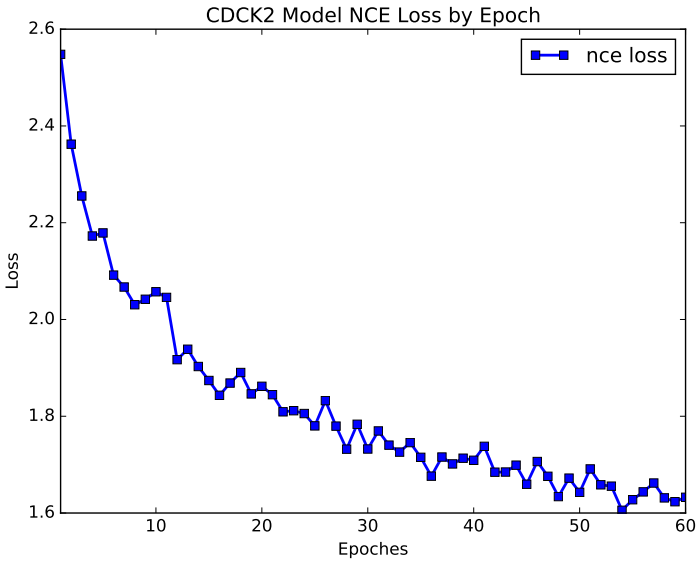


Figure 4.5: NCE Loss of CDCK2 on development set over training iterations

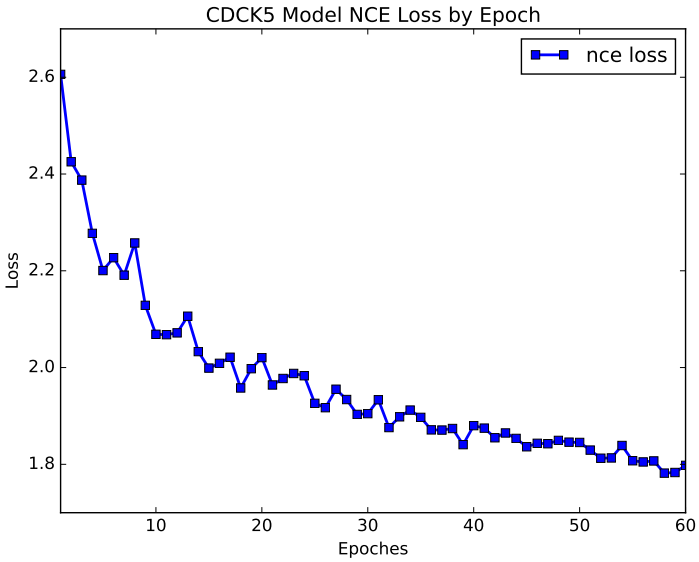


Figure 4.6: NCE Loss of CDCK5 on development set over training iterations

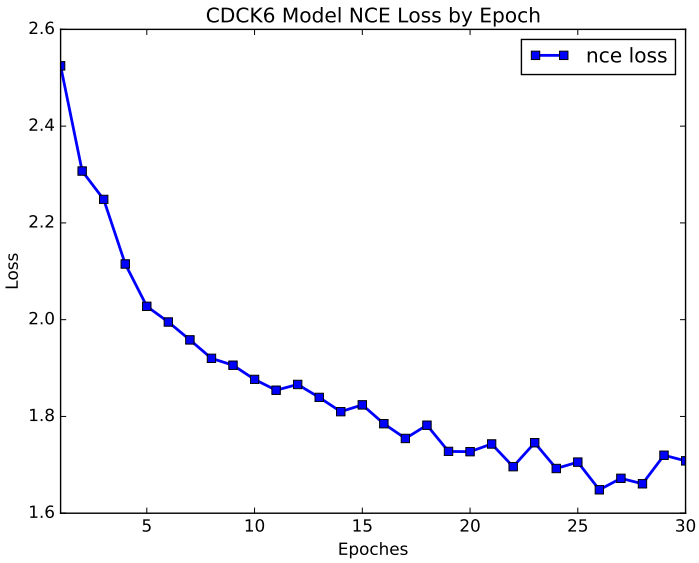


Figure 4.7: NCE Loss of CDCK6 on development set over training iterations

Feature	Feature Dim	Summarization	LDA Dim	1st EER	2nd EER
MFCC	24	average pooling	24	9.211	13.48
CDCK2	256	average pooling	200	5.887	11.1
CDCK5	40	average pooling	40	7.508	12.25
CDCK6	256	average pooling	200	6.809	12.73

