Universitat Politècnica de Catalunya

Bachelor's Thesis

Multi-speaker Neural Vocoder

A dissertation submitted to the Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona Universitat Politècnica de Catalunya by

Oriol Barbany Mayor

In partial fulfillment of the requirements for the degree in Telecommunications Technologies and Services Engineering

Advisor: Dr. Antonio Bonafonte

Barcelona, June 2018

Abstract

Deep learning has revolutionized almost every engineering branch over the past decades and have also been successfully applied to text-to-speech, where it yields state-of-the-art performance and overcomes classical approaches.

This work focuses in the implementation of a speech synthesis system based in Recurrent Neural Networks (RNNs) that holds many speakers with a unique model. Despite the fact that other systems only share some layers across speakers but maintain independent blocks for every identity, this dissertation explore the possibilities of implementing an adaptation of the end-to-end model SampleRNN conditioned to both speech parameters and speaker identity that allow an entire shared framework.

Speech parameters, which in this thesis are extracted with Ahocoder, are proofed to be intrinsically dependent of the speaker. This allows to use the simplest presented adaptation of the SampleRNN as a low-rate vocoder that significantly reduces the bit-rate given that only acoustic parameters have to be sent instead of the whole waveform in order to decode an utterance. This behavior proof the efficiency of the proposed shared structure, but hinders other tasks such as voice conversion due to the futility of speaker identity in the model. Some experiments regarding the model parameters have been conducted to optimize the results and a novel approach of non-causality named as look ahead is proposed.

With the aim of tackling the voice conversion problem given the speaker dependencies, a bottle neck approach inspired by the VQ-VAE model and an adversarial framework are proposed to obtain latent speaker-independent representations from speech parameters. This also allows a new kind of speaker interpolation named as *eigen-voice* representation that aims to mimic a new and unknown speaker given a short utterance. Promising results arise from this proposal as it could be used to tackle one of the only inconveniences of deep learning, which is the huge amount of data that the models demand to learn the structure of an input signal, and could also be used for other tasks like hardening the security of speech-based biometric systems.

Resum

Durant les últimes dècades, l'aprenentatge profund o *deep learning* ha revolucionat pràcticament totes les branques de l'enginyeria i ha estat aplicat amb èxit en la síntesi de veu, on obté els millors resultats sobrepassant amb diferència els anteriors assolits amb sistemes clàssics.

Aquest treball se centra en la implementació d'un sistema de síntesi de veu basat en xarxes neuronals recurrents amb un únic model per varis locutors. Encara que altres sistemes únicament comparteixen algunes capes entre parlants però mantenen blocs independents per a cada locutor, aquesta tesis explora les possibilitats d'implementar una adaptació del model SampleRNN condicionant tant a paràmetres propis de la parla com a la identitat del parlant que permet una estructura compartida.

Es demostra que els paràmetres de la parla, extrets mitjançant l'Ahocoder en aquesta tesis, són intrínsecament dependents del locutor. Aquest fet permet usar l'adaptació més simple presentada del SampleRNN com un vocoder de baixa velocitat que redueix significativament la taxa de bits transmesa ja que només es necessita enviar els paràmetres propis de la parla per descodificar un senyal de veu. Aquest comportament demostra l'eficiència de l'estructura compartida proposada, però dificulta altres tasques com la conversió de veu per la futilitat de la identitat del locutor en el model. S'han portat a terme varis experiments pel que fa a paràmetres del model per tal d'optimitzar els resultats i es proposa un nou enfocament de no causalitat anomenat *look ahead*.

Amb l'objectiu d'abordar el problema de la conversió de veu donat les dependències amb el locutor, es proposa una estratègia de coll d'ampolla inspirada pel VQ-VAE i una arquitectura adversativa per tal d'obtenir representacions latents dels paràmetres de la parla sense informació del parlant. Això també permet un nou tipus d'interpolació de locutors anomenada representació *eigen-voice* que pretén imitar la veu d'un locutor nou i desconegut amb un petit fragment de senyal de veu. Aquesta proposta aporta resultats prometedors ja que es podria fer servir per afrontar un dels únics desavantatges de l'aprenentatge profund, que és l'enorme quantitat de dades que es necessiten perquè un model aprengui l'estructura del senyal d'entrada, i també per altres tasques com la millora de la seguretat en sistemes biomètrics basats en la veu.

Resumen

Durante las últimas décadas, el aprendizaje profundo o *deep learning* ha revolucionado prácticamente todas las ramas de la ingeniería y ha estado aplicado con éxito en la síntesis de voz, donde obtiene los mejores resultados sobrepasando con diferencia los anteriores obtenidos con sistemas clásicos.

Éste trabajo se centra en el desarrollo de un sistema de síntesis de voz basado en redes neuronales recurrentes con un único modelo para varios locutores. Aunque otros sistemas únicamente comparten algunas capas entre hablantes pero mantienen bloques independientes para cada locutor, ésta tesis explora las posibilidades de implementar una adaptación del modelo SampleRNN condicionado a parámetros propios del hable y a la identidad del locutor que permite una estructura compartida.

Se demuestra que los parámetros del habla, extraídos mediante Ahocoder en ésta tesis, son intrínsecamente dependientes del locutor. Éste hecho permite usar la adaptación más simple presentada de SampleRNN como un vocoder de baja velocidad que reduce significativamente la tasa de bits transmitida puesto que solo necesita enviar los parámetros del habla para decodificar una señal de voz. Éste comportamiento demuestra la eficiencia de la estructura compartida propuesta, pero dificulta otras tareas como la conversión de voz por la futilidad de la identidad del locutor en el modelo. Se realizaron varios experimentos referentes a los parámetros del modelo con la intención de optimizar los resultados y se propone un nuevo enfoque de no causalidad al que se refiere como *look ahead.*

Con el objetivo de abordar el problema de la conversión de voz dadas las dependencias con el locutor, se propone una estrategia de cuello de botella inspirada por VQ-VAE y una arquitectura adversativa para obtener representaciones latentes de los parámetros del hable sin información del hablante. Esto también permite un nuevo tipo de interpolación de locutores llamada representación *eigen-voice* que pretende imitar la voz de un locutor nuevo y desconocido con un pequeño fragmento de señal de voz. Esta propuesta aporta resultados prometedores porque se podría usar para combatir uno de los únicos inconvenientes del aprendizaje profundo, que es la enorme cantidad de datos que se necesitan para que un modelo aprenda la estructura de una señal de entrada, tanto como para otras tareas como la mejora de la seguridad en sistemas biométricos basados en voz.

"The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves."

Dartmouth Artificial Intelligence Conference, 1955

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Dr. Antonio Bonafonte for the continuous support of my bachelor's thesis and related research, for his patience, motivation, and immense knowledge in both speech synthesis and deep learning. His guidance helped me in all the time of investigation and writing of this thesis.

I thank Georgina Dorca, who loaned me her source code from the bachelor's thesis and helped me understand it in early stages when all seemed a mess to me, as well as Santiago Pascual, who helped me fine-tune the architecture and tackle some difficulties with Pytorch.

My sincere thanks also goes to the TALP research center and the administrators of the computing resources who gave me access to the laboratory and research facilities. Without they precious support it would have not be possible to conduct this research.

Besides the research fellows, I would like to thank all who kindly helped in the development of several experiments by taking their subjective evaluations.

Last but not the least, I would like to thank my family: my parents and my sister for supporting me throughout writing this thesis and my life in general.

Revision history and approval record

Revision	Date	Purpose
0	10/04/2018	Document creation
1	14/06/2018	State of the art chapter revision
2	30/06/2018	Introduction and Methodology chapters revision
3	01/07/2018	Final revision

Document Distribution List

Name	e-mail
Oriol Barbany Mayor	oriolbm96@gmail.com
Dr. Antonio Bonafonte	antonio.bonafonte@upc.edu

Written by:		Reviewed and approved by:	
Date	10/04/2018	Date	01/06/2018
Name	Oriol Barbany Mayor	Name	Dr. Antonio Bonafonte
Position	Project Author	Position	Project Supervisor

Contents

List of Figures

List of Tables

List of Abbreviations

ADAM	Adaptive Moment Estimation
ANN	Artifitial Neural Network
BPTT	Back-Propagation Through Time
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
IAF	Inverse Autoregressive Flow
LPC	Linear Predictor Coding
LSB	Less Significant Byte
LSTM	Long Short-Term Memory
MFCC	Mel-Frequency Cepstral Coefficient
MLP	Multi-Layer Perceptron
MOS	Mean Opinion Score
MSB	Most Significant Byte
NLL	Negative Log-Likelihood
NLP	Natural Language Processing
QRNN	Quasi-Recurrent Neural Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SPSS	Statistical Parametric Speech Synthesis
SST	Speech-to-Speech Translation
TBPTT	Truncated Back-Propagation Through Time
TTS	Text-to-Speech
VAE	Variational AutoEncoder

1 Introduction

This project was carried out at the Speech Processing Group (VEU) which belongs to the Signal Theory Department of the Universitat Politècnica de Catalunya (UPC) and works on language engineering and speech processing. Its main areas of research are speech coding, robust speech features extraction, text-to-speech conversion, speech recognition, speech-to-speech translation and spoken language. Its main objective is to develop verbal communication systems with the aim of overcoming linguistic barriers and enhancing the accessibility of information systems.

The purpose of this project is to take advantage of the state-of-the-art technologies of speech synthesis, in particular of the SampleRNN, an unconditional end-to-end neural audio generation model. During the development of this thesis, SampleRNN was rearrange in a way that output utterances were conditioned and therefore could be used as a Text-to-Speech (TTS) system, and speaker identity could be chosen. With that model, several applications including voice synthesis, a low-rate vocoder, voice conversion and mimicking of previously unknown voices could be achieved.

1.1 Statement of purpose

This work aims to develop a vocoder, which is the second stage of a TTS system as the one depicted in figure 1.1. The mission of the vocoder therefore is to synthesize speech given some parameters extracted either from raw text or from other speech signals in case of voice coding. The classical approach for vocoder is the so-called Statistical Parametric Speech Synthesis (SPSS), but new approaches based on deep learning have outperformed these results. Deep learning is based in learning very complex functions that model the distribution of the training data in such a way that it can generalize to unseen data. When referring to speech synthesis, a particular architecture called Recurrent Neural Network (RNN) is normally used in order to model the temporal evolution of the input signals. Initially, each model was trained with only one speaker, but recent studies showed the advantage of holding many speakers with a same shared structure. This allows to achieve better results because speech signals from different speakers have a common behavior that is exploited with a Deep Neural Network (DNN).

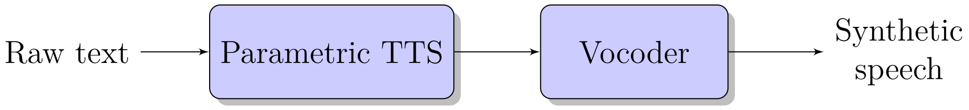


Figure 1.1: Block diagram of a two-stage Text-To-Speech system

The main project initial idea was provided by the advisor of this work, Antonio Bonafonte, who was first contacted by the author with the aim of supervising the development of alternative projects that also involved deep learning although that the applications were mainly centered on the music field. At the beginning of this project, two proposals were considered, including both an speaker dependent and independent vocoder.

In the first option, the speaker was conceived as one of the inputs of the vocoder and therefore the content of the acoustic conditioners was thought as not to be enough to describe its identity. The second option, which was focused on speech coding, bandwidth expansion and speech enhancement, was about considering the opposite case where speaker is not relevant. Despite the thesis was oriented to the speaker dependent vocoder proposal, results found during this dissertation showed that acoustic parameters that are normally used to condition a model in a TTS system such as Mel-Frequency Cepstral Coefficients (MFCCs) and the fundamental frequency or pitch are very related to the intrinsic characteristics of its generator. This is why in the first stage of the work, results could be used for a low-rate vocoder instead of its original purpose which included voice conversion. Nevertheless, new deep learning architectures developed during this dissertation got latent representations without speaker information that allowed to achieve the main objectives of the chosen proposal.

1.2 Project requirements and specifications

The main requirement of this project is to implement a multi-speaker vocoder based on Deep Learning and able to generate voice from speech parameters interfered from an input signal using a system like Ahocoder (Erro et al., 2014). This vocoder is able to generate voices attributed to different speakers within the training database with a unique model to hold them all. Once the speech parameters are independent from the speaker, it can also change the speaker identity and therefore perform voice conversion. Moreover, the final step of the thesis consists on being able to mimic any voice that could be expressed using the known speakers from the corpus and to synthesize voice from this representation. Some parallel work was derived from each of that proposals involving applications like speech coding and some new architectures were developed to achieve the project requirements.

Project specifications at first glance were set as a TTS system with a score of 3.5 or more in the subjective 5-scale Mean Opinion Score (MOS) test in naturalness. State of the art vocoders have recently outperformed baseline statistical parametric and concatenative speech synthesizers reducing the gap between natural speech and best previous model by more than 50% and reaching scores around 4.2 in North American English[1]. It is so justified to set an achievable target based on the order of magnitude of the current vocoders.

1.3 Methods and procedures

This project was conceived as the continuation of the work published in the SampleRNN paper, which provides an open-source code[2] and therefore promotes advances in the speech synthesis fields as it avoids start replicating the paper from scratch. The original implementation was in Theano, a Python library for fast numerical computation that can be run on the Central Processing Unit (CPU) or Graphics Processing Unit (GPU) but an implementation in the deep learning library Pytorch was preferred as a baseline model[3]. Moreover, this code created by Kozakowski and Michalak (2017) allows training the model with an arbitrary number of tiers, whereas the original implementation allows maximum 3 tiers, and use Gated Recurrent Units (GRUs) as recurrent layers.

[1] Results presented on *Wavenet: A generative model for raw audio* (Kleijn et al., 2017)

[2] https://github.com/soroushmehr/sampleRNN_ICLR2017

[3] https://github.com/deepsound-project/samplernn-pytorch

From the beginning, this dissertation was conceived as a continuation of a previous bachelor's thesis carried by Dorca Saez (2018) and supervised by Antonio Bonafonte. The generated code was therefore recycled and used as starting point since some adaptations and improvements have been added on the baseline Pytorch model. The code is written in Python 3 as the programming language and it uses the previously mentioned library Pytorch, which is a deep learning framework that provides Tensors and Dynamic neural networks in Python with strong GPU acceleration. Additionally, some Bash scripting has been used to start the pertinent experiments and generate samples with the desired models, which all have been run with GPUs at the computer cluster *CALCULA* pf VEU group.

1.4 Work Plan

The project was structured in the Work Packages exposed below. The breakdown of each one and the Gantt diagram with the planned schedule were presented as an independent document and are included in appendix B.

WP1	Environment familiarization
WP2	Theoretical learning
WP3	Multi-speaker Neural Vocoder
WP4	Voice Conversion
WP5	*Eigen-voice* representation
WP6	Documentation

1.5 Incidences

The major incidence was related to data preparation. This step took more long than expected and some troubles in the training process arose. Modifications on PyTorch parameters such as learning rate, weight normalization or scheduler were applied and results became as expected.

Moreover, models have been training between 3 and 6 days, which sometimes delay the modifications as some of the evaluations cannot be done in early stages of the experiments were output is basically random noise. The huge amount of required memory due to the large database needed for this project also lagged the resource allocation on the CALCULA servers. Some of the hypothesis were wrong at first glance, which made the project found non-desired yet interesting and usable solutions as explained in section 1.1.

Overall, this obstacles made the project move forward slowly than predicted, which excluded the integration of a parametric speech synthesis model acting as the first stage of the TTS system from the desired results and reduced the time scheduled for some other work packages such as the *eigen-voice* representation. This is further justified in chapter 6, where a future work section is presented with the desired evolution of this work including the aforementioned integration.

2 State of the art

Speech synthesis, also known as TTS, is defined as the artificial production of human voice. The aim is therefore to generate an audio signal that utters an input text with as naturalness and intelligibility as possible. Naturalness describes how closely the output sounds like human speech, while intelligibility is the ease with which the output is understood. In this chapter, some of the current approaches will be discussed and differentiated under the umbrellas of classical and deep learning-based approaches. The aim is to give a brief introduction to the background of this project making special emphasis on the deep learning papers that inspired the methodology explained in chapter 3.

2.1 Classical approaches

Despite deep learning-based speech synthesizers have yield the state-of-the-art performance, the yet most widely used system called concatenative units (section 2.1.1) differ from these and will be reviewed. In addition, SPSS, which has leveraged the speech synthesis research during the last decade will be analyzed (section 2.1.2). With these two methods, a trade-off intrinsic to the nature of each one shows up. On the one hand, concatenative synthesis offer high-quality results, but on the other hand they are not as robust neither they offer as much flexibility when dealing with voice characteristics as SPSS.

2.1.1 Concatenative synthesis

The large popularity of this system and the reason why it has been operating during many years, lies in the fact that it uses real speech, which is recorded in a variety of prosodic contexts to fit as many usage cases as possible, and thus it has an unquestionably high level of naturalness. In one of the most well-known approaches of concatenative synthesis (Hunt and Black, 1996), it is proposed to segment signals from a large speech database by phonemes (although some other works segment by diphones[1]). Then, those segments, also known as units, are concatenated with the aim of generating the desired utterance given a target phoneme sequence extracted from text.

When concatenating units to generate quality speech, it is important to take into account the discontinuities of e.g. phase and pitch as well as the differences in prosody between those units. As it was mentioned before, the same units are usually recorded with different expressiveness to adapt the target prosody on every context. Although that having a large database means a high probability of founding a matching unit, some processing tools are often needed to smooth discontinuities and force the prosody to match. These transformations degrade the quality but offer a higher flexibility without needing a huge database.

The selection of units is therefore performed by minimizing a mixture of a target cost C^t and a concatenation cost C^c that are pondered by the weight ρ as in equation (2.1). The

[1] A diphone is a voice unit with the length of a phoneme but defined in between of two phonemes as it is the more stable point and the less influenced by co-articulation, which is the effect of neighboring phonemes.

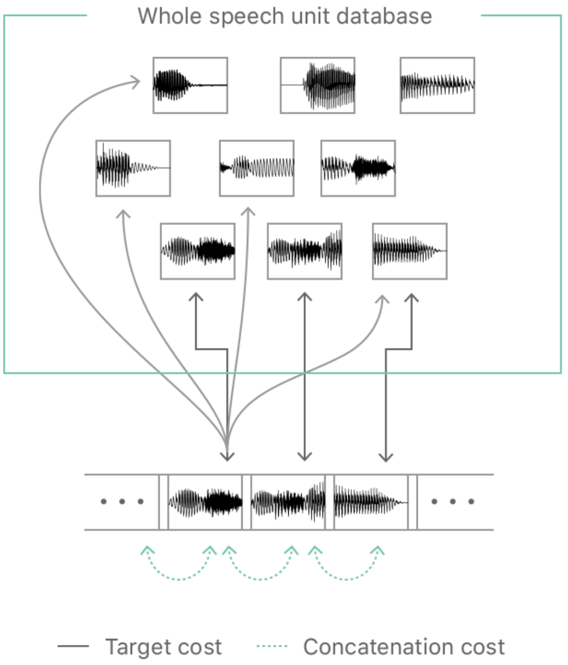


Figure 2.1: Illustration of the unit selection method based on target and concatenation costs from Deep Learning for Siri's Voice (Apple)

minimization of this total cost aims to find the pre-recorded unit or segment from the database u_i that best matches a set of targets t_i derived from the text using the system front-end. For instance, this target could contain the desired duration, prosodic information, identity of speaker, etc. Optimal unit selection can be performed with the Viterbi algorithm (J. Viterbi, 1967). However, to obtain near real-time synthesis on large speech databases, containing tens of thousands of units, a pruned version is used.

$$\hat{u}_1^N = \arg\min_{u_1^N} \rho \sum_{i=1}^{N} C^t(u_i, t_i) + (1-\rho) \sum_{i=2}^{N} C^c(u_{i-1}, u_i) \tag{2.1}$$

The target cost C^t measures the perceptual similarity between a unit from the database u_i and a target unit t_i, i.e. the worthiness of getting a certain unit to match the desired prosody. It is defined as a sum of $p-$weighted differences between the target elements and the features of u_i, which are extracted from the source sounds, and usually consist of a prosodic feature vector with the pitch and duration of the phonemes.

On the other hand, the concatenation cost takes into account the discontinuities between successive units u_i and u_{i-1}. Thus, it is defined zero when evaluating the concatenation cost of consecutive units that have been segmented from a single speech signal of the database. These two costs are depicted in figure 2.1.

Overall, concatenative synthesis offer quality results but demands a large database to fit all prosodic contexts.

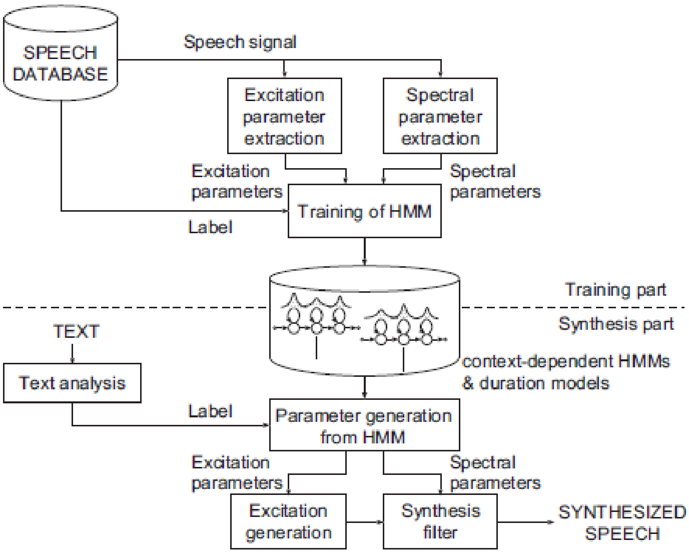


Figure 2.2: HMM-based speech synthesis system from Black et al. (2009)

2.1.2 Statistical Parametric Speech Synthesis

A different approach is SPSS, being the most popular variation the Hidden Markov Model (HMM)-based generation synthesis, which offers more customization since it is based on choosing the desired speaker characteristics (such as the duration of phonemes, expressiveness, identity, etc.) and synthesizing the artificial utterance with mathematical models, i.e. not directly dealing with real data once models are trained. This means that the same database used in Concatenative Synthesis could be used to train the models, but the system could be implemented without that data, as the needed parameters are already learned. This happens again with all the DNN-based approaches, and therefore this is also considered a machine learning technique. SPSS popularity lies in its flexibility to change prosodic and spectral content of the generated voice. Another highlight of this technique is its low memory footprint once the model is trained due to the lack of an intrinsic speech database like in concatenative synthesis, even though that quality is not as high than with real speech segments.

