Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Voice Conversion using Deep Learning

Degree's Thesis
Audiovisual Systems Engineering

Author: Albert Aparicio Isarn
Advisors: Antonio Bonafonte and Santiago Pascual de la Puente

Universitat Politècnica de Catalunya (UPC)
2016 - 2017

Abstract

In this project we present a first attempt at a Voice Conversion system based on Deep Learning in which the alignment between the training data is intrinsic to the model.

Our system is structured in three main blocks. The first performs a vocoding of the speech (we have used Ahocoder for this task) and a normalization of the data.

The second and main block consists of a Sequence-to-Sequence model. It consists of an RNN-based encoder-decoder structure with an Attention Mechanism. Its main strengts are the ability to process variable-length sequences, as well as aligning them internallly.

The third block of the system performs a denormalization and reconstructs the speech signal.

For the development of our system we have used the Voice Conversion Challenge 2016 dataset, as well as a part of the TC-STAR dataset.

Unfortunately we have not obtained the results we expected. At the end of this thesis we present them and discuss some hypothesis to explain the reasons behind them.

Resum

En aquest projecte presentem un primer itent en la realització d'un sistema de Conversió de Veu basat en Aprenentatge Profund (Deep Learning) en el qual l'alineament entre les dades d'entrenament sigui intrínsec al model.

El nostre sistema s'estructura en tres blocs principals. El primer bloc codifica la veu en paràmetres (*vocoding*). Hem usat el codificador Ahocoder per a aquesta tasca. A més a més, aquest primer bloc normalitza les dades.

El segon bloc consisteix en un model *Sequence-to-Sequence*. Consisteix en una estructura codificador-decodificador basada en Xarxes Neuronals Recurrents (RNN) amb un Mecanisme d'Atenció (*Attention Mechanism*). Els punts forts d'aquest model són la capacitat per a tractar seqüències de durada variable, alhora que les alinea internament.

El tercer bloc del sistema desnormalitza les seqüències i reconstrueix els senyals de veu.

Per a desenvolupar el sistema hem usat el conjunt de dades del *Voice Conversion Challenge* 2016. Hem fet servir també una part del conjunt TC-STAR.

Desafortunadament no hem obtingut els resultats que esperàvem. Al final d'aquesta tesis presentem aquests resultats i plantegem algunes hipòtesis que els expliquen.

Resumen

En este proyecto presentamos un primer intento en la realización de un sistema de Conversión de Voz basado en Aprendizaje Profundo (*Deep Learning*) en el cual el alineamiento de los datos de entrenamiento es intrínseco al modelo.

Nuestro sistema está estructurado en tres bloques principales. El primer bloque codifica la señal de voz en parámetros (*vocoding*). Hemos elegido el *vocoder* Ahocoder para esta tarea. Este bloque también normaliza los parámetros codificados.

El segundo bloque consiste en un modelo *Sequence-to-Sequence*. Este modelo está formado por una estructura codificador-decodificador basada en Redes Neuronales Recurrentes (RNN) con un Mecanismo de Atención. Sus puntos fuertes son la capacidad de procesar secuencias de longitud variable, a la vez que las alinea internamente.

El tercer bloque del sistema desnormaliza los parámetros, y reconstruye la señal de voz a partir de ellos.

Para el desarrollo del modelo hemos usado el conjunto de datos (*dataset*) del *Voice Conversion Challenge* 2016. También hemos usado una parte del conjunto TC-STAR.

Desafortunadamente no hemos obtenido los resultados que esperábamos. Al final de esta tesis los presentamos y proponemos varias hipótesis que los explican.

To my family: tot i que el que us explico us sona a ciència ficció, sempre m'heu prestat el vostre suport quan l'he necessitat. El que heu fet per mi tots aquests anys no té preu.

To all my friends at *Distorsió*, *L'Oasi* and *El Xollo*: Gràcies per les incomptables estones que he compartit amb vosaltres i pels bons moments que resten per escriure a la història de Revistes. Som tots una gran família.

To Lucas: ets l'únic que em comprèn i amb qui puc mantenir una conversa sobre el grau que ens ha fet qui som. Que això no es perdi mai!

Acknowledgements

First and foremost, I would like to thank my advisors, Antonio Bonafonte and Santiago Pascual for the invaluable help they have given me during the whole development of this project.

I would like to also thank the Signal Theory and Communications Department for allowing me to use its computing power.

Last, but not least, I would like to thank the Free Software and Deep Learning communities, to which I offer this project. I specially thank the authors of *Python*, *NumPy*, *TensorFlow* and *Keras*, for the great tools they have provided me.

Revision history and approval record

Revision	Date	Purpose
0	20/04/2017	Document creation
1	26/04/2017	Document revision
2	12/05/2017	Document revision
3	15/05/2017	Document approval

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Albert Aparicio	albert.aparicio.isarn@alu-etsetb.upc.edu
Antonio Bonafonte	antonio.bonafonte@upc.edu
Santiago Pascual de la Puente	santiago.pascual@tsc.upc.edu

Written by:		Reviewed and approved by:		Reviewed and approved by:	
Date	15/05/2017	**Date**	15/05/2017	**Date**	15/05/2017
Name	Albert Aparicio	**Name**	Antonio Bonafonte	**Name**	Santiago Pascual de la Puente
Position	Project Author	**Position**	Project Supervisor	**Position**	Project Supervisor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we present the statement of purpose of this project. Then, its requirements and specifications are stated, as well as the methods used for the development of our proposal. Finally, we mention the work planification, the incidents we have encountered and what modifications we have made to the initial plan.

1.1 Statement of purpose

Voice Conversion (VC) is a useful complement in Text-to-Speech (TTS) as well as in Automatic Translation.

In the case of TTS, VC allows the systems to give many different voices, having to map them just for Voice Conversion, which is simpler than modeling voices for TTS. In Automatic Translation, Voice Conversion also allows the system to output a translated message using the voice of the speaker who uttered the message.

There are also uses in the entertainment industry. VC could be used to dub the voice of an actor into another language, or to generate a voice that can no longer be recorded (ex. the voice of a deceased person). Another use would be to synthesize inexistent voices, which would be useful for videogame characters [13].

Nowadays, computing power, greatly enhanced by Graphics Processing Units (GPU), has reached such a level that it is possible to work with Deep Neural Networks with reasonably short training times when mapping the speech features from the source speaker to those of the target speaker.

The goal of this thesis is to develop a deep learning-based system able to convert a voice signal from a speaker into another that sounds as if it were uttered by a different one. The result signal must keep the linguistic and prosodic elements of the original signal unmodified. In order to fulfil this task, we have used the dataset from the 2016 Voice Conversion Challenge and part of the TC-STAR dataset (described in detail in section 3.1).

The main contribution of this project is a first approach of a Voice Conversion system in which the alignment of the input data is implicit in it.

This project has been carried out at the Signal Theory and Communications Department (TSC) of the Universitat Politècnica de Catalunya (UPC) during the 2016 Fall and 2017 Spring semesters. It has been developed at the Speech Processing investigation group (VEU) [43] as a contribution to its research project **DeepVoice: Deep Learning Technologies for Speech and Audio Processing** (TEC2015-69266-P (MINECO/FEDER, UE)).

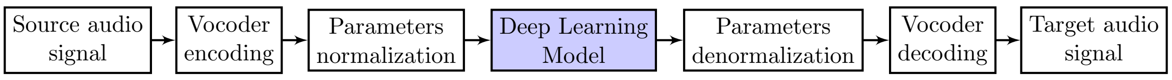


Figure 1.1: Block diagram of the proposed solution

1.2 Requirements and specifications

This project has been developed as an investigation effort on Voice Conversion, and future work is expected to build on what we propose here.

The main requirement of this project is to **develop a Deep Learning based model to map speech features from a source speaker to those of a target speaker in which there is no need to align the data prior to the conversion**.

The specifications of the project are:

- Prior to developing the system, a baseline system in which the data is aligned externally (by the use of the Dynamic Time Warping algorithm) has to be developed
- The new system outperforms the baseline system above in terms of naturality and similarity
- The model is implemented with TensorFlow and/or Keras
- The training and test data have to be prepared beforehand

1.3 Methods and procedures

This project aims to present a solution for voice conversion. The proposed solution consists of three main stages, depicted in Figure 1.1. The first stage performs a **Vocoding and Normalization** of the input signal. The second stage is the **Sequence-to-Sequence** model itself (explained in detail in Section 3.3.2). The third and last stage performs an inverse procedure of the first stage, as it **denormalizes the signal and generates voice using a vocoder to obtain the result speech signal**.

The models have been trained and tested using the public dataset from the Voice Conversion Challenge 2016 (see Section 3.1.1). This dataset contains clean parallel speech from 10 different speakers. We have also used a part of the TC-STAR dataset, which contains speech from 6 speakers (see Section 3.1.2).

This project has been developed using PYTHON 3 as the programming language of choice. In addition, we have used several libraries: *NumPy*, *TensorFlow*[2] and *Keras*[9]. We have used *Keras* for the baseline and the first Sequence-to-Sequence models, and *TensorFlow* for the last iterations of this architecture (reasons explained in Section 1.5). Additionally, some BASH scripting has been used in the vocoding stage.

All developed models have been trained on GPU-accelerated servers from the ”VEU” group at the Signal Theory and Communications Department from UPC.