Table 4.3: Speaker Verification Results on LibriSpeech test-clean-100 - Summarization with Average Pooling

the first trial list is significantly lower than that of the second trial list. This is expected since the second trial list contains no speaker-chapter overlap between enrollment and test, and thus the higher error rate. Secondly, CPC features show significant improvement over MFCC. Specifically, features from CDCK2 model recorded 5.887 and 13.48 EER, which are 36% and 18% relative improvements over the baseline. Although CDCK6 showed lower NCE loss and higher prediction accuracies during training, its features performed worse than the ones from CDCK2.

Feature w PCA	Original Feature	PCA Dim	PCA Variance Ratio
CDCK2-36	CDCK2	36	76.76
CDCK2-60	CDCK2	60	87.40
CDCK5-24	CDCK5	24	93.39
CDCK6-36	CDCK6	36	82.30
CDCK6-60	CDCK6	60	90.31

Table 4.4: CPC features applied with PCA Summary

The second approach to summarization in speaker verification is i-vectors, which also gives a fix-length utterance level feature for each utterance. However, as mentioned earlier, usually the feature dimension to i-vectors is below 60. A feature dimension of 256 will take weeks to train an i-vector extractor. Therefore, dimension reduction on frame-level CPC features is first performed before summarization. We chose Principal Componenet Anaysis (PCA) for reducing the CPC feature dimension because we do not want to introduce extra nonlinearity for the learned feature and PCA is a linear transform. Table 4.4 is the summary of the CPC features after PCA transform with their corresponding PCA variance ratio, and the feature dimensions are all smaller or equal to 60 after PCA.

Table 4.5 presents the result of various MFCC, CPC, and combinations of MFCC and CPC features for speaker verificaiton with i-vectors. We can see that i-vectors with MFCC alone got 5.518 and 8.157 EER on the two trial lists. We trained three i-vectors systems with CPC features after PCA: CDCK2-60, CDCK5-24, and CDCK6-60. We can see that these features achieved up to 11% EER relative improvement over the baseline on the first trial list. The relative improvements are much smaller compare to their counterparts in Table 4.3. Furthermore, on the second trial list, MFCC with i-vectors prevails CPC with

Feature	Feature Dim	Summarization	1st EER	2nd EER
MFCC	24	i-vectors	5.518	8.157
CDCK2-60	60	i-vectors	5.351	9.753
CDCK5-24	24	i-vectors	4.911	8.901
CDCK6-60	60	i-vectors	5.228	9.009
MFCC + CDCK2-36	60	i-vectors	3.62	6.898
MFCC + CDCK5-24	48	i-vectors	3.712	6.962
MFCC + CDCK6-36	60	i-vectors	3.691	6.765

Table 4.5: Speaker Verification Results on LibriSpeech test-clean-100 - Summarization with i-vectors

i-vectors.

Since MFCC and CPC are two very different feature extraction methods, they should capture different aspects of the speech signal, which may be complementary for speaker verification. We fused MFCC and CPC features before i-vectors by simply concatenating the two feature vectors. The last three rows of Table 4.5 show the results of fusing MFCC with CPC features after PCA. We can see that the best combinations attains 34% and 17% relative improvements over MFCC i-vectors on the two lists.

4.4 Feature Visualizations

It is a good practice to visualize speech features, and we visualize the CPC features and compare them to MFCC. Since CPC features from model CDCK2 and CDCK6 are 256 dimension, which may contain too much visual details, we chose to visualize CPC feature from CDCK5, which has 40 dimension. Figure 4.8 and 4.9 are visual comparisons of MFCC and CPC features on two randomly picked LibriSpeech test-clean-100 utterances: 2830-3980-0028 and

