ON DEEP LEARNING METHODS FOR SPEECH SYNTHESIS APPLICATIONS

CHAN KIN LOK

MPhil

The Hong Kong Polytechnic University

2023

The Hong Kong Polytechnic University

Department of Applied Mathematics

On Deep Learning Methods for Speech Synthesis Applications

CHAN Kin Lok

A thesis submitted in partial fulfilment of

the requirements for the degree of

Master of Philosophy

August 2022

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

CHAN Kin Lok

Abstract

Voice Cloning is a speech processing task that aims to synthesize speech with a specific target's voice. There is a resemblant topic named Voice Conversion in the field. The difference is that, while Voice Conversion techniques process existing audio data, Voice Cloning newly synthesizes speech from text. In this thesis, a popular open-source deep-learning-based Voice Cloning model is introduced. The structure of the neural network layers is studied and supporting literature is reviewed.

The objective of this project is twofold. First, we want to optimize the open-sourced model to boost its performance, especially in low-resources cases in which only a limited amount of data is available. The methods studied in this thesis are to optimize hyperparameters of the speech synthesis process and to finetune the model using a small dataset of target speakers. Improvement in speech quality and voice similarity is observed.

Another objective is to develop potential applications of Voice Cloning techniques. In this project, we investigate and propose an application in educational usage, that we can detect pronunciation errors by comparing speech data from real

humans and synthesized speech. Existing methods in field may require either professional language knowledge or numerous examples recorded from real humans. Our proposed method employed a TTS model to generate reference speech so that these are no longer necessary. In addition, applying Voice Cloning techniques could simplify the comparison procedure between teachers' and students' speech data.

Acknowledgements

Countless people supported me in completing this thesis. I wish to express my sincere gratitude to my supervisor Prof. Cedric Yiu for patiently answering my questions, fruitful discussion sections, and valuable feedback. I would also like to thank the teachers of the subjects I took in my study period for providing knowledge and expertise, and the fellow in university for technical supports. Finally, I shall thank my family for supporting my daily live, so that I could concentrate on my studies.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Text-to-Speech Synthesis

Text-to-speech (TTS) synthesis is a research topic with a long history. It aims to generate corresponding speech data according to input text. This "input text" is often natural language, but it could also be any other type of symbolic linguistic representation such as phonetic transcriptions. In implementations nowadays, it is often required that TTS models should be "end-to-end", which means both input and output of the models should be natural languages that are understandable to humans. It is preferred that all linguistic and mathematical processes are handled by models, no specific knowledge should be required from users.

A traditional technique of TTS is concatenative method, which synthesizes speech by concatenating small pieces of speech data in a database. Data stored in the database may differ in different models: some of them may be phonemes, others

may be entire words or phrases. This approach could generate high-quality speech in terms of naturalness and has long been state-of-the-art. However, its limitations also come from its database: pronunciation, accent, and intonation of synthesized speech are limited by the samples in the database.

Statistical parametric approaches were developed in the 1990s. The idea of this approach could be described in three stages. The first stage is to extract features from the input text sequence. That means building a model for computer vision so that the TTS model could understand human languages in text format. Natural Language Processing (NLP) skills may be applied here, but machine learning methods are also useful. The second stage is a statistical module that predicts acoustic features from linguistic features. Acoustic features are representations of audio signals but are easier to process. More details about acoustic features would be presented in Section 2.2. The last stage is constructing audio waveforms from acoustic features. Models for this purpose are called "vocoders". It is common for speech processing models to handle acoustic features. Vocoder itself could be a wide research topic that benefits many other speech processing fields. Hidden Markov Model-based frameworks are popular among parametric methods and have long been used until deep learning-based models become popular.

Since the 2010s, deep learning techniques have been applied to speech synthesis field. Many neural network-based methods are developed and become one of the main streams nowadays. Deep neural network (DNN) models are proven to have high non-linearity and could perform well in speech processing tasks including speech synthesis. Many attempts have been done since mid-2010s [52], [1], [44]. One of the popular models is Tacotron [53] and its improved version, Tacotron2 [42].

The advantage of this model is that this model could accept characters as input and produce a spectrogram as output. Thus, users are not required to have any linguistic knowledge nor use a specific vocoder for waveform construction. Details of Tacotron and vocoder would be given in Section 2.3.

1.2 Voice Cloning

Voice cloning is an extended concept of TTS. Apart from synthesizing speech from text, there is an extra target for mimicking the voice of specific target speakers. The term "Voice Cloning" may not be wisely used in the field still. Hence, there may be different explanations for the topic in different documents. To avoid ambiguity, in this thesis we define this research topic by listing its main goals:

1. Perform accurate speech synthesis.
2. Mimic target speakers' voice characteristics while synthesis.
3. Generate natural sounding audio waveform.

There is a similar research field called Voice Conversion. Voice conversion is a speech processing problem that aims to change some of the information in speech data while keeping other information unchanged. Changing speaker information while keeping its content is one of the typical voice conversion problems. The two fields seem similar, but they are not identical. One of the main differences is that

inputs to voice conversion models are speech data and the models directly process the input. In contrast, we restrict the input to voice cloning models to be text and a reference speech. Only feature extraction would be performed on the reference speech to quantify the voice characteristic inside.

Google's team proposed a framework for voice cloning in which transfer learning from speaker verification studies is used for learning speaker characteristics [18]. We refer to this original paper as SV2TTS in this thesis. The proposed model is composed of three neural network-based modules. The three modules are independent and trained in order. However, same as in many other studies from Google, a large set of internal speech data is used in training and no implementation is released to the public. Real-Time-Voice-Cloning (RTVC) is an open-source implementation of SV2TTS on Github [17]. It follows the framework proposed by SV2TTS and is trained by only open-source resources. It also slightly modified the architecture of neuron networks for faster training and inference. RTVC is one of the most popular projects in the field nowadays.

1.3 Phase Reconstruction

Spectrograms are a common acoustic feature used by many signal processing projects. The two TTS models, Tacotron and Tacotron2, mentioned above also predict spectrogram (in Hertz-scale or Mel-scale) from input text. A spectrogram is the magnitude of Short-Time Fourier Transform (STFT) output. STFT and its inverse transform (ISTFT) have perfect reconstruction properties. There is

no information loss in between. However, the above TTS models only predict the magnitude of STFT, extra phase information is required to reconstruct the waveform. This phase reconstruction has long been a research problem in the field. Many speech processing topics also face this problem since it is common to represent signals with acoustic features instead of their time-domain waveform.

Algorithm-based method has long been state-of-the-art. Griffin and Lim [14] proposed a simple iterative algorithm to approximate phase information from a given fixed STFT magnitude (STFTM). We refer to this algorithm as Griffin-Lim Algorithm or GLA in short throughout the whole thesis. Although the algorithm only requires low computation power and does not require extra information other than STFTM, there is a common belief that the convergence of GLA is slow, and the quality of the approximated signal may not be satisfying. We will study the algorithm both theoretically and practically. The details of GLA would be presented in Section 2.5. Numerical experiments and analysis results would be presented in Section 3.1.

Motivated by the deep-learning boom and the limitations of GLA, neural network-based waveform construction has become an active research topic. Many different models have been proposed since mid-2010s, such as WaveNet [34], WaveRNN [19], WaveGlow [38], etc. Neural models are often reported to have high performance in terms of producing natural-sounding waveforms. However, as common problems shared by most neural networks, this type of model requires high computational resources, including hardware such as GPUs and memory. Its performance may also be not stable, especially when the training data used are not enough to well generalize the network learning outcome. High time cost is also a problem in

training stage. Some models, such as WaveNet, are also known as slow in inference. Despite all the problems, neural models are still a popular research direction in the field.

1.4 Pronunciation Error Detection

With the quick development of globalization, the need of learning foreign languages increases day by day. In the past, people needed to go to school and take classes to learn new languages. Benefitting from the advance in technology, nowadays people can learn at home with the help of electronic devices. Computer-assisted language learning (CALL) has been studied over the past few decades. Compared to learning in classes, learning with computers allows students to learn at their own pace. There is also no pressure from teachers and classmates.

Oral practice is indispensable to language learning, no matter what the means of learning is. If students are learning from computers, it is necessary to build a function in CALL systems that can interact with students by providing feedback on their speaking exercises. Computer-assisted pronunciation training (CAPT) systems are designed to serve the purpose. It focuses on checking students' pronunciation and giving feedback. The core of such a system is automatic pronunciation error detection (APED).

In addition, theories [45] have been developed that infants learn to speak by imitating the sound they heard and improve their pronunciation through interaction and feedback from their caregivers. Some studies [33], [9] also provided evidence

that infants' hearing sense plays an essential role in their language learning, in the sense that they must listen to their voice while practising speaking. These suggest that infants "learn by ear", while many adults "learn by eye" when they learn foreign languages.

We also believe that voice cloning techniques would help in language learning because users can imitate pronunciation from words or sentences synthesized by a model that their voices are learnt. It should be more efficient that users learn from their own voices rather than others' voices.

1.5 Objectives and Contributions

The motivation of this study is to explore recent achievements in text-to-speech synthesis after the deep-learning boom in mid-2010s. We especially intended to study voice cloning related materials, including publications and open-source resources such as training data and pretrained models.

As a subtopic in our studies, we have also explored the performance of Griffin-Lim Algorithm and neural vocoders. Algorithm-based vocoders have long been used in the past few decades, but were quickly replaced by neural vocoders after the deep-learning boom. We have studied the mathematics and theoretical backgrounds of GLA and conducted experiments to investigate the advantages of neural vocoders in practical cases. We found that the performance of GLA could be affected by hyperparameters. We will introduce one hyperparameter we found essential and demonstrate how we choose the optimal hyperparameter in Chapter 3. Also, through

numerical experiments, we found that GLA perform well while it is standalone, but it could not function well as a part of a TTS system. Related results will be presented in Chapter 4.

The main contribution of this thesis is a new approach for Automatic Pronunciation Error Detection. APED is a task to check and score students' pronunciation by only machines. It is still an active study topic nowadays. One of the popular methods nowadays is to recognize phonemes from students' speech data using Automatic Speech Recognition (ASR) models and then compare them with the correct phonemes labelled by professionals. Another well-known approach is to compare students' attempts to teachers' demonstrations. In this thesis, we will show that voice cloning techniques could be useful for pronunciation error detection.

Our proposed method is a completely new approach that employs TTS models. It does not require advanced language knowledge nor reference speech recorded by teachers. It is also not necessary to train classification models to detect errors. We will present the procedure of the proposed APED method, together with some simulation results in Chapter 5.

The structure of this thesis is as followed. Chapter 2 is literature review. Basic deep learning and signal processing related knowledge is provided in Section 2.1 and 2.2. Details of the voice cloning model used in this project are presented in Section 2.3. Essential reference papers behind the voice cloning models are introduced in the sub-sections under Section 2.3 and 2.4. In Section 2.5, we explore the Griffin-Lim Algorithm and its convergence. Chapter 3 is methodology. We judge the usefulness of GLA by reviewing its theoretic background and conducting

some simple numerical experiments in Section 3.1. In Section 3.2, we introduce some metrics for TTS model evaluation. We then test the performance of the open-source voice cloning model employed. Chapter 4 presents what can be done for improvements with only limited resources, together with related results. We then discuss potential applications of voice cloning techniques in Chapter 5. Details of a new method for automatic pronunciation error detection are presented. We also provided simulation results to support this proposed approach. Chapter 6 concludes the whole thesis and points out some possible directions for further research.

Chapter 2

Literature Review

2.1 Deep Learning

Deep learning is a stream of machine learning that focuses on the usage of artificial neural networks (ANN). ANN itself has been widely used in different types of machine learning. Deep learning, as its name, tends to use "deep" networks that stack numerous neural network layers in models. Deeper networks also show higher performance in many tasks. Winners of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) would be a good example. The winner in 2012 used an 8-layer-deep network [21]. Winner in 2015 substantially increase the depth of their model to 152 layers [15], and proved to outperform other models.

As the deep learning techniques and models applied to different research fields are noticeably different, in this section only the basic ideas of deep learning are

included. Additionally, some more details about deep learning in speech processing and the natural language processing (NLP) field would be included in the later part.

2.1.1 Fully Connected Neural Network

Fully connected (FC) layers may be the most simple, classical and common structure in deep learning. It has some different names in different documents, such as dense layer. The mathematics in FC layers is relatively simple. An FC layer is defined by formula

$$y = \sigma(Wx + b),$$

where x and y are the input and output of the layer, W and b denote the weight matrix and bias vector. σ represents an activation function.

Neural networks in the early days often use Sigmoid functions as activation. Sigmoid function in the deep learning field usually refers to Logistic function defined as

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Rectified Linear Unit (ReLU) is one of the most commonly used activation functions nowadays. ReLU function is defined as

$$ReLU(x) = max(0, x).$$

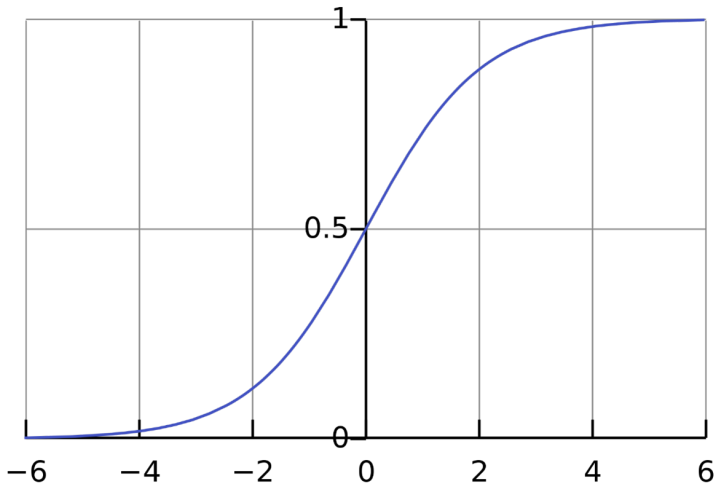


Figure 2.1: Logistic Function

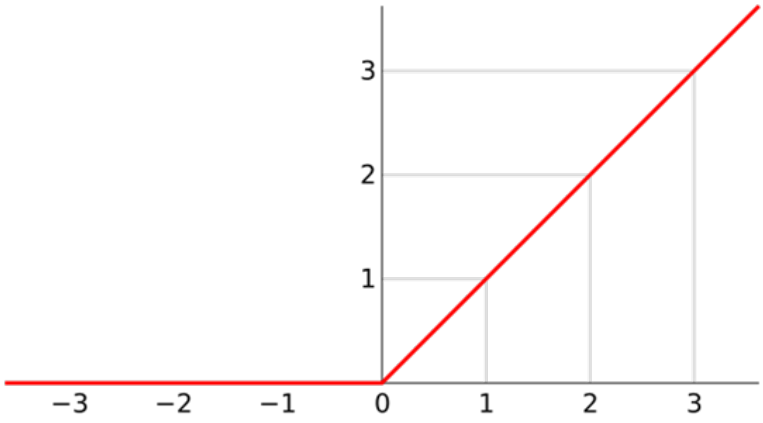


Figure 2.2: Rectified Linear Unit Function

The complexity of an FC layer is $O(n_{l-1}n_l)$, where n_l is the number of neurons in layer l, n_{l-1} is the dimension of input, which also means the size of the previous hidden layer or input layer.

