MFCC to wave conversion using WaveRNN

Alessandro Iucci, Florent Monin and Marie Sarbiewski

Project report for DT2112
Date: March 14, 2020
Supervisor: Jens Edlund, Éva Székely and Jonas Beskow
Examiner: Jens Edlund
School of Electrical Engineering and Computer Science

Abstract

The WaveRNN architecture has been found useful for Speech processing. For now, it has been used in complement of a Tacotron that generated Mel- spectrogram from text data, and then the WaveRNN generated the waveform from there. This study aims at implementing a vocoder with the help of WaveRNN, which means that only Mel Frequencies Cepstral Coefficients (it will be denoted as MFCC after this mention) should be fed to the WaveRNN, and then the Neural Network should be able to reconstruct the original voice from the MFCC.

Contents

Chapter 1

Introduction and motivation

Sequential speech generation methods have accomplished peak performance in various domains of signal processing, such as natural language processing, natural images processing or speech synthesis. However, they require a lot of computing time, and suffer from overfitting. This is the reason why the WaveRNN was invented. This is a Recurrent Neural Network (it will be denoted RNN after this mention) especially adapted to speech synthesis. Usually, it is combined with a Tacotron, in order to be able to generate text from speech (TTS generation).

One could ask themselves the question: could it be possible to use the WaveRNN alone, and then be able to use it as a vocoder? This is the question addressed in this study. A WaveRNN has been implemented, and used as a standalone component. This report describes in details the procedure used from data collection to the final discussion on the results.

Chapter 2

Background

2.1 A vocoder

The first vocoder was invented and realised by an engineer from Bells labs named Homer Dudley. His project was rather simple: it was only analog circuits, and no machine learning at all was involved (description of the process in [1]). Many vocoders were produced based on this architecture which used analog circuits able to approximately reproduce the human voice.

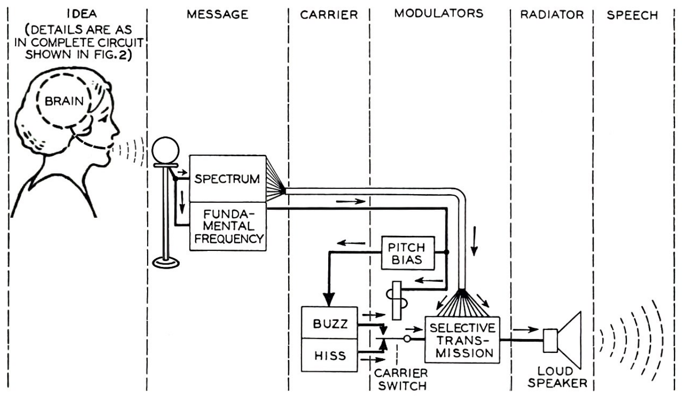


Fig. 7—Schematic circuit of the vocoder.

Figure 2.1: Scheme of the first Bell's vocoder

However, the results of the these kind of vocoders could never be perfect and a human ear could easily make the difference between a real human voice and a voice reproduced using a vocoder. Through time, different techniques were used to improve the result with the aim of making it so accurate that it could not be distinguished from a real human voice.

Indeed, the uses of vocoders are many and varied. The first use that comes to mind is of course sound compression and sound storage: if one can store only lighter encrypted files and then decode them easily as a sound later, one can store a lot of sound files in a small place. However, the uses can be more sophisticated, with for example sound encryption, messaging systems, and many other examples.

Nowadays, all the state-of-the-art vocoders are implemented with the help of deep neural networks, and even more precisely, most of them are Recurrent Neural Networks or Generative Adversarial Networks (they are then called Adversarial Vocoders, see [2]).

2.2 WaveRNN

WaveRNN is a special type of RNN. Indeed, usual recurrent networks are deep architectures that allow memory storage and time dependencies. The representation of that kind of network is made by a directed graph along a temporal sequence. They are often used in a sequential context such as image analysis, or text analysis, because they have the ability to store some results in a memory, so that a correlation between two inputs that are next to each other can be found.

The main objective of the WaveRNN is to provide the same advantages than classical RNN but requiring only a small number of operations at each step. Indeed, the classical architectures used in this field (Convolutional sequence models, see [3]) already have good results, but tends to have a relative deep architecture that needs a lot of training time and a lot of computation, due to the large number of layers.

The main principle of the WaveRNN is the following (see Figure 2.2) : we split a RNN in two parts :

- one for the 8 most significant bits of the 16-bits audios, named the **coarse**,
- one for the 8 least significant bits named the **fine**.