1.4 Work Plan

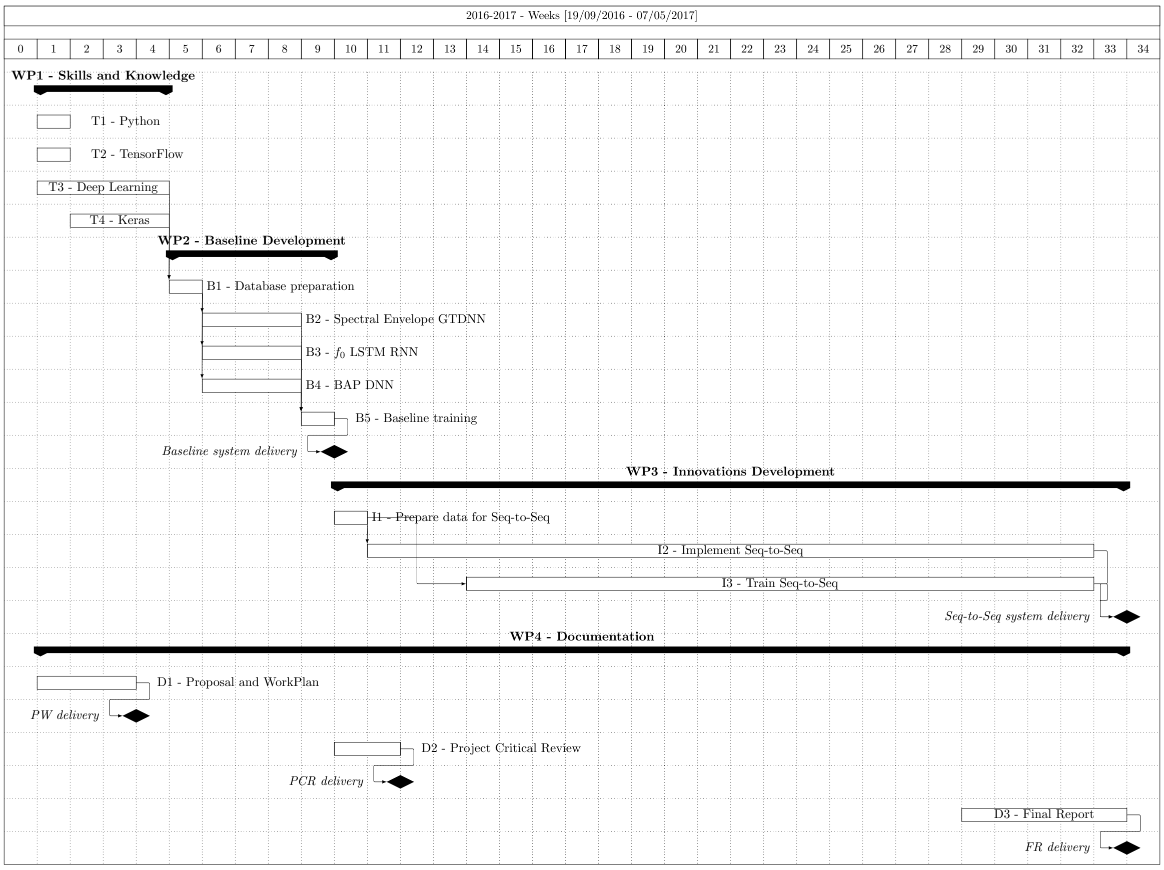
The project has been planned into several packages, detailed in Section 1.4.1. The planning in this section corresponds to the latest one, as the initial plan had to be modified. The reasons for this change will be explained in section 1.5.

1.4.1 Work Packages

- **WP1** - Skills and Knowledge
- **WP2** - Baseline Development
- **WP3** - Innovations Development
- **WP4** - Documentation

These Work Packages (WP) are detailed in the Gantt diagram of section 1.4.2. In it, each task in each WP is depicted in the diagram.

1.4.2 Gantt Diagram



1.5 Incidents and Modifications

One of the tasks in the **Innovations** package of the initial Work Plan of this project consisted in adding a Frequency Warping module to the Baseline system, and then developing a Generative Adversarial NN (GAN) model. However, due to the arisal of problems related to the Dynamic Time Warping algorithm used to align the dataset for the Baseline (see Section 2.2.1), we decided to change the direction of the project and develop a Sequence-to-Sequence model instead of the GAN one.

When we presented a modified version of the Work Plan in the Critical Review, there were five weeks allocated to the development of the Sequence-to-Sequence model. However, we have found the development of this model to be much more troublesome than we expected. Because of this, the WP3 of the Work Plan we present here has a lot more time allocated to this package.

Due to the problems explained above, the original WP4 (Systems Evaluation) has been removed from the project.

Chapter 2

State of the art

In this section we will briefly explain Deep Learning and the reasons for its popularity. We also describe the architectures used in this project. In addition, we will also explain Voice Conversion, its main challenges and commonly used techniques, using both classical and Deep Learning methods.

2.1 Deep Learning

Deng and Yu defined Deep Learning as "*a class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification*" [11, p. 199-200].

Deep Learning (DL) has risen in popularity amongst researchers in Computer Vision, Image Processing, Natural Language Processing and Speech Processing, among others. Its main strength is its ability to model complex non-linear mapping functions.

To do that, they must be trained with datasets containing a substantial amount of examples. This constraint is what has allowed DL to flourish in the recent years, as sizeable datasets have been released to the public; some of them belonging to Deep Learning contests, such as ImageNet.

These bigger, *deep* models require a considerable amount of computing power, as they have numerous parameters to tune. Because of this requisite, only in recent years (from early 2010's on), with the increasing availability of GPU- accelerated computing, researchers have been able to train these Deep Learning models with relatively short training times.

2.1.1 Deep Neural Networks (DNN)

One of the main architectures of Deep Learning is the Deep Fully-Connected Neural Network. It is made of several stacked Neural Networks. Figure 2.1 portrays an example of a Deep Neural Network.

In a Deep Neural Network, its input is a vector of features $\boldsymbol{x} \in \mathbb{R}^K$, where K is the number of input units of the DNN. The outputs of a DNN can be classes in the case of a classification problem or predicted feature vectors in the case of a regression problem. In both cases, the output is $\boldsymbol{y} \in \mathbb{R}^L$, where L is the number of output units of the DNN.

In a classification problem, it is common to make the DNN output the predicted class as a one-hot encoded vector. A one-hot encoded vector is a vector in which L is the number of classes. In this vector the element y_j is set to 1 when the class is j and the rest of the elements are set to 0.

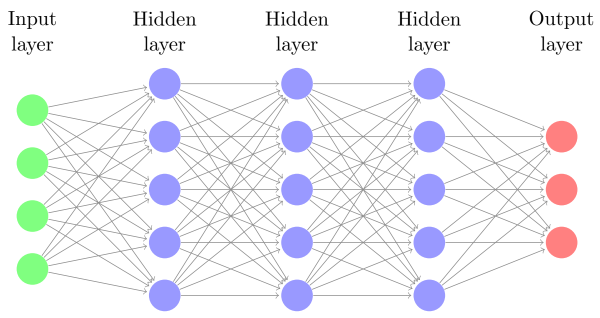


Figure 2.1: Example of a Fully-connected Deep Neural Network [10]

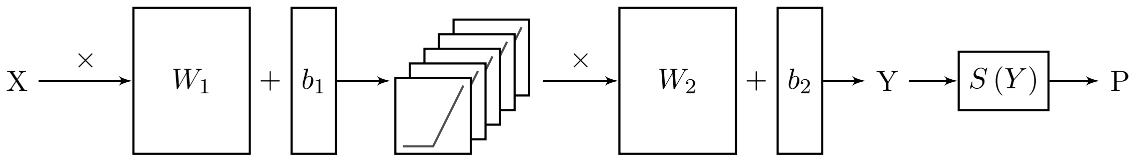


Figure 2.2: DNN-based logistic classifier. The first layer has a ReLU activation, while the second uses a Softmax activation

Each layer consists of a weighted linear combination of the layer's inputs. The result of this combination is then fed into a non-linear function, known as the *activation* function. This activation function is the element that enables a Deep Neural Network to model non-linear functions. If there were no activation functions, a DNN would become an affine transformation of its inputs. Equation (2.1) shows the mathematical expression of a DNN layer:

$$Y = f\left(W \times X + b\right) \tag{2.1}$$

In this equation, W, X and b represent the weight matrix, input matrix and bias matrix, respectively. f represents the non-linear activation. The values in both W and b are automatically adjusted in when training the model.

The most common activations are the **Hyperbolic Tangent (tanh)** (2.2a), the **Sigmoid** (2.2b), the **Softmax** function (2.2c) and the **Rectified Linear Unit (ReLU)** (2.2d):

$$\tanh\left(t\right) = \frac{e^t - e^{-t}}{e^t + e^{-t}} \tag{2.2a}$$

$$\sigma\left(t\right) = \frac{1}{1 + e^{-t}} \tag{2.2b}$$

$$S\left(\boldsymbol{z}\right)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \ \boldsymbol{z} \in \mathbb{R}^K, \ j = 1, \ldots, K \tag{2.2c}$$

$$ReLU(x) = \max(0, x) \tag{2.2d}$$

The resulting architecture of a multiple-layer DNN is shown in Figure 2.2:

2.1.1.1 Training of a Deep Neural Network

The training of a Deep Neural Network is done by iteratively adjusting the weights and biases so that a **loss function** (ex: Mean Square Error (MSE), Cross Entropy) is minimized by means of the Backpropagation algorithm [6, 23]. The result of this function is an indication of the prediction errors from the model.

In order to adjust the weights of the DNN, we use an **Optimizer**. One of the most well known of them is *Gradient Descent*. This optimizer updates the weights by adding to them the result of multiplying the gradient in that weight's node in the graph [23] by the learning rate. Equation (2.3) shows the expression of the weight update [44, ch. 4.3, p. 20]:

$$\Delta \boldsymbol{w} = -\eta \frac{\partial L}{\partial \boldsymbol{w}} \tag{2.3a}$$

$$\boldsymbol{w}[n+1] = \boldsymbol{w}[n] + \Delta \boldsymbol{w} \tag{2.3b}$$

In the equation, $\boldsymbol{w}$ is the weight vector, n is the current iteration of the training, η represents the Learning Rate and L the loss function.

Fundamentally, the convergence towards the minima of the loss is affected by the **Learning Rate** and the **Momentum**. The Learning Rate is a multiplicative factor that regulates the amount of the gradients of each node in the graph that are used to adjust the weights. The Momentum is an operation that dictates how much the gradients affect the current variation of the parameters. Momentum prevents the evolution of the parameters to oscillate [32].

Nowadays there are several easily available optimizers that ease this process. The ones most used in this project are RMSProp [16] and Adam [24]. The explation of the inner workings of these optimizers is out of the scope of this project; refer to [38] and [37] for implementation details, respectively.

Some other aspects to be taken into account in this iterative process are the batch size and number of training epochs. The batch size dictates how many sequences are processed before a loss value is computed. Larger batch sizes result in shorter training times, at the cost of a greater memory requirement and viceversa. The number of epochs is the number of times that the model processes the whole training database. Multiple epochs help the loss converge to the minima and also help with the optimization of the parameters for our problem.

In order to prevent overfitting i.e. lack of generalization due to excessive training, the dataset is split into three parts, `training`, `validation` and `test`. The training partition is the biggest of the three and is used to adjust the parameters of the model. The validation partition is used to evaluate the training and optimize the hyperparameters (size of the layers, learning rate, choice of optimizer, etc.) as well as stopping the training before the model is overfitted. The test partition is used to evaluate the overall performance of the system. Only the training partition affects directly the weights of the model.

Dropout is a regularization technique used to prevent overfitting "*by randomly omitting half of the feature detectors on each training case*" [18]. It makes each unit in the model

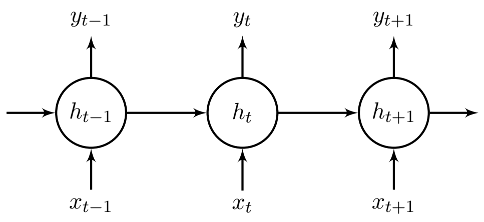


Figure 2.3: Recurrent Neural Network. h_j, x_j and y_j represent the inner state (parameters), input and output, respectively, of timestep j

"*learn to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate*" [18]. The probability with which a unit is disabled is one of the hyperparameters that are tuned with the validation partition.