2.1.2 Convolution Neuron Network

Convolution Neuron Network (CNN) is well-known for its ability in the computer vision field. CNN is often used to learn local patterns in input features. The philosophy of CNN is based on the assumption that recognition of patterns should not be affected by the size and location of the patterns. For instance, a filter that recognizes the existence of birds' beaks in images should function equally well, no matter whether the beak is occupying a large part or a small part of the image, or the beak is located in the top-right or bottom-left corner. In some cases, sub-sampling could be done between CNN layers. It is based on another assumption that lower resolution would not affect recognition.

A convolutional layer is defined as

$$output_j = \sum_{i=1}^{C_{in}} Weight(j,i) * input_i + bias_j, for 1 \leq j \leq C_{out},$$

where $*$ is the convolution operator, C_{in},C_{out} are the number of input, output channels of a CNN layer respectively, i,j are the index of channel.

Using CNN layer could be beneficial in the sense that, a convolutional filter is equivalent to an FC layer but with fewer connections between nodes and parameters are reused. While convolution, weights in filters are fixed. Each element in output

feature maps is only computed from a small area of input but not the whole input like what is done in FC layers.

Time complexity of a CNN is given by

$$O(\sum_{l=1}^{L} C_{l-1} \times k_l \times C_l \times m_l),$$

where l is the index of CNN layer, L is the depth of the network, k is kernel size, and m is output feature size. Note that this formula mainly considers the case of 1-D convolution, which is common in speech processing. In some other fields such as image processing, 2-D convolution is also wisely used. In such cases, one should be aware that k_l and m_l are in 2-D shape. For instance, $k = 9$ for a 3×3 filter.

2.1.3 Recurrent Neural Network

Recurrent Neural Network (RNN) is the variation of artificial neural networks that could learn patterns better from data in time series structure. By introducing a "hidden state" mechanism, the order of input would be considered and memorized. Output feature may change according to input order.

The figure 2.3 [36] below illustrated the structure of a simple RNN. It is shown that a hidden state h_t is computed from input x_t and stored in RNN cells at each time step t. At time step $t + 1$, this hidden state is considered, together with the next input x_{t+1}, for updating hidden state h_{t+1}. The mathematics in this RNN

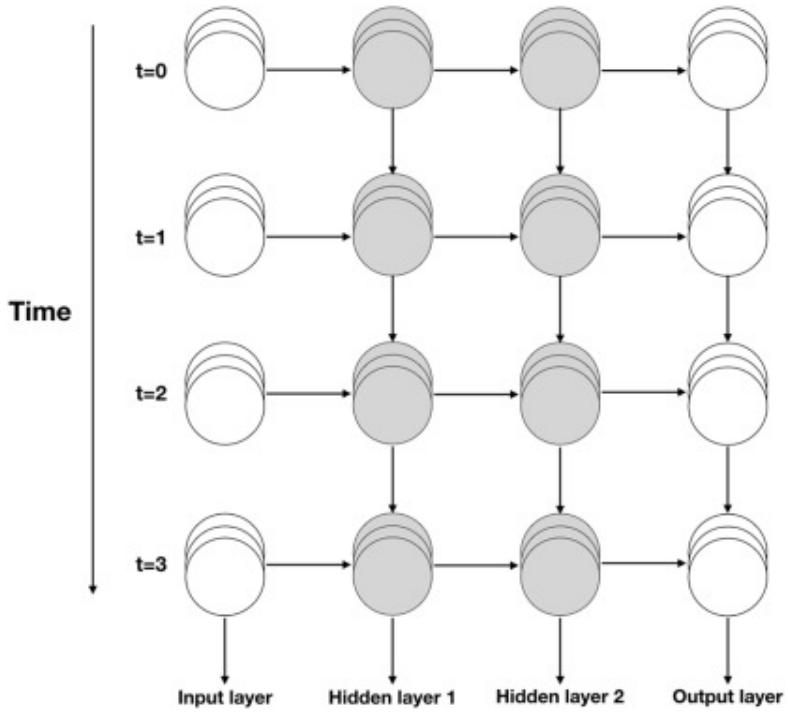


Figure 2.3: Simple Recurrent Neural Network

structure could be summarized by

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h),$$
$$y_t = \sigma_y(W_y h_t + b_y),$$

where W, b and σ are the weight, bias and activation function.

Long Short-Term Memory

Nowadays, the term RNN used in many documents refers to Long Short-Term Memory (LSTM) [16] instead of the "simple" RNN introduced above. LSTM can help to overcome the gradient vanish problem in simple RNN. Gradient vanish occurs

because, during backpropagation, when the network is deep, partial derivatives of loss to parameters in the previous layers become very small. Optimization of learnable parameters could not continue since the gradient is small and could hardly point out a decent direction.

LSTM layers are composed of an input gate, a cell gate, a forget gate and an output gate. The four gates control the information flow in and out of the layer, and also what to store in LSTM cells.

An LSTM layer is defined by the following equations:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}),\\
g_t &= tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}),\\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}),\\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}),\\
c_t &= i_t \odot g_t + f_t \odot c_{t-1},\\
h_t &= o_t \odot tanh(c_t),
\end{aligned}
$$

where i_t,g_t,f_t,o_t are the input, cell, forget and output gate at time t respectively, c_t,h_t are cell state and hidden state at time t. σ is Sigmoid function, $\odot$ is element-wise product. x_t denotes the input to LSTM cells. It is shown above that the number of learnable parameters in an LSTM layer is about 4 times that in an FC layer of the same size. It caused a larger memory and higher computation power requirement.

The Time Complexity of an LSTM network is given by the following:

$$O(\sum_{l=1}^{L}(4C_{l-1}C_l + 4C_l^2 + 3C_l)),$$

where l is the index of LSTM layers, L is the depth of the network, and C_l is the number of cells in layer l. This complexity would be double for bi-directional layers, as the computation is done for both forward and backward directions.

Gated Recurrent Unit

As stated above, LSTM contains a relatively large number of parameters and requires higher computation power, a variation is Gated Recurrent Units (GRU) [4]. GRU can be considered as a simplified LSTM in the sense that it omitted the output gate. While having fewer learnable parameters, it is reported that GRU can achieve similar performance as LSTM. A GRU unit is defined by the following equations:

$$\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}),\\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}),\\
n_t &= tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn})),\\
h_t &= (1 - z_t) \odot n_t + z_t \odot h_{t-1},
\end{aligned}$$

where r_t,z_t,n_t are the reset, update, new gate respectively, and h_t is the hidden state at time t. The input to GRU cells is x_t, or the hidden state of the previous layer for multilayer GRU.

2.2 Acoustic Features

Signals can be described as time series. Analog signals are usually described as continuous waves. In contrast, digital signals are discrete data points. For audio signals, the time series may be the amplitude of air pressure over time. Sampling rate is the number of sample points recorded in one second. For example, the sampling rate of CDs is typically 44.1kHz, i.e., 44100 samples per second. Direct processing of signals in time domain is difficult since there is a lack of information about the components of the signal. Thus, concise and precise representations of digital signals are required. Acoustic features are the representations that we are looking for. There are many types of acoustic features used in different fields. In this chapter, only the features used in this project are introduced.

2.2.1 Fourier Analysis

Signals are time series as mentioned. Information that changes over time could be easily observed by plotting the waveform. However, it is very difficult to know the sinusoid component of the signals only from the time-domain waveforms, no matter how high the sampling rate is. Fourier analysis can be used to compute frequency components of signals.

Fourier transform is a mathematical method to transform signals into spectrums, a representation of amplitudes over frequency. In other words, it transforms signals from time domain to frequency domain. As shown in figure 2.5 below, one could gain high resolution in frequency domain after transformation. But at the same

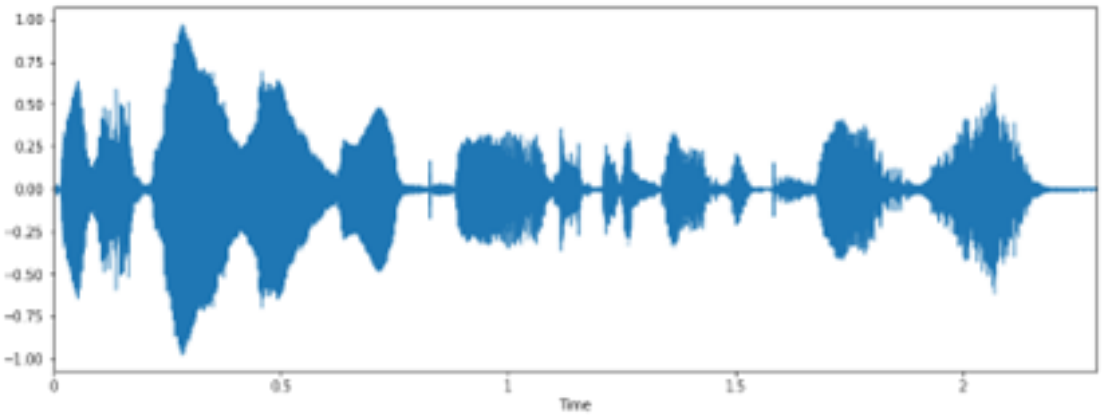


Figure 2.4: Digital Signal

time, almost all information about time is lost and can only be recovered by applying the inverse transform to the spectrum. Note that both time-domain signal and frequency-domain spectrum are discrete in this example, so Discrete Fourier Transform is actually done here.

Discrete Fourier Transform and its inverse transform is given in equation 2.1 and 2.2, respectively.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, k = 0, ..., N-1 \tag{2.1}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{-j2\pi kn/N} \tag{2.2}$$

where $x[n]$ denote time-domain signals, $j = \sqrt{-1}$.

Note that $e^{-j2\pi kn/N}$ are the roots of unity. The transformation gives a complex number result. Its magnitude can be drawn as "Magnitude Spectrum", and its phase can be drawn as "Phase Spectrum" similarly. Some documents or applications

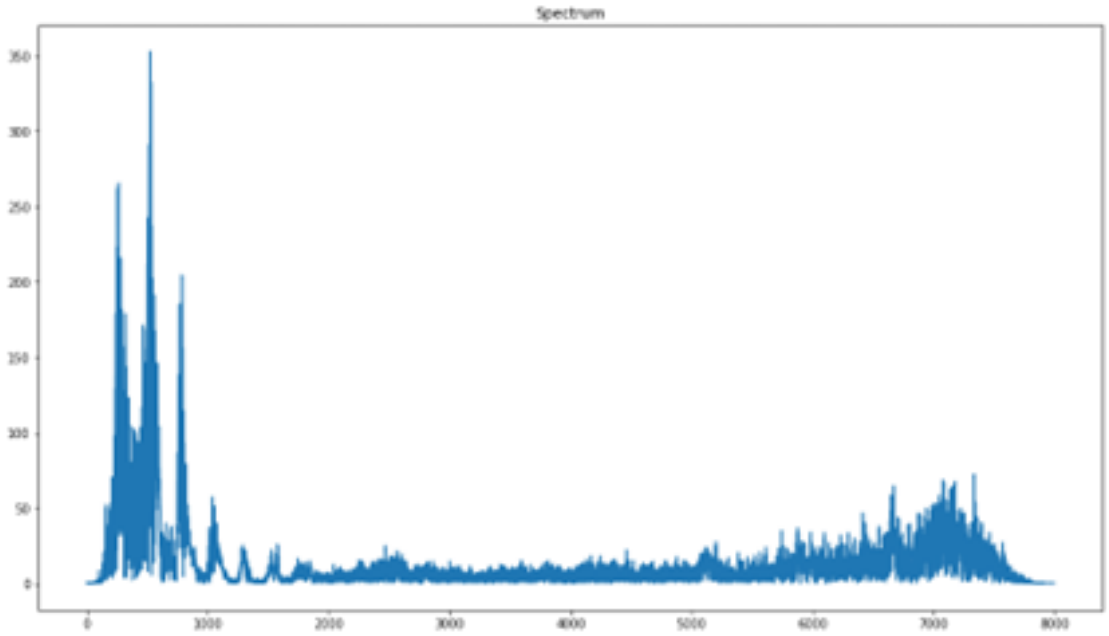


Figure 2.5: Spectrum

may refer spectrum to the powered spectrum, i.e. squared magnitude. Readers should notice the terminology difference among different documents. Without specification, spectrum in this thesis refers to magnitude spectrum.

Fourier Transform mentioned in this thesis should be referred to as Fast Fourier Transform (FFT). FFT is an algorithm that computes Discrete Fourier Transform (DFT) and its inverse efficiently. Compared to the complexity of DFT $O(N^2)$, the FFT algorithm could reduce the complexity to $O(Nlog_2N)$. It is also well-known that the FFT algorithm could have the most efficiency when N is a power of 2.

As pointed out above, it becomes a problem that time-domain information is lost after FFT. Since many signals in real life are not periodic, time-domain information is still preferable in many cases. One of the solutions to this problem is Short-Time Fourier Transform (STFT). During STFT, input signals are cut into small

segments by window functions along the time axis. Each segment is transformed to frequency-domain by Fourier Transform. The output is named spectrogram and is often visualized in heatmap format. Spectrograms become popular in signal processing since it reserves a certain level of time-domain and frequency-domain information at the same time. As shown in figure 2.6 below, there are three axes in a spectrogram: time domain along the horizontal direction, frequency domain along the vector direction, and amplitude indicated by colour. As for spectrum, only the magnitude of the STFT result is plotted in spectrograms, so spectrograms can also be recognized as a real number matrix.