Each part feeds into a **softmax** layer and the prediction of the 8 fine bits is conditioned on the 8 coarse bits, resulting in a **Dual Softmax Layer**.

Finally, there is an output layer that is able to make an efficient prediction for 16-bits samples, using two small output spaces of size 2^8 (so the output space size is $2^9 = 512$) instead of one output space of size $2^{16} = 65536$. This method reduces significantly the amount of time needed in the training phase, as much as the amount of memory needed on the GPU for the training. To get more details and clarifications on the WaveRNN, see [4].

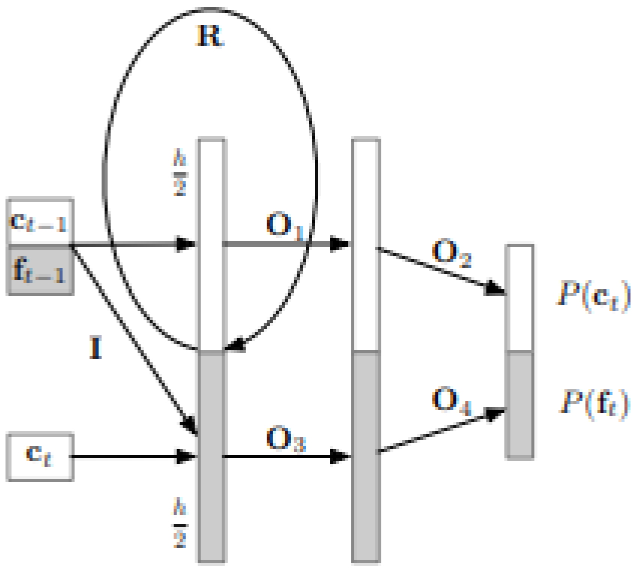


Figure 2.2: The architecture of the WaveRNN with the dual softmax layer. c represents the **coarse** of the sample and f represents the **fine** of the sample. The multiplication by R (matrix of the RNN) happens for both the coarse and fine bits simultaneously, then output of the gates is evaluated for the coarse bits only and c_t is sampled. O_1 and O_3 are the transition matrices from the RNN to the softmax layer, and O_2 and O_4 are the transition matrices from the softmax layers to the dual softmax layer.

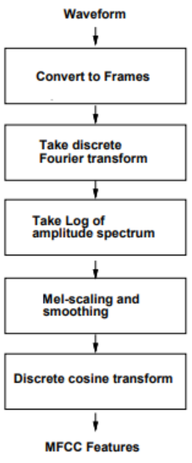
2.3 Mel Frequencies Cepstral Coefficients (MFCC)

The MFCCs are the result of a particular transformation applied to the waveform. Indeed, the waveform in its raw form can not be used like this: there would be a huge amount of coefficients to compute in the corresponding deep architecture. The principle of the MFCCs is quite similar to the Fourier transform : one has to convert the waveform into frames, take the discrete Fourier transform and then to take the $\log$ of the amplitude spectrum. After that, a Mel-scaling and smoothing is necessary, and a discrete cosine transform ends the process. So the final expression of the MFCCs is :

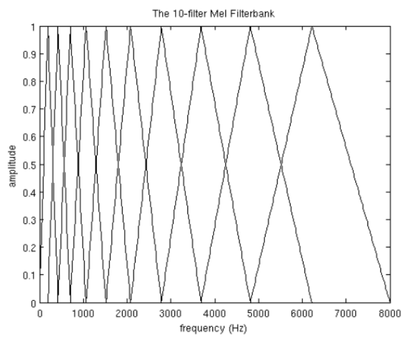
$$\left|\mathcal{F}^{-1}\left\{\log\left(|\mathcal{F}\{f(t)\}|^2\right)\right\}\right|^2$$

where $\mathcal{F}$ is the Fourier transform and $f(t)$ is the original signal on wave form.

MFCC are one of the dominant features used for speech recognition. With only few coefficients (generally 13, as in this paper), one is capable of modelling a speech element quite accurately.


Waveform
Convert to Frames
Take discrete Fourier transform
Take Log of amplitude spectrum
Mel-scaling and smoothing
Discrete cosine transform
MFCC Features

(a) Detailed computation of the MFCC, according to [5]



(b) Windowing for the frame conversion

Figure 2.3: Definition of the MFCCs

Chapter 3

Methods

In this section we will illustrate which data we used to train our neural network inspired to the WaveRNN framework and how we managed to train it and get the desired output.