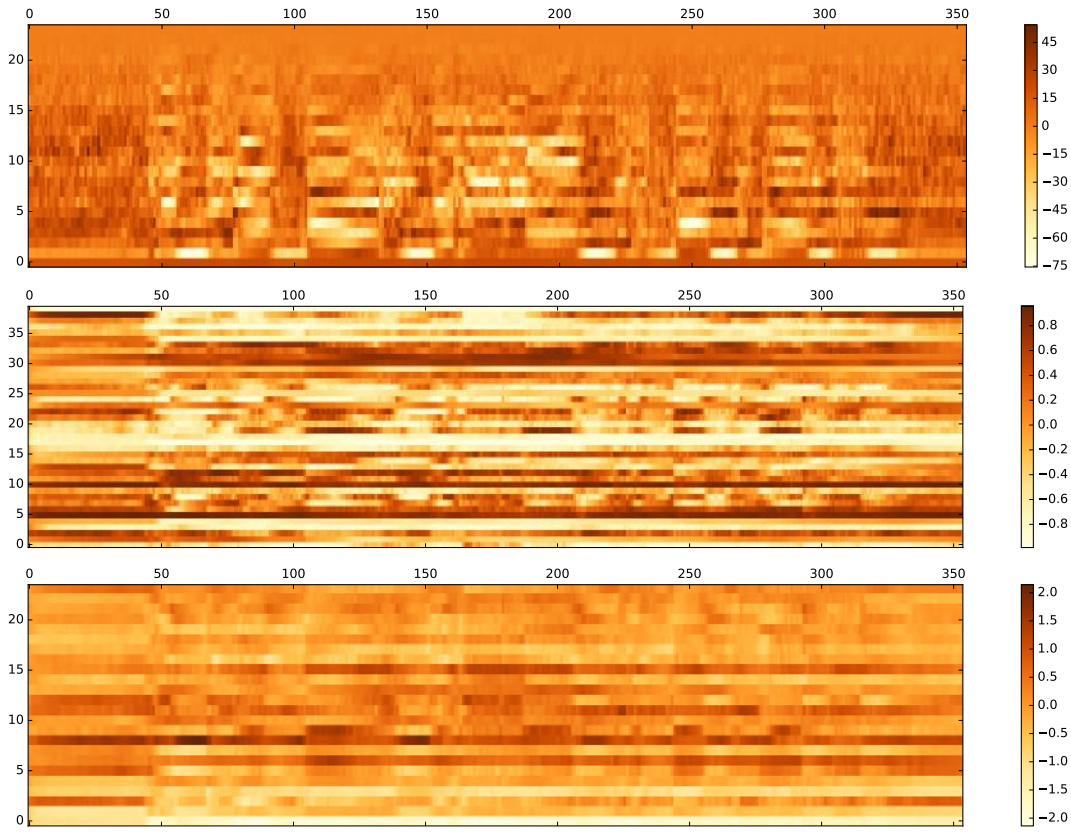


Figure 4.8: A visual comparison of MFCC (top), CPC (middle), and CPC with PCA (bottom) for utterance 2830-3980-0028

5105-28241-0017. We also visualize CPC features with PCA transform, CDCK5-24. Looking at the visualizations, CPC and MFCC bear very little similarity that they differ in structure and magnitude. However, one observation worth noting of the CPC features is that there are several feature bins whose values remain in a small range over time, which signifies that the CPC features learn some global information that lasts over time.