2.1.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a particular kind of Neural Network. Their main point is that they capture patterns over time all respecting the ordered context of the sequence. To do so, RNNs introduce a temporal dimension in their layers. This dimension is expressed by its **timesteps**. Each of them is formed by a whole DNN, connected to the previous timestep. Figure 2.3 depicts a Recurrent Neural Network. Equations in (2.4) describe how RNN layers connect to each other [17].

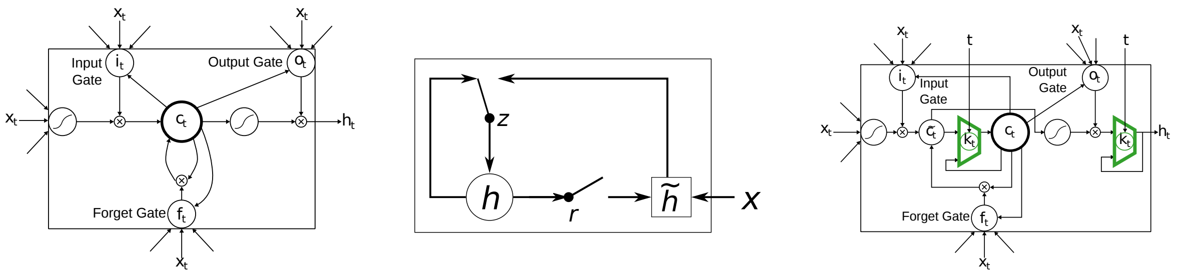
$$h_t = f\left(W_{xh}\, x_t + W_{hh}\, h_{t-1} + b_h\right) \tag{2.4a}$$
$$y_t = f\left(W_{hy}\, h_t + b_y\right) \tag{2.4b}$$

Recurrent Neural Networks present two problems that mar the training process. They are the **vanishing gradient** and **exploding gradient**.

The vanishing gradient problem occurs when the gradient of the loss is multiplied a many times by a number g that is $|g| < 1$. In this case, the gradients reach values close to 0 when propagated several timesteps back. An exploding gradient problem is the opposite of a vanishing one. In this case, the gradient of the loss is multiplied repeatedly by big numbers, so it tends to infinity (or else a number too big to handle).

Both these problems result in the RNN not being able to propagate the gradient far enough, thus not training properly. In these cases, the RNN is only capable of capturing short time dependencies.

There are several proposed architectures that adress these problems. The most well-known of them are the *Long Short-Term Memory (LSTM)* cell and the *Gated Recurrent Unit (GRU)*. We briefly explain them below. We also show a recently proposed architecture, the *Phased Long Short-Term Memory (PLSTM)* cell.



(a) LSTM cell [30] (b) GRU cell [8] (c) PLSTM cell [30]

Figure 2.4: Internal structure the LSTM, GRU and PLSTM cells

2.1.2.1 Long Short-Term Memory

In order to address the vanishing and exploding gradient problems, in 1997 Hochreiter and Schmidhuber proposed two modifications over the "vanilla" RNN. First, the gating idea to control the flow of information through the cell. Secondly, a separate cell state that could better retain long-term memory [19].

These cells contain several of so-called *gates* that control the flow of information that enters and leaves the cell. There are three gates, named *Input*, *Forget* and *Output*. They are trained to allow all, a part or none of the information in the input to be taken into account to adjust the weights in the DNN of a timestep. Figure 2.4a depicts its internal structure.

2.1.2.2 Gated Recurrent Unit

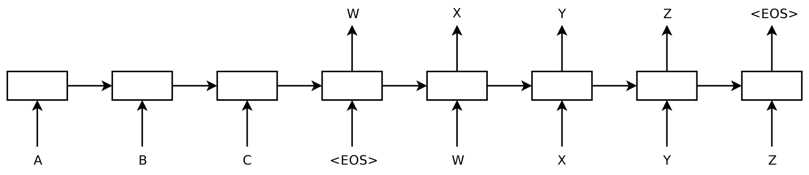
The *Gated Recurrent Unit* was proposed by Cho et al. in 2014 [8]. It features a structure similar to that of an LSTM cell, but with the Forget and Input gates combined into a single *Update* gate and the Cell state and the Hidden state merged together [31]. This results in a cell that performs well with less parameters to be trained. Figure 2.4b shows its internal structure.

2.1.2.3 Phased LSTM

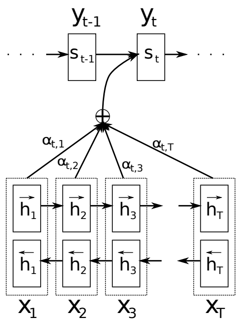
The *Phased Long Short-Term Memory* cell was presented by Neil et al. in 2016. It builds upon the LSTM cell by adding a time gate [30]. This gate is controlled by an internal clock, and only on a small time it is allowed to register new information. This allows the cell to retain information for a long number of timesteps without being modified. This cell is appropriate for large sequence lengths, and has been used in some of our Sequence-to-Sequence model iterations for this reason. Its internal structure is shown in Figure 2.4c.

2.1.3 Sequence to Sequence

We have used a Sequence-to-Sequence approach proposed in 2014 by Sutskever et al. [35]. The greatest strength of a Sequence-to-Sequence system is that it can work with sequences of different lengths in its input and output. This architecture was initially intended to be



(a) Sequence-to-Sequence [35]. The blocks from the beginning to the 'C' constitute the encoder. The blocks from the first 'EOS' and on form the decoder.



(b) Attention Mechanism [5]

Figure 2.5: Structures of (2.5a) Sequence-to-Sequence model and (2.5b) Attention Mechanism

used in Machine Translation but it is also useful for Dialog Generation (Chatbots) and Image Captioning. We have used it for the purpose of Voice Conversion, as it poses an analogous problem.

The Sequence-to-Sequence learning approach consists of an encoder-decoder structure. These two blocks are both deep Recurrent Neural Networks (RNN). The output of the encoder is a fixed-length context vector. The context vector contains a compact representation of the input information, in a fixed-length format. This context vector is replicated and injected into each of the units of the decoder, where the information is mapped to the output target information. The decoder output is thus, a sequence. Figure 2.5a shows the encoder-decoder structure of a Sequence-to-Sequence model.

In the Voice Conversion case, the input parameters are the source vocoded speech. The context vector can be understood as a representation of the uttered message in a "speaker-agnostic" space. The output parameters are the target speaker vocoded speech features.

2.1.3.1 Attention Mechanism

The Attention Mechanism was proposed in 2014 by Bahdanau et al. [5]. It builds upon the Sequence-to-Sequence approach, making some improvements.

Whilst the input of the decoder in a Sequence-to-Sequence model is a constant, fixed-length context vector for all sequence elements, with the Attention Mechanism each sequence element uses a different context vector.

Each context vector is computed from a weighted combination of *annotations* $(h_1, \ldots, h_{T_x})$. The encoder maps the input sequence to these annotations.

Equation (2.5) below shows the weighted combination of annotations to form a context vector c_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.5}$$

Each weight α_{ij} is computed by the expression in equation 2.6:

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)} \tag{2.6}$$

where $e_{ij} = a\left(s_{i-1}, h_j\right)$ is an *alignment model* (s_{i-1} is the hidden state of the encoder at timestep $t-1$). This alignment model "*scores how well the inputs around position j and the output at position i match*" [5].

These distinct context vectors establish a **mechanism of attention** with which the decoder can select those elements of the input sequence that are relevant. This mechanism relieves the encoder from compressing all the information in the input sequence in a fixed-length vector, thus letting the information "*be spread throughout the sequence of annotations*" [5].

2.2 Voice Conversion

In a Voice Conversion task, features from two speakers are involved. The objective is to alter the features of the source speaker so that the resulting speech signal sounds as if they were uttered by the target speaker. The challenge of Voice Conversion is to obtain a speech signal that is highly similar to that of the target speaker, while sounding natural at the same time.

Traditionally, techniques based on Gaussian Mixture Models (GMM) have been used mainly to map frame-to-frame spectral features [22, 34, 39]. According to Ling et al., Hidden Markov Models (HMM) with single Gaussian state-output PDFs are also used for this kind of mapping [26]. Some authors, such as Abe et al. have used vector quantization and spectral mapping [3]. Some previous work also includes the use of Dynamic Frequency Warping to compensate for the oversmoothing resulting of the use of GMMs [40].

Techniques like GMMs, Linear Regression Models and Partial Least Squares Regression assume the mapping from source to target to be linear, whereas Kernel Partial Least Squares Regression, Neural Network approaches and Restricted Boltzmann Machine approaches assume the mapping to be non-linear [47].

Parametric methods like GMMs, while they are capable of producing highly similar speech, they tend to oversmooth the resulting signal. This oversmoothing is caused by the use of Mean Square Error or Maximum Likelihood as they criteria for the training. These methods produce statistical averaging and neutralize the details of the speech [47].

There are several methods that attempt to solve this oversmoothing effect. Some approaches improve the acoustic models used to process the signal, such as Trajectory HMM, Product of Experts and Gaussian Progress Regression. Other approaches opt to modify the speech parameter-generation algorithm. Examples of these approaches are Global Variance [7], Segment-wise representation and minimizing Kullback-Leibler divergences [26]. A method often used to preserve details is Frequency Warping. This method attempts to minimize the spectral difference, or to maximize the correlation between the converted and target speech [47].

Another challenge of Voice Conversion is the mapping of prosodic features, such as pitch (F_0), duration or intensity, as they also contain relevant information about the speaker [7].

Deep Learning has been successfully applied in fields such as Computer Vision, Natural Language Processing and Speech Processing. In the case of Voice Conversion, it is a field in which some successful Deep Learning techniques have already been employed. Narendranath et al. proposed a system to transform the vocal tract features with the use of Artificial Neural Networks [29]. Chen et al. used several Neural Networks to map the features of the speakers [7]. Mohammadi and Kain used generatively-trained DNNs to model the mapping relationship between the spectral envelopes of source and target speakers [28]. Hsu et al. proposed a system to convert voice from non-parallel data [20].

The interest of the community in this problem is shown in the organization of the Voice Conversion Challenge (VCC) [41], where several research teams propose solutions for Voice Conversion.

2.2.1 Thesis proposal

When converting voice, we need the frames to be aligned. This is necessary because in order for the model to learn how to map the features from one speaker to the other, the spoken message must remain unchanged. If it was not, then the model would be learning an additional unintended mapping.

Up to this point, the alignment of the training data has been done using the Dynamic Time Warping algorithm. We think, however, that it is not adequate for this task because it introduces repeated frames into the signal.

If the model is fed signals with repeated frames it will learn a different statistic from the one it will encounter when inferring data i.e. when converting new data with no target to compare with. We believe this difference reduces the potential of the approach.