3.1 Dataset

The dataset we used is LJSpeech [6] which is a public domain speech dataset consisting of 13.100 short audio clips of a single speaker reading passages from 7 non-fiction books. A transcription of each audio is provided for each clip and the clips vary in length from 1 to 10 seconds and have a total length of approximately 24 hours. Each audio file is a single-channel 16-bit PCM WAV with a sample rate of 22050 Hz. The audios are all of a equal and good quality, of the same women reading text.

3.2 Training phase

The implementation of the WaveRNN-based project [7] we used had a structure composed of two different neural networks combined to have as input a text and as output the waveform of the corresponding text as shown in 3.1
The two networks are:

- A Tacotron which created a Mel spectrogram from the corresponding input text
- The actual WaveRNN which reproduced the waveform starting from the Mel spectrogram created by the Tacotron

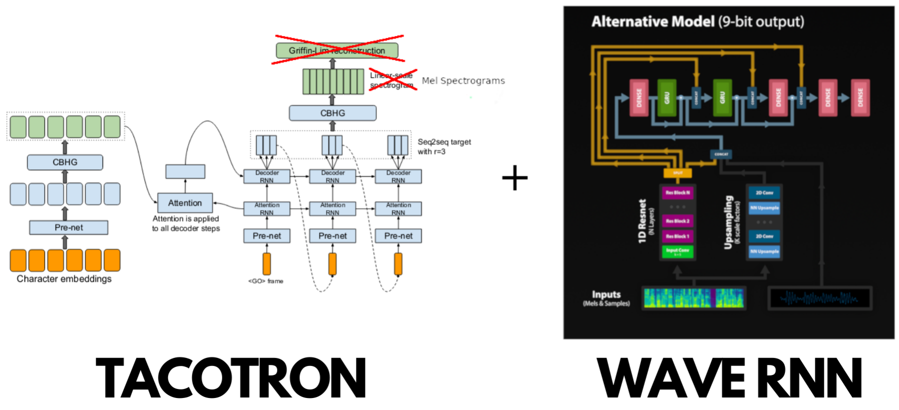


Figure 3.1: WaveRNN + Tacotron structure

and together they created a TTS (text-to-speech) network.
However our focus was to reproduce the waveform and corresponding audio starting from the most relevant MFCC instead of having as input the text. This is the reason why in our project we unlinked the two networks and only used a modified version of the WaveRNN so that we could have as input the actual MFCC instead of the Mel spectrogram.
Our first attempt was extracting from the audio files the Mel spectrogram using the tools included in the `utils.dsp` library and training the network with it (see our GitHub [8]). The input of the WaveRNN in the training phase was made by two components:

- The Mel spectrogram which directly goes through an up-sampling process and at the same time through a 1D Resnet
- The actual audio waveform expected as output which is combined with the first component only after the Mel spectrogram was pre-processed as in the previous point

Once we made sure that the Mel spectrogram generated with the library tools and given as input to the WaveRNN performed good both in terms of training and generation phase, we proceeded with the next step which was changing the input, using the most relevant MFCC instead of the whole Mel spectrogram.
We chose to use 13 most relevant coefficient because, after doing some research on which was a good estimate of the needed number of coefficients, we found out 13 is a good amount to make sure to have most of the information preserved.
The biggest issue we had to face while coding this input changes was to have the new inputs to match the expected input size without changing the characteristics of

the audio, so that we could feed it to the neural network. This turned out to be particularly challenging also because the code was not of easy understandable and it was not commented, so even with a carefully coding inspection, the slightest change turned out to be particularly difficult to implement.
To achieve this, we first had to generate the MFCC with windowing parameters that were sensible - a window that is sized small enough to capture the information of only a vowel, or a consonant, but long enough that it wouldn't create too much data. The parameters were also chosen so that the ratio between the length of this input and the length of the output was, in average as close to an integer as possible. Indeed, the network can only multiply the temporal size of the data by an integer, through upsampling. We ended up using windows of 10ms, and shift between windows of 5.3ms. This lead to a ratio of almost $1 : 117$ between the MFCC and the original waveform. From there, we were able to modify the upsampling network to suit our needs. From there, it was easy to modify the 1D Resnet to also give us a fitting shape for its output (see [8]).
This process was successful for the first two attempts getting us good results as explained in the results chapter, but we did not get anything satisfying when adding the pitch as explained in the pitch section

3.3 Generation phase

The generation phase was carried on every 10 epochs to see the evolution of the resulting output and we obviously used a test set of audios from which we extracted the Mel spectrogram and the 13 most relevant MFCCs. This time the part of the input representing the waveform which we used in the training process to speed up the network learning, was all set to zeros because the objective of the network itself is to recreate that waveform, so we are not supposed to have it in advance. On the other hand, the initial waveform was used in the evaluation process to check the performance and loss of our model.