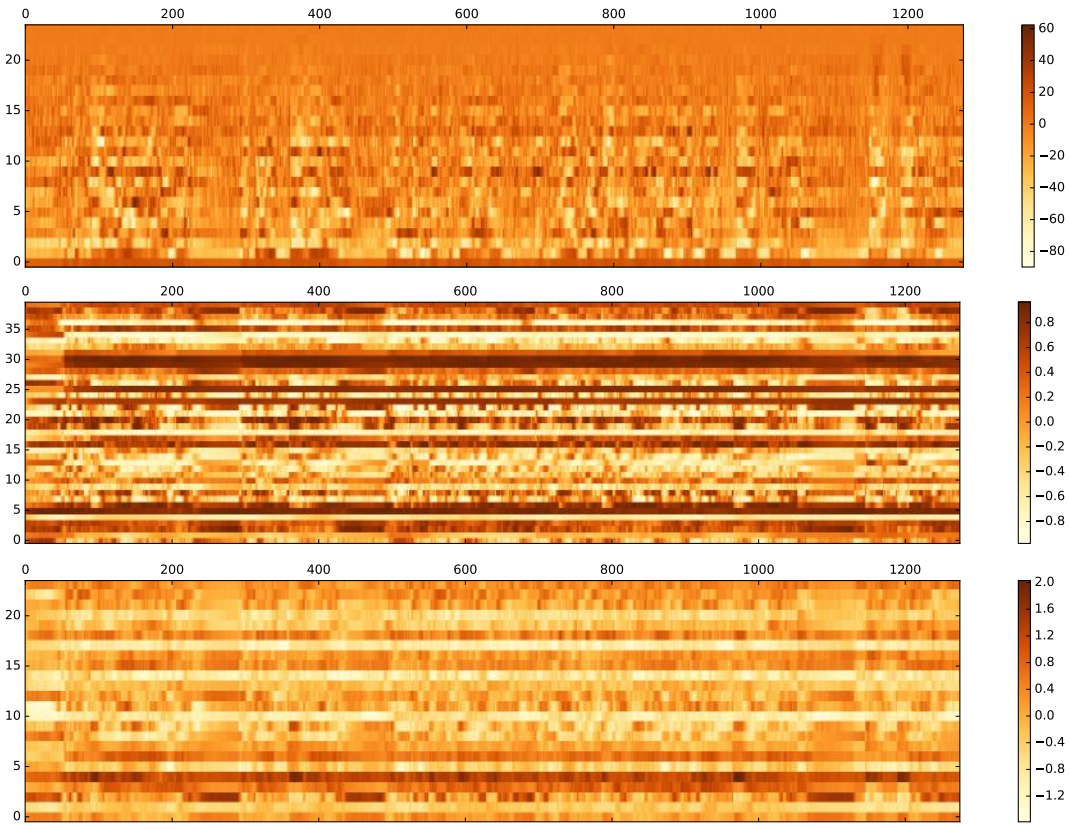


Figure 4.9: A visual comparison of MFCC (top), CPC (middle), and CPC with PCA (bottom) for utterance 5105-28241-0017

4.5 Speaker Verificaiton DET Curves

To examine the tradeoff between false alram and miss rate, we plotted the Detection Error Tradeoff (DET) curves for the CPC and MFCC based speaker verification system. Figure 4.10 and 4.11 are DET curves for MFCC and CPC fusion-based i-vectors speaker verification system. For both trial lists, we can see that the fusion features reduced the miss and false alarm probabilities compared to the baseline.

Figure 4.12 and 4.13 are DET curves for CPC i-vectors speaker verification

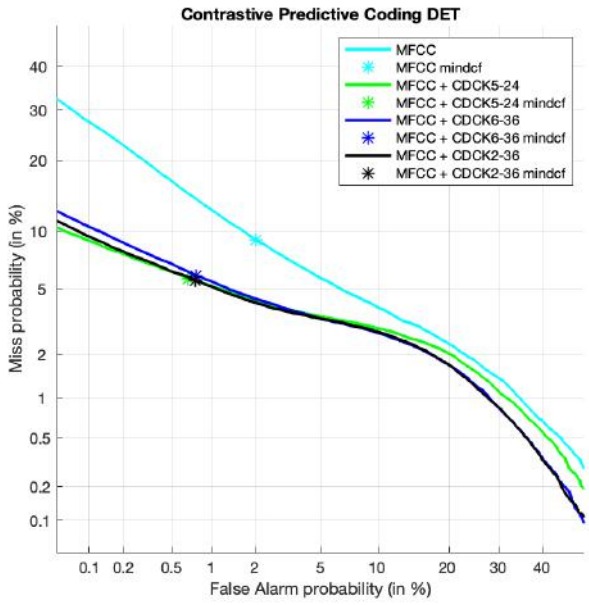


Figure 4.10: 1st trial list DET curve for CPC and MFCC feature-level fusion i-vectors speaker verification system

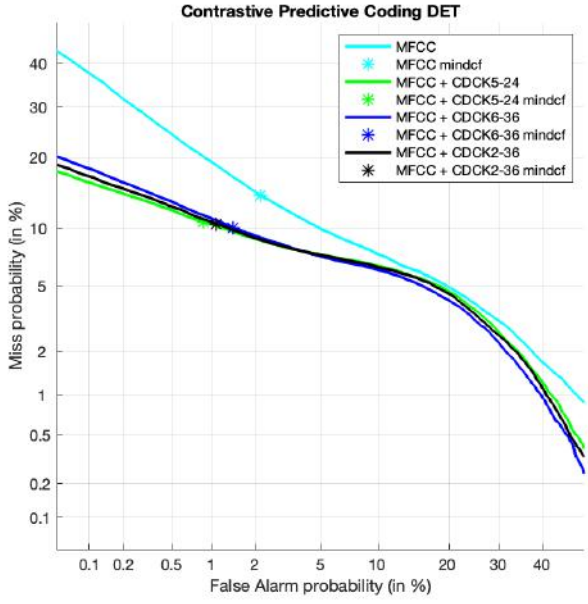


Figure 4.11: 2nd trial list DET curve for CPC and MFCC feature-level fusion i-vectors speaker verification system