Short-Time Fourier Transform and its inverse transform is given in equation 2.3 and 2.4, respectively.

$$\text{STFT}(x[n]) = X(m,\omega) = \sum_{n=0}^{L-1} x[n]w[n-m]e^{-j2\pi\omega n/L} = Ae^{j\theta} \tag{2.3}$$

$$\text{ISTFT}(X(m,\omega)) = x[n] = \sum\sum X(m,\omega)w[n-m]e^{-j2\pi\omega n/L} \tag{2.4}$$

where $x[n]$ denote time-domain signal and $w[n]$ denote window function with length L, A is amplitude and θ is phase, $j = \sqrt{-1}$.

In this thesis, FFT and its inverse is done through Python package *Scipy* [50]; STFT, its inverse and most of the other signal processing mathematics are done through another package *librosa* [28].

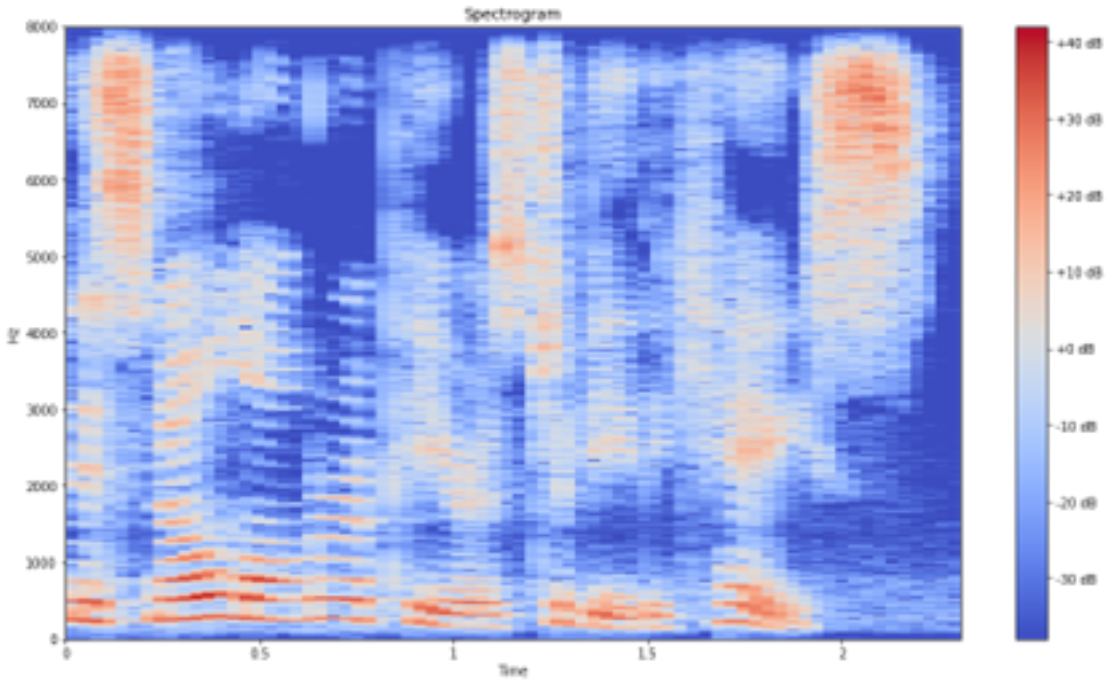


Figure 2.6: Spectrogram

2.2.2 Mel Scale Frequency

It is common in speech processing to use Mel-spectrograms instead of the linear spectrograms introduced above. Mel-spectrograms differ from linear spectrograms in the scale of frequency domain. Human hearing sense is nonlinear, which means the pitch heard by human ears is not linearly related to the frequency. For instance, the frequency difference between 400 Hz and 500 Hz, and 10400 Hz and 10500 Hz, is the same, i.e., 100 Hz. However, human is hard to detect the difference between the latter pair, compared to the former pair. It is observed that humans are more sensitive to lower frequencies than higher frequencies. This nonlinearity can be better observed in music. The frequency of standard A440, as known as A4 in Scientific Pitch Notation (SPN), is 440 Hz. The frequency of A3 and A5 is 220

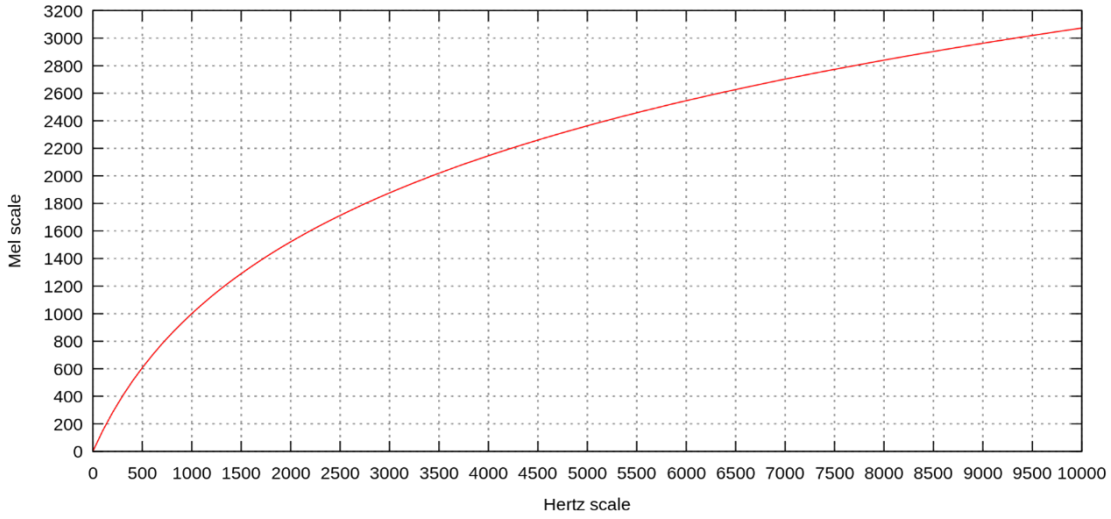


Figure 2.7: Relationship between Mel scale and Hertz scale frequency

Hz and 880 Hz respectively. In human perception, the difference of pitches is 1 octave, but the difference measured in Hertz is not the same. It is observed that the frequency in Hz would be doubled for every octave higher.

The Mel-scale is developed to adapt to human hearing senses. The transformation between Mel-scale and Hertz-scale is given by

$$m = 2595 \log_{10} \left(1 + \frac{f}{700}\right),$$

where m and f represent the frequency in Mel-scale and Hertz-scale. Similarly, humans do not hear the amplitude of sound linearly but logarithmically. It is also common to indicate amplitude in Decibel scale in spectrograms.

Spectrogram is a kind of basic acoustic feature. It is very popular in deep learning methods of speech processing since it can keep most of the information in audio and it is easy to compute. Mel-spectrogram is mainly used in this report.

2.3 Voice Cloning

The ultimate goal of voice cloning is to generate output speech in which the voice is similar to that of input reference speech and the content is the same as input text. The SV2TTS model [18] proposed by Google's team approaches the goal in three steps, using three independent neural network modules, namely Speaker Encoder, Synthesizer and Vocoder. The Speaker encoder extracts speaker information from reference speech, the synthesizer predicts spectrogram from speaker embedding and input text, and the vocoder generates waveform from spectrograms. This pipeline is shown in the figure 2.8 below. The three blocks represent the three neural networks, and the items not in boxes are either input, intermediate product or output of the model.

Encoder aims to extract speaker information from utterances. It takes speech data as input. Input audio data are first preprocessed to log Mel spectrogram. The filterbank energies are fed to the encoder network. Network output is a speaker embedding vector to represent voice characteristics. This vector is used for a speaker verification task while training. By feeding back the loss, it is expected that the network can learn to extract speaker information.

Synthesizer is a typical network used in TTS that predicts Mel spectrogram from input text. It is a sequence-to-sequence model, using an encoder-decoder structure with an attention mechanism. Input character sequence is first mapped to character embedding sequence by a learnable lookup table, then transformed to a hidden representation by the encoder layers. Speaker embeddings are concatenated to the hidden representations and are used as the memory in the attention mechanism.

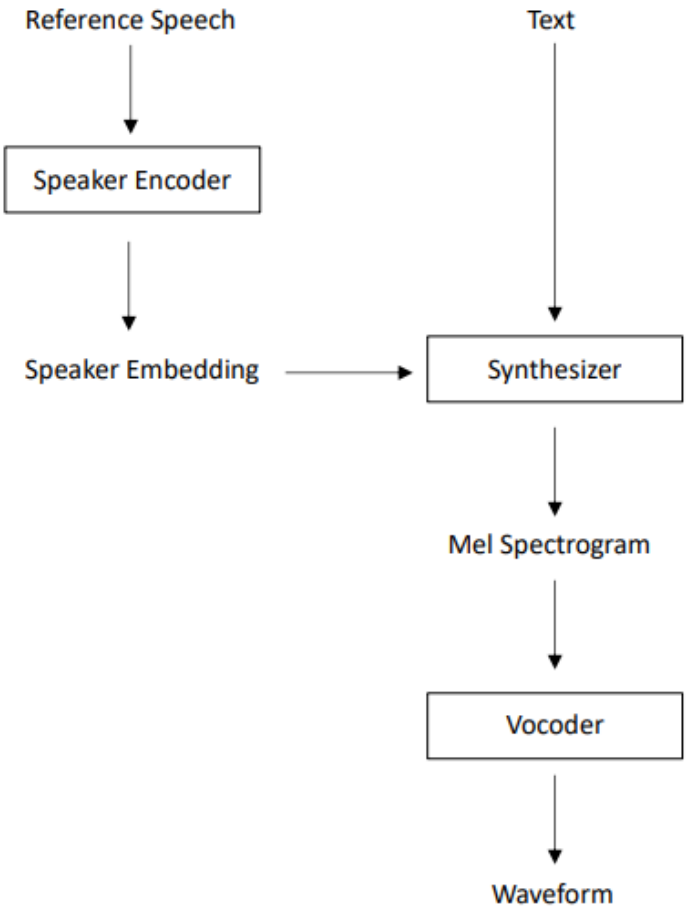


Figure 2.8: Model Overview

The decoder consumes the memory and predicts Mel spectrogram frames with assistance from attention. The decoder also determines when to terminate the decoding procedure by itself.

Vocoder is another typical component in speech processing that reconstruct signals from acoustic features. Spectrogram is a popular acoustic feature in many kinds of speech processing. However, it is also known that phase information is lost in spectrograms. The Vocoder module in this thesis is used to predict the original waveform from spectrograms.

2.3.1 Speaker Encoder

The first neural network, namely Speaker Encoder, aims to represent speaker information in an utterance with a vector. This vector is also known as speaker embedding. Extracting speaker information is an essential task in speaker verification and speaker recognition field. SV2TTS performed transfer learning from those fields to voice cloning field. In this section, we introduce what are the common methods to calculate speaker embeddings, and how the Speaker Encoder is trained.

Before the development of deep learning-based methods, it was common to use i-vector to represent human voice characteristics [7]. It represents the speaker information of an utterance as a fixed and low dimensional vector, typically 400 dimensions, using Gaussian mixture model. d-vector is a neural network-based method. Its performance is compatible to i-vector [48]. Utterances are cut into small frames and each frame is fed into a deep neural network (DNN). The output

of the last layer is then used in the output layer for speaker recognition task, in which the label is a one-hot vector that represents speaker identities. It is expected that the DNN learns to output vectors that are highly relative to speaker voice characteristics so that those vectors can be used for identity classification. The average of those outputs from each utterance frame is called d-vector and is often considered as the speaker embedding of that utterance.

A potential limitation of d-vector is that only frame-level information is considered by the network while training. x-vector is proposed for improvement, and it outperformed the traditional i-vector method [43]. One of the modifications made is that statistical pooling is done after feeding frames to neural networks. For example, the mean and variance of the output vectors are calculated and used as input to another neural network to perform utterance-level consideration. x-vector is the last activation before the output layer. This idea is the same as d-vector.

The model used in SV2TTS and RTVC is General End-to-End [51]. As shown in Figure 2.9 below, the network is constructed by a 3-layer LSTM and one linear projection layer. Using dense layers in the network, as done in the d-vector method, may cause a problem that the model only accepts input with fixed length. The d-vector method introduced above cut audio into small fixed-length pieces before input to the model. Using LSTM instead of dense layers may avoid this problem. There is no information about what the projection layer should be in the original paper. It is assumed to be a fully connected linear layer in RTVC implementation. The number of cells in each LSTM layer is also reduced in RTVC implementation.

Training data in each mini-batch is a set of audio clips that includes N speakers

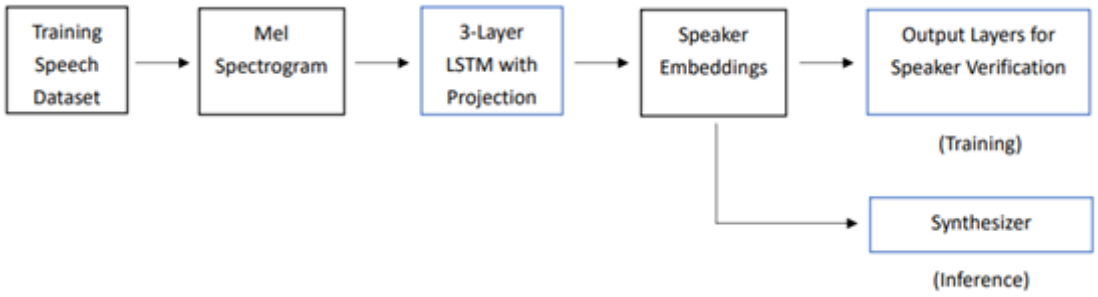


Figure 2.9: Pipeline of Speaker Encoder

and M utterances from each speaker. The data are pre-processed to 40-channel Mel spectrograms. Log filterbank energies are the input to the neural network. Data are denoted as x_{ji} ,where $1 \leq j \leq N, 1 \leq i \leq M$, corresponding speaker embeddings are denoted as e_{ji}.

For any speaker k, centroid can be calculated from all his speaker embeddings as

$$c_k = \frac{1}{M} \sum_{m=1}^{M} e_{km}.$$

While training, cosine similarity can be calculated from each pair of embedding and centroid, scaled by learnable weight and bias as below:

$$S_{ji,k} = \omega \cdot \cos(e_{ji}, c_K) + \beta.$$

This $S_{ji,k}$ is then put into Softmax function. It is expected that, for positive examples, i.e., when embedding and centroibelonggs to the same speaker, this $S_{ji,k}$, could be as close to 1 as possible. Similarly, for negative examples, i.e. embedding and centroid belonging to different speakers, this value should be close to 0.