Other milestone of this technique is the so-called speaker adaptation, which allows to incorporate new speakers despite a hypothetic low amount of data by fulfilling the model by adapting or combining those corresponding to other speakers (Black et al., 2009). SPSS also allows speaker interpolation, which is based in combining the characteristics of the already trained speakers to synthesize new voices. This concept will be further reviewed and a new proposal based on it, is exposed in section 3.3.

In a first stage, the speech signal is represented in a parameterized way, meaning that it gets encoded in a vocoder that creates an acoustic feature vector in a windowed frame. The linguistic parameters of each phoneme in a given phonetic and prosodic context are then represented by a stochastic generative model, usually a HMMs, and finally parametric representation is followed and some speech parameters of both types spectral like MFCCs and excitation such as pitch are extracted (as depicted in figure 2.2).

2.2 Deep Neural Network-based Speech Synthesis

Over the past decades, deep learning has revolutionized almost every engineering branch such as that of interest which is Natural Language Processing (NLP), that includes but is not limited to speech synthesis, machine translation, information retrieval and speech recognition. Although the idea of simulating the behavior of neurons was presented 75 years ago (McCulloch and Pitts, 1943), it hasn't really been large-scale applied since last years, when the symbiosis of new algorithms, the availability of huge amounts of data to handle and an increase of computation power enhanced by GPUs has allowed to implement functional DNNs that operate in reasonably short times. With the deep learning approach, speech synthesis has surpassed the trade-off that was briefly exposed in section 2.1 by providing high quality speech as unit selection synthesis does with the flexibility and robustness of SPSS.

In those problems called time series, which are basically about forecasting sequential discrete data with a natural order, there are two major deep learning architectures: RNNs and Convolutional Neural Networks (CNNs). The predicted values are based on the previously observed ones and in a DNN this is achieved by either having temporal connections that provide memory to the network or by taking a time span of sample as inputs respectively. Raw audio signals are challenging to model because they have correlations at both neighboring samples and between ones thousand of samples apart.

For those readers who are not familiarized with neural networks and their mentioned architectures, please refer to appendix A before continuing with this section as some previous knowledge is assumed.

In this appendix, first the basic concept of Artifitial Neural Networks (ANNs) and the Multi-Layer Perceptron (MLP) are introduced. Also some fundamental concepts that will be mentioned during this work such as classification and regression, parameter optimization, backpropagation, stochastic gradient descent, mini-batch, or overfitting are explained in there. Then, the appendix introduce advanced architectures that will be used in a variety of experiments such as RNNs, that allow to model temporal series, and their deep variants such as the well-known GRU and Long Short-Term Memory (LSTM). CNNs are also explained and their use in both its original purpose which is image processing and as a substitute of RNNs is detailed. Finally, the concept of embeddings to represent categorical features is also discussed. All these concepts are the fundamentals of all the architectures proposed for speech synthesis and that are explained in this chapter and in the proposal.

An unintuitive improvement to DNN-based speech synthesis had been driving by focusing the problem as it was about classification instead of regression even when data is implicitly continuous. Categorical or Multinoulli distribution is more flexible and can more easily model arbitrary distributions because it doesn't assume any distribution of the data. This implies that output is no longer a real value but a discrete one represented by a class. In case of having 8-bit integer values to quantize the signal, this means that there are 256 possible discrete values and thus 256 classes. Reconstructed speech signals achieve less quality degrading when non-linear instead of uniform quantization is applied. Therefore, a previous $\mu-$law companding transformation (ITU-T. Recommendation G. 711, 1988) is used before classifying into the hypothetic 256 classes.

A model to hold many speakers out of the same shared DNN structure also achieve better results than learning the parameters of a single isolated speaker Fan et al. (2015). In Pascual de la Puente (2016) it is proposed to have some common layers to exploit the linguistic information shared by all speakers, with a multi-output architecture with isolated paths for each of the speakers. This suggests that internal representation is shared among multiple speakers. In this case, the learning of the base shared structure can be transferred for a new speaker to achieve speaker adaptation like in SPSS (see section 2.1.2) with limited training data, achieving good results in naturalness and similarity to the original speaker.

Voice conversion is another NLP problem which consists in changing the speaker given a signal uttered by another subject. It could be though that adding an input with the speaker ID would be enough to train the network to synthesize with another speaker given some speech parameters. Nevertheless, these speech parameters are not independent from the speaker and this is the latent representation that is used to train the network. This means that the speaker label can be obviated and no effective voice conversion is achieved with this approach. This conclusion was stated by Chou et al. (2018), who introduced the idea of training like in a Generative Adversarial Network (GAN). This concept will be further reviewed in section 3.2.2.

Some relevant works regarding DNN-based speech synthesis have been already exposed, but the most relevant papers for the development of this thesis are exposed in the following sections. First of all, Wavenet and their parallel variant are exposed for being this approach the one that yet achieves better quality metrics. After this, the baseline model for this project is more extensively exposed as chapter 3 is entirely based on it. Therefore, some equations of the model and the reason of multi-level architecture with modules running at different clock rates is argued. The following paper is VQ-VAE, which inspired the bottle-neck experiment exposed in section 3.2.1 and presents an interesting topic called as voice-style transfer. The last paper was included for the techniques that were proposed on it for adapting the state of the art models for a real-time TTS that could be implemented on a mobile CPU. This was a great milestone due to the good results obtained despite of the huge compression of the model and the parallelization in generation.

2.2.1 WaveNet

WaveNet (van den Oord et al., 2016) is a DNN for generating raw audio waveforms. The model is fully probabilistic and autoregressive, with the predictive distribution for each audio sample conditioned on all previous ones. A single WaveNet can capture the characteristics of many different speakers with equal fidelity, and can switch between them by conditioning on the speaker identity.

The joint probability of a waveform $\boldsymbol{x} = \{x_1, \ldots, x_T\}$ is factorized as a product of conditional probabilities (equation (2.2)), which distribution is modeled by a stack of convolutional layers. There are no pooling layers as the output of the model has the same dimensionality as the input, which is formed using the previously predicted samples. The use of causal convolutions, ensures that the model cannot depend on any future steps, and the dilation allows to increase the receptive field by orders of magnitude without needing many layers or large filters that would greatly increase computational cost. The result of this combination are the so-called causal dilated convolutions, which are depicted in figure 2.3.

$$p(\boldsymbol{x}) = \prod_{t=1}^{T} p(x_t | x_1, \ldots, x_{t-1}) \tag{2.2}$$

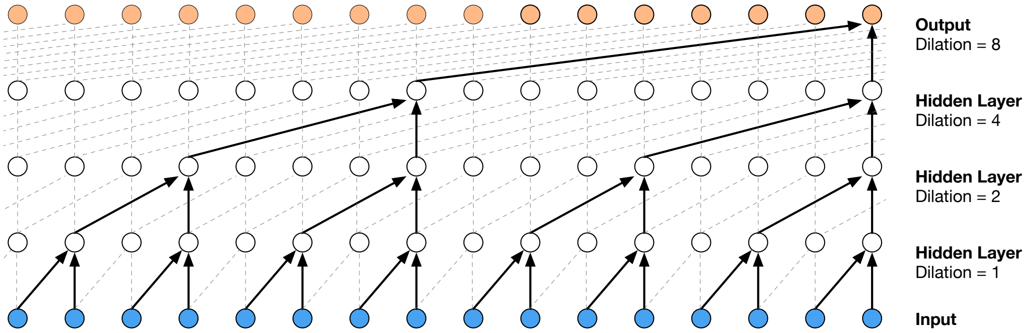


Figure 2.3: Visualization of a stack of dilated causal convolutional layers from van den Oord et al. (2016)

WaveNet doubles the dilation for every layer up to a limit and then repeats it by stacking one or more of the same blocks like the one shown in figure 2.3. Exponentially increasing the dilation factor translates into exponential growth of the receptive field with depth, meaning that with 32 layers 1024 samples can be encompassed. Note that this is done by only considering two inputs for each convolution.

WaveNet can be conditioned in order to produce speech with the required characteristics given a desired utterance. In a multi-speaker setting, the speaker could be chosen by feeding its identity as an extra input and, in the same manner, linguistic features about the text would be feed for TTS. In this case, the joint probability of the waveform $\boldsymbol{x}$ is conditioned by an additional input $\boldsymbol{h}$ and can be expressed as in equation (2.3).

$$p(\boldsymbol{x}|\boldsymbol{h}) = \prod_{t=1}^{T} p(x_t|x_1, \dots, x_{t-1}, \boldsymbol{h}) \tag{2.3}$$

The model can be conditioned on other inputs with both global and local conditioning. On the one hand, global conditioning is characterized by a single latent representation $\boldsymbol{h}$ that influences the output distribution across all timesteps, e.g. a speaker embedding in a multi-speaker setting. On the other hand, local conditioning is a second time-series h_t with a lower sampling frequency than the audio signal, e.g. linguistic features in a TTS. The time-series has to be transformed using a learned up-sampling that maps it to a new time-series with the same resolution as the audio signal. This concept of global and local conditioners will be further and deeper reviewed in chapter 3.

The achieved receptive field of WaveNet is of 240 milliseconds, which encompasses a lot of audio samples, e.g. with a typical sampling frequency of 16kHz this would mean a scope of 3840 samples. Nevertheless, WaveNet was found to sometimes have unnatural prosody by stressing wrong words in a sentence, which could be due to the long-term dependency of fundamental frequency (F_0) contours. This was solved by also taking into account other linguistic features to condition the model as the F_0, which was predicted with an external model at a lower frequency (200 Hz) and thus allowed to learn long-range dependencies.

Parallel WaveNet

Training a WaveNet is quite fast, as the CNN architecture allows to parallelize all time steps as they outputs and inputs are known. This approach take ground truth $\boldsymbol{x}$ as input samples, following thus what is called as professor forcing (Lamb et al., 2016). When generating samples, however, each input sample must be drawn from the output distribution before it can be fed in as input at the next time step. Therefore, WaveNet is poorly suited to parallel processing and hard to deploy in a real-time production setting. The resulting update called parallel WaveNet, is capable of generating high-fidelity speech samples with 1000× speed-up relative to original WaveNet. This translates to more than 20 times faster than real-time (van den Oord et al., 2017a).

Inverse Autoregressive Flows (IAFs) are stochastic generative models whose latent variables are arranged so that all elements of a high dimensional observable sample have a known distribution and therefore can be generated in parallel. The goal of this work was to marry the best features of both models: the efficient training of the original WaveNet and the time-saving sampling of IAF networks. This is achieved with what they refer as Probability Density Distillation.

The procedure was to use a fully-trained WaveNet model as a teacher to train the IAF, also called as the student in this particular framework. The IAF is smaller than the original WaveNet and, although the architecture is also based in dilated convolutions, the generation of each sample does not depend on any of the previous generated samples. This means that generation can be done in parallel and therefore is better suited to modern computational hardware.

The student network is randomly initialized and is fed random white noise and speaker conditioners as an input with the aim of generating a continuous audio waveform as output once trained. Therefore, to generate the correct distribution for timestep x_t, the IAF can implicitly infer what it would have output at previous timesteps $x_1, \ldots, x_{t-1}$ based on the noise inputs $z_1, \ldots, z_{t-1}$ and the known speech conditioners, which allows it to output all x_t in parallel given z_t. Transforming uncorrelated noise into structured samples requires more training time compared to CNNs but the inference is much faster as required in production environments.

During training, the waveforms generated by the student network are then fed to the teacher, which scores each sample based on the similarity of original WaveNet's outputs. The loss function to backpropagate is the Kullback-Leiber(KL) between the teacher's and the student's distribution. Some extra loss functions are added, including perceptual loss to avoid bad pronunciations, contrastive loss to reduce the noise, and a power loss to help match the energy of real human speech.

2.2.2 SampleRNN

SampleRNN (Mehri et al., 2017) is a model for generating unconditional audio sample by sample. It combines memory-less modules with RNNs in a hierarchical structure as depicted in image 2.4, where the lowest tier processes individual samples and each higher module operates on an increasingly longer timescale and at a lower temporal resolution. This is done to tackle the dependencies that raw audio signals have both with neighboring samples and over very long time spans.

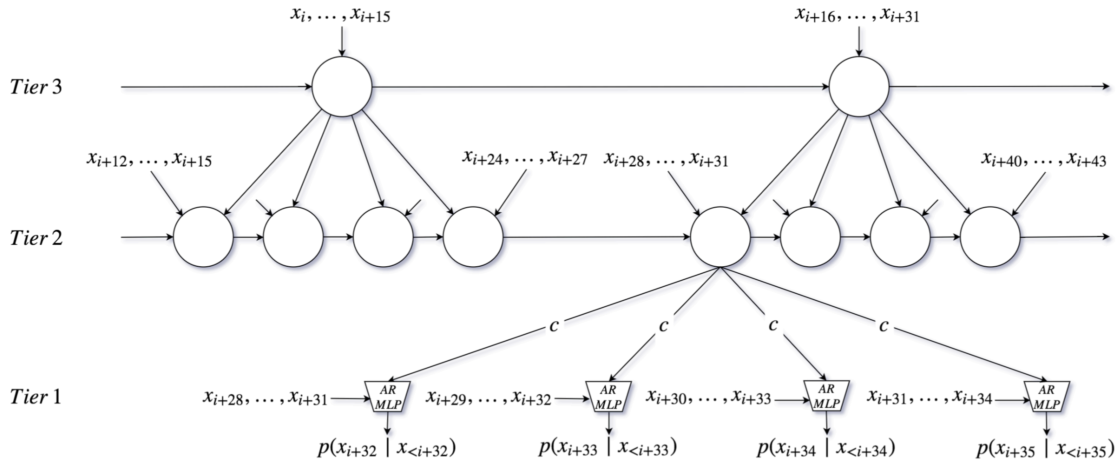

$x_i, \dots, x_{i+15}$
$x_{i+16}, \dots, x_{i+31}$
Tier 3
$x_{i+12}, \dots, x_{i+15}$
$x_{i+24}, \dots, x_{i+27}$
$x_{i+28}, \dots, x_{i+31}$
$x_{i+40}, \dots, x_{i+43}$
Tier 2
c
c
c
c
Tier 1
$x_{i+28}, \dots, x_{i+31}$
AR MLP
$x_{i+29}, \dots, x_{i+32}$
AR MLP
$x_{i+30}, \dots, x_{i+33}$
AR MLP
$x_{i+31}, \dots, x_{i+34}$
AR MLP
$p(x_{i+32} \mid x_{<i+32})$
$p(x_{i+33} \mid x_{<i+33})$
$p(x_{i+34} \mid x_{<i+34})$
$p(x_{i+35} \mid x_{<i+35})$

Figure 2.4: Unrolled model at timestep i with $K = 3$ tiers Mehri et al. (2017). As a simplification, only one RNN and up-sampling ratio $r = 4$ was used for all tiers.

Due to the unconditional nature of SampleRNN, its outputs are basically random audio that maintain the structure of the input, being this speech or music in the author's proposal. This means that it cannot be considered a vocoder that could be used for TTS. The conditioning of this model to speech and speaker parameters was done during this dissertation and the process is explained in chapter 3.

As mentioned before and in contrast to WaveNet, SampleRNN has different modules operating at different clock-rates and thus have more flexibility in allocating the amount of computational resources in modeling different levels of abstraction. This can be used to allocate very limited resources to the module responsible for sample level alignments and more resources to model long dependencies which vary slowly, e.g. identity of phoneme being spoken. The entire hierarchy is able to model the joint probability of a waveform (equation (2.2)) and is jointly trained end-to-end by back-propagation.

The frame-level modules, i.e. all tiers excluding the first one with $k = 1$, are deep RNNs that operate on non-overlapping frames of different frame sizes $FS^{(k)}$ and summarize the history of its inputs into a fixed-length conditioning vector $\boldsymbol{h}_t^{(k)}$ with t related to the clock-rate of that tier for the next module downward. This hidden state is a function $\mathcal{H}$, representing one of the known memory cells reviewed in section A.3, that depend on the previous hidden state and an input (equation (2.4)).

$$\boldsymbol{h}_t^{(k)} = \mathcal{H}\left(\boldsymbol{h}_{t-1}^{(k)}, \boldsymbol{inp}_t^{(k)}\right) \tag{2.4}$$

For the top tier $k = K$, $\boldsymbol{inp}_t^{(K)}$ is the input frame, i.e. the previously predicted samples, and for intermediate modules ($1 < k < K$) it is a linear combination of the conditioning vector from the higher tier and the current input frame with a learned up-sampling done with either a fully-connected layer or an equivalent 1D-convolution with a unit size kernel[2] (equation (2.5)). The purpose of the up-sampling is to map the vector $\boldsymbol{c}$ into a series of $r^{(k)}$ vectors (equation

[2] 1D-convolutions with kernel size 1 are popular in deep learning literature because they are equivalent to fully-connected linear layers but accept inputs with three dimensions, being this the number of filters also named as channels. The inconvenient of linear layers is that, by definition, they have an input shape of (N_{batch}, L) and don't accept extra dimensions.

(2.6)), being $r^{(k)}$ the ratio between temporal resolutions of the modules.

$$\boldsymbol{inp}_t^{(k)} = \begin{cases} \boldsymbol{W}_x^{(k)} \boldsymbol{f}_t^{(k)} + \boldsymbol{c}_t^{(k+1)}; & 1 \leq k < K \\ \boldsymbol{f}_t^{(k=K)}; & k = K \end{cases} \tag{2.5}$$

$$\boldsymbol{c}_{(t-1)*r+j}^{(k)} = \boldsymbol{W}_j^{(k)} \boldsymbol{h}_t^{(k)}; \quad 1 \leq j \leq r \tag{2.6}$$

The sample-level module at time t outputs a distribution over a sample x_{t+1} conditioned on the $FS^{(1)}$ preceding samples and the conditioning vector from the upper tier $\boldsymbol{c}_t^{(k=2)}$. The first tier has a small associated frame size and, given that no memory is needed for considering correlations to nearby samples, a MLP with a softmax output is used to speed up the training (equation (2.7)).

$$p(x_{t+1}|x_1, \ldots, x_t) = Softmax(MLP(\boldsymbol{inp}_t^1)) \tag{2.7}$$

The input vector is computed with equation 2.5 as in the other tiers. Nevertheless, the frame used in the first tier $f_t^{(1)}$ is not formed by raw audio but with samples such as e_t, which represent the raw audio sample x_t after passing through an embedding layer of $FS^{(1)}$ outputs of q different discrete values, e.g. $q = 256$ for an 8-bit quantization (equation 2.8).

$$\boldsymbol{f}_t^{(1)} = flatten([e_{t-FS^{(1)}+1}, \ldots, e_t]) \tag{2.8}$$

In order to tackle the computation charge of training RNNs and thus improve the efficiency of this process, Truncated Back-Propagation Through Time (TBPTT) is used. It basically consists in splitting each sequence into short subsequences (best results were reported to be obtained with a length of 512 samples) with the aim of propagating gradients only to the beginning of each subsequence instead of the whole sequence. Despite TBPTT is used, SampleRNN can still mimic the existing long-term structure of the data because of the hierarchical architecture.

2.2.3 VQ-VAE

VQ-VAE stands for Vector Quantized-Variational AutoEncoder (VAE), and it was presented within the paper titled Discrete Neural Representation Learning (van den Oord et al., 2017b). A VAE is a two-stage network which first stage encodes a given input into a latent representation of smaller dimension with a known distribution, typically a unit Gaussian. The second stage, decodes the latents to obtain a reconstruction as similar to the input as possible.

This architecture therefore have a bottle-neck that provokes a feature compression followed by an inverse structure that achieve the input size. VQ-VAE has been successfully tested in different types of data such as audio, image and video.

Unlike VAE, VQ-VAE has discrete instead of continuous latent codes and the samples drawn from these index an embedding table; that is why they reference the Vector Quantifier in the naming. These latent embeddings plus a one-hot embedding for the speaker, are then used to

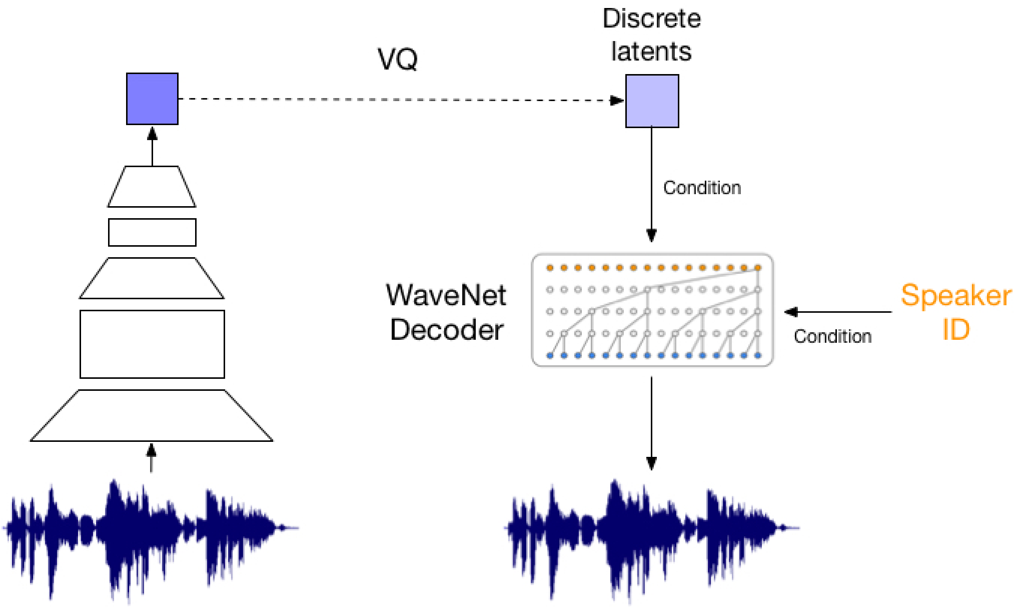


Figure 2.5: Framework of a VQ-VAE applied to speech synthesis with independent speaker conditioner to allow voice conversion from Aäron van den Oord's (principal author of VQ-VAE) webpage

condition a the decoder network in training, which in the case of audio synthesis has a dilated convolutional architecture similar to WaveNet decoder as depicted in figure 2.5.