3.4 Pitch

We then decided to combine the MFCCs with the information on the pitch of the waveform to compare the performances, as we were expecting the convergence to speed up dramatically since the information about the pitch is quite precious and hard to learn for a neural network with only the spectrogram as input.
Therefore, we extracted the pitch using the tool REAPER [9] and we tried to substitute to the less relevant MFCC this new information (see [8]). Unfortunately, once

again we had several issues with the size of the generated pitch vector, so to feed it to the network and combine with MFCCs we had to upsample the pitch vector.
The training and generation phases were carried on exactly as specified previously, but changing the input from 13 MFCCs to 12 MFCCs + pitch.
Unfortunately the result was not as expected and it was probably due to a wrong adaptation of the network, as further discusses in the discussion chapter.

Chapter 4

Results

4.1 Spectrogram as an input

We first trained the original WaveRNN network using the LJSpeech dataset. To see the convergence of the network, we computed the Mean Squared Error over the spectrogram of the input and of the generated output. This allows to quantify the quality of the audio in an objective way. We can see in Figure 4.1 this convergence.

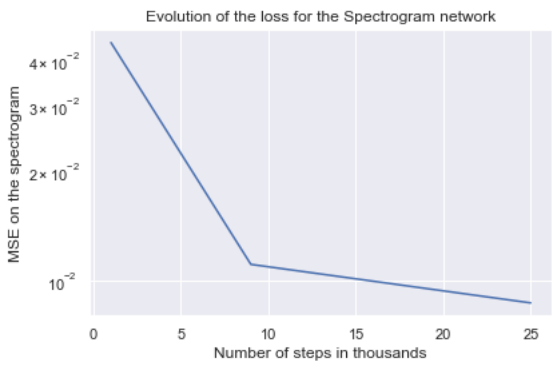


Figure 4.1: Error of the network during the training

4.2 MFCC as an input

We then trained a modified version of the network, to take as an input 13 MFCC components of the original waveform.

The Figure 4.2 shows the convergence of the network, and how the error lowers after a serveral thousand steps.

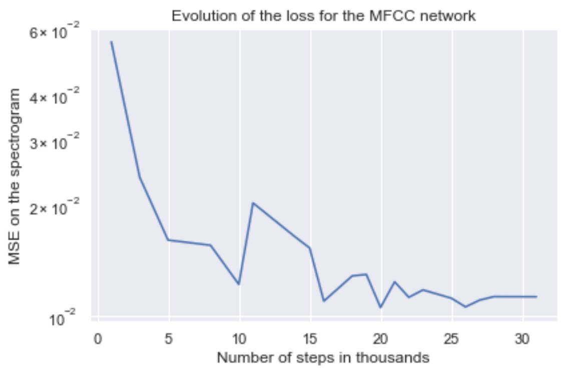


Figure 4.2: Error of the network during the training

4.3 MFCC+Pitch as an input

We then trained a modified version of the previous network, to take as an input 12 MFCC components of the original waveform, and the pitch. The Figure 4.3 shows

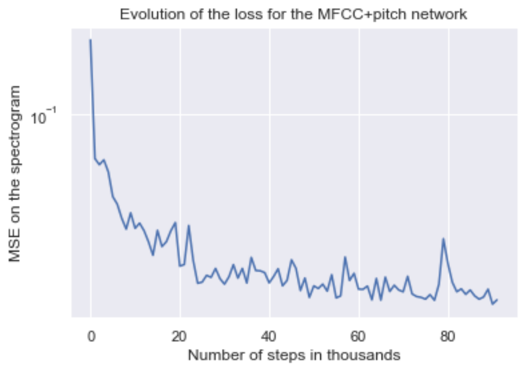


Figure 4.3: Error of the network during the training

the convergence of the network, and how the error lowers after a serveral thousand steps. All the generated audios can be found on the Soundcloud [10].

Chapter 5

Discussion

5.1 Network convergence

To see the convergence of the network, we first tried to compute the Mean Squared Error (MSE) between the input and the output directly, but this loss didn't show any evolution at all. We concluded that it wasn't a good metric for the quality of the audio. We then tried to compute the MSE between the spectrograms of the audios. This proved to be much more efficient at showing the improvement of the quality that we could subjectively hear in the audios.
The three figures 4.1, 4.2 and 4.3 show a convergence of the network. However, when listening to the outputs of the network, one can clearly hear a difference in quality between the output of the MFCC+Pitch network and the MFCC network. This is partly shown by the value of the loss, which is lower on the MFCC network than on the MFCC+Pitch network. This indicates that the loss alone isn't a good indicator of the audio quality, but it is still able to show the evolution of the output. To further evaluate the output, it would be needed to have a human evaluation of the quality of the output.