Cross entropy is used in loss function. For each pair of embedding and centroid, the loss is given as followed:

$$L\left(e_{ji}\right) = -S_{ji,j} + \log \sum_{k=1}^{N} \exp\left(S_{ji,k}\right).$$

Hence, the total loss is given by:

$$L_G(x; w) = \sum_{j,i} L\left(e_{ji}\right).$$

It is worth mentioning that, to avoid bias in the calculation of similarity, for positive examples, i.e. $j = k$, $S_{ji,k}$ is given by the following modified equation:

$$c_j^{(-i)} = \frac{1}{M-1} \sum_{\substack{m=1 \\ m \neq i}}^{M} e_{jm},$$

$$S_{ji,k} = \omega \cdot \cos(e_{ji}, c_j^{(-i)}) + \beta.$$

The loss encourages speaker embeddings to be close to the centroid of the speaker, at the same time far away from other speakers' centroids. While inference, the similarity part is omitted as we are not interested in speaker verification. e_{ji} is named "speaker embedding" and its usage will be mentioned in the following parts.

2.3.2 Synthesizer

The second network is the synthesizer, which predicts Mel spectrograms from input text. Tacotron 2 [42] is used but without the WaveNet vocoder. This network should be trained after finished training the speaker encoder. A trained speaker encoder is needed to generate embeddings for each utterance in training data. The speaker encoder is frozen and used as a black box in this stage. Text in the transcript of training data is another input to the network for training. Tacatron2 has an encoder-decoder architecture. The encoder converts input text to a hidden feature representation. The decoder predicts Mel-spectrogram by this representation and attention mechanism. Apart from the original papers, many of the details and notation in this report are based on an open-source implementation [27]. Tacotron 2 is a complicated TTS model, we summarized its architecture in Figure 2.10.

Input character sequences are first converted to character embeddings by a learnable lookup table and then are passed through 3 convolutional layers. The output of the last convolutional layer is passed to a bi-directional LSTM. A bi-direction LSTM is actually two independent RNNs, one read the input sequence in order, and another read in reverse order. Using bi-directional LSTM allows the network not only to consider the input text sequence from left to right but also from right to left. The output of the forward and backward RNN are concatenated together, then further concatenate to speaker embeddings for the encoder hidden state. This hidden state is used as memory in the attention module after being concatenated with speaker embeddings.

The attention module is a hybrid attention mechanism [5] which combined content

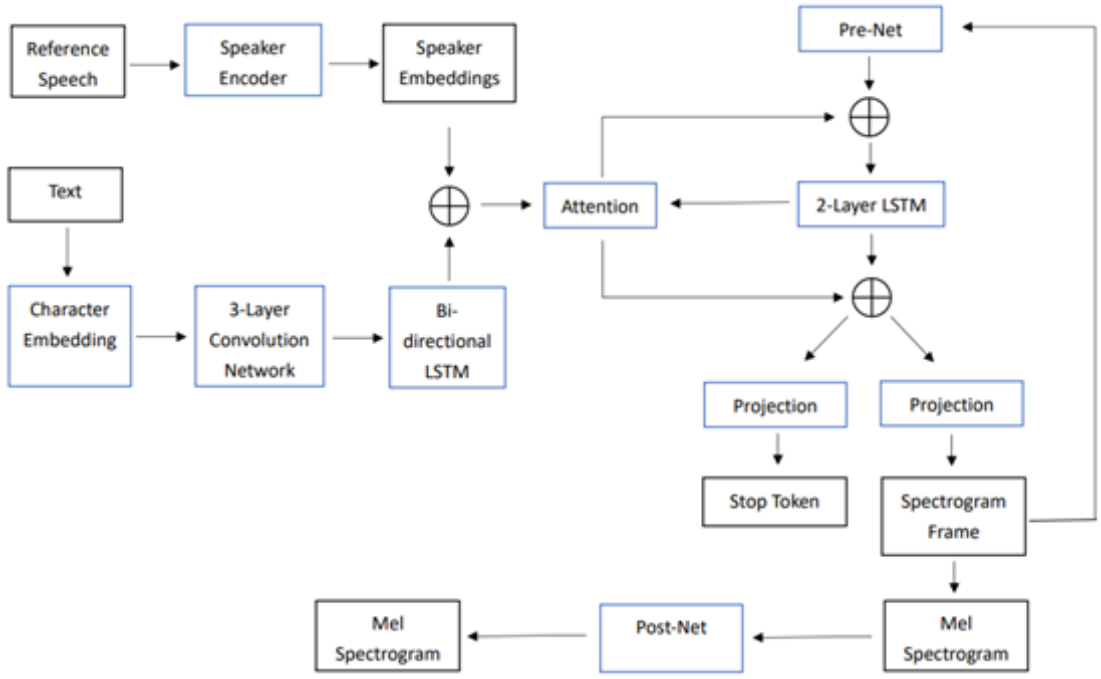


Figure 2.10: Detailed architecture of Synthesizer

based [3] and location based [13] attention. The attention mechanism calculates context vectors from each encoder annotation, current decoder hidden state and location features.

Decoder steps start from feeding the previous predicted Mel spectrogram frame to a Pre-Net, which is constructed by 2 fully connected layers, making the model autoregressive. In contrast, while training, the corresponding ground truth Mel spectrogram of the previous decoding step is used instead of the predicted one. This idea is called "Teacher forcing" and is often used in the training of recurrent neural networks (RNN).

The output of Pre-Net is concatenated with the previous context vector and then feed to a decoder RNN network of 2 LSTM layers. This decoder hidden state is

used to compute a new context vector as mentioned in the attention part above. Then the hidden state and the context vector are concatenated together. On one hand, this new vector is input to a projection layer to predict a new frame of Mel spectrogram. On the other hand, it is also put into another projection layer to compute Stop Token, which determines whether the whole decoding process is finished or not.

The attention mechanism is summarized here with the help of mathematical representations. Denote encoder hidden state sequence as $\{h_j\}$. For each decoding step i, the decoder output of the previous step y_{i-1} is first fed into Pre-Net. The output of Pre-Net py_i, concatenated with context vector c_{i-1}, is then feed into the LSTM layers for a new decoder hidden state s_i.

In attention module, attention energy is given by

$$e_{ij} = v^T tanh\left(W_a s_i + V_a h_j + U_a f_{i,j} + b_a\right),$$

where W_a, V_a, U_a and b_a are learnable weights and bias. Location feature $f_{i,j}$ is given by a convolution operation

$$f_{i,j} = F_a * c\alpha_{i-1},$$

where

$$c\alpha_{i-1} = \sum_{j=1}^{i-1} \alpha_{i,j}.$$

Attention weights can be computed after having attention energies by putting them

into Softmax function:

$$\alpha_{ij} = \frac{exp\left(e_{ij}\right)}{\sum_{k=1}^{T} exp\left(e_{ik}\right)}.$$

Finally context vector is a linear combination of encoder hidden state based on attention weights:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j.$$

After finished decoding, the predicted Mel spectrogram would pass through a Post-Net to improve overall quality. The Post-Net is constructed by 5 convolution layers with *tanh* function as activation, and a projection layer. Residual connection is done as the final step of Post-Net.

The loss of synthesizer is the sum of L1 and L2 loss of spectrogram before Post-Net, L2 loss after Post-Net and cross entropy of stop token. L2 regularization is used additionally but excludes all types of bias and parameters in RNN, projections and embeddings layers. The training target is the ground truth Mel spectrogram of the training data. The loss function is summerized as

$$\begin{aligned} loss =& \frac{1}{n}\sum_{i=1}^{n}\left(y_{real,i} - y_i\right)^2 + \frac{1}{n}\sum_{i=1}^{n}\left|y_{real,i} - y_i\right| + \frac{1}{n}\sum_{i=1}^{n}\left(y_{real,i} - y_{final,i}\right)^2 \\ &+ \lambda\sum_{j=1}^{p}\omega_j^2 + \frac{1}{N}\sum_{n=1}^{N}\left(-\sum_i y_{label,i} log\left(y_{s,i}\right)\right), \end{aligned}$$

where the subscript *real* denotes the ground truth training target, $final$ denotes the Post-Net output.

2.3.3 Vocoder

In SV2TTS, the authors used WaveNet as the vocoder of the framework. It was a modified WaveNet in which the slow inference speed is overcome, and spectrograms can be used as input instead of the high-level acoustic features mentioned in the original document. However, there is no public implementation or codes available as usual. In RTVC, the developer used a modified version of WaveRNN instead. This vocoder is made open-source on Github but is not supported by any publication. The details can only be found in the codes. Some explanations can also be found in the thesis of RTVC.

The predicted Mel spectrograms generated by the above synthesizer and the ground truth audio waveform are split into small segments. While training, the inputs to the model are the Mel spectrogram segment of time t and the waveform segment $t-1$. Output is the corresponding audio waveform in time t. Based on the released code, the loss function used is cross entropy.

The pipeline of the vocoder is illustrated in Figure 2.11. Input Mel spectrogram is used in two parallel processes, called ResNet and Up-sampling network. In the up-sampling network, the Mel spectrogram is up-sampled to match the length of the target waveform by 2d-convolution layers. In ResNet, the Mel spectrogram is first fed to a 1D convolution network, then passed to several Residual Blocks. The exact number of Residual Blocks used is set by users in advance. Each Residual Block is composed of a convolution layer, with batch normalization and ReLU activation function, followed by another convolution layer with batch normalization. Finally, residual connection is done before output. The overall output is fed to

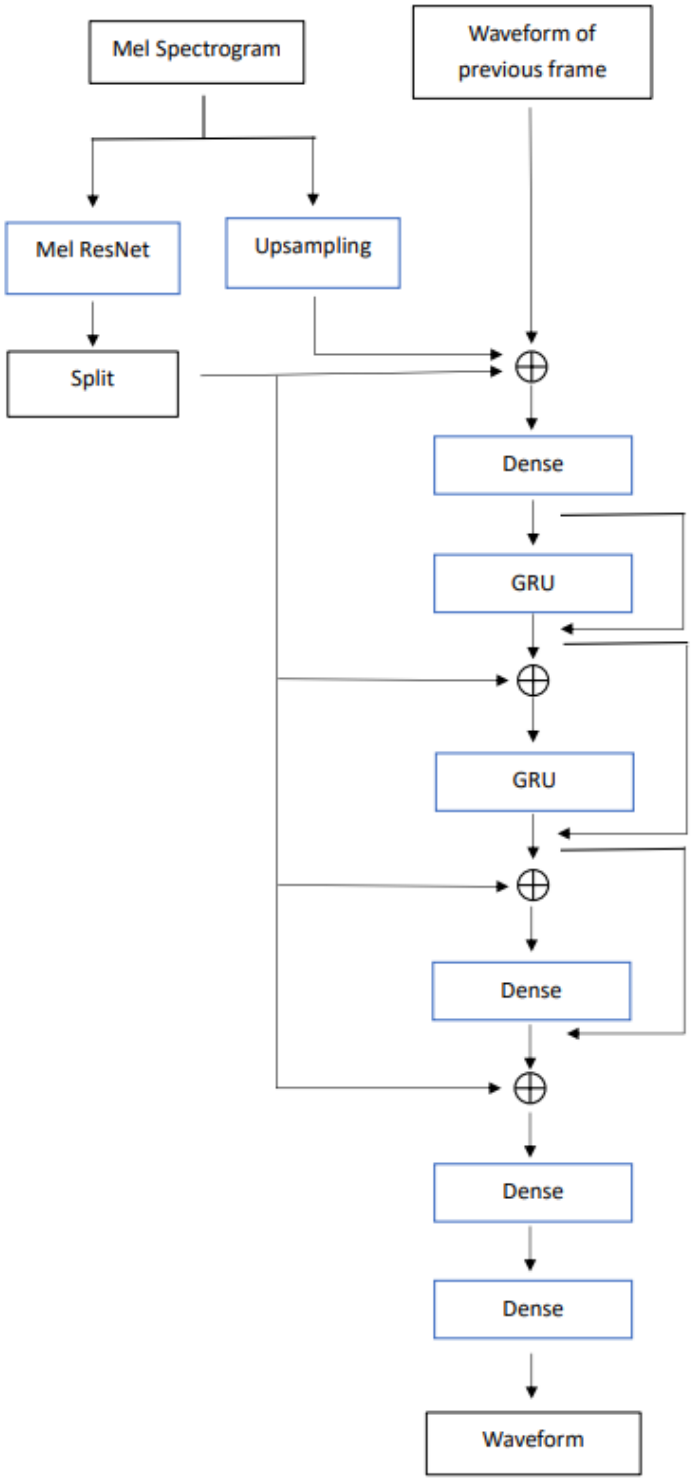


Figure 2.11: Detailed architecture of Vocoder

a convolution layer again. The output feature would be split into four equal parts along the channel dimension and are used to condition the upsampled Mel spectrogram mentioned above. At this point, the previous waveform segment, the upsampled Mel spectrogram and the first feature segment are concatenated and used as input to the later layers. The main body of vocoder is composed of six neural networks. Those are 1 fully connected network, 2 GRU networks, and another 3 fully connected networks, in order. The three other feature segments are added into the network after the first GRU layer, after the second GRU layer, and after the second FC layer, respectively.

The training target is ground truth waveform compressed to 9-bit using mu-law. Thus, the model considers the signal generation problem as a 512-class classification problem. The probability distribution of the 512 classes is given by putting model output to Softmax function. While inference, this 9-bit output would be decoded using mu-law.

2.4 Attention Mechanism

As supplementary information, we explore the initiation and development of attention mechanism (AM) in this section. It will cover two main types of attention: cross-attention and self-attention.

Cross-attention was developed in the mid-2010s for sequence-to-sequence Natural Language Processing (NLP) problems using encoder-decoder type models. Attention is proposed to assist the alignment between features extracted by encoder

and outputs predicted by decoder. Another major function of attention is that it helps the model determine output length, which is critical in sequence-to-sequence models. The encoder-decoder structure is commonly used since then.

Self-attention further extended the mechanism in the sense of using attention inside the encoder or decoder. Recent research results show that training the encoder solely could build a good foundation model that could be applied to many different downstream tasks with a relatively light finetune. Mathematical representations would be given in this section to briefly summarize the computation of the mechanisms.