The reconstruction has the same text contents, but the waveform is quite different and prosody in the voice is altered. This means that the VQ-VAE learns a high-level abstract space that is invariant to low-level features and only encodes the content of the speech. This experiment confirms that important features are often those that span many dimensions in the input data space (in this case phoneme and other high-level content in waveform).

The main interests in this work lie in the voice-style transfer, which is in line with one of the main purposes of this dissertation. It consists in conditioning the decoder of the VQ-VAE on a speaker embedding which is different that the one who generated the latents, and therefore allowing voice conversion. The success of VQ-VAE in this task suggests that the encoder factors out speaker-specific information in the encoded representation as they have same meaning across different voice characteristics. This behavior arises naturally because the decoder gets the speaker-id for free so the limited bandwidth of latent codes gets used for other speaker-independent phonetic information. In fact, the latent codes discovered by the VQ-VAE were shown to actually be very close related to the human-designed alphabet of phonemes. This was confirmed as the accuracy of mapping every of the 128 possible latent values to one of the 41 possible phoneme values by taking the conditionally most likely phoneme, took a value of 49,3% with an unsupervised classifier, while a random latent space would result in an accuracy of 7,2% (prior most likely phoneme).

2.2.4 Efficient Neural Audio Synthesis

The work proposed by Kalchbrenner et al. (2018), describe a set of techniques aimed to reduce the sampling time and computational complexity to implement a real-time TTS system on a mobile CPU. On the one hand, the computation time attributed to each layer is reduced by

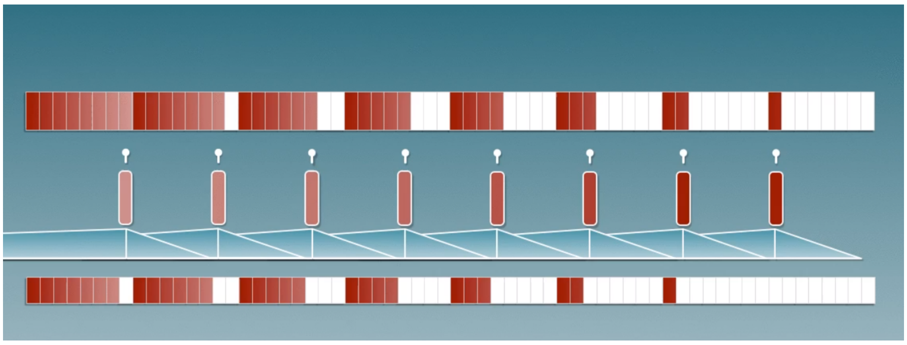


Figure 2.6: In this model, there are $B = 8$ sub-tensors expressed by a triangle that represents the conditioning on the already generated samples in the row below. The delay is $\tau = 8$, and the darker samples represent the first column of each row of the 2-D shape of sub-tensors. Image extracted from an animation created by DeepMind

sparsifying the model during training, i.e. deleting parameters in the network. It is demonstrated that large models with sparsity of up to 96% work better and faster that small dense models.

On the other hand, also better *CUDA kernel implementations*[3] are done making use of persistent memory, which reduces the cost when transferring weights and updates from memory to GPU to perform hardware accelerated computations. Moreover, the number of layers is also directly reduced by changing the architecture of WaveNet layers from CNNs to RNNs (WaveRNN), which can deliver highly non-linear transformations with a single layer.

These methods also allow an increase of the resolution of WaveNet from 8 to 16 bits and their sampling frequency from 16 to 24 kHz that compensates the quality degradation attributed to the model simplification. If the same methodology of outputting a categorical distribution of 256 classes in the case of using 8-bit quantization was applied to the 16 bit case, more than 65000 classes would be needed. To handle this increase in resolution, a dual softmax layer is used, which means that the output is split into Most Significant Byte (MSB) and Less Significant Byte (LSB), thus requiring only 256 classes for each byte, i.e. a total of 512 classes. This architecture also allow weight and computation sharing across both parts of the output giving more importance to MSB, which is predicted before LSB and conditions it.

The number of generated samples is also reduced by pipelining in generation. This approach is named as Subscale WaveRNN and consists in expressing the output tensor $\boldsymbol{u}$ into B sub-tensors of sample rate B times smaller. These are arranged as a 2-dimensional signal with a certain delay between them. The samples are then generated in parallel across the B sub-tensors without following the natural order of the original tensor $\boldsymbol{u}$ and filling the gaps determined by the delay hyper-parameter, which determines how much future is seen. This behavior is depicted in figure 2.6.

[3] The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements for the execution of compute kernels, which are functions executed by many GPU threads in parallel.

3 Methodology

With the purpose of building a model for audio generation, SampleRNN is adapted to output coherent utterances by conditioning the input under some speech parameters obtained with Ahocoder (Erro et al., 2014). The baseline model was also conditioned on features extracted with Ahocoder, but the model was only trained with one speaker. Therefore, the model is first adapted to hold many speakers by adding an extra input to define its identity and changing the database. This implementation is detailed in section 3.1.

This architecture was aimed to perform the tasks of both speech synthesis and voice conversion. Nevertheless, results found with the first implementation showed the intrinsic relationship between the speech characteristics and its generator. This is why results obtained with the first model could be used for a low-rate vocoder but not for voice conversion due to the futility of the speaker identity. This means that the parameters that condition the model, could be transmitted and used in reception to synthesize the desired speech signal. The advantage of using a similar system to the one presented here, is that different rates could be achieved when generating, thus achieving what is known as bandwidth extension.

New deep learning architectures exposed in section 3.2, got other features by combining the existing ones extracted with Ahocoder to get an independent representation named as latent. These latents, will ideally not have speaker information but the linguistic content of the original features will be conserved after the transformation. This allow to generate speech that synthesize the same text with the same intonation and phoneme length but with a speaker identity that could be swapped as desired. This is known as voice conversion. Last section of this chapter presents a novel kind of speaker interpolation titled as *eigen-voice* representation that aims to imitate a new and unknown speaker given a short recording of it.

3.1 Multi-speaker Neural Vocoder

The baseline model is not the one implemented by Mehri et al. (2017), authors of the original paper. Instead of this implementation, which was done using Theano, a Python library for fast numerical computation that can be run on the CPU or GPU, an implementation in the deep learning library Pytorch was preferred. This code created by Kozakowski and Michalak (2017) allows training the model with an arbitrary number of tiers, whereas the original implementation allows maximum 3 tiers, and use GRUs as recurrent layers. Nevertheless, the optimal distribution of the model was found to be of $K = 3$ tiers as suggested in SampleRNN.

The idea of substituting the supposedly real-valued outputs with a Multinoulli distribution is preserved because results obtained in the original paper shows that it outperforms the regression approach, which is told to be almost indistinguishable from random noise. The loss function used to compare the targets with outputs is the Negative Log-Likelihood (NLL), also known as multi-class cross-entropy, which is typical for $M-$class classification problems. Concretely, the used variant is called size average, which is based in averaging the losses for each mini-batch of batch size N, and its expression is shown in equation (3.1). Note that in this configuration, the output of the network is not only computed by mini-batches but also by sequences. This means that in the case of SampleRNN and its variants, $N = batch\_size \times sequence\_length$.

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} y_n \log(Softmax(f_\theta(\boldsymbol{x}_n))) = \frac{1}{N}\sum_{n=1}^{N} y_n l_n \tag{3.1}$$

NLL evaluates the likelihood between a target y_c, which is one-hot encoded, and the log-probabilities l_n obtained at the output of the model $f_\theta(\boldsymbol{x}_n)$ with a Log-Softmax layer, which is basically applying a softmax layer to get the probability distribution followed by a logarithm operation. From now on, when referring to the output of the network, the softmax layer will be considered within the model and not an external layer but before. Equation (3.1) is an exception to emphasize the importance of applying a softmax function (equation (A.3)) at the outputs of the network. Its use is mandatory before computing the NLL because it assures that the probabilities inside the logarithm never reach an absolute zero. Unlike in frame-level modules, the input of previously predicted samples of the sample-level module is quantified after a $\mu-$law companding transformation as recommended in ITU-T. Recommendation G. 711 (1988) because reconstructed speech signals achieve less quality degrading when non-linear instead of uniform quantization is applied. The audio is quantified with $b = 8$ bits, which means that there are 256 classes at the output of the $q-$way Softmax of SampleRNN.

The optimizer in charge of training SampleRNN using TBPTT is the so-called Adaptive Moment Estimation (ADAM) (Kingma and Ba, 2014). ADAM is a stochastic gradient descent algorithm with adaptive learning rate that not only changes the weights following the direction based on the gradient of the loss function with respect to the parameters, but also based on the values of the previous updates. This implies a certain inertia or moment on the update of parameters, which is determined by the betas that apply on both the updates of biased first and second moment estimations. Regarding the learning rate of this optimizer, this value diminish proportionally to the variation of gradients between batches.

The gradients computed are clipped before the TBPTT in order to avoid the exploding gradient problem (refer to section A.3 for more details). The trade-off between batch size and learning rate was tested based on some works like L. Smith et al. (2018). In Masters and Luschi (2018) it is confirmed that the use of small batch sizes achieve the best training stability and generalization performance for a given computational cost in a wide range of experiments. Based on this works, the batch size used in the model is 128.

3.1.1 Data structure

As mentioned in the explanation of SampleRNN (section 2.2.2), frame-level modules are deep RNNs that operate on non-overlapping frames of different frame sizes of size FS. In the current implementation, $FS^{(2)} = 20$ and $FS^{(3)} = 4$. The number of input samples on each level is calculated as the cumulative product of the frame sizes, which means that for level 2 and 3 there are respectively 20 and 80 samples at the entry of each tier. The sample-level module, i.e. the tier $k = 1$, have a frame size of $FS^{(1)} = 20$, which is the same as in the upper module $k = 2$. The reason why the parameters exposed above take this values lies in the fact that, when conditioning the input samples with the acoustic characteristics, it is mandatory that the window size of the input is multiple of the one used in the extraction of characteristics. Ahocoder computes the speech parameters every 5ms, which is equivalent to 80 samples as the sample rate of the audio is 16kHz. The sequence length, which determines how many samples are included in each TBPTT pass is set to 1040, which means that there will be 13 sequences of 80 samples for every pass on the tier 3 and 52 sequences of 20 samples in the tier 2. Regarding the number of hidden units used for all GRUs, it was set to 1024 as best results were found with this value in the original paper (Mehri et al., 2017).

These numbers as well as the batch size, which has a size of 128 as explained above, are shown in red in figure C.1 to ease the reading. In the same picture, the numbers of a single sequence are specified, but it has to be considered that in a single pass not only one sequence is processed. Note that speaker have also 13 values for every sample in a batch because audios have different lengths and when rearranging data to fit the desired shape, it happens that different speakers are in the same sample of a batch. In this case, the most repeated speaker is assumed to be in the whole sample because the ending and starting samples of an audio, where the change of speaker identity happens, is composed of silence. As well as with the conditioners, speaker is checked every 5 ms, which make the assumption of silence coherent and let the model be simpler without compromising output quality. The speaker ID is then passed through an embedding layer with 6 dimensions and 6 different values. This approach was chosen instead of a one-hot encoding vector, which would offer similar results, to ease the incorporation of a speaker recognition system for purposes like the one explained in section 3.3. Nevertheless, both options are valid and others like deciding the speaker with a picture of him and a CNN to recognize it could also be implemented.

Every audio of the speech dataset is fed to Ahocoder (Erro et al., 2014), which uses a sliding window of 5 ms and extracts some features from it. These parameters are aimed to represent the nature of the windowed segment of speech signal in such a manner that it could be decoded with them. For every window, which translates to 80 samples as the audio files are sampled at 16 kHz, Ahocoder computes the following features:

- MFCCs of order $p = 40$. These represent the short-term power spectrum and are the result of taking the inverse Fourier transform of the logarithm of the estimated spectrum in the Mel frequency scale, which considers the critical bands of the human hearing system.
- The logarithm of the fundamental frequency ($lf0$) also named as pitch contour.
- The maximum voiced frequency (mvf), which is used in various speech models as the spectral boundary separating periodic and aperiodic components during the production of voiced sounds. Recent studies (Drugman and Stylianou, 2014) have shown that its proper estimation and modeling enhance the quality of statistical parametric speech synthesizers.

The pitch contour signal behaves very different for voiced and unvoiced signals. In the first case, it is a continuous signal and in the second case, it does not exist and Ahocoder represents it as $-\infty$. To tackle this non-linearity, the output of Ahocoder is post-processed and the pitch contour is interpolated to be log-linear in the unvoiced segments. With the aim of allowing the model to learn the real distribution of the pitch, a flag indicating whether the 5 ms sequence is voiced or unvoiced (uv) is added to the conditioner vector, which looks as illustrated in figure 3.1. This vector of conditioners is denoted as $cond_i = [cc_1, \ldots, cc_{40}, lf0, mvf, uv]$ in figure C.1, where the conditioner dimension is substituted by its value: 43.

The last layer of the frame level tiers is a transposed 1D convolution. It is also known as a fractionally-strided convolution or a deconvolution even if it is not an actual deconvolution operation. It is used to learn an optimal up-sampling and therefore not have to choose between predefined interpolation methods like the bi-cubic or the bi-linear one. To sum up, the transposed convolution operation forms the same connectivity as the normal convolution but in the

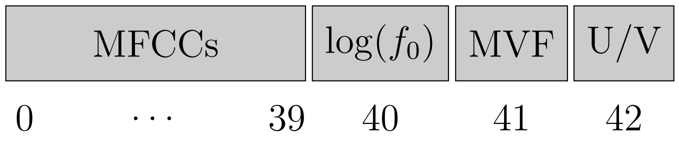


Figure 3.1: Structure of the conditioner vector

backward direction. In the presented model an external parameter of bias was also introduced and added to the output of the transposed convolution which is set to not have an own bias.

The output of the frame level layers is used by the lower tier in a number of sub-sequences determined by the up-sampling ratio, which takes the value of 4 as in the SampleRNN implementation. This is the so-called upper conditioning and would be used by 4 different modules, which are illustrated as a concatenation in the dimensions shown in red.

The first tier, also called as sample-level module, have an embedding layer that was introduced by the authors of SampleRNN and showed to improve the results of the model. The input sequence have samples quantized with 8 bits, which means that it can take 256 different values. The dimensionality is preserved but the activations is not one-hot encoded but is a vector with learned values. A fully-connected layer, implemented with an equivalent 1-D convolution with kernel size 1, follows the embedding layer to increase the dimensionality to 1024 (number of hidden units used for all GRUs) and then add its output to the conditioner of the upper tier which now match the dimension. After that, two more fully-connected layers followed by a softmax layer with a logarithm applied at the end of it, is added to output the desired Multinoulli distribution of the sample at a given timestep conditioned on the previous ones. The architecture of the sample-level module is depicted in figure C.2.

It is important to highlight the structure of the audio samples in each of the modules of the model. The data is arranged in a matrix of two dimensions to have as much columns as the batch size. This matrix is then stored in memory to avoid loading the data from disk every time that `Dataloder`, the Pytorch function in charge of asking for data to use it as inputs of the network, asks for it. For every sample in the batch, 1040 samples being this the sequence length are taken to perform TBPTT. Data is overlapped 80 samples, being this the samples in a frame of the tier 3. The overlap in the first batch will be a vector of zeros, which will be preceded by 1039 samples as depicted in image 3.2. Note that the sequence length is 1040 but the last sample is the one to be predicted. A reset flag is added to initialize the hidden states when there is no data of the same files that would be useful to predict the following samples. This will be active only in the first batch where the overlap are zeros.

This data can be denoted as stateful, which means that the first vector of the second batch will be the continuation of the last vector in the first batch and allows to exploit the memory capabilities of the RNNs. In case that the total number of samples per file is not multiple of 80, which would mean that the last vector of acoustic characteristics was not computed with 80 samples and therefore is not aligned with data, the last $cond_i$ vector as well as the last window of data will be discarded if there are less than 60 remaining samples. Otherwise, these will be padded with zeros to fulfill the 80 samples and align with the conditioners.

Note that the number of input samples of this module is now expressed as a flat vector with size 1059 for every index of the batch.

Once the data is arranged as in 3.2 inside the dataset function, the data-loader from Pytorch gets both sequences of inputs and targets. Each of the frame level modules then take the samples

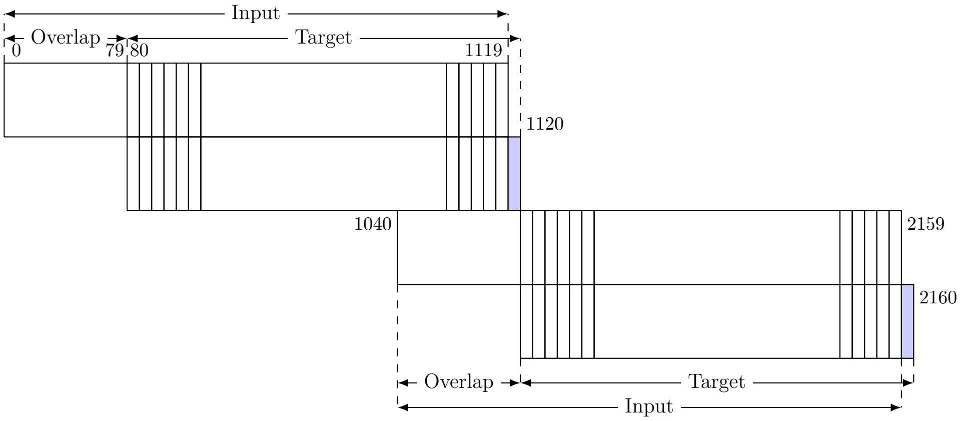


Figure 3.2: Structure of the audio data

from index $fsamples[3] - fsamples[i]$, to $1119 - fsamples[i]$, where $fsamples[3] = 80$ and $fsamples[2] = 20$. This means that only 1040 samples are taken of the 1120 (from index 0 to 1119 for the first batch) but starting on different points for every frame level module. Regarding the first tier, the samples from index $fsamples[3] - fsamples[1]$ with $fsamples[1] = 20$ to the end are taken, resulting in the number exposed above: $1059 = 1119 - (80 - 20)$.

3.1.2 Normalization

The speech parameters extracted with Ahocoder are normalized before feeding them to the model with the aim of controlling the magnitude of the gradients in training. The chosen normalization function was a simple feature scaling, which bounds each of the conditioners from 0 to 1 and is computed with equation (3.2). Note that each element of the conditioner vector is independently normalized.

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x} - \boldsymbol{x}_{\text{min}}}{\boldsymbol{x}_{\text{max}} - \boldsymbol{x}_{\text{min}}} \tag{3.2}$$

With the hypothesis of having speaker-dependent conditioners, a normalization for each speaker was proposed as a derivate experiment to isolate the speech features to the source. This supposition can be clearer by thinking about the pitch of each speaker. In human speech, this value varies from 50 to 500 Hz, and adult males use to have lower fundamental frequencies than females or children. The pitch varies to give intonation to a phrase but if normalized by speaker it have similar profiles for same level of expressiveness. Some plots illustrating the effect of this normalization are shown in chapter 4 for some of the other speech parameters whose behavior is not so evident as in the pitch case.

In both cases, maximum and minimum values for each of the parameters were found for the training partition of the speech dataset, so it could happen that some features of the other partitions overpass the bounds.

3.1.3 Look ahead

The main difference between a vocoder that is decoding a live conversation that is remotely encoded and the one that generates speech in a TTS system is that in the second case, the complete sequence of speech parameters that will be fed to the vocoder are known. This means that in contrast with a possible phone call were both ends are talking at real time, after synthesizing a speech signal, all parameters that will condition the model are already extracted either from raw text or from another speech signal like in Ahocoder. Note that by using deep learning and more concretely architectures derived from RNNs, the model has memory and thus to provide context only the current and following frames are needed because all the previous ones are remembered and stored as the hidden states. This contrast with other classical approaches were this kind of context would need a window centered on the sample of interest.

With this idea in mind, the causality that speech synthesizers inherited from the vocoders used in decoding is questioned and another experiment is derived from it. The proposal is then to not only use the 43 parameters extracted from the current 80 samples but also the ones computed with the following window. This means that in this experiment there will be 86 features conditioning the output, being these the ones corresponding to the current and future frames except from the last 5 ms segment of audio which will have the same features twice.

3.2 Voice Conversion

Voice conversion is a technique to modify a speech waveform which freely converts non-para-linguistic information while preserving linguistic information (Toda et al., 2016). This means that the changes in intonation, pauses and the spoken text is exactly the same but the speaker changes. To develop this technique, a deep understanding of how to effectively factorize speech acoustics into its individual components such as linguistic, non-linguistic, and para-linguistic information using various technologies, such as machine learning, is needed. Moreover, voice conversion has great potential to develop various applications not only for flexible control of speaker identity of synthetic speech in TTS systems but also as a speaking aid for vocally handicapped people such as dysarthric and laryngectomees patients, as a voice changer to flexibly generate various types of emotional and expressive speech, for vocal effects to produce more varieties of singing voices, for enhanced mobile speech communication using wide-band speech, accent conversion for computer assisted language learning, and so on. Therefore, it is worthwhile to study this technique for both scientific purposes and industrial applications.

Although a simple conversion function, such as a global linear transformation or frequency warping with a constant warping rate for modifying the spectral envelope, is capable of changing speaker identity, it is insufficient to convert a specific source speaker's voice into another specific target speaker's voice. A more sophisticated conversion function to effectively model a nonlinear mapping between source and target voices needs to be developed to convert speaker identity. This is why some approaches using deep learning have been derived from this challenge by assuming that the linear transformation explained in section 3.1.2 would not be enough, but a neural network could learn a non-linear function capable of freeing the linguistic parameters of the speaker information.