As a means to deal with this problem we have applied a Sequence-to-Sequence (Seq2Seq) approach. Sequence-to-Sequence learning has been used in different tasks involving the mapping between sequences of different lengths, such that any alignment between both is learned end-to-end within the neural model.

For this thesis we propose two systems, a baseline system and a Sequence-to-Sequence system. For the baseline, we have adapted a proposal from Interspeech 2016 by Chen et al.[7]. It will be explained in detail in section 3.3.1. The Seq2Seq system consists in a Sequence-to-Sequence model with Attention Mechanism. This model will be detailed in section 3.3.2.

Chapter 3

Methodology

This chapter cataloques the various stages that form our proposed systems. First, we will explain the datasets we have used. Then, how the data has been prepared for use with our models. Last, but not least, we detail each of the models we have developed and their various iterations.

3.1 Datasets

We have used two different datasets for this project: the dataset from the Voice Conversion Challenge (VCC) 2016, and the TC-STAR dataset.

Initially we used the VCC dataset. It was used to train the Baseline model, and the first two iterations of the Sequence-to-Sequence (Seq2Seq) model. However, while developing the third iteration, we found it not to contain enough data, so we switched to the TC-STAR dataset. The TC-STAR datset was also used to pretrain the second iteration of the Seq2Seq model.

3.1.1 Voice Conversion Challenge 2016 (VCC)

This dataset is public and was initially provided to the contestant teams of the Voice Conversion Challenge 2016 [33]. Its results were published in Interspeech 2016 [46].

This dataset is based on the DAPS dataset [41, 1]. It contains parallel short sentences in English from 10 different speakers, 5 female and 5 male. It contains 162 sentences for training and 54 for testing, making approximately 9 min 20 s of training speech per speaker, and 30 s of test speech per speaker.

Each sentence is encoded in mono at 16KHz, 16 bits and saved in a separate WAVE (`.wav`) file.

3.1.2 TC-STAR English Training Corpora for ASR: Recordings of EPPS Speech

The TC-STAR dataset was put together by the European project of the same name from 2004 to 2006 [36]. It contains English speech from European Parliament Plenary Sessions (EPPS). This corpus contains a total of 290 hours of speech from different speakers.

In this project, we have used only a part of this dataset, specifically, the speech from speakers 72,73,75,76,79 and 80. Speakers 72,75 and 76 are female; speakers 73,79 and 80 are male.

We have used all six speakers in the pretraining of the 2nd iteration of the Seq2Seq model (see Section 3.3.2.3). In total, there are approximately 41.5 hours of training speech between them. For testing, there are approximately 3.25 hours of speech.

In the case of speakers 75 and 76, there are approximately 1h 30 min of parallel speech per speaker. We used them in their entirety, apart from about 45s from each, which were reserved for testing purposes. These two speakers have been used in the Attention Mechanism Seq2Seq model (see Section 3.3.2.4).

Each sentence is encoded in mono at 16KHz, 16 bits and saved in a separate WAVE RIFF (`.wav`) file [36].

3.1.2.1 Silence removal

We found this dataset to contain significant misalignments in the sequences that increased the difficulty of the alignment, causing the Attention Mechanism in the Attention Seq2Seq model (see Section 3.3.2.4) to perform badly.

In order to ease the task of the Attention Mechanism, we preprocessed this dataset by reducing the leading and trailing silences in both source and target sequences to a maximum of 100 milliseconds.

3.2 Preparation of the datasets

In this section we show how the datasets have been parametrized and prepared to be used with our models. We first detail the vocoding and normalization process and then we explain the layout of the data in the tensors we enter into the models.

3.2.1 Vocoding

We have used a high quality vocoder to obtain parameters from the audio files in the dataset. The vocoder of choice has been Ahocoder by Erro et al. [14, 12]. This vocoder extracts Mel cepstral features (MCP), as well as the pitch of the signal in log scale ($\log(f_0)$) and its Maximum Voiced Frequency (MVF). The signal is reconstructed using a harmonic model for frequencies below the MVF. It adds a noise-like component for frequencies above the MVF.

We have used the default settings of Ahocoder v0.99. With these settings we obtain from the vocoder three parameter files (`.mcp`, `.lf0`, `.vf`) from each audio file. These files contain a row of parameters for each frame (5ms) of speech:

- `.mcp` - 40 cepstrum parameters
- `.lf0` - pitch value
- `.vf` - maximum voiced frequency

These files contain binary data, in which each parameter is a floating point value. We have used the `x2x` tool from SPTK [42] to convert them to ASCII format, so they could be loaded into Python with NumPy's `loadtxt`.

The pitch and MVF parameters present major discontinuities at the points where the signal changes from Voiced to Unvoiced, and viceversa. To mitigate them, we linearly interpolate these parameters. In order to be able to recover later the Voiced/Unvoiced information, we save a flag for each frame indicating whether it contains Voiced or Unvoiced speech.

3.2.2 Normalization

The speech parameters we have used have been normalized in order to control the magnitude of the gradients in the training. During the development of this project we have used two different normalizations. The first one we used is a Standard score normalization, and the second is a Feature scaling.

3.2.2.1 Standard score normalization

This normalization ensures that the result data distribution (assumed to be normal) has zero mean and unit variance. Equation (3.1) shows its expression.

$$\boldsymbol{x'} = \frac{\boldsymbol{x} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{3.1}$$

$\boldsymbol{x}$ represents a vector from the dataset. $\boldsymbol{\mu}$ is a mean vector, containing the mean of each parameter in the dataset. $\boldsymbol{\sigma}$ is a standard deviation vector, containing the standard deviation of each parameter in the dataset. $\boldsymbol{x'}$ represents the normalized vector. The fraction in the equation represents an element-wise division.

3.2.2.2 Feature scaling

This normalization guarantees that $x'_i \in [0, 1]$, where x_i is the ith dimension of an input parameter vector. Equation (3.2) depicts this scaling:

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

$\boldsymbol{x}_{max}$ is a vector containing the maximum value in the dataset for each parameter. $\boldsymbol{x}_{min}$ contains the minimum value in the dataset for each parameter. All operations in the equation are element-wise.

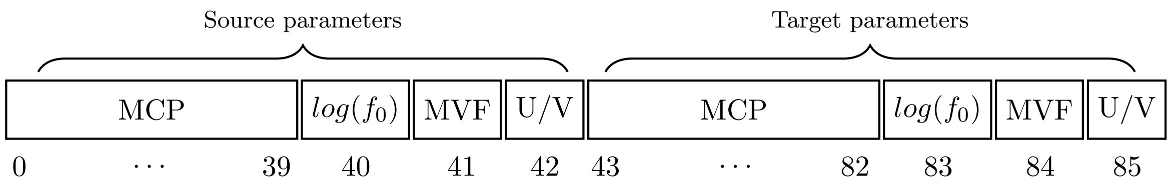


Figure 3.1: Baseline layout of the parameters of a frame. The width of each group of parameters is not in real scale.

3.2.3 Layout of the dataset tensors

The sequences of the datasets we have used are laid in matrices, so they can be easily fed into the model.

Due to the variability of their lengths, we must add some padding in order to standardize their lengths; we have used a padding with zeros. The sequences belonging to source speech are padded on the left i.e. before the first frame; the target sequences are padded on the right i.e. after the last frame.

This choice of left and right is related to the order with which the model is going to receive the frames. The decoder starts predicting target frames right after the last source frame has entered the model. The first target frame cannot be a padded frame because the model needs to be fed the ground truth data for that frame, thus we pad after the sequence has ended.

3.2.3.1 Layout in Baseline model

In the baseline model, the dataset was laid out into two 2-dimensional matrices, one for training data and one for test data.

These matrices contain one aligned frame on each row. The parameters from the vocoder are placed in columns. Figure 3.1 depicts the placement of the parameters in a frame.

3.2.3.2 Layout in Seq2Seq model

The data used for the Seq2Seq model is laid out in 3-dimensional tensor pairs. There is a pair for training data and one for test data. Each pair consists of a tensor for source data and another for target data. They are organized such that each row represents a sequence, each column, a timestep, and each position in the 3rd dimension represents a single parameter.

Figure 3.2 shows the organization of the 3- dimensional tensors, and Figure 3.3 depicts the organization of the parameters in a frame.

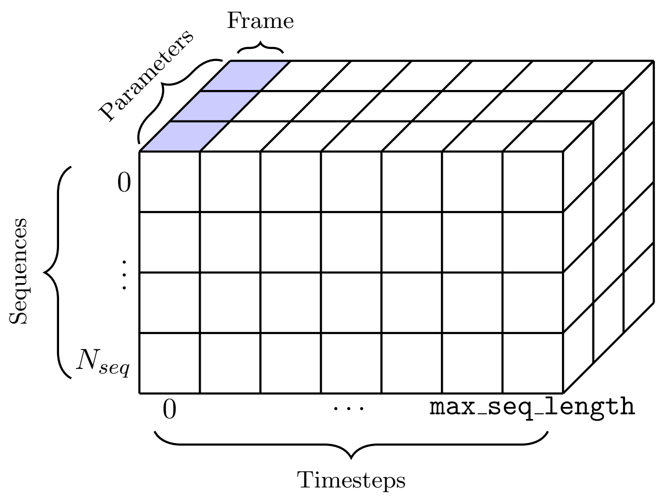


Figure 3.2: Layout of the 3-dimensional tensor of data for the Seq2Seq model

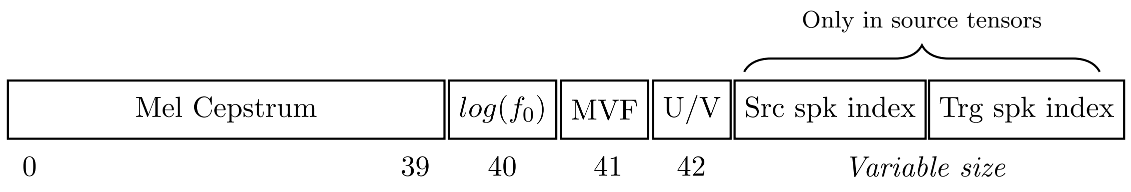


Figure 3.3: Seq2Seq layout of the parameters of a frame. The source and target speaker indexes take different number of parameters depending on the iteration of the model. In the explanation of each model we will specify how many parameters do these indexes take. The width of each group of parameters is not in real scale.

3.3 Proposed Models

We have developed two models for this project, a Baseline model, inspired by the proposal of Chen et al. [7], and a Sequence-to-Sequence model.

We have focused most of our efforts to this last model, and have encountered several obstacles that have hampered our progress. As a result of these difficulties, we have developed several iterations of the Seq2Seq model, each building upon the previous one. Section 3.3.2 details each and every one of the models.