5.2 Differences between the networks

The first network, trained with the full Mel Spectrogram was our test quality, as we knew thanks to fatchord [7] that it would produce an output of a great quality. With similar amount of training, the MFCC network performed surprisingly well, and that can be seen both in the audio recordings [10], and on the value of the error on the figure 4.2, compared to the error on the figure 4.1.
However, we expected the MFCC+Pitch network to perform better than it did. Even after extensive training (80k steps), the error did not reach the same amount as the

MFCC network. It seems that the Pitch conveys more information that the thirteenth cepstral coefficient. This poor result could be explained by the implementation of the network. Indeed, in order to add the pitch, which is a continuous component, to the MFCC, that are sample every so often, on a window of the signal, we had to add the pitch after the upsampling of the MFCC. That means that the auxiliary input that are fed through the network after the upsampling do not contain the information of the pitch. This might influence the network, and prevent it from correctly converging. To address this issue, one would need to further modify the structure of the WaveRNN [7], which is out of the scope of this project.

5.3 Generalizability of the results

As explained in the methods chapter, the audios all came from one dataset, composed with the voice of only one woman, speaking in a controlled environment. Hence, we could expect the loss, which even though computed on a testing part of the dataset, to be biased.
Indeed, the training dataset is still very similar to the dataset used to train the network. It would be interesting to see how well the networks perform on other audios, that come from different situations.
We performed a few tests on the MFCC network. This showed us however, that it would seem that the network is, although less performing, still able to produce a sensible output on very different data. Of course, the quality of the audio influences a lot on the quality of the output, as well as the language used. From our tests, it seems that only English is faithfully reconstructed.
Yet, Swedish recordings that were reconstructed were impressively understandable. A potential factor of the apparent success of the MFCC network, is the fact that the network only uses 13 coefficients. This type of feature extraction should limit overfitting, as it should only give to the network essential information. Hence, one could hope that the differences in quality in different audios would not affect too much the MFCC. This should help the generalization capacity of the model.

Chapter 6

Conclusions and further development

The project as we planned it was highly successful and the amazing results we got with the training of the network using only the 13 most relevant MFCC testify it. Both loss comparison and human evaluation of the generated audio files gave us results far beyond our expectations. This gives us plenty of opportunities for further exploitation and study of this vocoder.
Unfortunately, due to lack of time, we couldn't go deep in the study of the combination of MFCC with the pitch and fix what was wrong in our attempt of making it work, but it would surely be interesting to keep working on this task.
Another interesting approach would be seeing what happens with a modification of the pitch or of the MFCC and feeding the modified input through the vocoder.
In general, the utilities of this tool are multiple and varied and it would be interesting to apply our network in real applications and testing its behaviour in those situations.

Bibliography

[1] Mara Mills. **Media and Prosthesis: the Vocoder, the Artificial Larynx, and the History of Signal Processing**. Accessible at : `https://muse.jhu.edu/article/491050`. Accessed : 2020-03-03, pp. 107–109.

[2] Paarth Neekhara et al. “Expediting TTS Synthesis with Adversarial Vocoding”. In: (2019). `https://arxiv.org/abs/1904.07944`. Accessed : 2020-03-06.

[3] Kalchbrenner N. et al. “Neural machine translation in linear time”. In: (2016).

[4] Nal Kalchbrenner et al. “Efficient Neural Audio Synthesis”. In: **CoRR** abs/1802.08435 (2018). arXiv: `1802.08435`. URL: `http://arxiv.org/abs/1802.08435`.

[5] Beth Logan. “Mel Frequency Cepstral Coefficients for Music Modeling”. In: **Cambridge Research Laboratory** (). URL: `%5Curl%7Bhttps://www.academia.edu/download/30390942/musicir_paper.pdf%7D`.

[6] **LJ Speech Database**. `https://keithito.com/LJ-Speech-Dataset/`. Accessed: 2020-02-23.

[7] fatchord. **WaveRNN Vocoder + TTS**. `https://github.com/fatchord/WaveRNN`. 2019.

[8] Alessandro Iucci, Florent Monin, and Marie Sarbiewski. **speech-technology-group11**. `https://github.com/jucchee/speech-technology-group11`. 2020.

[9] google. **REAPER**. `https://github.com/google/REAPER`. 2019.

[10] **Waveforms generated by the different networks, as well as the original targets**. `https://soundcloud.com/speech-technology`. 2020.