The objective is therefore to design a network capable of learning this complex non-linear function that could take the acoustic features extracted with Ahocoder and output a transformation of them without speaker information. The identity of the speaker will be fed as an extra input as in the previous case, but this time the network will not be able to guess the speaker solely based on the acoustic features. This means that the model will be forced to give importance to the speaker identity that previously was meaningless to achieve good results in synthesis. Once this differentiation is done and the network is fully trained by feeding the identity of the speakers that and its correspondent features, this identity could be swapped in generation to obtain the desired voice.

3.2.1 Bottle-Neck approach

This approach to tackle voice conversion is inspired by VQ-VAE (section 2.2.3) and can be seen as the first stage of an auto-encoder. The purpose is to compress the speech features into a lower dimension called latent representation or code as in figure 3.3.

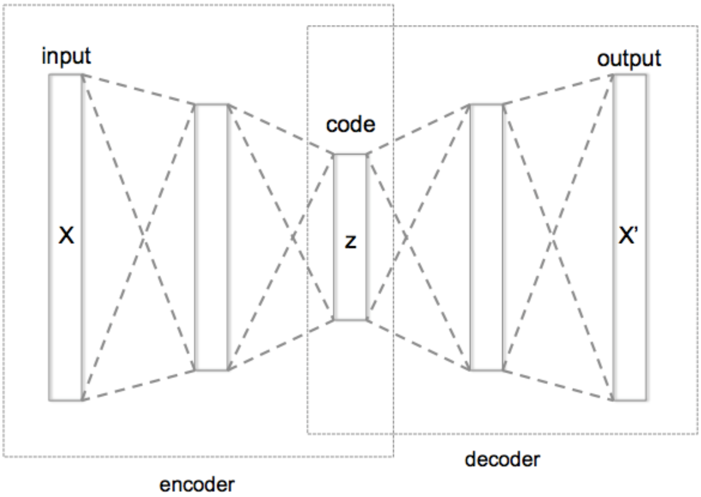


Figure 3.3: Schematic of an auto-encoder with 3 hidden layers from Wikipedia

By forcing a bottle-neck, the conditioners will ideally only preserve the linguistic information as the speaker identity is independently fit as in the SampleRNN model. Therefore, the unique difference with the baseline adapted model depicted in figure C.1 is that the output of Ahocoder will not only be applied one linear layer that expands it to the proper dimension that fit the number of hidden units og the GRUs. Different experiments have been run with a variety of values for the independent conditioner dimension, the lower number of channels before the expansion to 1024. An example with a narrower dimension of 10 is depicted in figure 3.4.

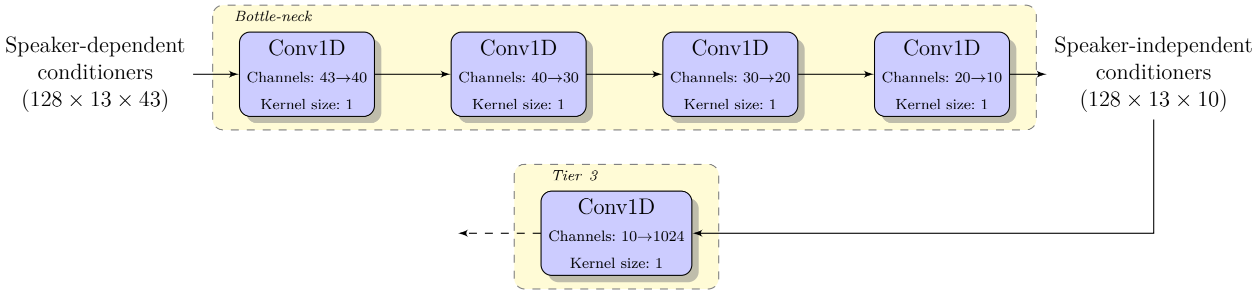


Figure 3.4: Example of a bottle-neck with independent dimension 10

3.2.2 Adversarial training

Another approach for voice conversion that was already introduced by the advisor of this thesis in the project proposal, is based in transforming the speech features to be independent of the speaker like in the bottle-neck approach but not necessarily with the same architecture. This new features are then used by SampleRNN and by a network that tries to guess the speaker from them. During the development of this thesis, a similar work was presented by Chou et al. (2018) and the architecture proposed for the discriminator was adapted to the needs of SampleRNN. The concept of adversarial training was in both cases inspired by Generative Adversarial Networks (GANs).

GANs were introduced by Goodfellow et al. (2014) and they are formed by a pair of simultaneously trained networks called generator and discriminator. On the one hand, the purpose of the discriminator is to assign a probability that tells if the inputs of that network are fake or not. On the other hand, the generator tries to fool the discriminator by synthesizing outputs from noise inputs that seem real. Once the loss of the discriminant is backpropagated, the gradients of the input are used to update the weights of the generative network and thus tell which parts of the generated object to change in order to yield a greater probability in being real as seen by the discriminator.

In the proposed formulation, the so-called discriminator is a classifier that tries to guess the speaker solely based on the latent representation of the conditioners which ideally are independent from the speaker once the network is trained. Note that the concept differs from the original GAN formulation as there is no fooling with fake data but with a transformation of it.

With the architecture of discriminant depicted in figure 3.5, Chou et al. (2018) reported a drop of verification accuracy from 0.916 to 0.451, which demonstrates that the desired independence between the linguistic information and the speaker is achieved. The conditionerCNN module was changed in a variety of experiments, where bottle-necks with different narrower dimensions and other approaches were the number of channels was not reduced were tried.

In order to train an adversarial model, two optimizers are needed to independently train both the discriminant to classify better and the rest of the network to, in this case, generate speech similar to the target with the premise of using the speaker-independent speech parameters. Both optimizers use ADAM and minimize the losses computed with the NLL function. One of the optimizers only applies back-propagates on the parameters of the discriminant and minimizes the classification loss $\mathcal{L}_2$. The other optimizer, applies on the rest of the model parameters, i.e. the ones belonging to SampleRNN and the conditionerCNN. The loss to minimize in this case is not only $\mathcal{L}_1$, which is determined by the likelihood of the output distribution and the target, but also on $\mathcal{L}_2$ with inverted sign to avoid the discriminant from guessing the speaker from the latents. The expression of the total loss minimized by the second optimizer follows equation (3.3), where λ is a hyper-parameter.

$$\mathcal{L} = \mathcal{L}_1 - \lambda \mathcal{L}_2 \tag{3.3}$$

The discriminant depicted in figure 3.5 includes some modules that have not yet been introduced. First of all and differing from all the previous diagrams of architectures, the activation functions are shown in the discriminant network because there are not Rectified Linear Units (ReLUs) between layers like in the other cases. They are Leaky ReLUs and, as it can be seen

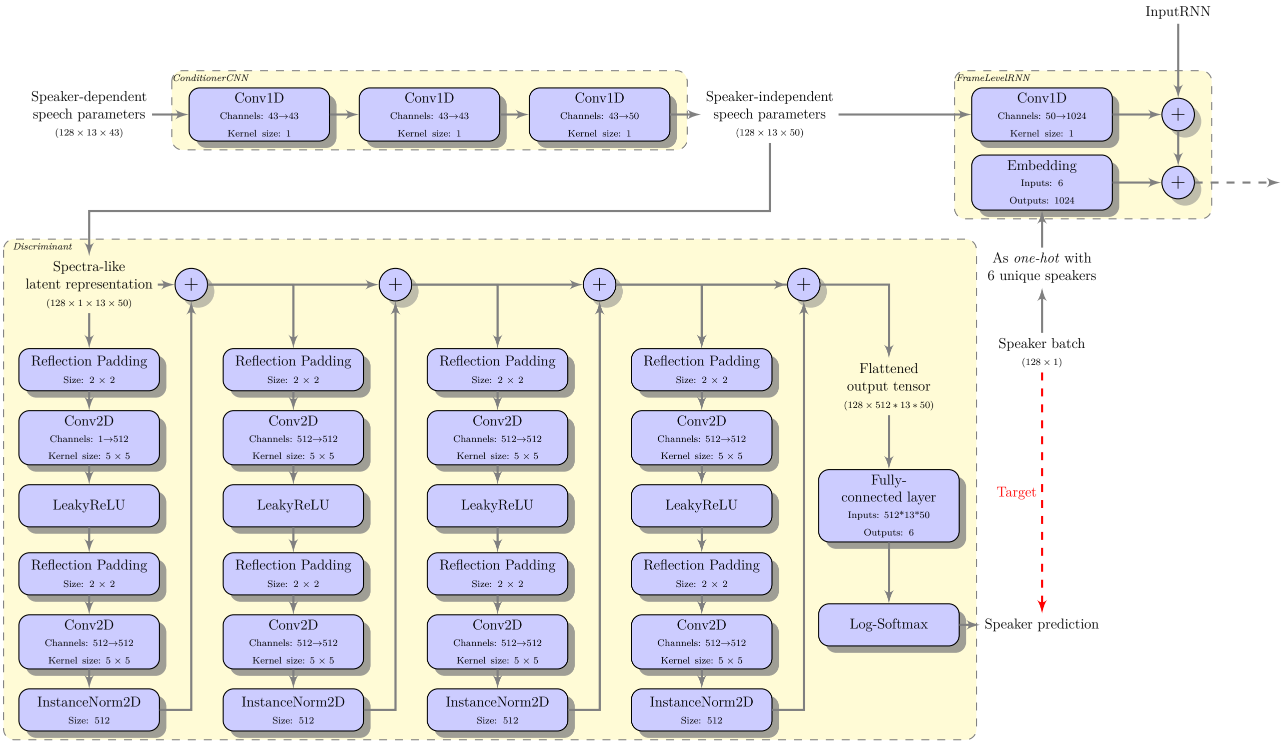

InputRNN
ConditionerCNN
Speaker-dependent speech parameters (128 × 13 × 43)
Conv1D Channels: 43→43 Kernel size: 1
Conv1D Channels: 43→43 Kernel size: 1
Conv1D Channels: 43→50 Kernel size: 1
Speaker-independent speech parameters (128 × 13 × 50)
FrameLevelRNN
Conv1D Channels: 50→1024 Kernel size: 1
Embedding Inputs: 6 Outputs: 1024
As one-hot with 6 unique speakers
Speaker batch (128 × 1)
Target
Discriminant
Spectra-like latent representation (128 × 1 × 13 × 50)
Reflection Padding Size: 2 × 2
Conv2D Channels: 1→512 Kernel size: 5 × 5
Conv2D Channels: 512→512 Kernel size: 5 × 5
LeakyReLU
InstanceNorm2D Size: 512
Flattened output tensor (128 × 512 ∗ 13 ∗ 50)
Fully-connected layer Inputs: 512*13*50 Outputs: 6
Log-Softmax
Speaker prediction

Figure 3.5: Diagram of the adversarial training architecture

in figure A.1, it is almost the same as the ReLU but a identity function with a slope of 0.01 for negative outputs instead of a constant 0. Also note that in this case convolutions are 2-dimensional. This was done to explode the temporal redundancy of speech conditioners. In order to do so, the latents are rearranged into a spectra-like representation with an extra dimension of 1 channel like if it was a gray-scale image. Before the convolutional layer, there is a reflection padding of size 2 × 2 that enlarges the input. A reflection padding example of the same size applied to a 3 × 3 matrix is shown below.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \longrightarrow 2 \times 2 \text{ Reflection Padding} \longrightarrow \begin{bmatrix} 9 & 8 & 7 & 8 & 9 & 8 & 7 \\ 6 & 5 & 4 & 5 & 6 & 5 & 4 \\ 3 & 2 & \mathbf{1} & \mathbf{2} & \mathbf{3} & 2 & 1 \\ 6 & 5 & \mathbf{4} & \mathbf{5} & \mathbf{6} & 5 & 4 \\ 9 & 8 & \mathbf{7} & \mathbf{8} & \mathbf{9} & 8 & 7 \\ 6 & 5 & 4 & 5 & 6 & 5 & 4 \\ 3 & 2 & 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

The padding is needed because in this case the kernel size is 5×5 and the output dimensions of the spectra-like representation would decrease in 5 units for every convolutional layer. In general, the output size after a convolution of kernel size K and padding P both symmetric in width and height dimensions, is given by the following expression, where the input has a structure of $(batch\_size \times channels \times W_1 \times H_1)$. In the case of wanting the same dimensions in both input and output of a convolutional layer with kernel size $K = 5$, a padding of size $P = 2$ is needed.

$$W_2 = W_1 - K + 2P + 1 \qquad ; \qquad H_2 = H_1 - K + 2P + 1$$

Within the discriminant, there is a pattern of stacking 2 convolutional layers with the respective paddings and activation functions that is repeated four times. Each output of these pseudo-modules is added to the result of the previous one except for the first case which is directly summed to the input. This is know as residual connections and was recently introduced by Wu et al. (2015). With network depth increasing, accuracy gets saturated and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error. Residual learning eases the training of DNNs and enables them to be substantially deeper without saturating the accuracy by copying the layers from the learned shallower model and then adding the input with an identity mapping. Moreover, the use of residual connections allows to perform iterative refinement of features (Jastrzębski et al., 2018).

Just before summing up the inputs and outputs of each residual connection module, an instance normalization is applied. This concept was introduced by Ulyanov et al. (2016) and was originally oriented to train high-performance architectures for real-time image generation. The normalization function is shown in equation (3.4).

$$\boldsymbol{Y} = \frac{\boldsymbol{X} - E[\boldsymbol{X}]}{\sqrt{Var[\boldsymbol{X}] + \varepsilon}} \tag{3.4}$$

For this project, $\varepsilon = 1 \cdot 10^{-5}$ as it is the default value of the instance normalization function implemented within the Pytorch library. Mean and standard deviation are calculated per-dimension separately for each object in a mini-batch and have a momentum of $\eta = 0.1$ that defines an update rule which follows the expression shown below, where $\hat{x}$ is the estimated statistic and x_t is the new observed value.

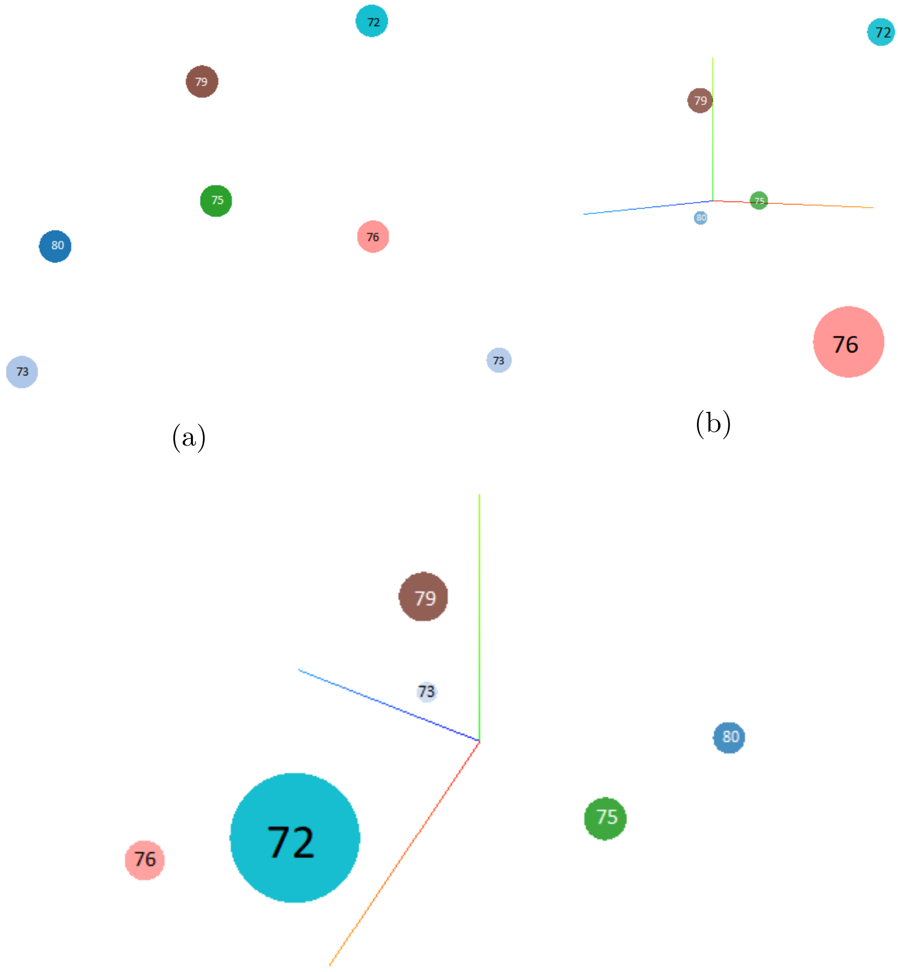
$$\hat{\boldsymbol{x}}_{new} = (1 - \eta) \cdot \hat{\boldsymbol{x}} + \eta \cdot \boldsymbol{x}_t$$

3.3 *Eigen-voice* representation

Deep learning achieves state-of-the-art results in speech synthesis and is able to generate different voices from the same model. Nevertheless, one of the only inconveniences of these methods is the huge amount of data that the models demand to learn the structure of an input signal for each of the speakers. For this work, a minimum of 14 minutes of speech signal is used to learn the representation of a single speaker for a multi-speaker model (see table 4.1). By achieving voice conversion, this model could read an input text with as many voices as the model have been trained with, but would not be able to speak with new voices.

In works like Pascual de la Puente (2016), experiments regarding speaker interpolation were tried with the aim of synthesizing new speakers. With a model which feeds the speaker identity as a one-hot vector, this would be as easy as putting various values into the input vector that would act as probabilities of belonging to each of the trained speakers. Inspired by a similar solution presented to a problem for facial recognition by Turk and Pentland (1991), the idea of this section is to create a basis of speakers that could ideally represent any new voice.

With the speaker embeddings computed on the tier 3 of SampleRNN and using the discriminant network depicted in figure 3.5 to predict the values of this embedding, the predicted speaker vector could be feed directly to the 1D-convolution as if it was the speaker embedding. A dimension of 6 could not be enough to represent the characteristics of many speakers, but some experiments to find the appropriate one are derived from this proposal. When the proper number of dimensions of the embedding is achieved, with a large enough database every speaker identity could be represented. The mapping of the speakers of the training database should be orthogonal vectors forming the basis or at least as much independent embeddings as the dimension of it. This idea could theoretically replicate any voice with a very short duration audio of the target speaker if the conditions exposed above are satisfied. For the current implementation, embeddings have a rank of 6, meaning that they are all linearly independent. This indicates that a higher dimension of embedding would be needed as some voices of teen speakers are not well modeled as it is exposed in section 4.3. The projections of the embedding obtained with the SampleRNN model can be seen in figure 3.6.



(a)

(b)

(c)

Figure 3.6: 2D (a) and 3D (b) and (c) embedding projections obtained with Principal Component Analysis dimensionality reduction technique visualized with Tensorflow Projector

4 Results

The results obtained for each of the experiments presented in chapter 3 as well as the value of the parameters that were fine-tuned in each of them are exposed below. Moreover, the database used in the development of this thesis will also be exposed. The results are presented under two big umbrellas of objective and subjective metrics. On the one hand, the comparison is done by analyzing the evolution of the loss function during the training and the mathematical similarity between target and generated signals. On the other hand, subjective metrics were acquired by performing perceptual tests to human listeners that rated the quality of the generated audio.

4.1 Speech dataset

The speech dataset used to train the model was formed by six voices from the TC-STAR project (Bonafonte et al., 2006), where half of them are males and the other half are females. TC-STAR project, is financed by European Commission and envisaged as a long-term effort to advance research in all core technologies for Speech-to-Speech Translation (SST). SST technology is a combination of automatic speech recognition, spoken language translation and TTS). The objectives of the project are ambitious: making a breakthrough in SST that significantly reduces the gap between human and machine translation performance. The project targets a selection of unconstrained conversational speech domains—speeches and broadcast news—and three languages: European English, European Spanish, and Mandarin Chinese. Accurate translation of unrestricted speech is well beyond the capability of today's state-of-the-art research systems. Therefore, advances are needed to improve the state-of the-art technologies for speech recognition and speech translation.

In some works like in Pascual de la Puente (2016), it is recommended to balanced the data per user so all of them have approximately the same amount of samples to train. Nevertheless, all the available database was used to investigate if it is really necessary to compensate the input data or there is no observable preference and thus a larger database can be used. Moreover, the approach of not balanced data was also taken to avoid the few length restricted by speaker denoted with the identifier 76, which barely have a quarter of the others.

Table 4.1 shows a summary of the used database, which is, as the union of corpus 11 and 33. The first one contains audios were transcriptions of the European Parliament are read and thus are based in spoken language while the second one contains the reading of news and therefore is based in written language. Each speaker is denoted by a numeric identifier and its sex either male (M) or female (F) is specified. It is also specified if the main purpose of each speaker was focused on baseline voices for TTS (B) or for voice conversion (V). In this work, all voices have been used for both purposes.

Speech recordings were originally sampled at 96 kHz with a resolution of 24 bits per sample. Nevertheless, with the aim of conserving the audio format specifications of SampleRNN, the database was down-sampled to 16 kHz and non-linearly quantified with $\mu-$law using 8 bits. The whole speech dataset was split into three subsets of 80%, 10% and 10% for training, validation and test respectively. Each audio was pseudo-randomly taken with the premise of approximately having the same proportion of speakers in every partition to avoid unseen identities.