3.3.1 Baseline

The Baseline model uses three Deep Neural Networks to map the source speaker to the target speaker. It uses a Recurrent Neural Network with GRU cells to map cepstral features, an RNN with LSTM cells to map the pitch of the signal, and a Fully-Connected DNN to map the Maximum Voiced Frequency (MVF) features.

These three networks have been implemented with Keras. The data used to train this model comes from the VCC dataset and is normalized with Standard score normalization.

The networks in this model are detailed in the following sections.

3.3.1.1 Cepstral features RNN-GRU

This RNN maps the 40 cepstral parameters of the source speaker to those of the target speaker, thus, the input and output sizes of this model are the same. This model works with tridimensional tensors of shape `(batch size, timesteps, data dimension)`. In this model, `batch size` is 1, as it takes one sequence at a time; `timesteps` is 50, meaning that the network can model a temporal evolution of up to 50 frames; `data dimension` is 40, being the 40 cepstral parameters to be mapped.

This model contains three layers: first a GRU layer with 70 stateful cells, second, a Dropout layer with probability $0,5$. Finally, acting as an output layer, there is a time-distributed Fully-Connected layer with 40 units.

This last fully-connected layer performs a projection of sorts, in order to allow the GRU layer to use as many cells as needed, keeping the output's dimension as 40. This last layer is time-distributed, meaning that it places a Fully-Connected layer at the output of each cell (each timestep) of the GRU layer.

This model has been trained for 50 epochs, with an RMSProp optimizer, with a learning rate of 0.0001 and an MSE loss.

3.3.1.2 Pitch RNN-LSTM

The task of this RNN is to map the pitch of the source to that of the target speaker. This network takes tridimensional tensors as its input. The shape of these tensors, for both input and output is `(batch size, timesteps, data dimension)`. In this case, `batch size` is 1, as it takes one sequence at a time; `timesteps` is 50, meaning that the network can model a temporal evolution of up to 50 frames; `data dimension` is 2, being the pitch value and a flag that detects if a frame contains voiced or unvoiced speech.

This model contains two layers: first an LSTM layer with 100 stateful cells and a time-distributed fully-connected layer with 2 units. This layer fulfills the same role as in the MCP network, projecting the output of the RNN to the desired dimensions.

This model has been trained for 50 epochs, with an RMSProp optimizer, with a learning rate of 0.0001 and an MSE loss.

3.3.1.3 Maximum Voiced Frequency DNN

This DNN maps the MVF features of the source speaker to those of the target speaker. In this case, the input and output shapes are different, because we input the features with context. This context means that the input layer receives the feature vector in (3.3):

$$(x_{t-C}, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_{t+C}, x_{uv}) \tag{3.3}$$

Where x_t is the feature in the current timestep, $x_{t\mp1}, \ldots, x_{t\mp C}$ represent C past and future frames, respectively; X_{uv} is the U/V flag, indicating if the current feature comes from a Voiced or Unvoiced frame.

The output of the DNN is a 2-dimensional vector, containing (y_t, y_{uv}), y_t being the prediction, and y_{uv} the U/V flag for the current timestep.

This network contains three fully-connected layers. The first two are of size 100, with a LeakyReLU activation [27] and a dropout layer with probability 0.5. The third layer is of size 2, with a linear activation i.e. no activation.

This model has been trained for 700 epochs, with an RMSProp optimizer, with an MAE loss. The learning rate in this model is variable. Its starting value is $5.5 \cdot 10^{-7}$, and it gets divided by 10 each time the validation loss does not decrease for 5 epochs.

3.3.2 Sequence-to-Sequence

The Baseline model presents a problem with the alignment of the dataset. By using the Dynamic Time Warping algorithm, we align each frame in the source data so that the model receives pairs of source-target frames that represent the same part of the message.

However, this alignment intrinsically introduces an inconsistency between training and

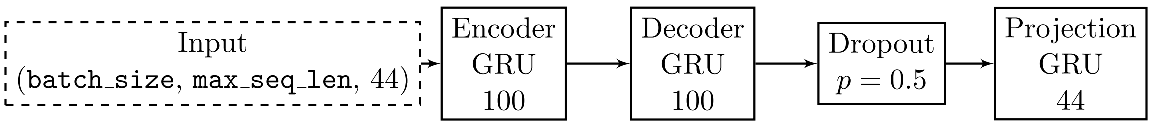


Figure 3.4: Graph of the 1st iteration of the Seq2Seq model

test data. The result frames from DTW does not maintain the temporal evolution of the original source signal, because many of the frames in a sequence get repeated one or more times, thus making the model learn from a signal that does not correspond to what will be input in a production stage, where the input signal is not altered in this aspect.

To deal with this problem, we propose a Sequence-to-Sequence model because it can natively learn the alignment between frames due to its architecture and the help of the Attention Mechanism.

3.3.2.1 First Sequence-to-Sequence iteration

This model is implemented with Keras. The dataset of this model does not use any speaker indexes, because it is trained to convert the speech from a single speaker to a single target speaker.

This first iteration uses three GRU layers and a Dropout layer, as depicted in Figure 3.4; the last GRU layer uses a linear activation. The dataset of this model has a Maximum Sequence Length of $\approx$ 8000 frames. It uses an MSE loss and Adam optimizer [24] with a Gradient Clipping of 10. The model has been trained for 50 epochs, with the default learning rate from Keras, 0.001.

3.3.2.2 Second Sequence-to-Sequence iteration

This second iteration was developed in an attempt to improve the poor results of the first iteration (see Section 4.2.1), having realized that we had not implemented a feedback connection in the decoder, which is part of the model proposed by Sutskever et al. [35].

Thus, this iteration introduces a feedback connection from the output of the graph to the decoder input, as well as featuring multiple inputs and multiple outputs. In addition, we have changed the layer type to the Phased LSTM, because at the time of implementing the model, this promising layer had recently been presented in NIPS 2016 [30]. Figure 3.5 depicts this graph.

This model has separate inputs for the dataset parameters and the speaker indexes. In this model, the speaker indexes are expressed as a one-hot encoding with 10 possible classes. This number of classes is due that we implemented this model to be able to convert from any to any of the 10 speakers in the VCC dataset. The source speaker index and target speaker index are input into a 10-to-5 Embedding before entering the Encoder. This Embedding is placed in the model to allow modularity between this model and the Pretraining version (see Section 3.3.2.3).

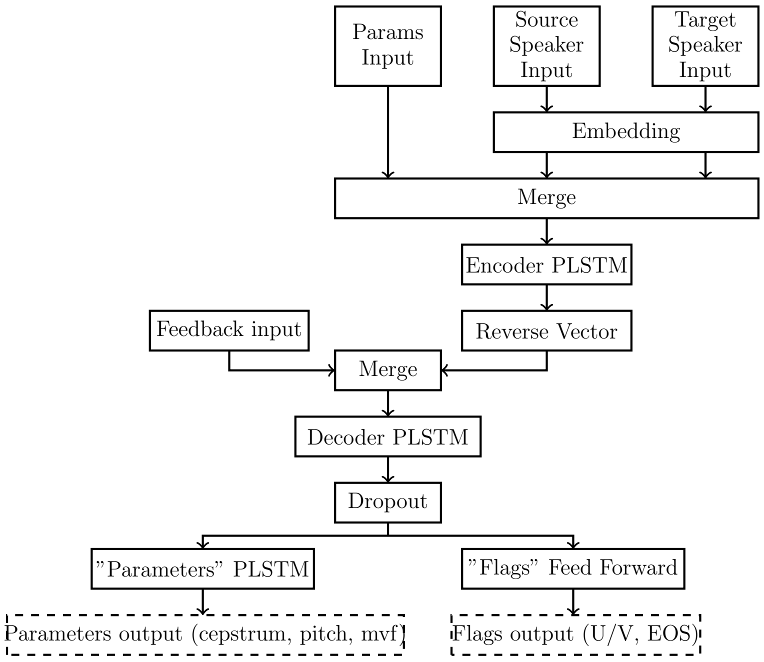


Figure 3.5: Graph of the 2nd iteration of the Sequence-to-Sequence model

The model has also two separate outputs, one for the parameters (cepstrum, pitch, MVF) and one for the flags (U/V, End-Of-Sequence). This allows the model to train separately and use different losses for each parameter group.

The feedback connection allows the decoder to use the output from the previous timestep to better model the temporal evolution of the parameters. To build this connection, we take the outputs from the decoder, we concatenate them to the output from the encoder.

We found the large sequence lengths to be a possible cause of the poor results of the 1st iteration (see Section 4.2.1), so we chose to change the GRU cells for PLSTM cells. We used an implementation from Francesco Ferroni [15].

Sutskever et al. in [35] indicate that feeding a reversed version of the source sequence to the encoder improves the performance of the model. They argument that doing so "*introduces many short term dependencies in the data that make the optimization problem much easier*".

The Encoder and Decoder layers have 256 cells, the Parameters PLSTM has 42 cells, the Flags fully-connected layer has 2 units. Both the Merge layers in the graph of Figure 3.5 concatenate their inputs. The ReverseVector layer flips the output of the encoder, to undo the reversal of the input data. The Dropout layer has probability $p = 0.5$.

This model uses an MSE loss for both its outputs. We used an Adam optimizer, with a 0.001 learning rate. We trained this model for 50 epochs.

3.3.2.3 Pretraining of the second iteration

Given the poor results of the 2nd iteration (see Section 4.2.2), and taking into account the low amount of data available in the VCC dataset, we decided on pretraining the second iteration as an autoencoder.

The idea was to train the model with a part of the TC-STAR dataset (see Section 3.1.2), but not to convert voice, just to encode it and decode it. The rationale was to let the model learn how to process voice, so when training it as a voice converter, it would be able to handle speech suitably.

We made a few changes to the 2nd iteration. We changed the input size of the speaker index Embedding from 10-to-5 to 6-to-5, which was trivial, and we changed the process of loading the dataset. We also changed the encoder RNN for a Bidirectional RNN, which is able to model temporal evolutions in both forward and backward direction.

In addition, we changed the loss in the Flags output for a Binary Crossentropy loss, as this output performs a classification task. Due to the fact that the TC-STAR is a larger dataset than the VCC one, we trained this model for 20 epochs instead of 50.

In order to load the data from the TC-STAR dataset, we could not load the whole of it, since it would require too much memory. What we did was to load it in batches with the use of a **Generator** function. This function reads the batches from the dataset as the training requires them. It results in a higher number of disk reads, but keeps the required memory bounded.

Apart from changing the reading of the data, we also had to ensure that the model would actually learn how to process voice, not just copy it. To make sure of that, we introduced some “frame noise”. This frame noise consists in replicating the frames of each sequence with the same probability distribution the DTW algorithm did.

3.3.2.4 Attention Sequence-to-Sequence

This third and last iteration of the Seq2Seq model has been implemented with TensorFlow instead of Keras. The reason for this change is that TensorFlow included an implementation of a Seq2Seq model with Attention Mechanism in their 1.0 version. Another reason is that this implementation coming from a reliable programming team, it rids us of any programming mistakes we might have committed while implementing the previous models.