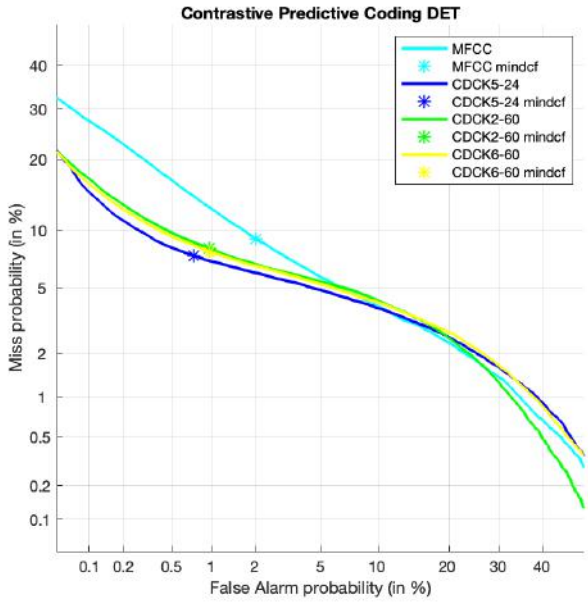


Figure 4.12: 1st trial list DET curve for CPC i-vectors speaker verification system

system. CPC i-vectors attained lower miss and false alarm probabilities only on the first trial list compared to MFCC i-vectors.

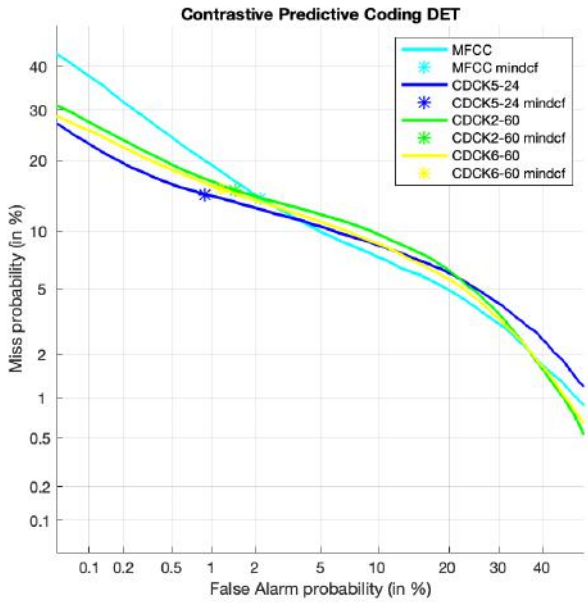


Figure 4.13: 2nd trial list DET curve for CPC i-vectors speaker verification system

Chapter 5

Discussion and Conclusion

5.1 CPC as an Alternative Feature for Speaker Verification

Common speech and speaker recognition systems employed deterministic Fourier-Transform-based features, such as MFCC, FilterBanks, or Peceptual Linear Predictive (PLP). In this work, we explored an unsupervised learned feature, CPC, for speaker verification task. We showed that CPC attains competitive speaker verification accuracy on LibriSpeech corpus, and it is presented as a potential alternative feature for future speaker verification research.

5.2 i-vectors is not an Ideal Summarization Method for CPC

i-vectors is one of the most popular features for speech analysis tasks. It is widely used for speaker recognition, language identification, speech recognition, etc. However, one constraint that i-vectors imposed on the input feature

is that it has multi-Guassian distributed. If the input feature does not comply to a multi-Guassian distribution, GMM-UBM and hence i-vectors would not likely to work. From our experiments, we observed that i-vectors is not an ideal summarization method, that summarizes frame-level feature into utterance-level feature, for CPC compared to MFCC i-vectors. Compare Table 4.3, which shows speaker verification EER of CPC features with average pooling, and Table 4.5, which shows the EER of CPC features with i-vectors. CPC shows very strong results over MFCC with average pooling as the summarization method. On the other hand, when i-vectors is used as the summarization method, CPC does not show clear advantage oer MFCC. One speculation is that CPC features are not multi-Guassian distributed, and hence there may be better summarization method, such as the x-vectors, which does not assume any input distribution on the input features.