	72 (F;B)	73 (M;B)	75 (F;V)	76 (F;V)	79 (M;V)	80 (M;V)
Whole database	07:58:18	07:29:47	02:01:16	01:09:55	01:50:34	01:56:06
Corpus 11	01:03:56	01:00:07	00:53:47	00:02:23	00:45:26	00:54:55
Corpus 33	00:00:00	00:00:00	00:11:51	00:11:41	00:11:28	00:12:09
Used database	01:03:56	01:00:07	01:05:38	00:14:04	00:56:53	01:07:04

Table 4.1: Table with database length broken down for each speaker and corpus

4.2 Learning strategy

As explained in chapter 3, all models were trained with stochastic gradient descent using a mini-batch size of 128 and minimizing the NLL with ADAM optimizer. The decay rates, which determine the inertia on the update of the biased first and second moment estimations, take the default values recommended by Pytorch library of $\beta_1 = 0.9$ and $\beta_1 = 0.999$. Learning rate is set to an initial value of $lr_0 = 10^{-4}$, as it was found to offer the best results. With a value of 10^{-3}, seemingly overfitting was reached at early stages and the generated samples didn't satisfy the requirements. By decaying the learning rate to a value of 10^{-5}, training was better than in the last case as it didn't collapse, but the evolution of the loss was very irregular.

An external learning rate controller called scheduler was also used to force decrease its value after some milestones because it was find that the output curves made a sudden change in the first epochs. This is why the scheduler was set two milestones at epochs 15 and 35 with $\gamma = 0.1$, which means that the supposedly fixed learning rate evolves as shown in the expression below. Note that ADAM changes the learning rate and thus the scheduler reduces it by a factor γ in every milestone but it will not take the values shown in the example.

$$lr = \begin{cases} lr_0 = 10^{-4} & epoch < 15 \\ lr_0 \cdot \gamma = 10^{-5} & 15 \leq epoch < 35 \\ lr_0 \cdot \gamma^2 = 10^{-6} & epoch \geq 35 \end{cases}$$

Weight normalization (Salimans and Kingma, 2016), which consists in decoupling the length and the direction of each weights, was also used to speed-up the convergence. It was applied to the 1-D Convolutional layers, where each weight $\boldsymbol{w}$ was re-parametrized into an scalar g representing the gain and a vector $\boldsymbol{v}$ as shown in equation (4.1). When the weight is decoupled, the gradient descent is applied on the loss functions computed with respect to these two new parameters $\nabla_{\boldsymbol{v}}\mathcal{L}$, $\nabla_g\mathcal{L}$. Weight normalization speeds-up the training and it is computationally cheaper in front of other approaches such as batch normalization.

$$\boldsymbol{w} = \frac{g}{||\boldsymbol{v}||}\boldsymbol{v} \tag{4.1}$$

4.3 Objective evaluation

Although the samples generated with models trained with around a hundred epochs sounded natural and were very similar to the targets after the incorporation of the learning controllers exposed above, some of the samples saturated. Normally this saturation ended up corrupting the audio from the start of this unwanted high energy noise to the end. Nevertheless, some of the corrupted signals obtained with the look ahead approach (see section 3.1.3), showed a saturation of only around 9500 samples (0.6s at a sample rate of 16kHz) that ended up with a normal behavior. This temporal saturation is depicted in figure 4.1. The plots of the output distribution and their corresponding generated waveforms for two audios representing normal behavior and saturation are included in section D.1.

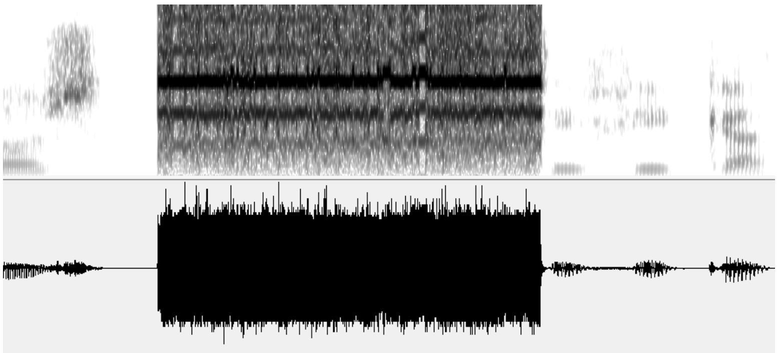


Figure 4.1: Waveform (below) and its spectrogram (above) of a generated signal that saturated on the middle samples

In order to tackle this instability, some research of this topic started but no similar works were found. The only idea was to change the approach referring the initial states of the RNN. This parameter derived in two experiments were its value was either learned during the training like in the default situation, i.e. like in all the other experiments, or a null tensor. Nevertheless, the saturation still prompted on some of the generated files and the only alternative was to select the non-corrupted audios for the perceptual test.

Regarding the experiment related to conditioners' normalization (section 3.1.2), it was first found that only the pitch feature, which inspired this approach to independently normalize with the aim of giving importance to the speaker identity, had the predicted behavior. Some plots of the feature evolution are included in section D.2, where the exact same sentence is uttered by the speakers from the voice conversion speech subset (see table 4.1).

As it can be seen in figure D.3, the independent normalization of the logarithmic pitch contour is very similar in all the audios of the different speakers, whilst the joint normalization would be enough to identify the speaker and thus no extra input parameter would be needed. For instance, note that in the joint normalization, it is very easy to distinguish between females (75, 76) and males (79, 80). This means that it would be impossible to change the speaker because the network doesn't need its identity for being this information included in the features. This is why this redundancy implies the futility of this input observed in the results.

Virtually all the other features resulted in very similar normalizations for both joint and independent approaches. The only exception was found with the MFCC of order 4, where the joint normalization acted as a scale factor for speakers 76, 79 and 80 due to a peak of speaker 75 at about sample 300. This is not an advantage for speaker isolation, but could be useful for a possible vocoder where the conditioners are quantized and the joint normalization could degrade the quality.

Besides the feature scaling, a z-score normalization, which imposes the normalized data to have zero mean and unitary standard deviation, was also thought to be implemented. Nevertheless, results obtained with the normalizations explained before showed that it wouldn't be enough to modify the conditioners to allow voice conversion. This was an impediment for the purposes of this thesis, but this behavior is useful from the perspective of a vocoder. Knowing the futility of the speaker identity input of the model, the features generated with Ahocoder could be used for speech coding. In order to demonstrate the potential of this proposal, some unknown voices were tested and promising results were found with them. This new speech database contained one male, on female and one kid. When generating, the speaker identity fed to the network in each case was the most similar one from the perspective of the pitch. Obtained results confirmed the hypothesis of the usefulness of the model as a vocoder, but also showed that a larger database would improve the quality. The major indicator of the need of extending the dataset was found when generating kid's voice, as it was not correctly synthesized due to the lack of similar data on the training dataset.

The bottle-neck approach achieved the desired Gaussian distribution, which is a characteristic of the latents of a VAE. An histogram obtained with TensorboardX is depicted in 4.2. This shows the evolution of the outputs between iterations 140000 and 20000, when the loss evolution was almost stationary.

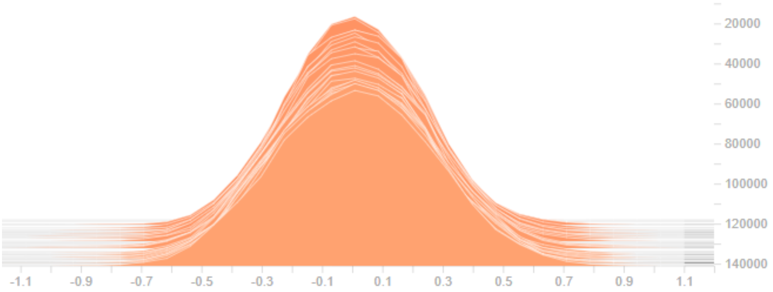


Figure 4.2: Histogram of features after bottle-neck

Although that the distribution of the outputs seemed to bring the desired results, the linguistic information was lost. This led to speech signals that were very similar to the unconditioned samples generated with the original SampleRNN implementation. For a dimension of the latents with a value of 10 and 30, a single voice belonging to speaker 72 could be identified. Nevertheless, with a value of 20, other speakers could be spotted in the results. The bottle-neck model was therefore discarded in a standalone version, meaning that only its use in the adversarial training framework showed the desired results. The loss of linguistic information due to the lack of features in the latent space was discarded because some known vocoder standards such as GSM or MELPe only use 8 to 10 Linear Predictor Coding (LPC) coefficients and yet obtain quality reconstructions. Nevertheless, the distance between the bottle-neck structure and the final samples generated with the whole model (see figure C.1), complicated the training of this new representation.

For the adversarial framework, batch size was reduced from 128 to 64 in order to avoid running out of GPU memory for the enlargement of the model respect to the other experiments. This is equivalent to increase the learning rate as exposed by L. Smith et al. (2018). The ConditionerCNN module (see figure 3.5) was composed of many layers that were aimed to get independent speakers thanks to the negative loss contribution of the classifier. The structure of this layers was in a first stage very similar to the bottle-neck, but other experiments were tried without narrowing the original dimension of 43 to avoid losing linguistic information as in the previous case.

The value for λ (see equation (3.3)) is taken from the implementation of Chou et al. (2018), where they suggest to linearly increase it from 0 to 0.01 in the first 50000 iterations to make sure the latent representation become speaker-independent gradually. This is needed because adding a the $\mathcal{L}_2$ term at the beginning of the training process, ends up causing problems to reconstruct the signal. Regarding the maximum value reached by λ, it provokes that the $\mathcal{L}_2$ term is smaller than $\mathcal{L}_1$ while in most cases, these must be comparable for the adversarial framework to work. Nevertheless, in this specific case, the small value is needed to maintain the main functionality which is generating quality speech.

In the experiment of the adversarial model where the ConditionerCNN module was the bottle-neck with latents' size 10, results obtained showed that the linguistic information was yet lost supposedly due to the similar argumentation given before. Nevertheless, the loss contribution of the classifier, achieved an unconditioned voice that sounded like another speaker, achieving thus relatively good results in one of the voice conversion fundamental metrics illustrated in figure 4.3. The objective is to also synthesize speech were the linguistic information is preserved and therefore the naturalness values is near 5 (Excellent), but achieving both is very difficult as it can be seen in the plot below, where some results for the Voice Conversion challenge are evaluated.

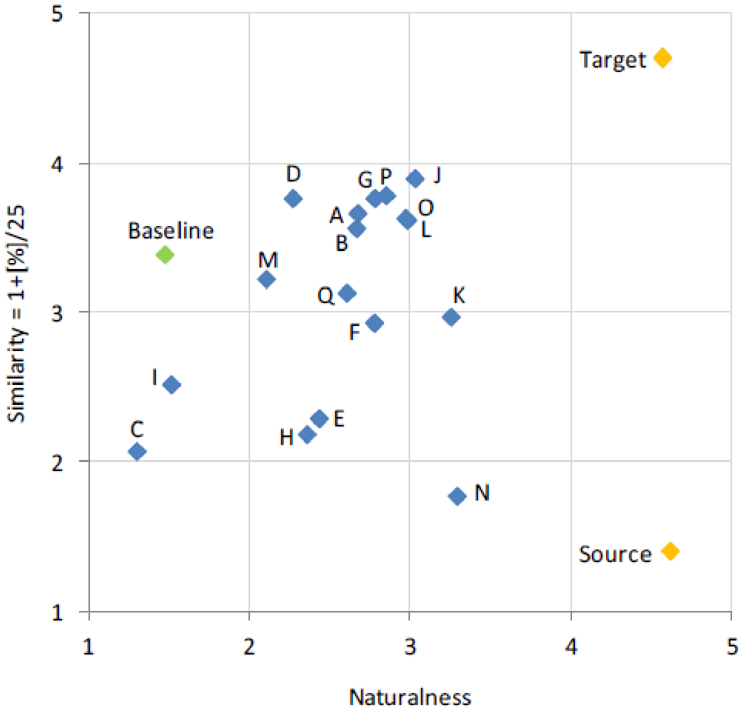


Figure 4.3: Overall naturalness MOS versus similarity to target speaker.Figure extracted fromToda et al. (2016)

Given the linguistic information loss of the bottle-neck, ConditionerCNN module was substituted by some linear layers that would allow to get an independent representation of the input features without the compression imposed in the previous case. This resulted into results were output was again conditioned and therefore naturalness was outperformed from the perspective of the previously described results. Nevertheless, the similarity to the target was lost like in all the previous experiments, thus showing a trade-off for the current adversarial architecture.

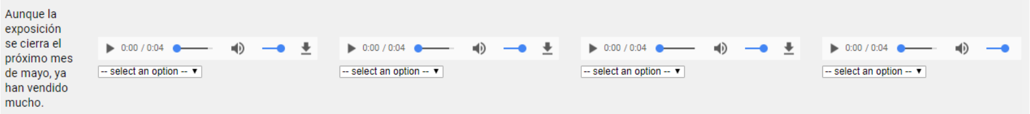
Overall, it was found that without this GAN inspired training, the speaker characteristics of the original speaker existing in the speech features inevitably degrades the performance of voice conversion.

4.4 Subjective Evaluation

The previous chapter compared the architectures and its results, but when analyzing a speech synthesizer, an unbiased subjective evaluation is also needed. A MOS test was therefore conducted to get a more in-depth comparison between four of the experiments regarding the variants of SampleRNN model exposed in section 3.1.2 and 3.1.3. MOS rates the naturalness in a scale of natural integers from 1 to 5, meaning these bounds bad and excellent quality respectively. The participants could listen the different recordings as many times as required to compare the systems to rate them. For each sentence, the transcription of the audio was provided to ease the listening, and audios of the different systems to compare synthesizing the same sentence, were disposed side by side as it can be seen in figure 4.4.

In the subjective evaluation there were two differentiated tests explained below:

- 8 audios from 4 males and 4 females generated with the variants of look ahead or not and joint or independent normalization are compared. All these models were developed during this thesis.
- 8 audios out of the same speaker are compared. This test is aimed to test the efficiency of the shared structure and therefore the best model of the previous ones from the perspective of the author, which is the model independently normalized with look ahead, is compared with Ahodecoder to have a reference of quality. Ahocoder/Ahodecoder is a high quality vocoder based on signal processing, where some coefficients necessary to estimate the original speech signal are estimated by Ahocoder. These coefficients are used in a signal model which decomposes the speech signal into and harmonic and an stochastic part. The low frequencies of the spectrum are modeled with the harmonic model, which is based in adding some sinusoids defined by an amplitude, frequency and phase extracted from the acoustic features. Regarding the high frequencies, these are modeled with the stochastic model, which is basically white noise filtered with the envelope of the signal obtained from the MFCCs provided by Ahocoder. The threshold dividing low and high frequencies is usually set a value, but in Erro et al. (2014) is a learned parameter. The authors of this work have above all centered their system in SPSS instead of in speech coding and thus Ahodecoder can be used as if it was a parametric TTS (see figure 1.1). The idea was to also compare the conditioned SampleRNN model developed by Dorca Saez (2018) to test the efficiency of the multi-speaker architecture. However, the final version of the model was not available and audio could not be synthesized before the deadline.



Aunque la exposición se cierra el próximo mes de mayo, ya han vendido mucho.

Figure 4.4: Screenshot of the first experiment of the test

The results exposed in tables 4.2 and 4.3 were obtained by averaging the evaluation of 25 volunteers for every system in each of the two tests. The samples included in the second test belonging to the best system developed in this thesis can be downloaded here. Some of the commentaries written by the volunteers who took the test, highlighted the difference in quality between males and females. This is why the following table does a separation between genders that shows that best results are indeed obtained with man voices. This was attributed to the unbalanced speech database, which contained one female speaker with only 25% of recordings compared to all other speakers (see table 4.1). Seemingly, this affected in the modeling of female voices, which had a louder background noise. This show the importance of a balanced database as stated in Pascual de la Puente (2016).

Normalization:	Independent	Joint	Independent	Joint
Look ahead:	No	No	Yes	Yes
Female:	3.3	3.3	3.8	3.6
Male:	3.6	3.6	4.0	4.8
Total:	3.5	3.5	3.9	3.8

Table 4.2: Table with subjective results comparing proposed methods

Note that the following table does not distinguish males and females. This was due to the planned comparison with the single speaker SampleRNN model, which was only trained with a male speaker that corresponds to the identifier 73. Some other comments also stated that male voices with lower pitch were worse modeled. In fact, the speaker 73 have the lower pitch of all the database and thus the results of this comparison could have been deteriorated by only considering this speaker.

Our system	Ahodecoder
3.8	4.2

Table 4.3: Table with subjective results comparing proposed and external state-of-the-art system

Overall, this chapter justified the learning strategy followed in this model and contrasted the results with both a subjective and an objective evaluation. While the users rate the SampleRNN system as natural with a high MOS score of 3.9 for the best proposed system, this evaluation decays when comparing to the Ahodecoder model which is based on signal processing and not on deep learning like other state-of-the-art vocoders. This high score is justified because the Ahocoder/Ahodecoder system was jointly optimized to achieve outstanding results with the own system and therefore not outputting more general features that could be used for other vocoders. The advantage of the proposed system is that it could be trained with any kind of parameters able to properly describe a speech signal. In fact, it has been verified that the features obtained with Ahocoder are not optimal for the system at least for voice conversion for its intrinsic dependence to the speaker. The planned strategy to find the parameters that fit the requirements of the system are described in section 6.1.

5 Budget

The cost of this project would only be attributed to researchers' salary as all of the used programs are open-source and the deployment resources have been provided by the TALP group of the Universitat Politècnica de Catalunya (UPC). Nevertheless, an estimation of the budget for this project if it was deployed without the help of an entity with that resources will be made using the cost of Amazon AWS EC2 servers [1]. There are a bunch of options but the cheapest yet valid GPU instance named `p2.xlarge` have been selected. Note that this price is expressed in United States dollars in the cost/hour column of table 5.1. However, the total amount referring to the server is expressed in euros by considering the upper-bound conversion 1$=0.85€.

Regarding to the salary that should be charged for the engineers, both a Junior and a Senior engineer were considered with the aim of simulating the author and the advisor of this work respectively. A total duration of 21 weeks will be considered as depicted in the Gantt diagram. The average working hours of the Junior engineer have been estimated to 40h/week and referring the Senior engineer, 1h/week attributed to the weekly meeting have been considered.

Concept	Amount	Cost/hour	Dedication	Weeks	Total
Junior engineer	1	15.00 €/h	40 h/week	21	12 600 €
Senior engineer	1	60.00 €/h	1 h/week	21	1 260 €
Server	1	0.99 $/h	140 h/week	21	2 499 €
				Total:	16 395 €

Table 5.1: Estimated budget of the project

[1] https://aws.amazon.com/es/ec2/purchasing-options/dedicated-instances/

6 Conclusions

In this bachelor's thesis, a vocoder acting as a second stage of a complete TTS system have been developed using deep learning techniques applied on a baseline implementation of the SampleRNN model. This vocoder conditions the output with speech features extracted with Ahocoder, but could be easily used with other modules that extract those parameters either from raw text or from other speech signals. The achieved model is capable of holding many speakers inside the same shared structure, so that every user shares its characteristics with the others, thus reducing the required number of parameters per speaker and letting them interact above all in the lower layers, where the linguistic information processing is performed. Although the unbalanced data, human listeners rate this system with state-of-the-art MOS scores and prefer the speech generated using the experiment with speaker independent normalization and look ahead approach. Both of that approaches were novelties introduced in this thesis and results show that they could be beneficial to other TTS systems. In the use of this model as a vocoder, promising results have been obtained although that it was known that for this purpose databases have to be larger and more variate to represent any speaker.

Regarding the speaker characteristics, it has been demonstrated that the original speaker existing in the speech features inevitably degrades the performance of voice conversion, thus requiring adversarial frameworks with a negative loss contribution. The architectures proposed to tackle the difficulties of voice conversion, were not conceived from the very beginning of this dissertation and thus where not presented on the proposal. Nevertheless, some difficulties arose referring when trying to change the identity of the generated speech. As this topic introduced in section 3.2 was really of interest, it was decided to broaden the thesis to include the necessary tools to achieve it. Inevitably, this reduced the amount of time scheduled for each of the work packages originally planned but did not narrow the original scope.

Voice conversion have also been treated and partially achieved, with the best results converting the voice as desired but not conserving the linguistic information lost with the bottle-neck structure. This was attributed to the depth of the model, and some strategies to tackle it are exposed in the following section and planned for the next months. Regarding the *eigen-voice* representation proposal, the methodology exposed in this thesis have been merely theoretical due to the difficulty of converting the voice of the generated speech. Nevertheless, it has great potential and could be applied to a bunch of different fields.

Overall, SampleRNN have been successfully adapted to conditional multi-speaker generation with results that achieve great quality and naturalness. With the aim of encouraging the development of similar works and given that the fact that the baseline code was open-sourced, it was decided to publish the source code developed during this work on the repository located on the author's GitHub page.

6.1 Future work

Differing from the vocoder exposed in chapter 3, a great amount of work when designing an end-to-end speech synthesizer is focused on parameterizing text (figure 1.1). An evolution of SPSS that uses DNNs for the representation of probability density functions of speech parameters given text was proposed by Zen et al. (2013). Therefore a next step would be to extract speech parameters directly from text instead of audio signals like in Ahocoder. The advantage of using a system based in DNNs is that the model could be jointly trained like in Tacotron (Wang et al., 2017), an end-to-end generative TTS model that synthesizes speech directly from characters and train the model from scratch with $< text, audio >$ pairs. With a similar framework,the optimal parameters mentioned in the closure of chapter 4 could be found. In fact, in the Neural Discrete Audio Representation paper (van den Oord et al., 2017b), the latents extracted with VQ-VAE that are then used to condition the Wavenet model to output the desired speech, are shown to be similar to the human-designed alphabet of phonemes. The features previously extracted with Ahocoder could be therefore substituted with this new representation, which probably is more efficient from the perspective of both voice conversion and audio synthesis as it achieves what authors call voice-style transfer (see 2.2.3).

The bachelor's thesis carried by Dorca Saez (2018) integrated MUSA, a parametric TTS system developed during the master's thesis of Pascual de la Puente (2016). Although the integration of both systems allowed to map text into speech parameters and then condition the SampleRNN model, it significantly degrades the quality. For this reason, the results presented in this dissertation were only extracted from more reliable parametric TTS modules that allowed to evaluate the presented model without introducing external errors.