2.4.1 Basics of Attention Mechanism

Attention could be described as the correlation matrix between two sets of vectors. In this session, we generally introduce the computations inside AM. Technical details of its applications would be given in the latter part.

Consider a vector q and a sequence of vector $K = \{k_1, k_2, \ \ldots, k_J\}$. The vector q is called query and the vectors in set K are called keys. The first step of AM is to calculate a correlation score between each pair of q and k_j. This score is called energy in some documents, and different methods may be used for the calculation. A simple and typical method is calculating the inner products of the vectors as below:

$$e_j = q \cdot k_j.$$

Another typical method is additive method as below:

$$e_j = w^T tanh\left(q + k_j\right).$$

Notice that e_j obtained in this stage are scalars and should be normalized, for instance, using Softmax function:

$$\alpha_j = \frac{exp\left(e_j\right)}{\sum_{j=1}^{J} exp\left(e_j\right)}.$$

The α_j here is called alignments or attention weights. Since Softmax function is used, the attention weight could be considered as a probability distribution of related items in the set of keys, and the sum of α_j equals to 1. This alignment is used to compute context vector c, which is a representation that contains weighted relative information inside, for further process. This context vector can also be considered as a linear combination as shown below:

$$c = \sum_j \alpha_j h_j.$$

2.4.2 Sequence-to-Sequence Models Using Attention Mechanism

Sequence-to-Sequence (seq2seq) models solved the problem that in some cases output length is not fixed and cannot be determined in advance. Consider a simple classification problem example like sentiment analysis of polarity, models trained for this task only need to output a scalar indicating which class the input belongs

to. More specifically, the models output a probability distribution of each class, and this distribution can be stored in a single vector, thus the size of model output is always fixed. Another type of problem is that output length could be computed in advance. An example from NLP may be Part-of-Speech (POS) tagging. Models for this task may simply take each word in the input sequence as a token and output one POS for each token, i.e., the number of output equals to the number of input tokens. However, many NLP tasks do not have this property. For instance, in machine translation, it is necessary to determine output size based on context.

Many seq-to-seq models shared an encoder-decoder structure linked by AM, such as [3] and [5]. There is a common idea that the input sequence is processed to latent representations by the model encoder network, and the decoder network predicts the output sequence by considering the latent representations with assistance from AM. There is no definite structure for the encoder and decoder network, while Long Short-Term Memory (LSTM) layers and convolutional layers are commonly used.

We now explain the AM using mathematic representations. Denote input sequence with length J as $X = \{x_1,\ x_2,\ \ldots, x_J\}$. Each x_j could be a scalar or vector in a fixed dimension. Encoder network extract features from input sequence to latent representation h_j. It is denoted as:

$$H = \{h_1, h_2,\ \ldots, h_J\} = encoder(X).$$

This representation is sometimes called encoder hidden state. Note that in general, the length of this representation sequence is the same as the input sequence, while the dimension of each vector inside may be different.

Decoder network has an autoregressive structure, which means that decoder output in each step would be fed back to the decoder network as input in the next step. Neural network layers in the decoder network generate a vector, namely decoder hidden state, and pass it to AM for alignment. For every step i,

$$s_i = decoder\left(y_{i-1}\right),$$

where y_i and s_i denote decoder output and decoder hidden state at step i.

In attention module, encoder hidden states are used for computing key vectors. Query vector is computed from decoder hidden state at each step, and AM returns context vector to the decoder. This context vector would be projected to decoder output by an extra neural network layer.

We can summarize the process as following equations together with the process introduced in the previous session, using inner product energy as an example:

$$\begin{aligned}
k_j &= W_k h_j, \\
q_i &= W_q s_i, \\
e_{i,j} &= \langle q_i, k_j \rangle, \\
\alpha_{i,j} &= \frac{exp\left(e_{i,j}\right)}{\sum_{j=1}^{J} exp\left(e_{i,j}\right)}, \\
c_i &= \sum_j \alpha_{i,j} h_j, \\
y_i &= W_p \left[s_i; c_i\right],
\end{aligned}$$

where W_k,W_q,W_p are learnable parameters. Note that their shapes are determined

by hyperparameters in advance.

It is worth mentioning that, in general, the first input fed to the decoder is a special token denoting the start of decoding or simply a zero vector. Similarly, a special token is set to be a possible output from the decoder, representing the termination of the decoding procedure. Seq-to-seq is achieved from the mechanism that the model decides whether to stop decoding using the stop token at every step.

2.4.3 Self-Attention and Transformer

Transformer model [49] extended the function of AM. Apart from just linking encoder-decoder, AM can be used as a kind of deep neural network layer. The proposed architecture is named self-attention. To avoid ambiguity, the AM introduced in the previous section would be called cross-attention. In this section, we describe the mechanism of self-attention and its application in the Transformer model.

Self-attention is calculated from almost the same procedure as cross-attention. Denote a vector sequence as $X = \{x_1,\ x_2, \ldots, x_N\}$, where it could be either the raw input sequence or output of previous hidden layers. Similar to cross-attention, key vectors are driven from each x_i, but in contrast, query vectors in self-attention are also given by x_i. Attention energy and alignment are calculated with the same method, but the context vector is computed from linear projection, namely value vector, instead of from x_i directly. The procedure is presented explicitly with

mathematical notations below:

$$
\begin{aligned}
q_i &= W_q x_i, \\
k_j &= W_k x_j, \\
v_j &= W_v x_j, \\
e_{i,j} &= \langle q_i, k_j \rangle, \\
\alpha_{i,j} &= \frac{exp\left(e_{i,j}\right)}{\sum_{j=1}^{J} exp\left(e_{i,j}\right)}, \\
c_i &= \sum_j \alpha_{i,j} v_j.
\end{aligned}
$$

This process would be repeated for every vector in the input sequence in each self-attention layer. Thus, the complexity of computing attention energy is $O(N^2)$.

Transformer kept the encoder-decoder architecture introduced in the previous session, meaning Transformer is also built for seq-to-seq usage. One of the major changes made in Transformer is that the encoder and decoder are mainly composed of self-attention layers.

The encoder of Transformer contains multiple identical blocks, each block is composed of two sub-layers. The first sub-layer is Multi-Head Self-Attention. The idea is the same as the self-attention presented, but computing multiple sets of queries, keys, and context vectors, then projecting to a final output sequence.

For a Multi-Head Self-Attention with H heads, the three vectors, q_i, k_j and v_j, are computed for each h, $1 \leq h \leq H$. Correspondingly, e_{ij}, α_{ij} and c_i are computed

for each h. Finally, the context vector of each h is combined as below:

$$c_i = W_o \left(\left[c_i^1, \ldots, c_i^H \right] \right),$$

where W_o is learnable weights.

Transformer then perform residual connection [15] and layer normalization [2] to output sequence. Residual connection could be written as

$$y = F\left(x; W\right) + x,$$

where F is a set of neural network layers with parameter set W. Layer normalization could be described as normalizing all vectors in layer output y by

$$y_i^{'} = \frac{y_i - \mu}{\sigma},$$

where y_i is the i-th vector component in y, μ and σ denote mean and standard deviation of the elements in y_i. The second sub-layer of the Transformer encoder is simply a fully connected (FC) network, with residual connection and layer normalization. This block is stacked serval times in the encoder. For the original Transformer, the authors mentioned they stacked six blocks in their model.

Decoder of Transformer shared a similar structure of stacking sub-layers. Before going into the blocks, it is worth mentioning that, the decoder of Transformer may not be autoregressive. A non-autoregressive decoder may be achieved by simply feeding a long sequence of START tokens to the decoder and ignoring outputs after the STOP token in the output sequence. Another method may be using an extra

model to predict output length.

The first sub-layer is masked multi-head self-attention. Transformer proposed causal masking in the decoder, which means forcing the decoder not to attend to keys in subsequent positions. For instance, under causal masking, query vector q_i could only attend to key vectors before position i. The second sub-layer is multi-head cross-attention, which aligns encoder hidden states and outputs of the previous sub-layer. By integrating the details of cross-attention and multi-head attention introduced multiple times in this chapter, readers could easily understand this concept. Thus, the details are omitted here. The last sub-layer is an FC layer, as of encoder structure. Note that residual connection and layer normalization are added after each sub-layer.

Transformer and self-attention techniques are further developed to BERT [8]. One important feature in BERT-like models is that the models first learn from big data by self-supervised learning, and then are finetuned for different downstream tasks. BERT-like models are reported to be especially good at NLP and have become state-of-the-art nowadays. Details would not be provided in this thesis as it is not closely related to our studies.

2.5 Griffin-Lim Algorithm

Spectrogram is a commonly used acoustic feature, especially for neural-network-based models. Spectrograms, in both Hertz-scale and Mel-scale, have been applied to different research topics in speech processing, such as Text-to-Speech Synthesis

(TTS) [53], [42], Voice Conversion (VC) [6], Speaker verification [51].

Spectrograms are given by taking the absolute value of the amplitude of the Short-Time Fourier Transform (STFT). It does not contain any phase information. To reconstruct waveform by Inverse Short-Time Fourier Transform (ISTFT), phase information is necessary. Otherwise, the speech intelligibility of reconstructed waveform would be low. However, predicting phase from its corresponding amplitude is difficult. Griffin-Lim Algorithm (GLA) [14] is an iterative algorithm that retrieves phase from spectrogram only. It is a common alternative to neural vocoders in neural network-based speech processing models. However, GLA takes only spectrograms in Hertz-scale as input. For models that generate Mel-spectrogram as output, it requires an extra step to convert Mel-spectrogram to a linear spectrogram. There is no analytic method for this transformation, only approximation could be found. Phase retrieval is more difficult in this case.

2.5.1 Problem Definition and Algorithm

In this chapter, we review related mathematics of GLA. Denote spectrogram as $S = |STFT(x[n])|$. Phase retrieval problem is defined as followed:

Problem 1 *Given S, find a signal x^* that solve the following optimization problem:*

$$\min_x \| |X| - S \|$$

$$s.t. X \in \mathcal{S}$$

where X is time-frequency domain representation of x, $\mathcal{S}$ is a set of spectrograms

whose amplitude is the same as given spectrogram S.

Griffin-Lim Algorithm is described in Algorithm 1, proof of global convergence followed.

Algorithm 1 Griffin-Lim Algorithm

Input: Spectrogram S
Output: Signal x_i
  Randomly initialize phase p_0
  Construct signal $x_0 = ISTFT(Se^{jp_0})$
  for each iteration i **do**
    $STFT(x_{i-1}) = s_{i-1}e^{jp_{i-1}}$
    Replace s_{i-1} by S
    $x_i = ISTFT(Se^{jp_{i-1}})$
  Until converge

2.5.2 Convergence Analysis

GLA is proved to have global convergence to a critical point by the authors [14], by showing the distance to critical point decreases in each iteration. We simplify the proof of convergence and present it below.

Denote time domain signal $x(n)$, $x_w(mS, l) = w(mS - l)x(l)$. Its STFT $X_w(mS, \omega)$ and inverse transform are given by Discrete-Time Fourier Transform (DTFT) and Inverse DTFT (IDTFT) in formula 2.5 and 2.6. S is a positive integer representing sampling period of $X_w(n, \omega)$ in n, $w(n)$ is window function. In this proof, we name this $X_w(n, \omega)$ as complex spectrogram for convenience.

$$X_w(mS, \omega) = \sum_{l=-\infty}^{\infty} x_w(mS, l)e^{-j\omega l} \tag{2.5}$$

$$x_w(mS, l) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X_w(mS, \omega) e^{j\omega l} d\omega \tag{2.6}$$

While we can do STFT to an arbitrary $x(n)$ for its $X_w(n, \omega)$, any arbitrary $X_w(n, \omega)$ may not be a valid complex spectrogram in the sense that there may not exist a real signal $x(n)$ whose STFT is $X_w(n, \omega)$.

Lemma 1 *We define the distance between a signal and a complex spectrogram as followed.*

$$\begin{aligned} D\left[x(n),\ Y_w\left(mS, \omega\right)\right] &= \sum_{m=-\infty}^{\infty} \frac{1}{2\pi} \int_{-\pi}^{\pi} \left|X_w\left(mS,\ \omega\right) - Y_w\left(mS, \omega\right)\right|^2 d\omega \\ &= \sum_{m=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \left[x_w\left(mS, l\right) - y_w(mS, l)\right]^2 \end{aligned}$$

Given a fixed $Y_w(mS, \omega)$, minimization of $D[x(n)]$ can be solved by taking gradient with respect to $x(n)$ and solve the equation of that gradient equals to zero, which gives:

$$x(n) = \frac{\sum_{l=-\infty}^{\infty} w(mS - n) y_w(mS, n)}{\sum_{l=-\infty}^{\infty} w^2(mS - n)}.$$

While initializing GLA, we have a fixed $|Y_w(mS, \omega)|$ as input. This spectrogram can be considered as a circle on polar complex plane. Its centre is the origin, and its radius is the magnitude of $Y_w(mS, \omega)$. This circle can also be considered as the constraint of solution set.