For this model, we have split the sequences of the TC-STAR dataset into chunks of 500 frames. Any chunk left that contains less than 500 frames is zero-padded. We have done this to ease the learning of the mapping between speech parameters.

This model, as the first iteration did, does not use speaker indexes. It has too been prepared to translate from one single speaker to another specific one, slightly simplifying the problem.

The encoder and decoder of the model use 512 LSTM cells each. The model has been trained with an Adam optimizer and an MSE loss. We have not used any Dropout layers in this model.

Below we present a summary table, listing the main features of each Seq2Seq iteration:

	Features
First iteration	Keras, Single Input-Output, One-to-One
Second iteration	Keras, Multi IO, Feedback, Many-to-many
Pretraining	Keras, Bidirectional RNN, Autoencoder
Attention Seq2Seq	TensorFlow, Attention Mechanism, One-to-One

Table 3.1: Summary table of the models of this project

Having presented all the models we have developed, in the following chapter we present the results we have obtained from each of them.

Chapter 4

Results

This chapter presents the results obtained with the techniques presented in Chapter 3 to improve the baseline system described in Section 3.3.1.

4.1 Baseline

Below are results of the most recent trainings of the Baseline system. We present objective metrics on the Test data and curves of the evolution of the losses.

We also present a problem we encountered when aligning the dataset with DTW.

4.1.1 Objective metrics

The following metrics have been computed in the process of predicting test data:

- Mel Cepstral Parameters Mean Cepstral Distortion (MCD): 8.25dB
- log-Pitch Root Mean Square Error (RMSE): 48.59
- Maximum Voiced Frequency RMSE: 2008.67
- Voiced/Unvoiced flags Accuracy: 90.7%

4.1.2 Loss curves

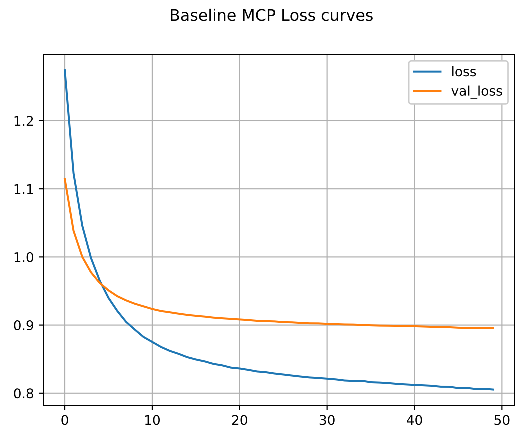
In figure 4.1 we present the loss curves of the last training of the Baseline.

Overall, this model gives intelligible results, although not being neither natural nor similar to the target speaker.

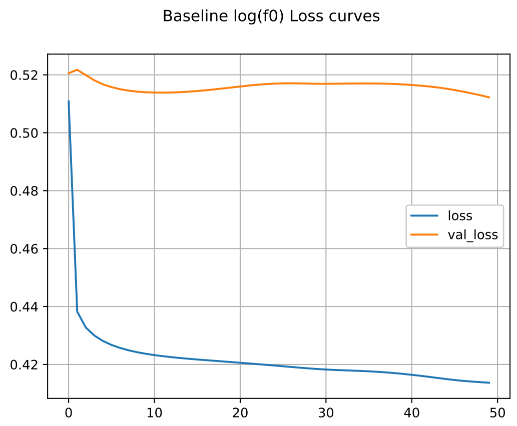
4.1.3 Data alignment

Initially we used the DTW implementation from SPTK. However, when we obtained the first predictions from the model we found that the results were far from acceptable. They showed mostly silent frames, with some spare ones containing what sounded like poor speech.

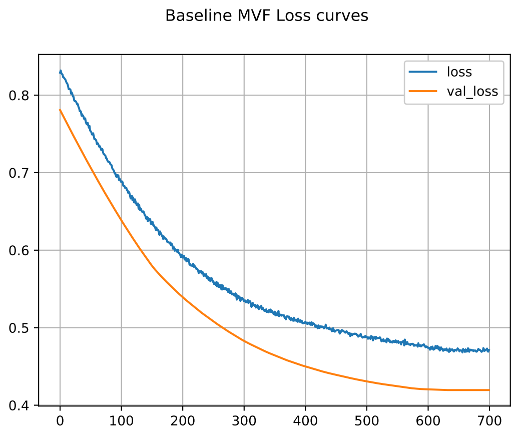
We investigated the source of this poor performance, and we found out that the implementation we were using did not perform well. We compared it to the implementation from



(a) MCP GRU-RNN (Section 3.3.1.1)

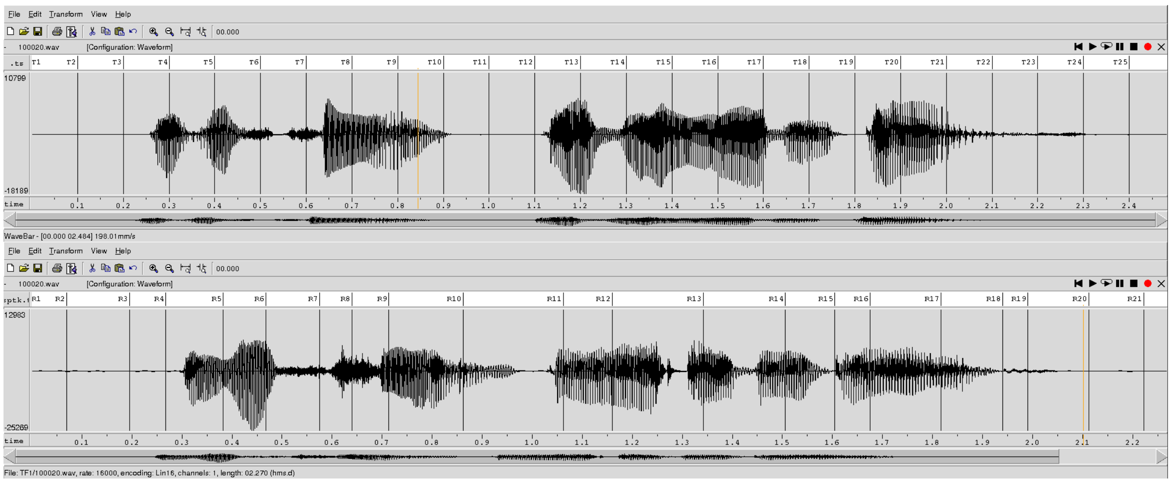


(b) $\log(f_0)$ LSTM-RNN (Section 3.3.1.2)

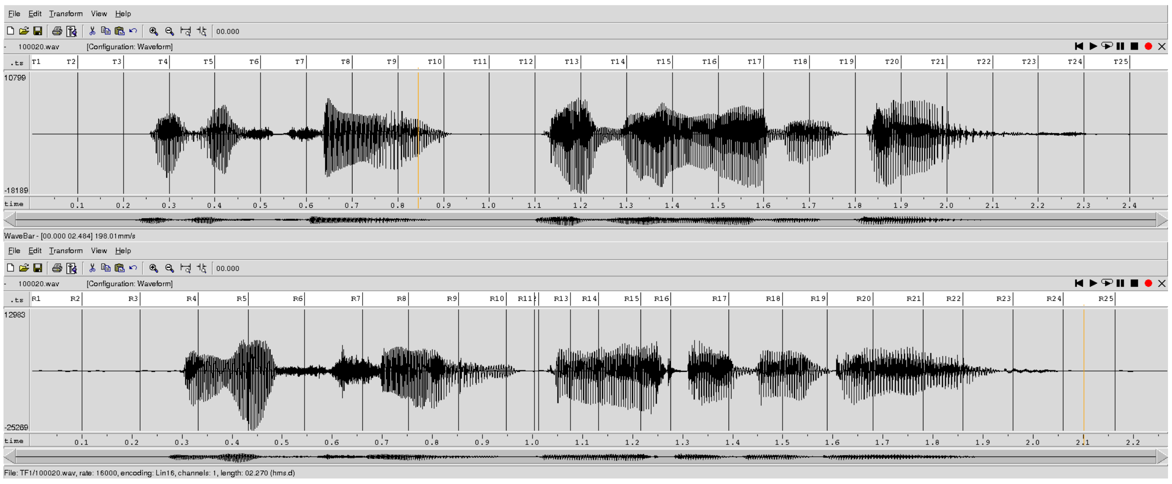


(c) MVF DNN (Section 3.3.1.3)

Figure 4.1: Loss curves of the training of the baseline model. There are graphs in which the validation curve is lower than the training curve. This is due to the fact that Keras evaluates the model with validation data after the weights have been updated.



(a) Alignment from SPTK. The boundaries are not coherent



(b) Alignment from UPC. The boundaries are coherent from source to target

Figure 4.2: Comparison of DTW alignments. The boundaries on the source signals are placed at each 0.1 seconds. The boundaries in target sequences are placed as projected by the alignment.

the VEU group and this comparison clearly showed the bad alignments from the SPTK DTW. From that point on, we used the DTW from the VEU group.

In figure 4.2 we show this comparison between DTWs. The screenshots in the figure are taken from WaveSurfer [25]. The top image in each subfigure shows a source file, and the bottom one, a target file. For the alignment to be correct, the vertical boundaries must be coherent from the source to the target file.

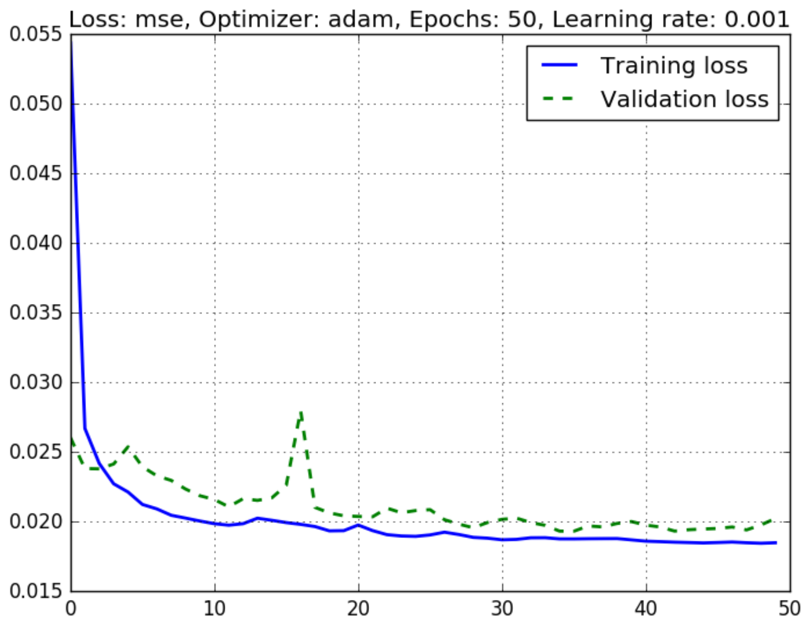


Figure 4.3: Loss curves of the training of the first Seq2Seq iteration (Section 3.3.2.1)

4.2 Sequence-to-Sequence

In this section we present the results of the trainings of the different iterations of the Sequence-to-Sequence model. We also present some objective metrics from the predictions of test data.