When referring to the use of the SampleRNN model as a vocoder, after incorporating more speakers to the database to have a wider range of different voices, some experiment such as alternative representations for low-rate coding, bandwidth extension and speech enhancement could be derived. The need of a larger database prompts when trying to synthesize voices that are very different from the ones in the database. This can be first solved by balancing the data, but some voices more similar to the kid of the vocoder database would be probably needed. In Kleijn et al. (2017), a similar proposal based in low-rate speech with the Wavenet model. Regarding to the recurrent units, an option to change from GRUs to Quasi-Recurrent Neural Networks (QRNNs) could be implemented with the aim of fastening-up the training and inference. See section A.3 for more details about RNN architectures.

Bottle-neck could be better trained by designing a full VAE. This framework is aimed to obtain the same features in both the input and the output, with latents of lower dimension (see figure 3.3). Once this module is independently trained, the first stage of this VAE, i.e. the bottle-neck previously depicted in figure 3.4, could be incorporated to the model with frozen coefficients. This means that the latent representation achieved with the training of the VAE would be preserved and no problems regarding the difficulty of training with the whole model would arise.

The adversarial training framework inspired by the work of Chou et al. (2018) was only the first stage of a proposed two-stage system. The second stage was another pair of generator and discriminator which was trained to create a residual signal to patch the output obtained in the first stage. This was designed to improve the quality of speech as they reported to obtain blurry spectra and artifacts. The training of the second stage is done after the first stage is already trained and their parameters remain frozen. This was nevertheless not implemented because the problem was in losing linguistic information and not in noisy outputs. The need of using this two-stage system is not discarded for future results with the adversarial framework model including the latents of the independently trained bottle-neck.

Bibliography

Albelwi, S. and Mahmood, A. (2017). A Framework for Designing the Architectures of Deep Convolutional Neural Networks. *Entropy*, 242(19).

Black, A. W., Zen, H., and Tokuda, K. (2009). Statistical Parametric Speech Synthesis. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 1229–1232.

Bonafonte, A., Höge, H., Kiss, I., Moreno, A., Ziegenhain, U., Heuvel, H. V. D., Hain, H., Wang, X. S., and Garcia, M. N. (2006). TC-STAR : Specifications of Language Resources and Evaluation for Speech Synthesis. *Proceedings of the Language Resources and Evaluation Conference LREC06*, pages 311–314.

Bradbury, J., Merity, S., Xiong, C., and Socher, R. (2016). Quasi-Recurrent Neural Networks. *CoRR*, abs/1611.0:1–11.

Britz, D. (2016). Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*.

Chou, J.-c., Yeh, C.-c., Lee, H.-y., and Lee, L.-s. (2018). Multi-target Voice Conversion without Parallel Data by Adversarially Learning Disentangled Audio Representations. *CoRR*, pages 2–6.

Dorca Saez, G. (2018). Neural Audio Generation for Speech Synthesis. Bachelor's thesis, Universitat Politècnica de Catalunya.

Drugman, T. and Stylianou, Y. (2014). Maximum Voiced Frequency Estimation: Exploiting Amplitude and Phase Spectra. *IEEE Signal Processing Letters*, 21(10):1230–1234.

Erro, D., Sainz, I., Navas, E., and Hernaez, I. (2014). Harmonics plus Noise Model based Vocoder for Statistical Parametric Speech Synthesis. *IEEE Journal on Selected Topics in Signal Processing*, 8(2):184–194.

Fan, Y., Qian, Y., Soong, F. K., and He, L. (2015). Unsupervised speaker adaptation for DNN-based TTS synthesis. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 4475–4479.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. *Advances in Neural Information Processing Systems 27*, pages 2672–2680.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hunt, A. J. and Black, A. W. (1996). Unit selection in a concatenative speech synthesis system using a large speech database. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 1:373–376.

ITU-T. Recommendation G. 711 (1988). Pulse Code Modulation (PCM) of voice frequencies.

J. Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269.

Jastrzębski, S., Arpit, D., Ballas, N., Verma, V., Che, T., and Bengio, Y. (2018). Residual Connections Encourage Iterative Inference. In *ICLR*, pages 1–14.

Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., Stimberg, F., van den Oord, A., Dieleman, S., and Kavukcuoglu, K. (2018). Efficient Neural Audio Synthesis. *CoRR*, abs/1802.0.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kleijn, W. B., Lim, F. S. C., Luebs, A., Skoglund, J., Stimberg, F., Wang, Q., and Walters, T. C. (2017). Wavenet based low rate speech coding. *CoRR*, pages 3–7.

Kozakowski, P. and Michalak, B. (2017). SampleRNN-pytorch. https://github.com/deepsound-project/samplernn-pytorch.

L. Smith, S., Kindermans, P.-J., Ying, C., and V. Le, Q. (2018). Don't decay the learning rate, increase the batch size. *ICLR*, pages 1–11.

Lamb, A., Goyal, A., Zhang, Y., Zhang, S., Courville, A., and Bengio, Y. (2016). Professor Forcing: A New Algorithm for Training Recurrent Networks. *NIPS*.

Masters, D. and Luschi, C. (2018). Revisiting Small Batch Training for Deep Neural Networks. *CoRR*, pages 1–18.

McCulloch, W. S. and Pitts, W. (1943). A Logical Calculus of the Idea Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y. (2017). SampleRNN: An Unconditional End-to-End Neural Audio Generation Model. *ICLR*, pages 1–11.

Olah, C. (2015). Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Pascual de la Puente, S. (2016). Deep learning applied to speech synthesis. Master's thesis, Universitat Politècnica de Catalunya.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868.

Toda, T., Chen, L. H., Saito, D., Villavicencio, F., Wester, M., Wu, Z., and Yamagishi, J. (2016). The voice conversion challenge 2016. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 08-12-Sept:1632–1636.

Turk, M. and Pentland, A. (1991). Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2016). Instance Normalization: The Missing Ingredient for Fast Stylization. *CoRR*.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 1–15.

van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., van den Driessche, G., Lockhart, E., Cobo, L. C., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. (2017a). Parallel WaveNet: Fast High-Fidelity Speech Synthesis. *CoRR*.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2017b). Neural Discrete Representation Learning. In *NIPS*.

Wang, Y., Skerry-Ryan, R. J., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q. V., Agiomyrgiannakis, Y., Clark, R., and Saurous, R. A. (2017). Tacotron: A fully end-to-end text-to-speech synthesis model. *CoRR*, abs/1703.10135.

Wu, S., Zhong, S., and Liu, Y. (2015). Deep Residual Learning for Image Recognition. *CoRR*, pages 1–17.

Zen, H., Senior, A., and Schuster, M. (2013). Statistical parametric speech synthesis using deep neural networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 7962–7966.

A Neural Networks

Artifitial Neural Networks (ANNs) were conceived with the aim of mathematically model human intellectual abilities by biologically plausible engineering designs and thus bringing the concept of artificial intelligence. Nevertheless, even the so-called Deep Neural Networks (DNNs), which are ANNs with many layers, are not as complex as a real neural network as they barely have hundreds of synapses in front of 10^{11} neurons with 10^{4} synapses each one on a real brain.

A.1 Multi-Layer Perceptron

An ANN is a set of neurons, which are basic units of the network. The activation of those is simply a linear combination of an input vector $\boldsymbol{x} = \{x_0, x, 1, \ldots, x_N\}$, which is pondered by a set of weights denoted as $\boldsymbol{w}$ plus a bias term b (equation (A.1)).

$$a = \boldsymbol{w}^T \boldsymbol{x} + b \tag{A.1}$$

Some neurological studies have already demonstrated that a non-linear behavior is shown in each neuron given an input stimulus. In order to mimic that, the activation described of equation (A.1) is applied a non-linear function f. Moreover, this allows to obtain arbitrary shape decision regions that fit to almost any kind of problem with the proper depth and width. Note that this functions have to be continuous but at the same time differentiable in order to optimize the weights. These functions are typically linear for small values and saturate for large ones. Some common names for non-linear functions used in neural networks are hyperbolic tangent, Rectified Linear Unit (ReLU) and sigmoid.

$$y = f(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{A.2}$$

The hooking of many of these basic units creates the Multi-Layer Perceptron (MLP), which is the simplest type of ANNs as it only have feed-forward connections. Neurons can be stacked in parallel and also concatenated, i.e. the output of one neuron is the input of the following giving therefore width and depth to the network respectively. The framework is thus formed by many layers which don't necessary have the same number of neurons in each of them as depicted in figure A.2. The leftmost layer of the network is called input layer while the rightmost is the output layer. In between layers are named as hidden, because its values are not observed in the training set as they are not target values neither inputs. As the number of hidden layers increases, the network can start being denoted as a DNN.

Deep-learning methods are capable of learning multiple levels of representation by composing non-linear modules. Each one transforms the output of the previous level, or raw input in case of the first layer, into a more abstract level representation. Therefore, with the composition of enough of such transformations, very complex functions can be learned.

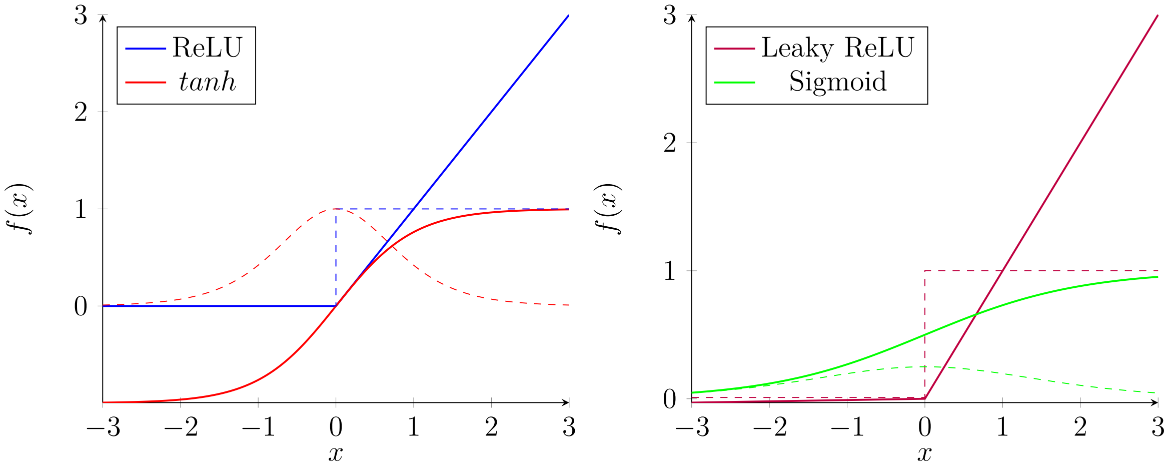


Figure A.1: Typical activation functions and their derivatives illustrated with dashed lines

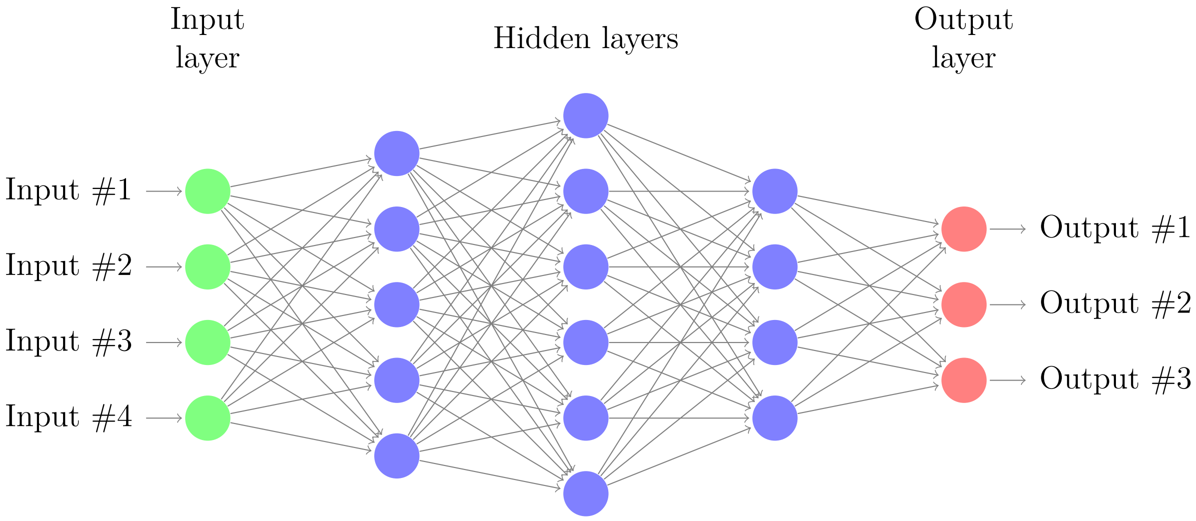


Figure A.2: Overview of a fully-connected MLP with three hidden layers

A.2 Optimization

There are essentially two types of supervised problems to tackle with DNNs; classification and regression. These deal with a discrete set of classes and a continuous domain of possible values respectively.

When classifying, the desired outputs, also called labels or classes, are in a form of a one-hot encoding vector, which have only one value set to 1, corresponding to the target class, and the 0 elsewhere, e.g. $[0, 0, 0, 1, 0]$. The loss, which depends on the parameters of the network, evaluates the difference between the targets and the outputs. Before comparing them, the final layer is often applied a softmax function (equation (A.3)) to express the probabilities of belonging to a class.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{A.3}$$

Regression aims to model and analyze several variables, when the focus is on the relationship between an output dependent variable and one or more independent variables which are treated as the inputs of the network. These outputs are a real value, such as an integer or floating point value, often representing quantities, like amounts and sizes.

For each of that problems, a loss function to minimize with respect to the parameters of the network is defined in order to train the model. In the regression case, loss functions depend on the distance from target to result, whereas in classification problems it only depends on either if class is well predicted or not, e.g. euclidean loss for regression and cross entropy for classification (equations (A.4a) and (A.4b) respectively).

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f_\theta(\boldsymbol{x}_i))^2 \tag{A.4a}$$

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \log(S(f_\theta(\boldsymbol{x}_i))) \tag{A.4b}$$

Virtually in all cases, functions to optimize are not convex and therefore have lots of minimums. Gradient Descent algorithms are used to move the parameters of the network in small steps which size is determined by the learning rate η in the opposite direction of the derivative of the loss. The update of the weights is shown in equation (A.5). Nevertheless, this algorithm does not necessarily achieves a global minima, meaning that suboptimal weights are typically used, but it is more important to find a minimum that generalizes well.

$$\boldsymbol{w} = \boldsymbol{w} - \eta\nabla_{\boldsymbol{w}}\mathcal{L} \tag{A.5}$$

In order to test if the model works well with unseen data, the whole dataset is usually chopped in three subsets for the purposes of training, validating and testing.

The first one is used to optimize the model by computing its outputs and then back-propagating the error to update the weights and biases. In each iteration, the gradient is computed with either all the training dataset or a fraction of it called batch. There are also two variants of the algorithm labeled as of Stochastic Gradient Descent, when only one example is used to update the weights, and as mini-batch Gradient Descent when only a few samples are considered. These are fast algorithms but have much more variance in the update than the Gradient Descent algorithm, which means that path to the minimum is not a straight curve as depicted in figure A.3. An epoch is defined as the pass of the entire training dataset to the network, which means that in the batch case it will be equivalent to the quotient between the dataset size and the batch size.

The validation set is used to fine-tune hyper-parameters of the DNN such as the number of layers or neurons. This data is therefore not explicitly used to train the model but it's not really unseen as the parameters are chosen to minimize the loss on it. This is the reason why a third partition of truly unseen data is needed to test the model.

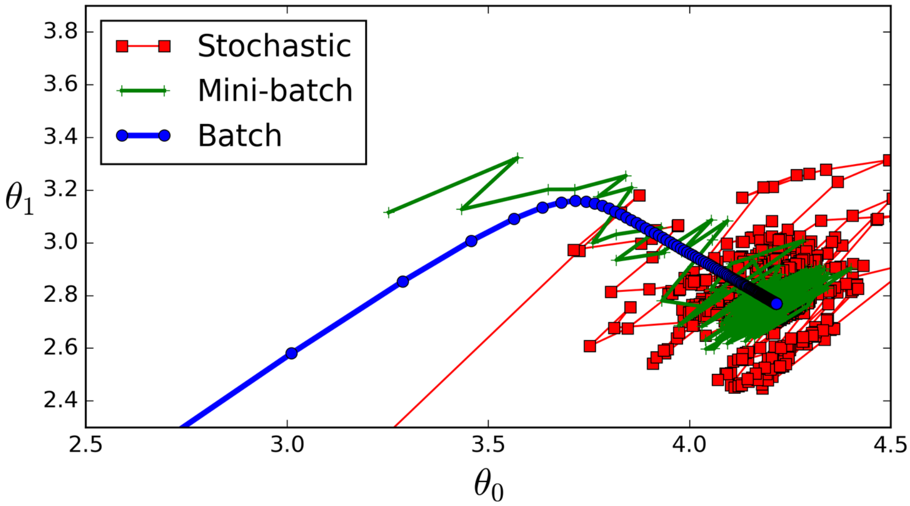


Figure A.3: Comparison between Gradient Descent algorithms from Hands-On Machine Learning with Scikit-Learn and TensorFlow chapter 4

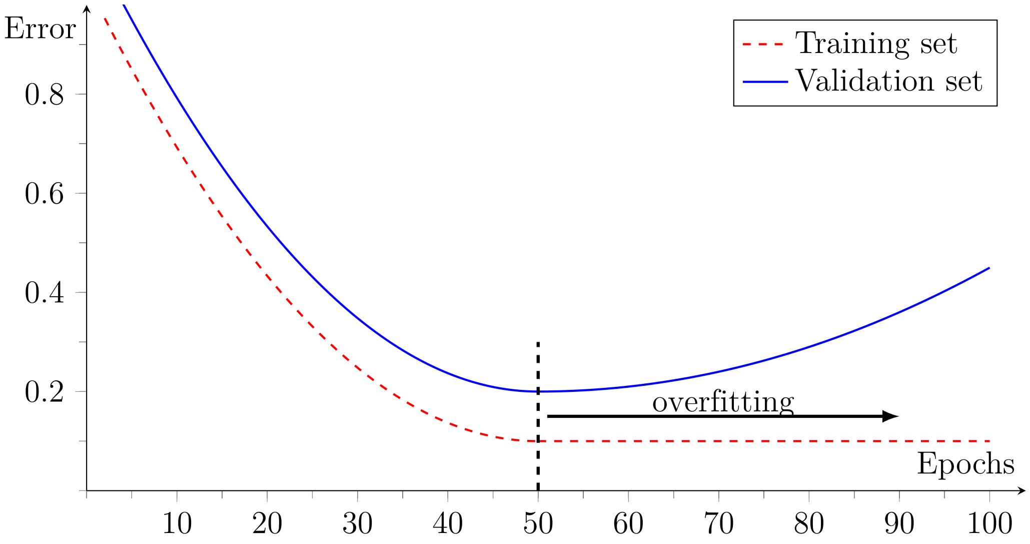


Figure A.4: Overview of a model with optimal weights reached at epoch 50

As it was stated before, the purpose when optimizing is to find a model that generalizes well and thus avoid overfitting. This phenomena is the production of an analysis that corresponds too closely or exactly to the training data, and may therefore fail to fit additional data or predict future observations reliably. It could be said that the model is too adapted or memorizes the training dataset, meaning that it doesn't generalize well. Overfitting can be detected when training loss keep diminishing while validation loss grow (figure A.4). The best model is therefore attributed to the parameters which give the best results in the non-trained elements, unless in those special cases that training dataset compromises all possible occurrences. This can be solved with early stopping, which make the training stop not only when the maximum number of iterations is reached, but also when the best model from the perspective of the validation loss has not been improved in a certain number of iterations. This is a simple yet effective solution to avoid overfitting but it requires more memory as a copy of the best and last parameters.

A.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) and their variants have leveraged completely the sequences processing and prediction problem, which makes them lead to interesting results in the speech synthesis field, where an acoustic signal of variable length has to be generated out of a set of textual entities. The idea behind RNNs is to make use of sequential information. Hidden layers have memory cells that condition future outputs. Theoretically, the retention of an RNN can be of arbitrarily long sequences, but in practice they are limited to looking back only a few steps. A typical RNN scheme is depicted in figure A.5, where the following notation is used:

- $\boldsymbol{x}_t$: Input at time step t.
- $\boldsymbol{s}_t$: Hidden state at time step t. It is calculated as the sum of the multiplication of the previous hidden state and the current input by a weight matrix respectively (equation (A.6)). For the computation of the first hidden state, $\boldsymbol{s}_{-1}$ is usually initialized to all zeros

$$s_t = f(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{W}\boldsymbol{s}_{t-1}) \tag{A.6}$$

- $\boldsymbol{o}_t$: Output at time step t. It is calculated solely based on the memory at time t: $\boldsymbol{s}_t$.