5.3 CPC Complements MFCC for i-vectors Speaker Verification

We observed that CPC complements MFCC for i-vectors speaker verification system. Table 4.5 contains results of CPC and MFCC feature fusion with i-vectors, which give improvements over both MFCC i-vectors and CPC i-vectors. Similarly, Figure 4.10 and 4.11 are the fusion i-vectors DET curves, which are better than that of CPC features 4.12 and 4.13. Therefore, we hypothesize that CPC complements MFCC for i-vectors based speaker verification system on the LibriSpeech corpus. However, whether this is true for all speech data is left for future work.

5.4 Future Work

Looking ahead, there are several directions for this work worth exploring. We listed five potential improvements and applications we would like to work on in the near future.

5.4.1 Density Estimation Methods

First of all, we followed (**oord2018representation**) and used Noice Contrastive Estimation for estimating the density ratio for learning high-level representation. There are other possible density estimation methods we can experimented with, such as the Importance Sampling. We are curious with the effectiveness of NCE and how it compares to other density estimation methods.

5.4.2 SRE16

Librispeech corpus is a relatively clean (little noise) datasets that was originally made for speech recognition. Although the results we presented show potentials, we have to tested on publicly recognized datasets. In addition, we manually created our own trial lists since LibriSpeech does not provide one. We could not compare our findings to other speaker verification systems. We are planning to conduct CPC model refinements and speaker verification experiments on NIST SRE16 with the data in Table 5.1[1].

[1]Based on https://github.com/kaldi-asr/kaldi/tree/master/egs/sre16/v1

Corpus	LDC Catalog No.
SWBD2 Phase 1	LDC98S75
SWBD2 Phase 2	LDC99S79
SWBD2 Phase 3	LDC2002S06
SWBD Cellular 1	LDC2001S13
SWBD Cellular 2	LDC2004S07
SRE2004	LDC2006S44
SRE2005 Train	LDC2011S01
SRE2005 Test	LDC2011S04
SRE2006 Train	LDC2011S09
SRE2006 Test 1	LDC2011S10
SRE2006 Test 2	LDC2012S01
SRE2008 Train	LDC2011S05
SRE2008 Test	LDC2011S08
SRE2010 Eval	LDC2017S06
Mixer 6	LDC2013S03

Table 5.1: Training Data List for SRE16

5.4.3 CPC x-vectors

As mentioned previously, i-vectors may not be the ideal summarization methods for CPC. We plan to conduct x-vectors (**snyder2018x**) speaker verification experiments after switching to SRE16.

5.4.4 Language Identification

We would also like to conduct CPC experiments on language identification[2]., which uses techniques from speaker recognition. Since CPC is designed to capture global information, it should learn some degree of language information in addition to speaker information of a speech signal.

[2]Based on https://github.com/kaldi-asr/kaldi/tree/master/egs/lre07

5.4.5 Domain Adaptation for Speaker Recognition

Finally, we would like to apply CPC for speaker recognition domain adaptation. Although there are signs that CPC may not generalize well to unseen conditions 4.5, we are interested to see how CPC can be used in that context.

Vita

Cheng-I Jeff Lai grew up in Taiwan. At age 15, he left home and rent a room at Taipei to study at Taipei Municipal Jianguo High School. At age 18, Cheng-I attended Johns Hopkins University with a desire to study biophysics until he met Prof. Najim Dehak, who convinced him the beauty and delicacy of human spoken language. He subsequently dedicated a good amount of his time on speech processing and speaker recognition research, with a focus on deep learning approahces to speech. In Cheng-I's Sophomore and Junior year, he interned at the Human Language Technology Center of Excellence (HLTCoE) and the Informatics Forum, University of Edinburgh. He will receive a Bachelor's degree in Electrical Engineering in December, 2018. Beginning February, 2019, Cheng-I will start as a research assistant at Center for Language and Speech Processing and also interview for Ph.D. programs.