In the i-th iteration of GLA, we have a time-domain signal $x^i(n)$ and its corresponding $X_w^i(mS, \omega)$. To minimize the distance $D[x^i(n), |Y_w(mS, \omega)|]$, the solution

is the followed:

$$\hat{X}_w^i\left(mS,\omega\right) = \left|Y_w\left(mS,\omega\right)\right| \frac{X_w^i\left(mS,\omega\right)}{\left|X_w^i\left(mS,\omega\right)\right|}.$$

However, as discussed above, $\hat{X}_w^i\left(mS,\omega\right)$ may not be a valid complex spectrogram. We can find the next iteration signal $x^{i+1}(n)$ by minimizing $D\left[x^{i+1}(n),\ \hat{X}_w^i\left(mS,\omega\right)\right]$ using Lemma 1. Thus, we have the following relationship:

$$\begin{aligned} D\left[x^{i+1}(n),\ \hat{X}_w^{i+1}\left(mS,\omega\right)\right] &\leq D\left[x^{i+1}(n),\ \hat{X}_w^i\left(mS,\omega\right)\right] \\ &\leq D\left[x^i(n),\ \hat{X}_w^i\left(mS,\omega\right)\right]. \end{aligned}$$

Since the phase of $X_w^i(mS,\omega)$ and $\hat{X}_w^i\left(mS,\omega\right)$ is the same, the distance between is only determined by their magnitude. $D\left[x^i(n),\ \hat{X}_w^i\left(mS,\omega\right)\right]$ can be written as

$$\begin{aligned} D\left[x^i(n),\ \hat{X}_w^i\left(mS,\omega\right)\right] &= \sum_{m=-\infty}^{\infty} \frac{1}{2\pi} \int_{-\pi}^{\pi} \left| X_w^i\left(mS,\omega\right) - \left|Y_w\left(mS,\omega\right)\right| \frac{X_w^i\left(mS,\omega\right)}{\left|X_w^i\left(mS,\omega\right)\right|} \right|^2 d\omega \\ &= \sum_{m=-\infty}^{\infty} \frac{1}{2\pi} \int_{-\pi}^{\pi} \left[\left|X_w^i\left(mS,\omega\right)\right| - \left|Y_w\left(mS,\omega\right)\right|\right]^2 d\omega. \end{aligned}$$

Hence, the convergence of GLA is proved by showing the distance between the sequence $\{x^i(n)\}$ and a fixed $\left|Y_w\left(mS,\omega\right)\right|$ decreases in every iteration, written as

$$D\left[x^{i+1}(n),\ \left|Y_w\left(mS,\omega\right)\right|\right] \leq D\left[x^i(n),\ \left|Y_w\left(mS,\omega\right)\right|\right].$$

Chapter 3

Methodology

3.1 Analysis of Griffin-Lim Algorithm

In this chapter, we will judge the usefulness of GLA in our project, especially as an alternative to neural vocoder for phase retrieval and waveform reconstruction from Mel-spectrogram. As stated in Chapter 2, the voice cloning model used in this project predicts Mel-spectrogram from text and speaker embeddings, and GLA predicts phase information from linear spectrogram magnitudes with global convergence to a critical point. We will describe the process of constructing waveform from Mel-spectrogram using GLA in this section. We will first show that linear spectrogram can be uniquely solved from Mel-spectrogram, then we will show that under certain conditions, STFT magnitude is a unique representation of signal. Hence, we conclude that there is a unique solution for waveform construction from Mel-spectrogram.

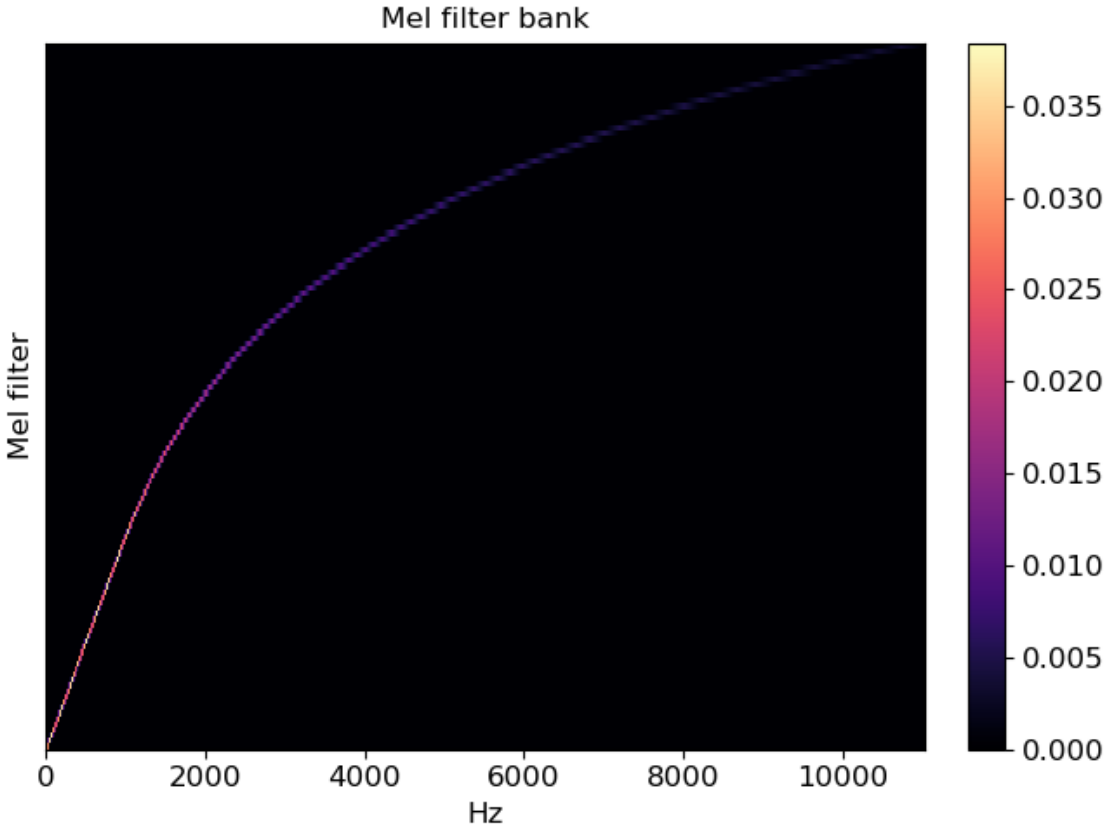


Figure 3.1: Visualization of digital Mel-filterbank

3.1.1 Convexity of Linear Spectrogram Approximation

Mel-spectrograms are given by simply passing linear spectrograms to a set of Mel-filterbank. Mel-filterbank is known as a set of triangular filters set according to Mel-scale. Figure 3.1 illustrated the idea of Mel-filterbank. The filter bank can be considered as a linear operator that transforms the frequency domain of a spectrogram from Hertz-scale to Mel-scale.

To the best of my knowledge, there is no analytic method to inverse the process. However, it is possible to solve the linear spectrogram from a given Mel-spectrogram

and Mel-filterbanks. We define the linear spectrogram approximation problem as below:

Problem 2 *Find a real number matrix S^* such that the Mel-spectrogram generated from it and the given Mel-spectrogram should be as close as possible.*

$$S^* = argmin \ \|FS - M\|_2^2 \, s.t. \ S \geq 0$$

where F denote given Mel-filterbank, M denote given Mel-spectrogram. All elements in S must be non-negative.

As a priori knowledge, it is also known that elements in both F and M are all non-negative by definition. Hence the problem can be redefined as a non-negative least squares (NNLS) problem at frame level as followed:

$$\underset{s \geq 0}{argmin} \left(\frac{1}{2} s^T Q s + c^T s \right)$$

where $Q = F^T F$, $c = -F^T m$. s and m is one frame of linear and Mel-spectrogram respectively.

NNLS problems have been proved to be convex due to the non-negativity constraints and Q being positive semi-definite [10]. Since every frame of spectrogram could be solved by convex optimization, we conclude that there exists a unique solution for linear spectrogram approximation, given a fixed Mel-spectrogram and Mel-filterbank.

3.1.2 Uniqueness of STFT Magnitude

The next step is to show the uniqueness of STFT magnitude (STFTM) to digital signal so that the critical point that GLA converge to is actually an optimum. Related work could be found in [32]. This work showed that under certain conditions, STFTM is a unique representation of signal. We present the key ideas in this section.

Lemma 2 *Denote $x[n]$ as a digital signal in the interval $0 \leq n \leq N$. Assume there are all zeros outside the interval and $x[0]$ is non-zero. The spectrum $|X(\omega)|$ and the first P samples points of $x[n]$, $0 \leq n \leq P \leq N$, uniquely specify the entire signal $x[n]$ if and only if $P \geq \lceil (N+1)/2 \rceil$.*

Proof 3.1.1 *First consider the case $P = \lceil (N+1)/2 \rceil$, where $M = N + 1$. Auto-correlation $R[n]$ of $x[n]$ is given by*

$$R[n] = x[n] * x\,[-n] = \sum_{m=0}^{M-1-n} x[m]x\,[n+m].$$

On the other hand, according to Wiener–Khinchin Theorem, this autocorrelation can also be computed from power spectra. Combined with the knowledge of the first

P samples in $x[n]$ a system of linear equations can be formed as the following:

$$\begin{bmatrix} x[0] & 0 & 0 & 0 \\ x[1] & x[0] & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x\left[(M/2)-1\right] & x\left[(M/2)-2\right] & \dots & x[0] \end{bmatrix} \begin{bmatrix} x\left[M-1\right] \\ x\left[M-2\right] \\ \vdots \\ x\left[M/2\right] \end{bmatrix} = \begin{bmatrix} R\left[M-1\right] \\ R\left[M-2\right] \\ \vdots \\ R\left[M/2\right] \end{bmatrix}.$$

Since the large matrix is a lower triangular matrix and $x[0] \neq 0$ by assumption, there is unique solution for $x[n]$ for $n = M/2, \dots, M-1$. The cases $P > \lceil (N+1)/2 \rceil$ can be proved similarly.

With the help of Lemma 2, the following conditions are proposed for ensuring uniqueness of STFTM:

1. Window function $w[n]$ has finite length $N_w > 2$
2. No zero in window function $w[n]$ for $0 \leq n \leq N_w$
3. At least 50% overlapping for window functions, denote as $L \leq \lfloor N_w/2 \rfloor$
4. Signal is one-sided
5. At most L consecutive zeros between any two non-zero sample points
6. Knowing first L consecutive samples of the signal starting from the first non-zero sample

With the help of Lemma 2, for any time index t, signal segment x_t can be solved from signal segment x_{t-1} and spectrogram frame s_t. Condition 6 above helps to solve the first segment according to the original work [32], but some scholars mentioned that condition 6 is not always necessary and claim that the above conditions are rather sufficient conditions than necessary conditions [46].

3.1.3 Numerical Experiments

Combining section 3.1.1 and 3.1.2, we concluded that GLA is sufficient for waveform reconstruction from Mel-spectrogram, at least conditionally. In this section, we justify the practical performance of GLA by conducting some numerical experiments. Before presenting the experiment settings and results. It is necessary to state that in the later part of this thesis, if there is no ambiguity, GLA would refer to the Fast-GLA [37]. The calculation is done by Python package *librosa* [28].

Algorithm 2 Fast Griffin-Lim Algorithm

Input: Spectrogram S
Output: Signal x_i
  Define $c_i = Se^{jp_i}$
  Randomly initialize phase p_0
  Fix $c_0 = Se^{jp_0}$
  for each iteration i **do**
    $t_i = STFT(ISTFT(c_{i-1}))$
    $c_i = t_i + \alpha_i(t_i - t_{i-1})$
    Update α_i
  Until converge

It can be easily shown that when $\alpha = 0$, Fast GLA reduces to the ordinary GLA. Readers may refer to the original document for the analysis of the value of α. In this thesis, it is set at $\alpha = 0.99$.

We present waveform reconstruction results measured by PESQ with different sizes of window function below. PESQ [39] is known as the international standard ITU-T P.862 that mimics Mean Opinion Score (MOS) in subjective listening test. In ITU-T P.862, PESQ is defined as a real-number score in the range $[1, 4.5]$ that measure speech quality in narrow-band between 300-3400 Hz. In P.862.2, a function is introduced to map PESQ to MOS score. In P.862.3, the PESQ-MOS is extended to wide-band assessment between 50-7000 Hz.

In all the experiments, sampling rate is fixed at 16000, hop size is set at one-quarter of window size, and the number of Mel Filter Bank Channel is fixed to be 80. Results presented below are calculated by a Python implementation of P.862.3 named *pesq*. Readers should also note that, due to the PESQ-MOS mapping function, the actual range of MOS is about $[1.04, 4.64]$. We call the scores presented as PESQ, although it is actually the MOS mapped.

The first experiment is to test the reconstruction ability of GLA from spectrograms computed with different window sizes. Considering the algorithm efficiency and sampling rate of signal, the window size we tested is from 2^8 to 2^{11}, meaning 16 ms to 128 ms in time domain. Additionally, we also tested window size 400 and 800, corresponding to 25 ms and 50 ms length. Fast GLA is run on an evaluation set of 22 speech data. Mean PESQ is plotted in Figure 3.2 every 10 iteration, from 10 to 200.

As shown in Figure 3.2, after 200 iterations, the algorithm converged or approached to a neighbourhood of optimum. The best performance is achieved by setting the window size to 512, resulting mean PESQ 4.47. Window sizes ranging from 400 to

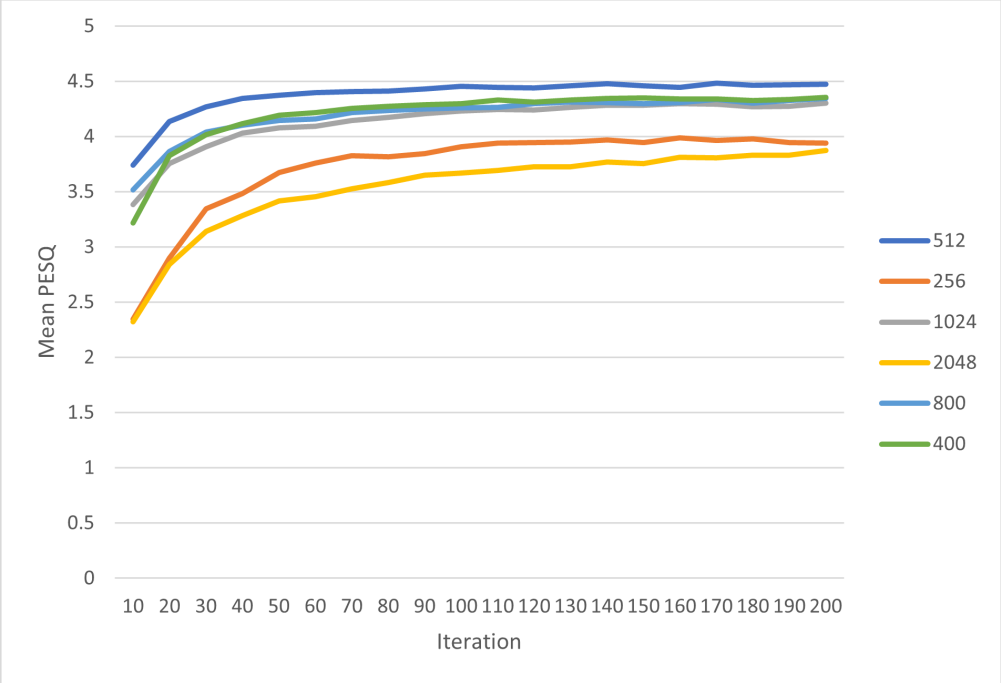


Figure 3.2: Result of waveform reconstruction from linear spectrogram

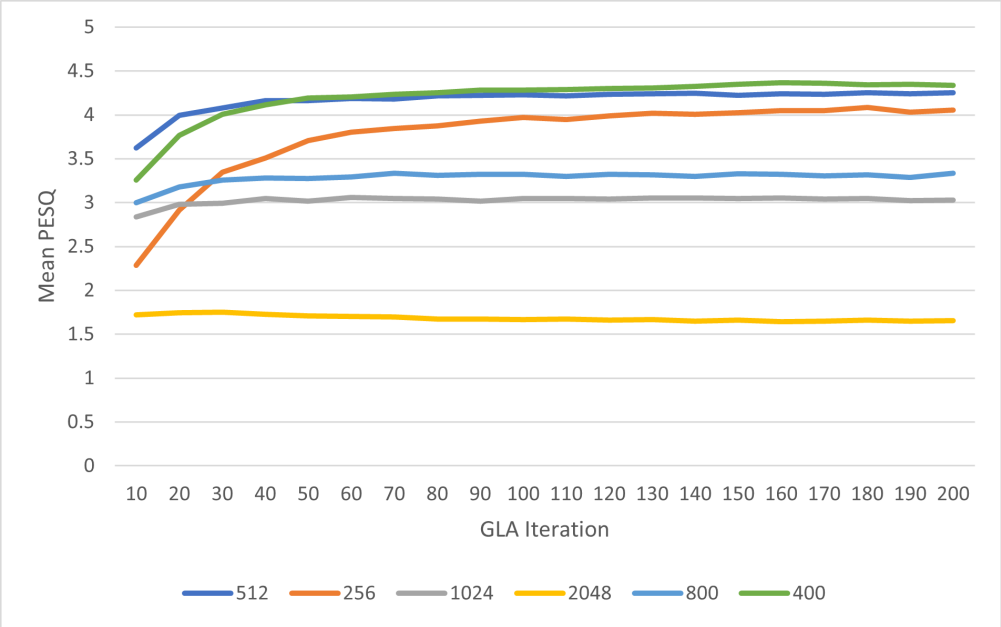


Figure 3.3: Result of waveform reconstruction from Mel-spectrogram

1024 have slightly lower performance, while extremely large or small windows, i.e., 256 and 2048, perform the worst.