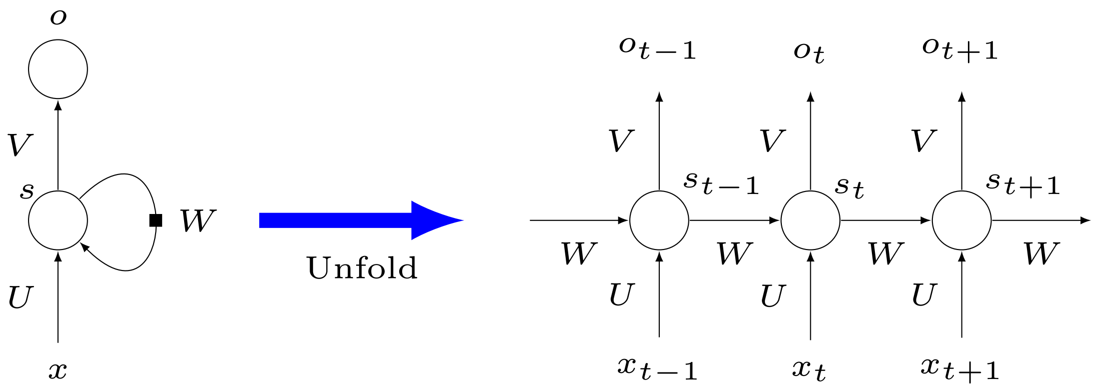
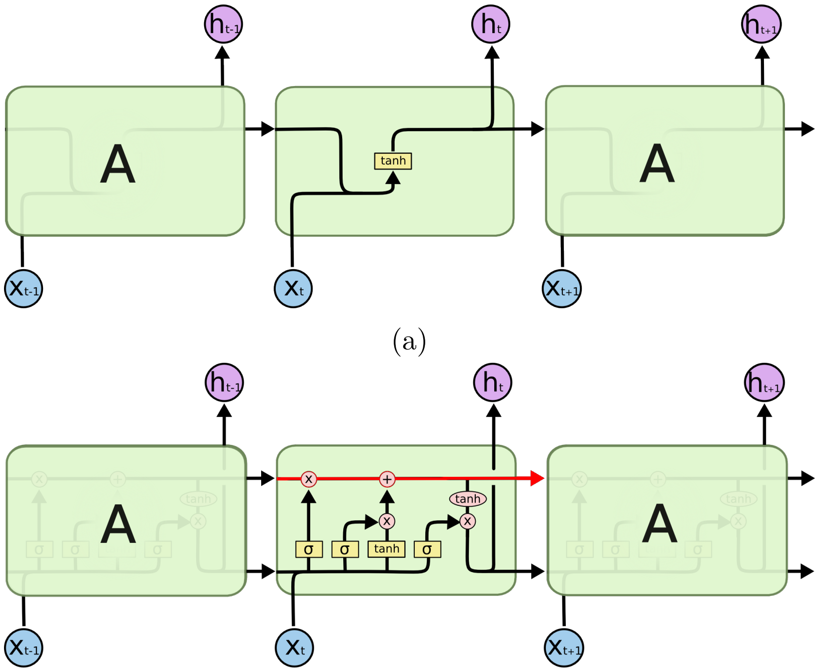


Figure A.5: Unfolded RNN representation inspired by Britz (2016)

Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps. This is called Back-Propagation Through Time (BPTT) and it basically means that, in order to compute the gradient at t, $t-1$ steps have to be backpropagated and then summed up. The loss with respect to the weight matrix $\boldsymbol{W}$ can be then computed with the rule chain based the relation of parameters expressed on equation (A.6):

$$\frac{\partial E_t}{\partial \boldsymbol{W}} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{s}_t} \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_k} \frac{\partial \boldsymbol{s}_k}{\partial \boldsymbol{W}}$$

The activation functions have null derivatives at both ends and it turns that the gradient values shrink exponentially fast, eventually vanishing completely after a few time steps. This is the so-called vanishing gradient problem and doesn't allow RNN to have long-range dependencies. All kind of neural networks have the potential of generating this problem, but RNNs are especially affected because of its dependencies with all past inputs.



(a)

(b)

Figure A.6: Comparison between a regular RNN (a) and a LSTM (b) (Olah, 2015)

The inverse case is known as the exploding gradient problem, and it basically means that gradients take enormous values and thus in long sequences taking a value of NaN (not a number). This problem is therefore obvious to detect and have simpler solutions such as clipping the gradient at a given threshold.

One of the most common solutions to vanishing gradients are the Long Short-Term Memorys (LSTMs), a version of RNNs. This architecture was introduced by Hochreiter and Schmidhuber (1997) and uses a gating mechanism to capture long-term dependencies. The chain structure is the same as in simple RNNs, but the repeating module has a different structure of four layers, represented by yellow boxes in figure A.6. The cell state (red horizontal line in figure A.6b) is the key of LSTMs, and it is based on having minor linear interactions with the outputs of the sigmoid layers, also known as gates. The outputs of these three gates (named forget, input and output from left to right) are between zero and one, which describes how much of each component should be discarded or remembered.

A commonly used variation of the LSTM is called Gated Recurrent Unit (GRU) and was introduced by Cho et al. (2014). Overall, this approach combines the forget and input gates into a single update gate, while the other is known as the reset gate. Moreover, GRUs merge the cell and hidden states as they don't have an output gate. Seeing this, it's obvious that this architecture have fewer parameters than LSTMs and this may train faster or need less data to generalize.

RNNs and their variants exposed above, are a powerful tool for modeling sequential data, but the dependence of each time-step's computation on the previous time-step's output limits parallelism and makes RNNs unwieldy for very long sequences. Quasi-Recurrent Neural Networks (QRNNs) are the state of the art to neural sequence modeling and alternate convolutional layers (see section A.4), which apply in parallel across time-steps, and a minimalist recurrent pooling function that applies in parallel across channels. Due to their increased parallelism, they are up to 16 times faster at train and test time (Bradbury et al., 2016).

A.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are typically conceived for treating with images but are also used in time series in front of RNNs to avoid typical problems intrinsic to that architecture such as the vanishing gradient and the lack of parallelism. Their basic idea is to deal with very high-dimensional inputs without needing a lot of parameters like a typical neural network would. In computer vision problems, this means that no features have to be extracted from images but the full image can be used as an input of the CNN and therefore, the grid-like topology of the input can be exploited. In order to reduce the number of parameters, each hidden unit is only connected to a subregion of size the receptive field.

The units that share the same parameters are organized into the same feature map and thus the same features are extracted at every position making the network translation invariant. This is a very desirable characteristic on image classification problems as the subject to recognize is not always located at the same position of the picture. The feature map is computed through a discrete convolution that gives name to the architecture. This means that a certain learned kernel slides over the input tensor computing dot products and shifting the stride number of samples.

Hidden units in the same neighborhood are pooled through a pooling layer that reduces the number of features. This layers typically take the maximum or the average of the non-overlapping neighborhoods and thus reduce the dimensionality and provide invariance to small local changes. Pooling approach is not as aggressive as taking strides larger than 1, i.e. skipping input samples. It also aims to reduce the dimensionality but on the feature map and there is no overfitting risk as no parameters are added. Although it yields to better results, the model is more expensive to compute as convolutions are running with lower stride.

Both convolutional and pooling layers are combined and usually followed by a regular neural network like in the architecture depicted in figure A.7. The fully connected layer can also be expressed as a convolution of kernel size 1 as it behaves like a small neural network.

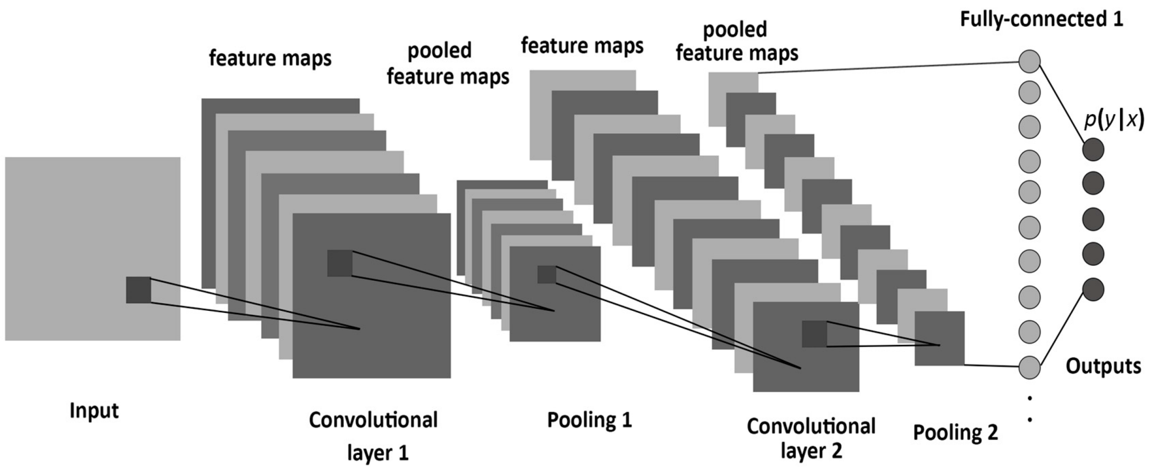


Figure A.7: Structure of a CNN for image classification from Albelwi and Mahmood (2017)

As it can be seen, there is a bunch of possibilities when combining layers and tuning each of the hyper-parameters that this architecture offer. Some layers can also be combined and run in parallel building what is known as inception modules. The outputs of these are then concatenated at the top of the modules and combined.

A.5 Embeddings

Embedding are mappings from discrete values to dense real value vectors that relate similarities to distances in the new feature space. This is a kind of unsupervised learning as the closeness of objects is interfered by the computer itself, i.e. there are no target values or labels like in supervised learning. A clear example of the use of embeddings is in machine translation problems, where words do not have a natural vector representation. Similar words occur in alike contexts and thus become close vectors. This means that a class for every possible word, which would be unmanageable, is no longer needed.

The training process is quite simple. Following the example of machine translation, every word of each sentence would be mapped to an embedding which is initialized randomly. The aim is to update the model with the premise of predicting the context of a word given a window around it. The result is that words with semantic and syntactic analogy which are used in similar contexts are mapped to close vectors in the feature space. When comparing embeddings, the length of the vector is not relevant, so no euclidean norm is used to check the similarities but cosine distance (equation (A.7)).

$$d(\boldsymbol{v}_1, \boldsymbol{v}_2) = \frac{\boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{||\boldsymbol{v}_1|| \cdot ||\boldsymbol{v}_2||} \tag{A.7}$$

The resulting feature map can be then projected into a 2-D space in order to study the learned relationship between inputs as depicted in figure A.8. The distance is taken into account when doing the projection, so similar inputs can be then analyzed and clustered into categories in an intuitive way.

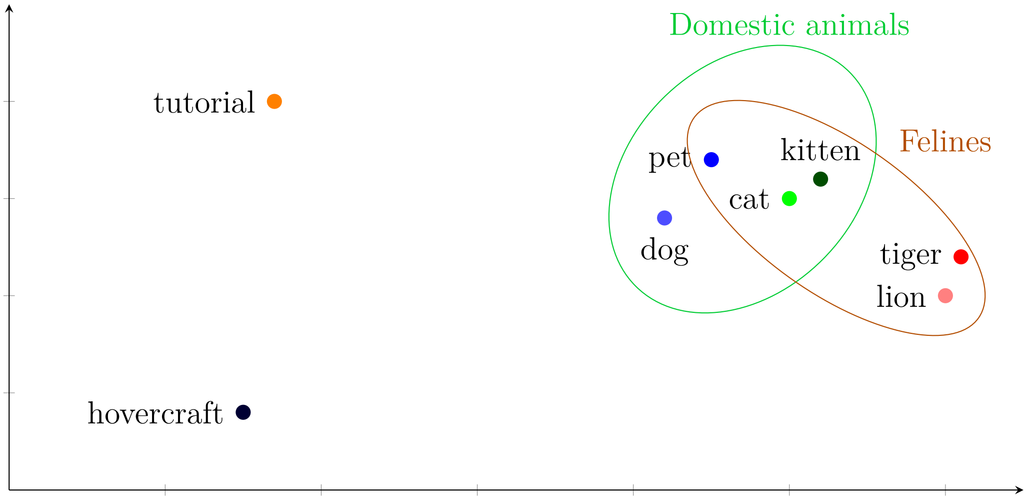


Figure A.8: 2-D projection of the feature map resulting of applying an embedding

B Work Plan

B.1 Work Packages

Project: Environment familiarization	**WP ref: WP1**	
Major constituent: Develop programming skills	Sheet 1 of 3	
Short description: Familiarize with the PyTorch Deep Learning framework as well as with the remote deployment environment used to take advantage of VEU group resources.	Planned start date: 22/01/2018 Planned end date: 01/03/2018	
	Start event: IDL course End event: Train first model	
Internal task T1: Introduction to Deep Learning (IDL: 230325) Internal task T2: Udacity Deep Learning course Internal task T3: Create environment and clone initial source code to VEU servers Internal task T4: Run a model with few modifications to evaluate the first results.	Deliverables: IDL certificate Udacity certificate	Dates: 30/01/2018 14/02/2018

Table B.1: Work Package 1 breakdown: Environment familiarization

Project: Theoretical learning	**WP ref: WP2**	
Major constituent: Study the theory behind newly developed vocoders	Sheet 1 of 3	
Short description: Research the state of the art technologies of speech synthesis and understand both their implementation and the novel ideas that they brought.	Planned start date: 25/01/2018 Planned end date: 15/04/2018	
	Start event: Project kick-off End event: Thesis drafting	
Internal task T1: Wavenet Internal task T2: Parallel Wavenet Internal task T3: SampleRNN Internal task T4: Neural Audio Generation for Speech Synthesis Internal task T5: Tacotron Internal task T6: Efficient Neural Audio Synthesis Internal task T7: Deep learning applied to Speech Synthesis Internal task T8: Relevant references of the above papers	Deliverables: -	Dates: -

Table B.2: Work Package 2 breakdown: Theoretical learning

Project: Multi-speaker Neural Vocoder	**WP ref: WP3**	
Major constituent: Adapt model	Sheet 2 of 3	
Short description: Feed phoneme information and speaker identity to generate speech. The talker either could or not be the same (voice conversion).	Planned start date: 15/03/2018 Planned end date: 15/05/2018	
	Start event: End of WP2 End event: Start of WP4	
Internal task T1: Database preparation Internal task T2: Adaptation of the model Internal task T3: System evaluation	Deliverables: Model outputs	Dates: 15/05/2018

Table B.3: Work Package 3 breakdown: Multi-speaker Neural Vocoder

Project: Voice Conversion	**WP ref: WP4**	
Major constituent: Obtain speaker-independent conditioners	Sheet 2 of 3	
Short description: Design an architecture to achieve voice conversion.	Planned start date: 15/05/2018 Planned end date: 15/06/2018	
	Start event: End of WP3 End event: Start of WP5	
Internal task T1: Design bottleneck and GAN-like approaches Internal task T2: Training and optimization Internal task T3: System evaluation	Deliverables: Model outputs	Dates: 15/05/2018

Table B.4: Work Package 4 breakdown: Voice Conversion

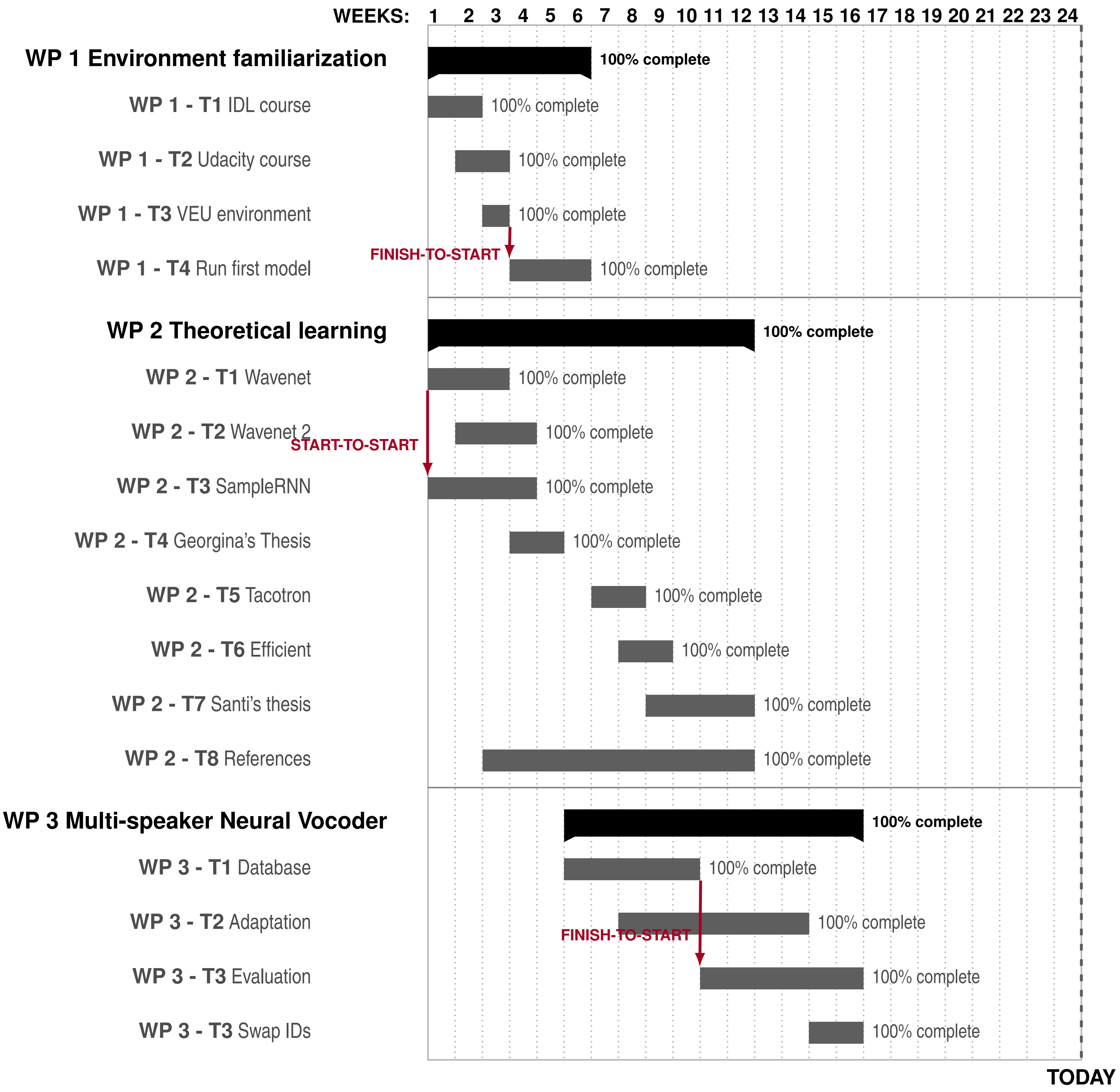
Project: *Eigen-voice* representation	**WP ref: WP5**	
Major constituent: Mimic new voices	Sheet 2 of 3	
Short description: Reproduce a voice corresponding to an unseen speaker with few samples by projecting its voice to the *eigen-voice* space.	Planned start date: 15/06/2018 Planned end date: 01/07/2018	
	Start event: End of WP4 End event: Final system evaluation	
Internal task T1: Integrate speaker recognition system with embeddings as outputs Internal task T2: Joint training and optimization Internal task T3: Produce speech in line with WP3 and/or WP4 Internal task T4: Build discriminant network to enhance voice biometrics	Deliverables: Model outputs	Dates: 01/07/2018

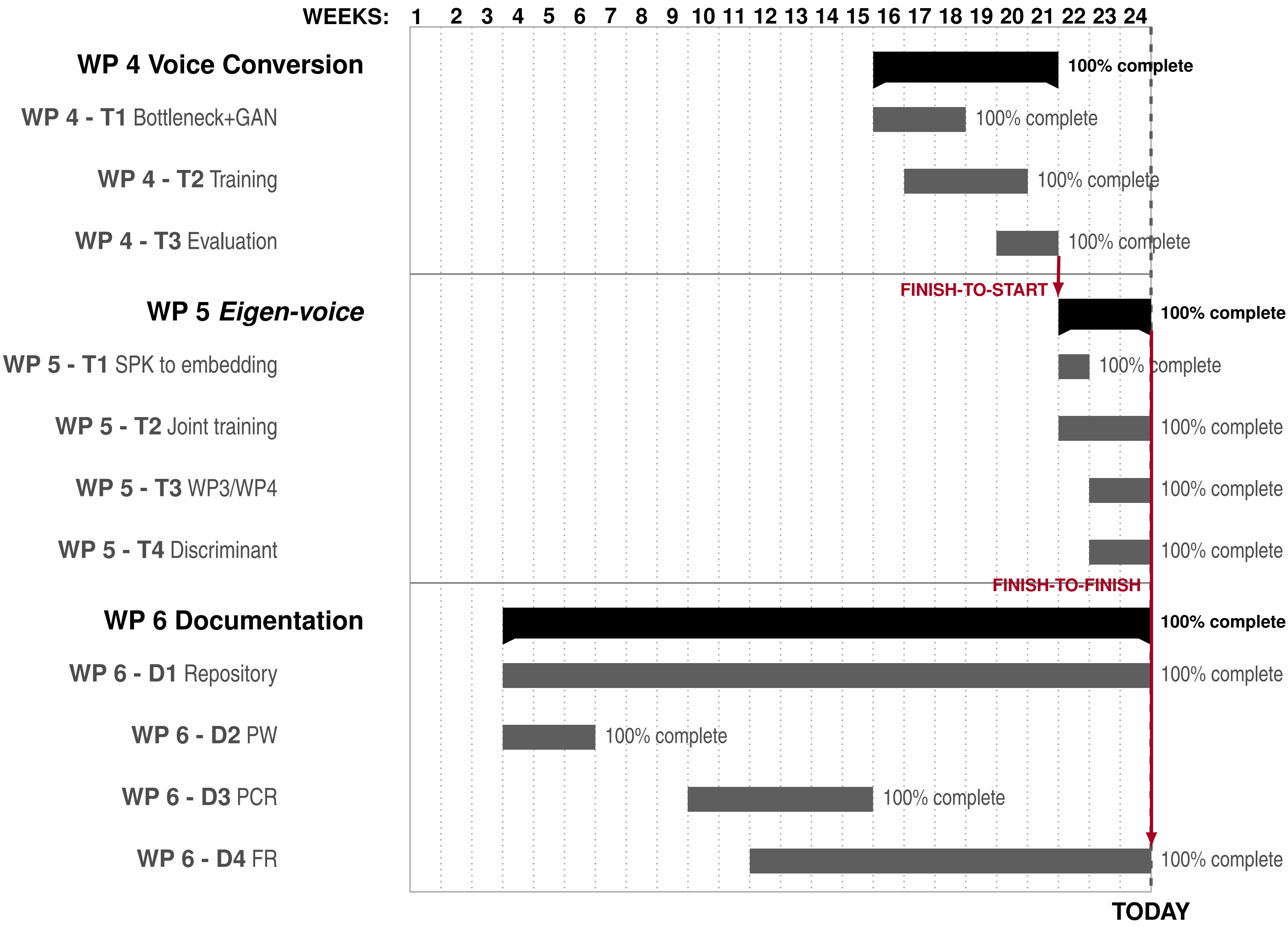
Table B.5: Work Package 5 breakdown: *Eigen-voice* representation