4.2.1 First Seq2Seq iteration

Figure 4.3 shows the loss curves of a training of the first iteration of the Seq2Seq model.

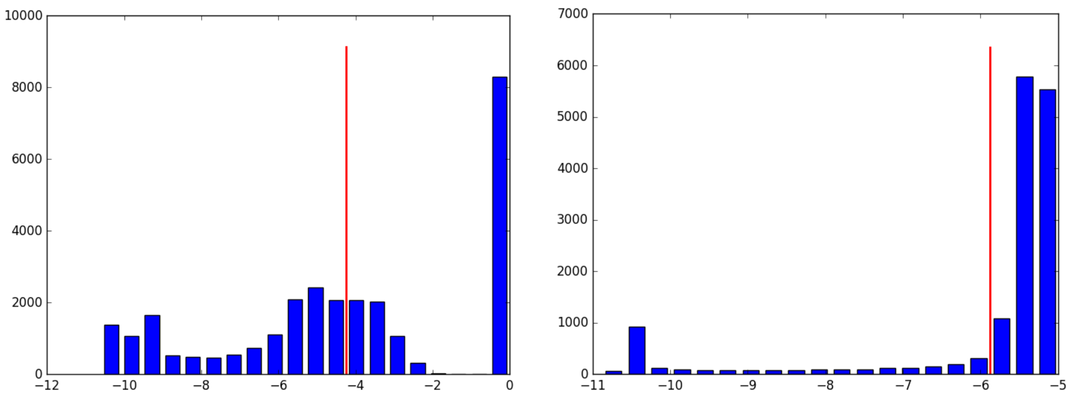
Figure 4.4 shows the histograms of the 0th cepstral parameter, both from ground truth data and prediction.

This model did not perform at all well. Although the loss curves look coherent, the predicted signals were silent, mainly due to low variability in the output signal and a bad prediction of the U/V flags, which caused the output signal to contain many frames marked as Unvoiced that were Voiced in the ground truth signals.

4.2.2 Second Seq2Seq iteration

Figure 4.5 depicts the loss curves of the second iteration.

The signals from this model sound extremely distorted, due to a high cepstrum distortion. To mitigate this distortion, we decided on pretraining this model (see Sections 3.3.2.3 and 4.2.3).



(a) Histogram of ground truth data (b) Histogram of predicted data

Figure 4.4: Histograms of predictions with the first seq2seq iteration. The red line indicates the mean of the distribution

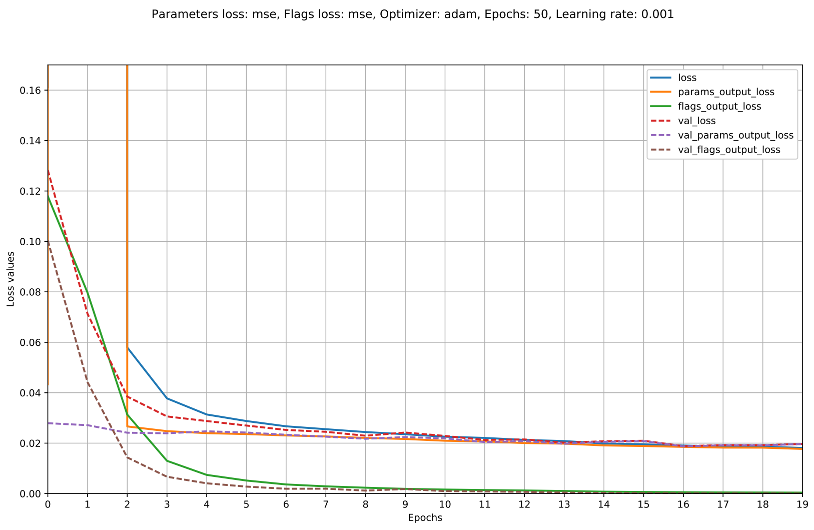
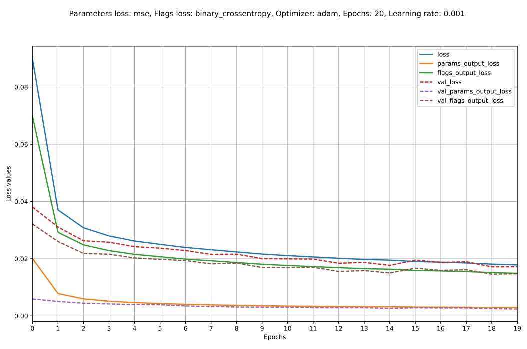
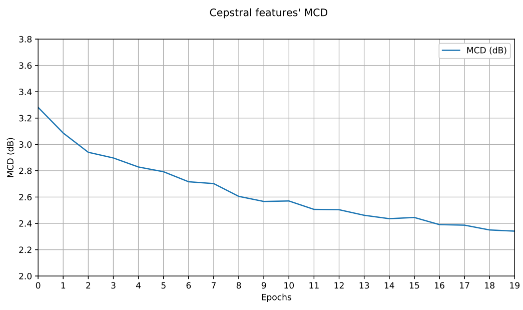


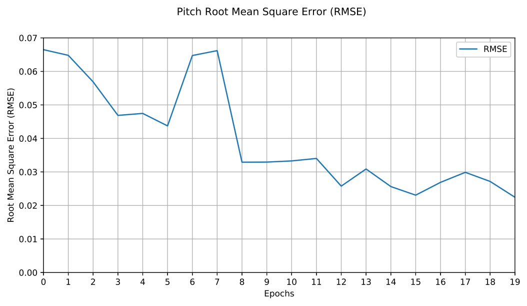
Figure 4.5: Loss curves of the training of the second iteration (Section 3.3.2.2)



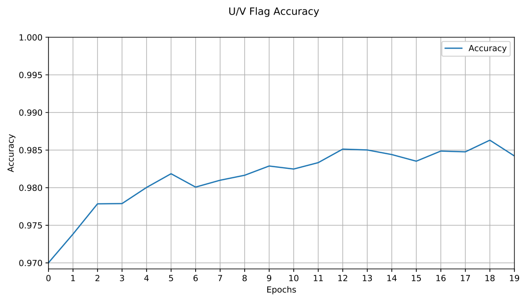
(a) Loss curves



(b) MCD curves



(c) RMSE curves



(d) Accuracy curves

Figure 4.6: Graphs of the pretraining of the second iteration (Section 3.3.2.3)

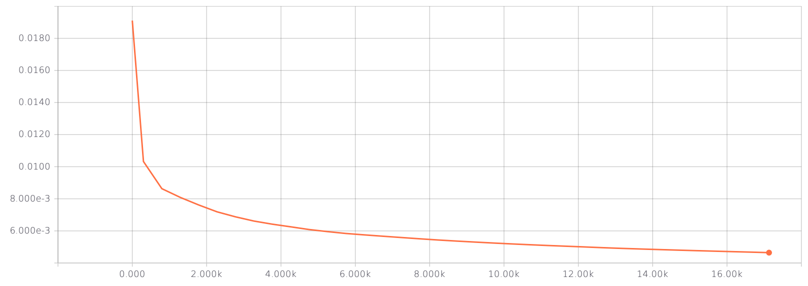
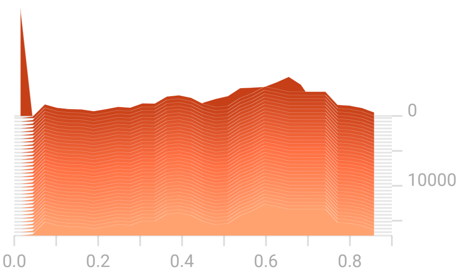


Figure 4.7: Training loss curve of Attention Seq2Seq model

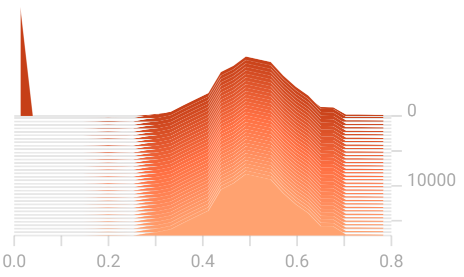
4.2.3 Pretraining of Second Seq2Seq iteration

The pretraining of the second iteration did not give satisfactory results. It resulted in almost silent Unvoiced frames, as it happened with the first iteration. The resulting training graphs are displayed in Figure 4.6

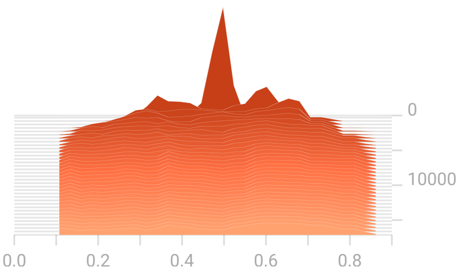
The reason for this bad results was that the feedback connection was not working. We tested this hypothesis by feeding ground truth (target) data into the feedback loop, instead of feeding the prediction from the previous timestep. When we did this, the model did output intelligible and fairly natural speech.



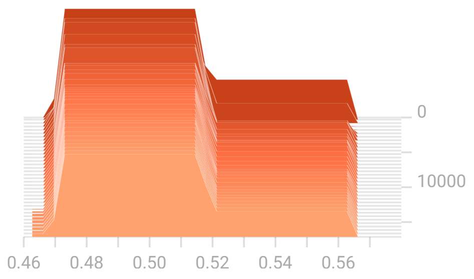
(a) Ground truth 0th cepstrum parameter



(b) Ground truth 39th cepstrum parameter

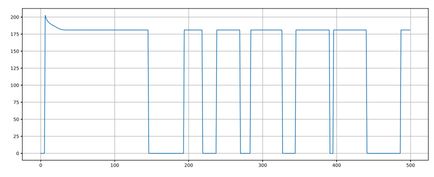


(c) Predicted 0th cepstrum parameter

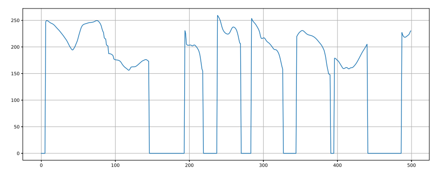


(d) Predicted 39th cepstrum parameter

Figure 4.8: Histograms of ground truth and decoder checkpoints over time from Attention Seq2Seq training



(a) Predicted data



(b) Target data

Figure 4.9: Pitch curves of a predicted and target sequence. The y axis' units are Hz.