We further arrange another experiment to investigate how approximating linear spectrogram from Mel-spectrogram affect the reconstruction ability of GLA. Results are plotted in Figure 3.3. The approximation is done by the method introduced in chapter 3.1.1. It is observed that a relatively small window size performs better in this task. Window length 512 is still one of the best window sizes in terms of PESQ, while length 400 is compatible with it in higher iterations. Note that length 800 and 1024, which have high scores in the previous test, perform significantly worse than other window lengths. Length 2048 performs the worst in both tests.

We argue that GLA, or at least Fast GLA, performs well, given that input spectrograms are accurate and under appropriate settings. The parameter optimized in the experiments is the window size of STFT. It is observed that $L = 512$ is the best window size under the current experiment environment. It resulted in high PESQ in both tests, suggesting that it can adapt to models that produce either linear or Mel-spectrogram. Window size 400 is the main competitor. However, considering the computation efficiency of FFT, setting window size as a power of 2 should be the first choice.

Moreover, reconstruction from Mel-spectrogram is in general less satisfying that that from linear spectrogram. It leads to a conjecture that convergence of GLA heavily depends on the input spectrogram since the algorithm aims to find a time-domain signal that could give an STFT outcome same as the input spectrogram. Hence, GLA does not function well if its input contains errors. We will investigate

this problem in the later part of the thesis.

3.2 Studies of Neural Network Models

In this section, we investigate the performance of RTVC, the popular open source model on the Internet, and present how we tried to improve the voice mimicking ability.

3.2.1 Performance Indicators

Before studying the model, we introduce the performance indicators used in this thesis. In many documents, the metric used for measuring the quality of generated speech is Mean Opinion Score (MOS). MOS is given by a subjective listening test by a group of listeners. However, it is time-consuming and costly for conducting a subjective listening test. Objective measurements are developed for quick evaluation. PESQ used in the previous section is one of those objective measures. It is widely used internationally but it is developed mainly for telecommunication situations. Some main limitations are: PESQ only accept audio in sampling rate of 8 kHz or 16 kHz; it requires a reference speech and is very sensitive to time alignment between reference and tested speech. In speech synthesis, it is rare that reference speech is available and synthesized speech could have a good time alignment with reference. Thus, PESQ is not a suitable metric in our case.

One commonly used objective metric is Mel Cepstral Distortion (MCD) [11], [47],

[22]. It is mainly used in voice conversion field. It first calculates Mel Cepstral Coefficients (MCEPs) from each frame of reference and tested speech, then compute root mean squared error (RMSE) among the two sets of MCEPs. Finally, MCD is given by taking average along frames and transforming into decibel scale. For reference, it is believed that automatic speech recognition (ASR) models could correctly recognize speech data whose MCD are below 8 [54]. In our thesis, we calculate MCEPs with the help of python packages *pyworld* [31], [30] and *pysptk*.

M-th order MCEPs $c_\alpha(m)$ is given by

$$H(z) = exp\sum_{m=0}^{M} c_\alpha(m)\tilde{z}^{-m}, \tag{3.1}$$

where

$$\tilde{z}^{-1} = \frac{z^{-1} - \alpha}{1 - \alpha z^{-1}}.$$

α here is the all-pass constant to be set by users. According to the manual of *pysptk*, 0.42 is set for audio samples in 16k Hz sampling rate. M denotes the order of MECP. In our case, M is set at 25. Furthermore, the dimension of a M-th order MCEP vector is $M + 1$. The extra zeroth cepstral dimension at $i = 0$ is said to be related to overall signal power [20].

Mel Cepstral Distortion is given by

$$\frac{10\sqrt{2}}{\ln 10}\frac{1}{T}\sum_{t}\sqrt{\sum_{i}\left(C_{ti} - \hat{C}_{ti}\right)^2}, \tag{3.2}$$

where C_{ti} and $\hat{C}_{ti}$ denote the MCEPs of frame t in reference and tested, i denotes the order of MCEPs taken from one frame.

MCD measures similarity by calculating the error between the acoustic features of two audio samples. One may notice that in equation 3.2, the length of reference and tested speech must be the same. It can be evaded by aligning reference and tested speech by Dynamic Time Wrapping (DTW) [40], [41] and calculate the normalized distance along the alignment path as an indicator.

DTW is a simple algorithm for aligning two time series. It detects similar patterns with different phases by shifting data points of the series. Given two time series $X = \{x_1,\ x_2, \ldots, x_I\}$ and $Y = \{y_1,\ y_2, \ldots, y_J\}$, where $I, J \in \mathbb{N}$, X and Y could be series of scalars or vectors. We can then define a distance measure between the data points in the two series as

$$d\left(i,j\right) = \left\|x_i - y_j\right\|.$$

By comparing all the data point pairs in the two sequences, we could have a local cost matrix

$$C \in \mathbb{R}^{I \times J} : c_{i,j} = d\left(i,j\right).$$

One may notice from this definition that the complexity of DTW is $O(IJ)$.

The DTW algorithm aims to align the two sequences by finding a path which runs through the low-cost areas in the cost matrix. The alignment path is a sequence of points $P = \{p_1, p_2, \ldots p_K\}$ where $p_k = (i,j)$. There are several constraints for this path:

1. Boundary condition: $p_1 = (1,1),\ p_K = (I,J)$

2. Monotonic condition: $i_{k-1} \leq i_k$ and $j_{k-1} \leq j_k$

3. Continuity condition: $i_k - i_{k-1} \leq 1$ and $j_k - j_{k-1} \leq 1$

Combining the above conditions, we can conclude that the relation between two consecutive points in the alignment path must be one of the followings:

$$p_{k-1} \begin{cases} (i_k, j_{k-1}) \\ (i_{k-1}, j_k) \\ (i_{k-1}, j_{k-1}) \end{cases}$$

We can then define a cost function as the weighted sum of cost along the alignment path:

$$\sum_{k=1}^{K} d(p_k) \cdot w_k.$$

w_k is a nonnegative weighting coefficient according to the step pattern used. Definitely, a longer sequence would result in a higher total cost. Thus, the objective of the DTW optimization is the time-normalized distance as followed:

$$D(X, Y) = \min_P \frac{\sum_{k=1}^{K} d(p_k) \cdot w_k}{\sum_{k=1}^{K} w_k}.$$

Since w_k is independent with the path P, the above objective function can be rewrite as

$$D(X, Y) = \frac{1}{N} \min_P \sum_{k=1}^{K} d(p_k) \cdot w_k,$$

where $N = \sum_{k=1}^{K} w_k$.

There are two basic types of step patterns: symmetric and asymmetric. For symmetric form, the weighting coefficient is defined as

$$w_k = (i_k - i_{k-1}) + (j_k - j_{k-1}) .$$

For asymmetric form, the weighting coefficient is defined as

$$w_k = i_k - i_{k-1}.$$

One may notice from the above definition that, the sum of weighting coefficient would be $N = I + J$ for symmetric form and $N = I$ for asymmetric form. Equivalently, one may also define $w_k = j_k - j_{k-1}$ and the sum of weighting coefficient would be $N = J$.

For the evaluation in our experiments, the symmetric step pattern is used. We used a python package *dtw-python* for DTW computation in this thesis [12].

Another objective metric used in this thesis is NISQA [29]. There are a few different versions of NISQA models. We employ the version built for TTS. It is a deep-learning-based method predicting subjective MOS from audio samples. The authors also addressed the problem that subjective listening tests for MOS are costly and existing objective assessment models such as PESQ and POLQA are not suitable for quality evaluation for synthetic speech since they are designed for transmission distortions.

Input signals to the model are first transformed into Mel spectrograms. Window functions of length 20 ms and hop size 10 ms are used in the computation. The Mel spectrograms are in 48 channels with a maximum frequency of 8 kHz as the training data used are mainly in 16 kHz sampling rate. The authors also mentioned that normalization of speech levels of input signals is not performed so that preprocessing procedure could be simplified, and the model would learn to handle it automatically. The spectrograms are then divided into segments with a fixed length of 150 ms as input to the network. As mentioned, the hop size of the spectrograms is 10 ms, resulting in an input size of 48×15.

NISQA model is composed of CNN and LSTM layers. The CNN consists of 6 convolutional layers. The number of filters in each layer is 16, 32, and 64 for the last 4 layers. The output features are then passed to a fully connected layer for a 20-dimensional vector representation before being passed into the LSTM layer. The LSTM layer is bi-directional with 128 cells. It considers the time dependencies in the feature sequence and evaluate the overall speech quality. Some common training techniques such as pooling, dropout and batch normalization are used. The authors made their codes open-sourced, readers may refer to the original documents and the codes for details.

Training and validation were done with numerous corpora such as data from Blizzard Challenge and Voice Conversion Challenge. For Blizzard Challenge, data of all the challenges from 2008 to 2019 are used except for 2017 and 2018. Similarly, for Voice Conversion Challenge, data from the 2016 and 2018 challenge were used. It is worth mentioning that, before training on synthetic speech quality evaluation, the authors pretrained the model with datasets from speech communication network

degradation domain. POLQA scores of those telecommunication datasets are used as MOS estimation in this pretraining. Using such a large amount of data resulted in a high correlation between predicted scores and subjective MOS. The authors reported an average correlation of 0.77 in testing data.

To study the evaluation ability of NISQA, we carried out a numerical simulation. We randomly chose 30 audio samples from the dataset and computed spectrograms from the samples. Then we reconstructed audio waves using Fast-GLA for 10 iterations and saved the reconstructed wave in each iteration. Thus, in total there are 300 products for scoring. Finally, we calculated PESQ and NISQA scores for each product.

Results are plotted as a scatter plot in Figure 3.4. The correlation coefficient is about 0.433. It is observed that there is a medium level of correlation between PESQ and NISQA. After performing linear regression analysis, a significant linear relationship between the two scores can be shown. However, the R^2 of the regression model is only 0.18, suggesting that the speech quality measured by PESQ and NISQA is not the same.

Another open source MOS prediction model is MOSNet [26]. After comparing the two models, we employ NISQA in this thesis since MOSNet tends to output scores around 3, as mentioned in the original paper.

Among the two indicators, MCD could mainly measure pronunciation and voice similarity, while NISQA focuses on the naturalness of speech, as mentioned in the original paper. MCD is calculated from MCEPs, but we consider that the idea could be similarly applied to other frame-level acoustic features, such as spectrograms.

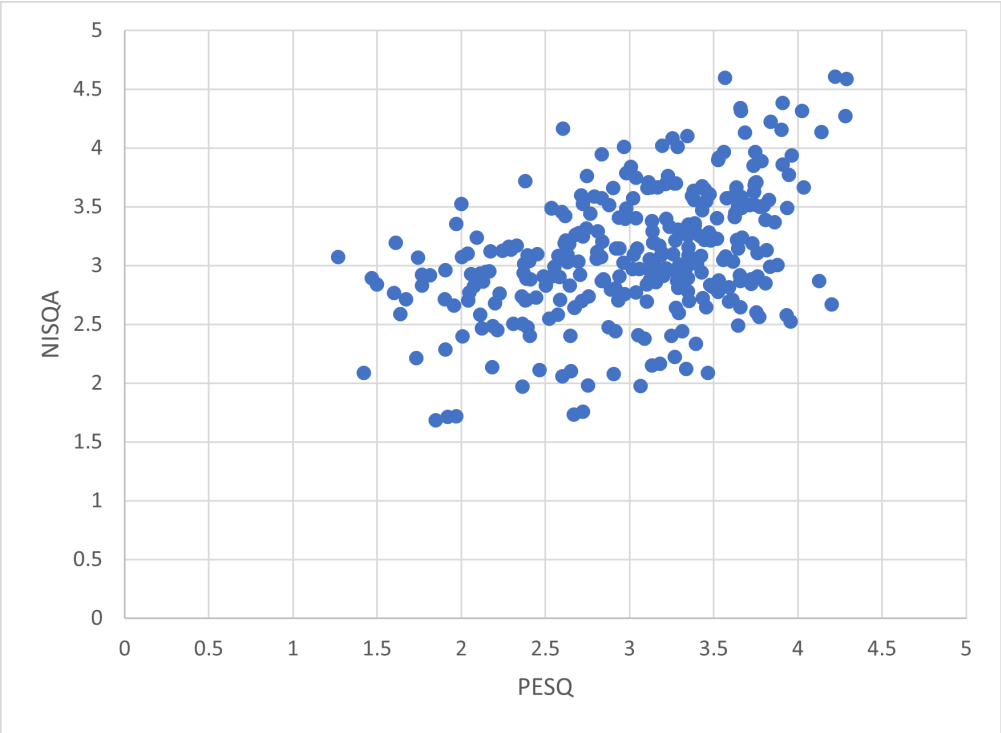


Figure 3.4: Scatter plot of PESQ and NISQA score

In this sense, we made the claim above since MCD focuses on the components of signals and NISQA is a regression model mimicking human perception. It is also supported by evaluation results presented later in this section.