Project: Documentation	**WP ref: WP6**	
Major constituent: Deliver all reports	Sheet 3 of 3	
Short description: Write the different project reports with the correspondent format and review its redaction and rigorousness. List bibliography used all along the thesis.	Planned start date: 16/02/2018 Planned end date: 02/07/2018	
	Start event: Start of WP2 End event: Final Delivery.	
Documentation task D1: References repository	Deliverables:	Dates:
Documentation task D2: Proposal and Workplan	PW	15/04/2018
Documentation task D3: Project Critical Review	PCR	07/05/2018
Documentation task D4: Final Report	FR	02/07/2018

Table B.6: Work Package 6 breakdown: Project documentation

B.2 Gantt diagram




WEEKS: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
WP 4 Voice Conversion
100% complete
WP 4 - T1 Bottleneck+GAN
100% complete
WP 4 - T2 Training
100% complete
WP 4 - T3 Evaluation
100% complete
FINISH-TO-START
WP 5 Eigen-voice
100% complete
WP 5 - T1 SPK to embedding
100% complete
WP 5 - T2 Joint training
100% complete
WP 5 - T3 WP3/WP4
100% complete
WP 5 - T4 Discriminant
100% complete
FINISH-TO-FINISH
WP 6 Documentation
100% complete
WP 6 - D1 Repository
100% complete
WP 6 - D2 PW
100% complete
WP 6 - D3 PCR
100% complete
WP 6 - D4 FR
100% complete
TODAY

C Unfolded structure of SampleRNN modules

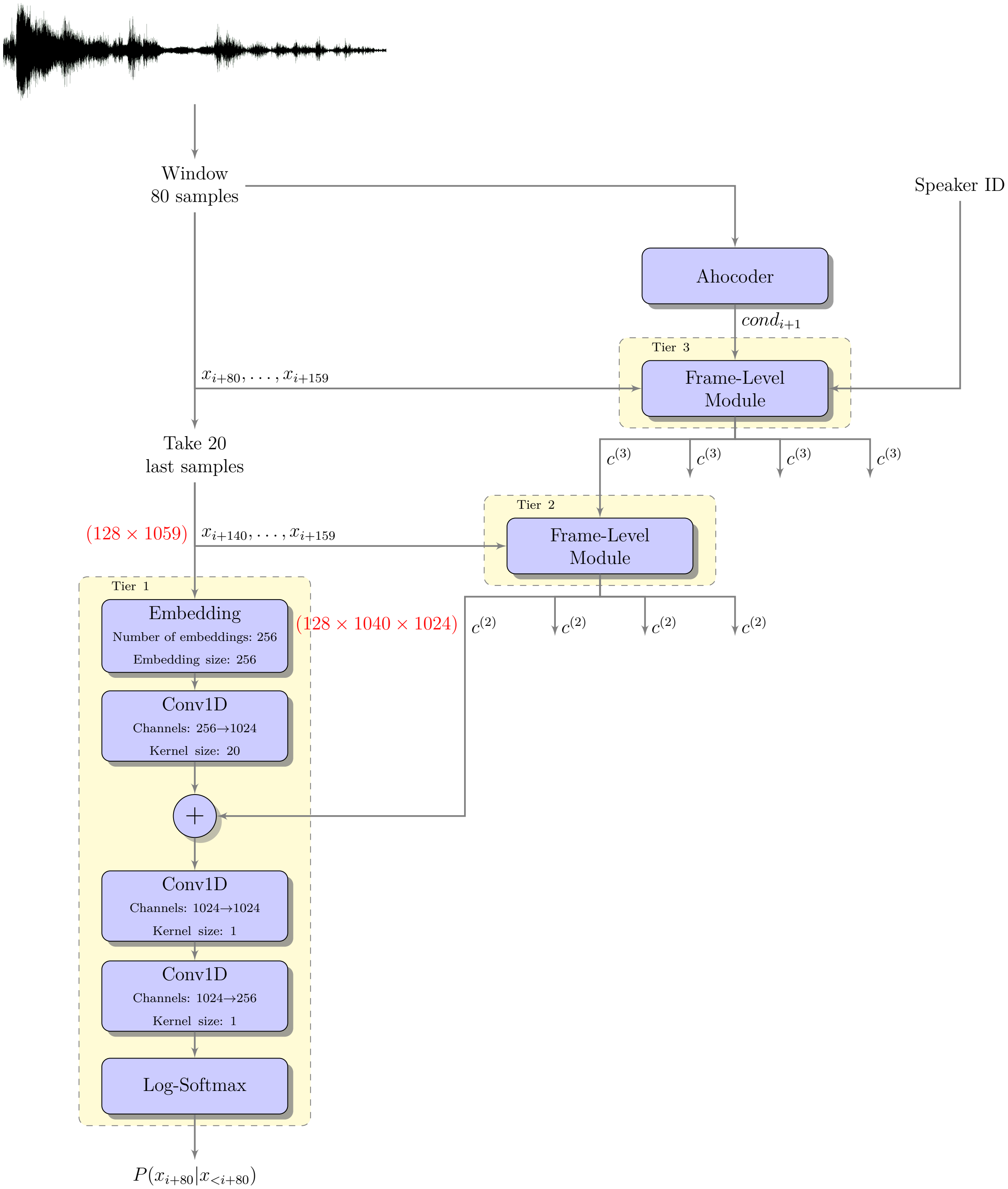


Figure C.2: Unfolded structure of the sample level tier of the adapted model

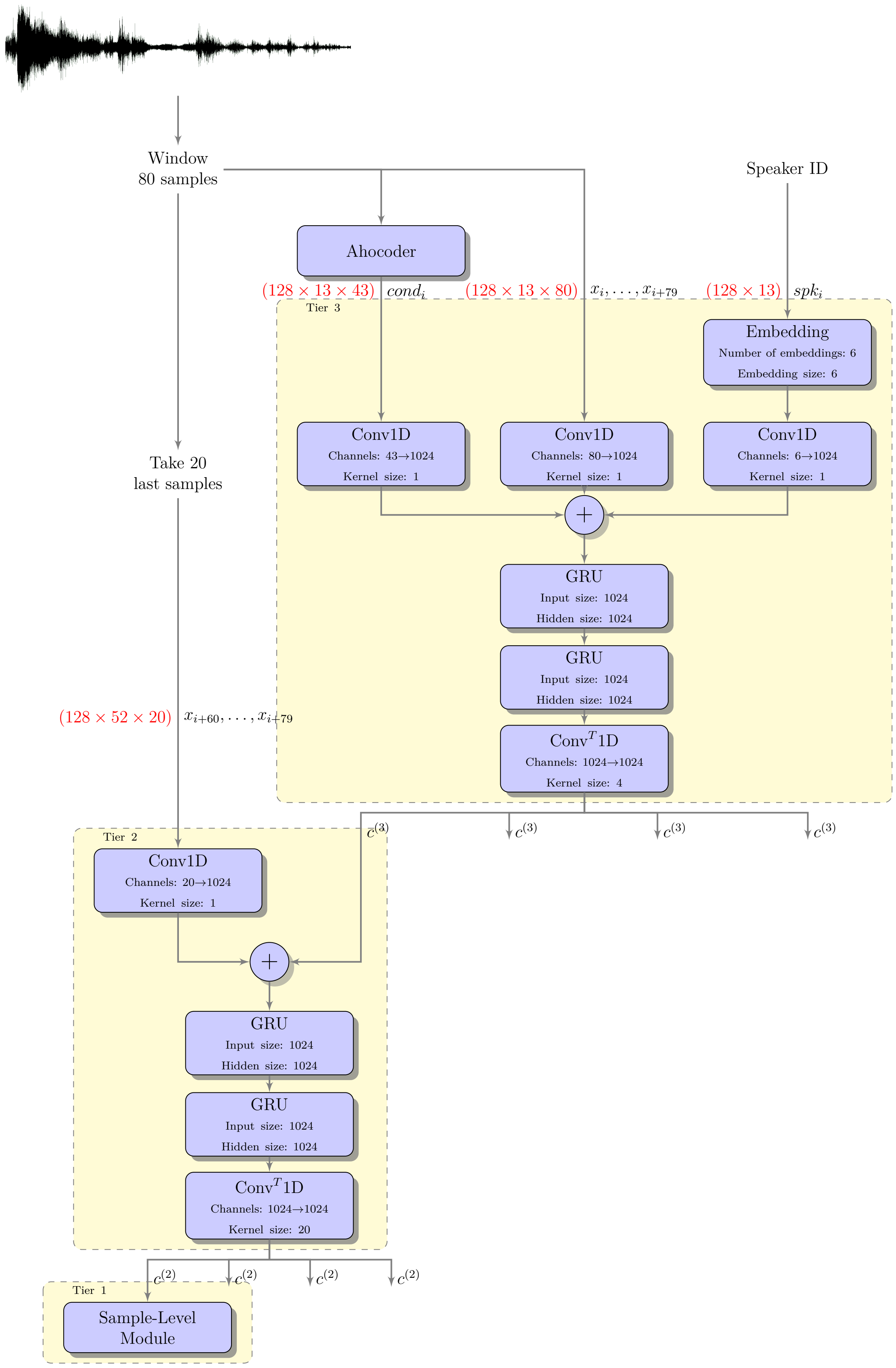
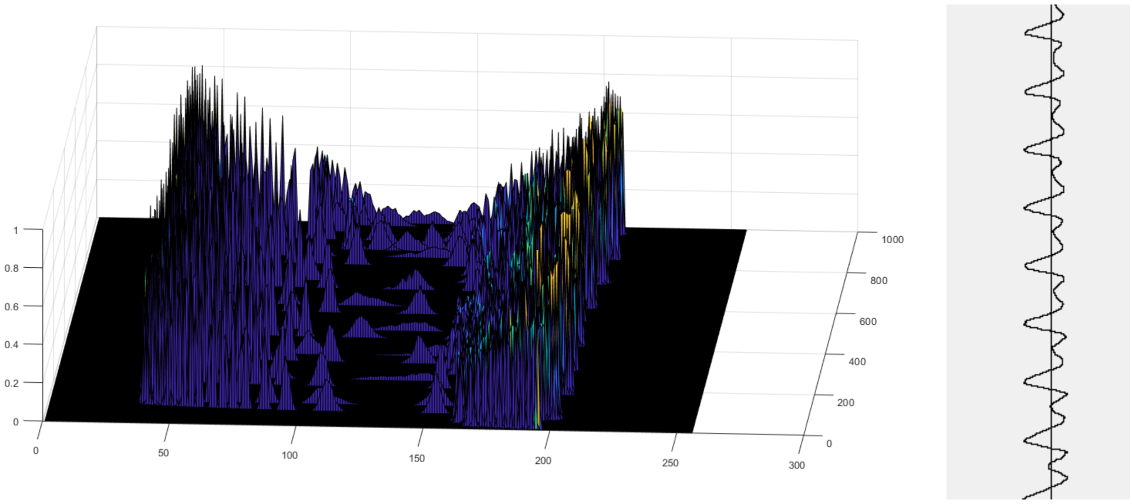


Figure C.1: Unfolded structure of the frame level tiers of the adapted model

D Resulting plots

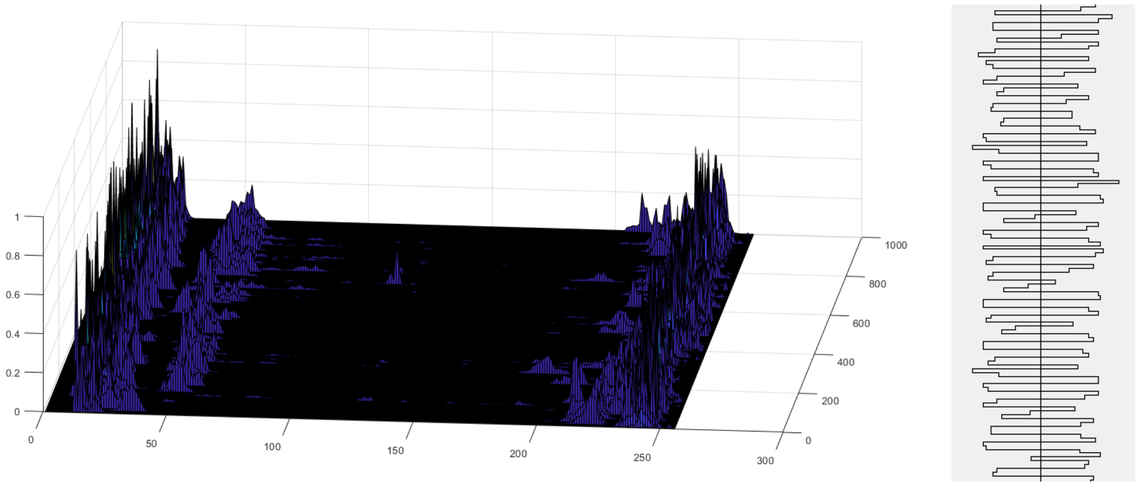
D.1 Output distributions



(a) Output distribution

(b) Waveform

Figure D.1: Correct output distribution behavior during 1000 samples (62.5 ms) and its correspondent waveform rotated to fit the axis of (a)

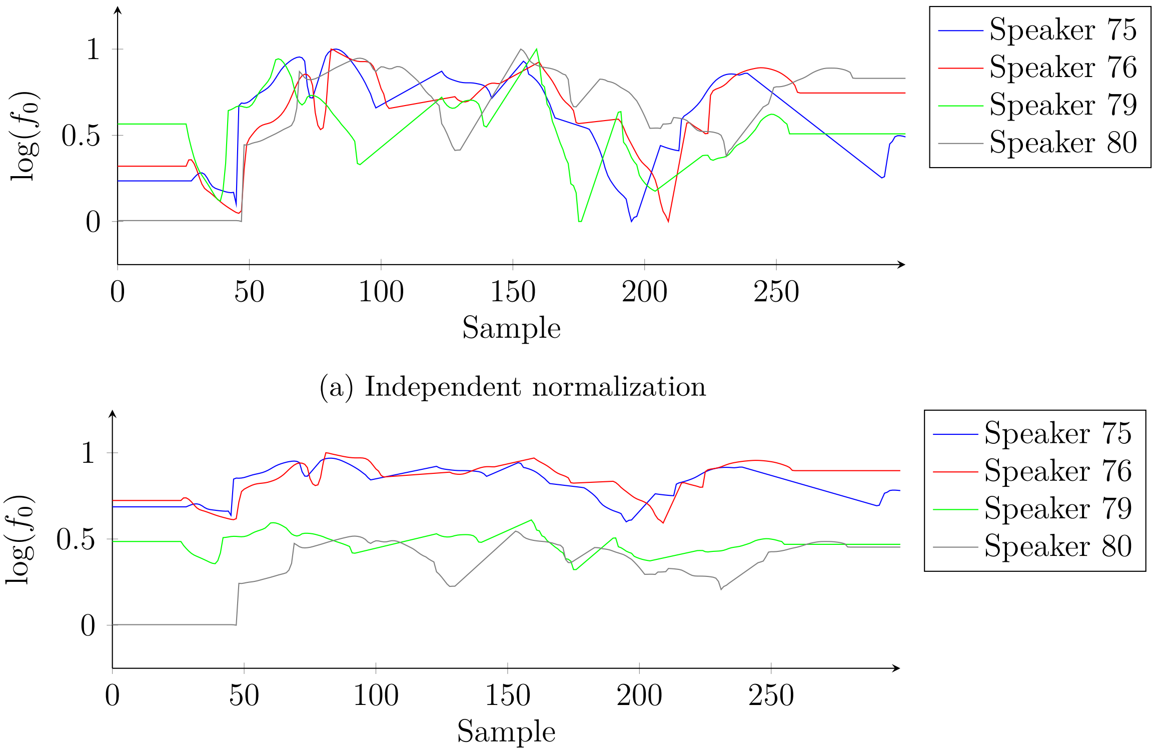


(a) Output distribution

(b) Waveform

Figure D.2: Saturated output distribution behavior during 1000 samples (62.5 ms) and its correspondent waveform rotated to fit the axis of (a)

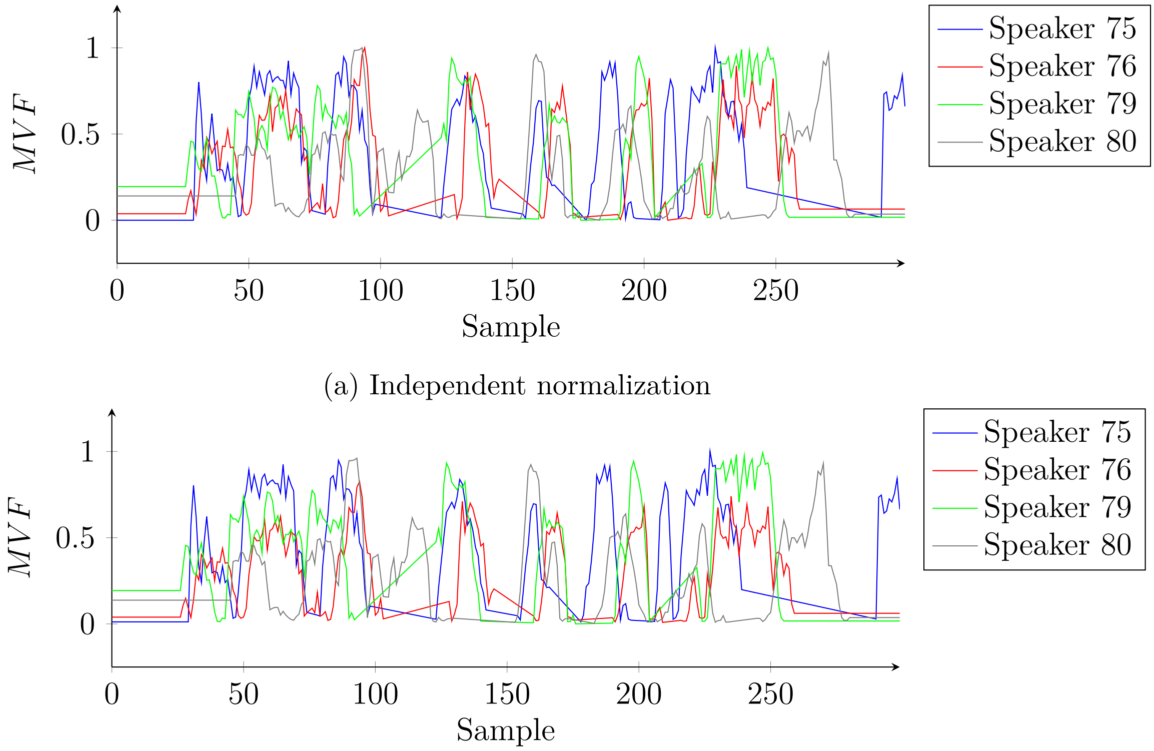
D.2 Normalization plots



(a) Independent normalization

(b) Joint normalization

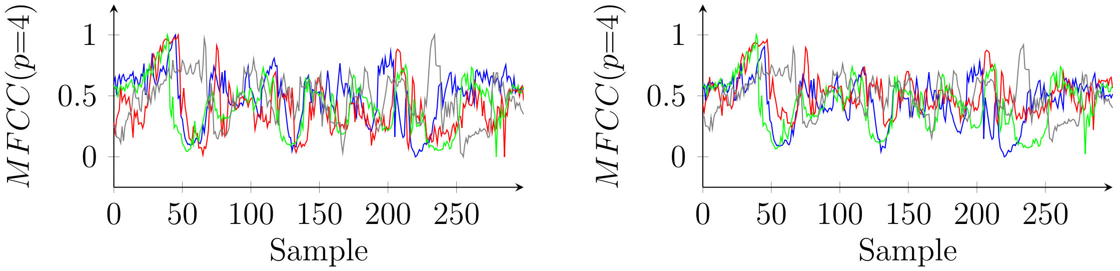
Figure D.3: Normalized logarithmic pitch contour



(a) Independent normalization

(b) Joint normalization

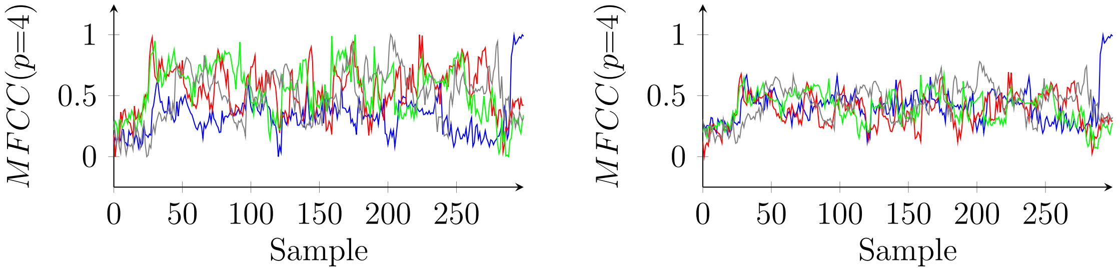
Figure D.4: Evolution of normalized maximum voiced frequency



(a) Independent normalization

(b) Joint normalization

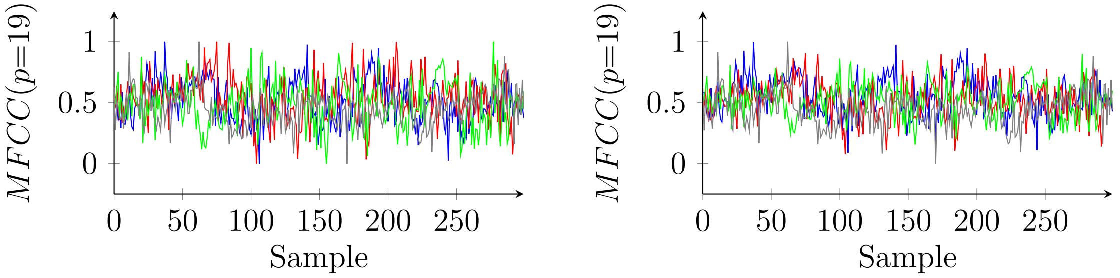
Figure D.5: Evolution of normalized Mel-Frequency Cepstral Coefficient (MFCC) order 3



(a) Independent normalization

(b) Joint normalization

Figure D.6: Evolution of normalized MFCC order 4



(a) Independent normalization

(b) Joint normalization

Figure D.7: Evolution of normalized MFCC order 19