4.2.4 Attention Sequence-to-Sequence

The main problem we encounter with this model is that it outputs signals with little to no variability i.e. they are almost constant. This effect can be seen in Figure 4.9. In this figure, we show the values of pitch (in linear scale) for each frame in a sequence. By using the U/V flags of the target data we have assigned null values to the unvoiced frames. Figure 4.9a shows the predicted signal, and Figure 4.9b, the target signal.

Chapter 5

Budget

This project has been developed using the resources provided by the VEU Group of UPC.

Thus, the main costs of this projects comes from the salary of the researches, the time spent in it and the server time used for the training of the system.

Although we have used the servers in the premises of the TSC Department, we have estimated their hourly cost by using the prices of Amazon AWS EC2 `p2.xlarge` servers [4].

We consider that my position is that of a junior engineer, while the Ph.D student and the professor who were supervising and advising me had a wage/hour of a senior engineer.

We will consider that the total duration of the project was of 34 weeks, as depicted in the Gantt diagram in Section 1.4.2.

	Amount	**Wage/hour**	**Dedication**	**Total**
Junior engineer	1	15.00 €/h	40 h/week	20,400 €
Senior engineer	1	40.00 €/h	4 h/week	5,440 €
Senior engineer	1	60.00 €/h	2 h/week	4,080 €
	Amount	**Hours**	**Cost/hour**	**Total**
Servers	1	720 h	0.99 €/h	712.8 €
			Total	30,632.8 €

Table 5.1: Estimated budget of the project

Chapter 6

Conclusions

In this project we have made a first approach at adapting a Sequence-to- Sequence model (originally proposed for the Automatic Translation task) for a Voice Conversion task. The reason for this approach was to make the alignment of the signals intrinsic to the model. We have used multiple libraries and frameworks. We have also implemented several data processing functions which can be re-used on later projects.

There have been signs that the system is actually learning and the trainings have not diverged. Despite our efforts, the system we have developed does not perform well. It does not output speech signals.

We have two hypothesis explaining the bad results of our system. The first one points the encoder as the culprit of the poor performance. We think it could be a reason, since we have checked that the decoder can output speech signal. This was tested during the pretraining, when we fed ground truth data into the decoder and the results were good (see Section 3.3.2.3).

The second hypothesis argues that the attention mechanism's performance falls short in what is expected of it. It is possible that this mechanism is not enough for aligning the source and target data when there is big misalignments between the sequences. In recent work, Wang et al. mention that the attention mechanism "*tends to get stuck for many frames before moving forward*" [45]. This could explain the Attention Mechanism not aligning properly.

In future developments, we intend to investigate the performance of the Attention Mechanism by extracting its alignment masks. In [45, fig. 3] there are examples of these graphs.

Another task to be approached in future work is to look for an efficient method for encoding long sequences, because we have had to split our dataset sequences into short chunks, and this can cause problems.

Future work can also be done in the Generative Adversarial Networks field, following the footsteps of Hsu et al. [21].

We have published the code from this project, as a contribution to the scientific community. The code can be found at `github.com/albertaparicio/tfg-voice-conversion`, and its support libraries in `github.com/albertaparicio/tfglib`.

Both projects are published under free copyleft licenses. The main project is licensed under the **GNU GPL v3+** and the support libraries, under the **GNU Lesser GPL v3+**.

Bibliography

[1] DAPS (Device and Produced Speech) Dataset - A dataset of professional production quality speech and corresponding aligned speech recorded on common consumer devices.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Masanobu Abe, Satoshi Nakamura, Kiyohiro Shikano, and Hisao Kuwabara. Voice conversion through vector quantization. *Journal of the Acoustical Society of Japan (E)*, 11(2):71–76, 1990.

[4] Amazon Web Services. AWS | Amazon EC2 Dedicated Instances - Dedicated Hosting. https://aws.amazon.com/ec2/purchasing-options/dedicated-instances/. Accessed on 4-5-2017.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the Third International Conference on Learning Representations (ICLR 2015)*, San Diego, USA, September 2014. arXiv: 1409.0473.

[6] Christopher M. Bishop and Geoffrey Hinton. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, New York, November 1995.

[7] Ling-Hui Chen, Li-Juan Liu, Zhen-Hua Ling, Yuan Jiang, and Li-Rong Dai. The USTC System for Voice Conversion Challenge 2016: Neural Network Based Approaches for Spectrum, Aperiodicity and F0 Conversion. In *Interspeech 2016*, pages 1642–1646, San Francisco, USA, September 2016.

[8] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, June 2014. arXiv: 1406.1078.

[9] François Chollet. Keras. https://github.com/fchollet/keras, 2015.

[10] Mark K. Cowan. neuralnetwork. http://www.ctan.org/pkg/neuralnetwork, July 2013.

[11] Li Deng and Dong Yu. Deep Learning: Methods and Applications. Tech Report MSR-TR-2014-21, NOW Publishers, Boston - Delft, May 2014.

[12] Daniel Erro, Iñaki Sainz, Eva Navas, and Inma Hernáez. Improved HNM-Based Vocoder for Statistical Synthesizers. In *Interspeech 2011*, pages 1809–1812, Florence, Italy, August 2011.

[13] Daniel Erro Eslava. *Intra-Lingual and Cross-Lingual Voice Conversion Using Harmonic Plus Stochastic Models*. Ph.D. Thesis, Universitat Polit??cnica de Catalunya, Barcelona, 2008.

[14] Daniel Erro Eslava. Ahocoder download - info. http://aholab.ehu.es/ahocoder/info.html, February 2017.

[15] Francesco Ferroni. PhasedLSTM-Keras. https://github.com/albertaparicio/PhasedLSTM-Keras, 2016.

[16] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. CSC321 - Introduction to Neural Networks and Machine Learning. Accessed 4-5-2017, January 2014.

[17] A. Graves, A. r Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[18] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*, July 2012. arXiv: 1207.0580.

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

[20] Chin-Cheng Hsu, Hsin-Te Hwang, Yi-Chiao Wu, Yu Tsao, and Hsin-Min Wang. Voice Conversion from Non-parallel Corpora Using Variational Auto-encoder. *arXiv:1610.04019 [cs, stat]*, October 2016. arXiv: 1610.04019.

[21] Chin-Cheng Hsu, Hsin-Te Hwang, Yi-Chiao Wu, Yu Tsao, and Hsin-Min Wang. Voice Conversion from Unaligned Corpora using Variational Autoencoding Wasserstein Generative Adversarial Networks. *arXiv:1704.00849 [cs]*, April 2017. arXiv: 1704.00849.

[22] A. Kain and M. W. Macon. Spectral voice conversion for text-to-speech synthesis. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, 1998*, volume 1, pages 285–288 vol.1, May 1998.

[23] Andrej Karpathy. CS231n Winter 2016: Lecture 4: Backpropagation, Neural Networks 1. https://www.youtube.com/watch?v=i94OvYb6noo, January 2016. Accessed 4-5-2017.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. arXiv: 1412.6980.

[25] Kåre Sjölander and Jonas Beskow. WaveSurfer. http://www.speech.kth.se/wavesurfer/, December 2011.

[26] Z. H. Ling, S. Y. Kang, H. Zen, A. Senior, M. Schuster, X. J. Qian, H. M. Meng, and L. Deng. Deep Learning for Acoustic Modeling in Parametric Speech Generation: A systematic review of existing techniques and future trends. *IEEE Signal Processing Magazine*, 32(3):35–52, May 2015.

[27] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[28] S. H. Mohammadi and A. Kain. Voice conversion using deep neural networks with speaker-independent pre-training. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 19–23, December 2014.

[29] M. Narendranath, Hema A. Murthy, S. Rajendran, and B. Yegnanarayana. Transformation of formants for voice conversion using artificial neural networks. *Speech Communication*, 16(2):207–216, February 1995.

[30] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences. *arXiv:1610.09513 [cs]*, October 2016. arXiv: 1610.09513.

[31] Christopher Olah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 2015. Accessed: 2017-04-28.

[32] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[33] Daisuke Saito, Fernando Villavicencio, Junichi Yamagishi, Toda Tomoki, Mirjam Wester, Zhizheng Wu, and Ling-Hui Chen. The Voice Conversion Challenge 2016. http://dx.doi.org/10.7488/ds/1575, December 2016.

[34] Y. Stylianou, O. Cappe, and E. Moulines. Continuous probabilistic transform for voice conversion. *IEEE Transactions on Speech and Audio Processing*, 6(2):131–142, March 1998.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[36] TC-Star Project. ELRA - ELRA-S0251 : TC-STAR English Training Corpora for ASR: Recordings of EPPS Speech. http://catalog.elra.info/product_info.php?products_id=1034.

[37] TensorFlow Authors. tf.train.AdamOptimizer. https://www.tensorflow.org/versions/r1.1/api_docs/python/tf/train/AdamOptimizer, 2017.

[38] TensorFlow Authors. tf.train.RMSPropOptimizer. https://www.tensorflow.org/versions/r1.1/api_docs/python/tf/train/RMSPropOptimizer, 2017.

[39] T. Toda, A. W. Black, and K. Tokuda. Voice Conversion Based on Maximum-Likelihood Estimation of Spectral Parameter Trajectory. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(8):2222–2235, November 2007.

[40] T. Toda, H. Saruwatari, and K. Shikano. Voice conversion algorithm based on Gaussian mixture model with dynamic frequency warping of STRAIGHT spectrum. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 2, pages 841–844 vol.2, 2001.

[41] Tomoki Toda, Ling-Hui Chen, Daisuke Saito, Fernando Villavicencio, Mirjam Wester, Zhizheng Wu, and Junichi Yamagishi. The voice conversion challenge 2016. In *Interspeech 2016*, pages 1632–1636, 2016.

[42] Keiichi Tokuda, Keiichiro Oura, Takenori Yoshimura, Akira Tamamori, Shinji Sako, Heiga Zen, Takashi Nose, Toru Takahashi, Junichi Yamagishi, and Yoshihiko Nankaku. Speech Signal Processing Toolkit (SPTK), December 2016.

[43] Universitat Politècnica de Catalunya. TALP - TALP - UPC Research Center. http://www.talp.upc.edu/. Accessed on 4-5-2017.

[44] Josep Vidal Manzano. Lecture notes of Pattern Classification (ETSETB - 230202). 2016.

[45] Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: Towards End-to-End Speech Synthesis. *arXiv:1703.10135 [cs]*, March 2017. arXiv: 1703.10135.

[46] Mirjam Wester, Zhizheng Wu, and Junichi Yamagishi. Analysis of the voice conversion challenge 2016 evaluation results. In *Interspeech 2016*, pages 1637–1641, 2016.

[47] Xiaohai Tian, Siu Wa Lee, Zhizheng Wu, Eng Siong Chng, and Haizhou Li. An Exemplar-based Approach to Frequency Warping for Voice Conversion. *IEEE Transactions on Audio, Speech and Language Processing 2016.*