3.2.2 Baseline Model Performance

In this subsection, we evaluate the baseline model accessible online with the indicators introduced above. MCD and NISQA are calculated from 30 synthesized audio samples. Those samples are mimicking 17 different speakers, including both male and female speakers. Mean and 95% confidence interval are reported in table 3.1.

In subjective listening, it is very clear that audio samples generated from GLA are full of artefacts, although the content is still audible. On the other hand, audio samples generated by neural vocoder have a lower level of artefacts.

Table 3.1: Mean and 95% C.I. of synthesized audio from baseline model

	MCD	NISQA
GLA	7.25 ± 0.343	1.47 ± 0.199
Neural Vocoder	5.86 ± 0.379	3.12 ± 0.110

As shown in table 3.1, a significant difference is recorded in mean NISQA scores, while the difference of MCD is relatively small. It could be explained by the introduction of evaluation measurements mentioned in the previous subsection that, MCD reflects pronunciation and voice similarity, and NISQA reflects the naturalness of speech. GLA generate samples have a similar mean MCD as neural

vocoder samples due to the audible content and similar voice characteristics. In contrast, the gap in NISQA comes from the high level of artefacts.

Despite the theoretical potential and the simulation results we presented in section 3.1, GLA could not show satisfactory performance in practical cases. This becomes one of the main topics we would like to study in this thesis. Another main topic is that for speakers unseen in training data, or the accent to mimic is different from training data, the model may not clone the voice well. Without developing new network architecture or gathering big data for further training, we hope to search if there are ad hoc methods to solve or alleviate the above problems.

Chapter 4

Results and Discussions

4.1 Transfer Learning and Finetune of Model

The method we employed to tackle the problems mentioned in the last part of Chapter 3 is transfer learning. Transfer learning helps to save costs, including the cost of data collection, computation power, training time, etc., by utilizing existing learning outcomes in the baseline model as initialization of new models. Good initialization is important in nonlinear optimization. Transfer learning could provide a certain level of guarantee on the learning outcome of the new model. In this section, we summarize the transfer learning done in our studies.

4.1.1 Modification of STFT Window Function

The first transfer learning trial is motivated by narrowing the performance gap of GLA between the simulation results in section 3.1.3 and the practical results in 3.2.2. The baseline model is trained with audio data in 16 kHz sampling rate, STFT window size used is 800 with frame shift 200. According to simulation results in section 3.1.3, it is shown that under this sampling rate and overlapping rate, window size 512 is a better choice in terms of phase retrieval. We reasoned that the low performance of GLA in section 3.2.2 is caused by the improper choice of hyperparameters, at least partially.

Thus, the first modification we made to the model is to finetune the model under this set of modified hyperparameters. Training data used in this trial is Librispeech [35]. This dataset is open-sourced and is also used by the baseline model. We made use of the two clean training data sets, namely *clean-100* and *clean-300*, that included around 400 hours of English speech data. We aimed to investigate how the window function setting would affect synthesis results, and therefore passed the synthesized Mel spectrogram to GLA for waveform reconstruction. In this sense, neural vocoders are not necessary. However, if readers are not using GLA and want to keep the neural vocoder structure used in the baseline model, it is necessary to train a neural vocoder correspondingly, since they must share the same set of hyperparameters.

4.1.2 Single Speaker Finetune

The second transfer learning we have done aims to boost the mimicking ability of the model. The learning outcome of baseline model may not generalize well, since it is trained with only the Librispeech corpus, in which speakers are mainly using the US accent, and the amount of data is not very sufficient. Synthesis quality may be unsatisfactory if the voice characteristic of the target speaker that users want to mimic is not similar to the speakers in training data.

We search for ad hoc methods, given there are only limited data on the target speaker. Using a model pretrained in large corpora as initial guess, and finetune using a small dataset is the method we studied in this thesis. The benefits of transfer learning are stated above. We expect that the pretrained model would have enough learning on basic TTS tasks, and extra training with the target speaker's data would specialize the model to the specific speaker. The training data used here could be relatively small, for instance, around 20 minutes. We consider this an ad hoc method used when the model on hand is unsatisfactory. If the users could afford it, training a model with large corpora that includes data from numerous speakers for well-generalized learning outcomes would be preferable.

4.2 Speech Quality

The first transfer learning done is very simple. Training data are preprocessed with a new set of hyperparameters and would be used to train a new synthesizer model, using baseline model parameters as initialization. This modified model is trained

for about 120k training steps. Although hyperparameters are changed in the hope of improving audio quality outputted by GLA based on the simulation result in section 3.1.3, we still prepared a neural vocoder for comparison.

Resulting MCD and NISQA MOS prediction can be found in Figure 4.1 and 4.2, respectively. From each model, 30 speech samples are synthesized by the baseline model and the modified model we trained. The set of 30 sentences is randomly sampled from 17 different speakers from the test corpus of LibriSpeech and the data are unseen in training. Scores calculated are presented in the form of box plot so that readers can have a more comprehensive idea of the distribution. Mean scores and 95% C.I. can also be found in Table 4.1. Note that the scores of baseline model have already been presented in section 3.2.2.

Table 4.1: Mean scores and 95% C.I. of synthesized audio from pretrained and finetuned model

	MCD	NISQA
Baseline with GLA	7.25 ± 0.343	1.48 ± 0.199
Baseline with Vocoder	5.86 ± 0.379	3.12 ± 0.110
Modified with GLA	7.14 ± 0.448	2.30 ± 0.219
Modified with Vocoder	5.54 ± 0.426	3.25 ± 0.138

MCD is not affected much by this finetuning process. It is shown in Figure 4.1 that MCDs of samples from both the baseline model and modified model are similar or slightly improved. However, using neural vocoder instead of GLA may lead to a certain level of improvement. For both models, samples generated using GLA have a mean MCD of around 7.2, and that of samples from neural vocoder is around 5.8 and 5.5.

On the other hand, the mean MOS predicted by NISQA is considerably improved

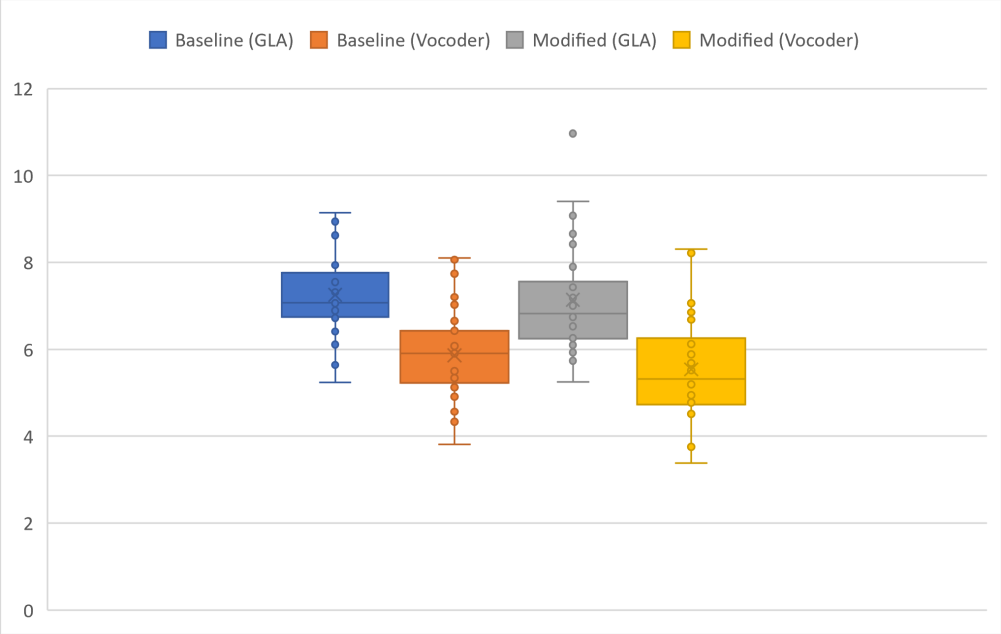


Figure 4.1: MCD of synthesized audio samples generated by baseline and modified model, using GLA and neural vocoder

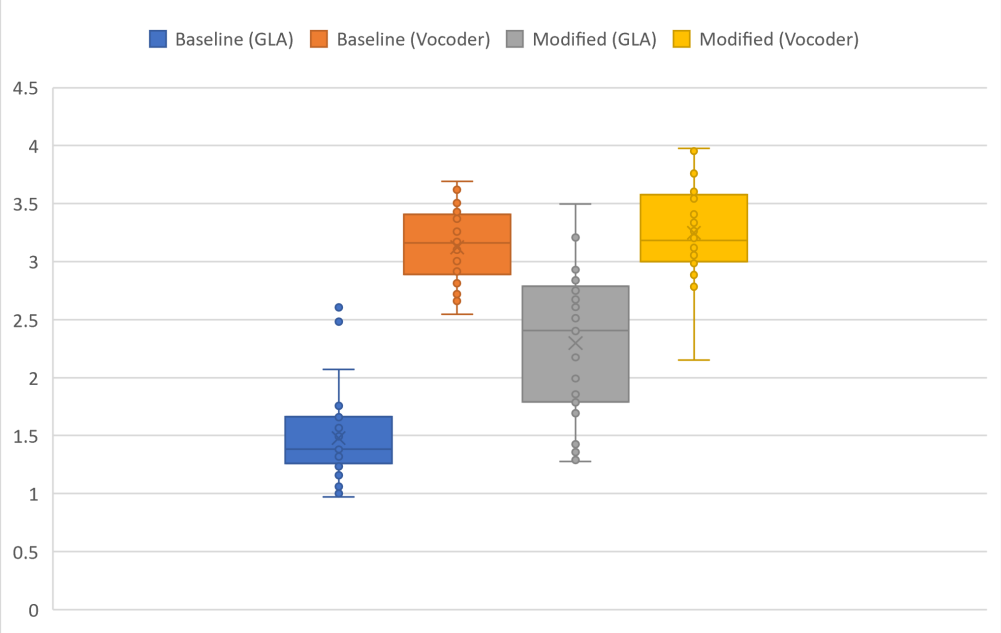


Figure 4.2: NISQA MOS prediction of synthesized audio samples generated by baseline and modified model, using GLA and neural vocoder

by this modification. From Figure 4.2, it is shown that the predicted MOS scores are originally about 1.5 in the baseline model and are enhanced to 2.3 in our modified model. Despite this significant difference, this result is still not yet lived up to our expectations. For comparison, audio samples generated using neural vocoder could have a mean predicted MOS slightly higher than 3.

4.3 Voice Similarity

The second transfer learning aims to boost voice similarity and speech naturalness for one single speaker by finetuning the model with only the target speaker's speech data. Training data used in this transfer learning experiment is from a female speaker chosen from the *train-other-500* dataset of LibriSpeech. In total there are about 24 mins speech data from this female speaker. We split it into training and testing sets, in which the length of testing data is around 5 mins.

For convenience, we call the model trained in section 4.1 as the "pretrained" model, and the single-speaker model trained in this section the "finetuned" model. The MCD and MOS predicted by NISQA is shown in Figure 4.3 and 4.4 as box plot, respectively. Mean and 95% C.I. is shown in Table 4.2.

Table 4.2: Mean scores and 95% C.I. of synthesized audio from baseline and modified model

	MCD	NISQA
Pretrained with GLA	6.34 ± 0.207	2.78 ± 0.067
Pretrained with Vocoder	6.55 ± 0.223	3.22 ± 0.051
Finetuned with GLA	5.90 ± 0.139	2.82 ± 0.063
Finetuned with Vocoder	6.20 ± 0.135	3.10 ± 0.086

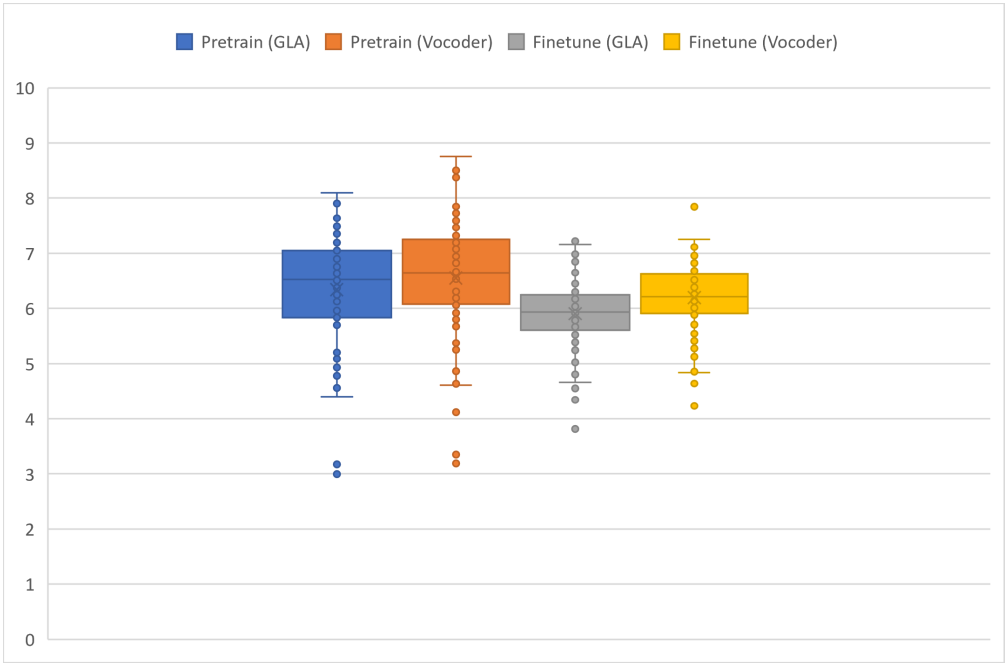


Figure 4.3: MCD of synthesized audio samples generated by pretrained and finetuned model, using GLA and neural vocoder

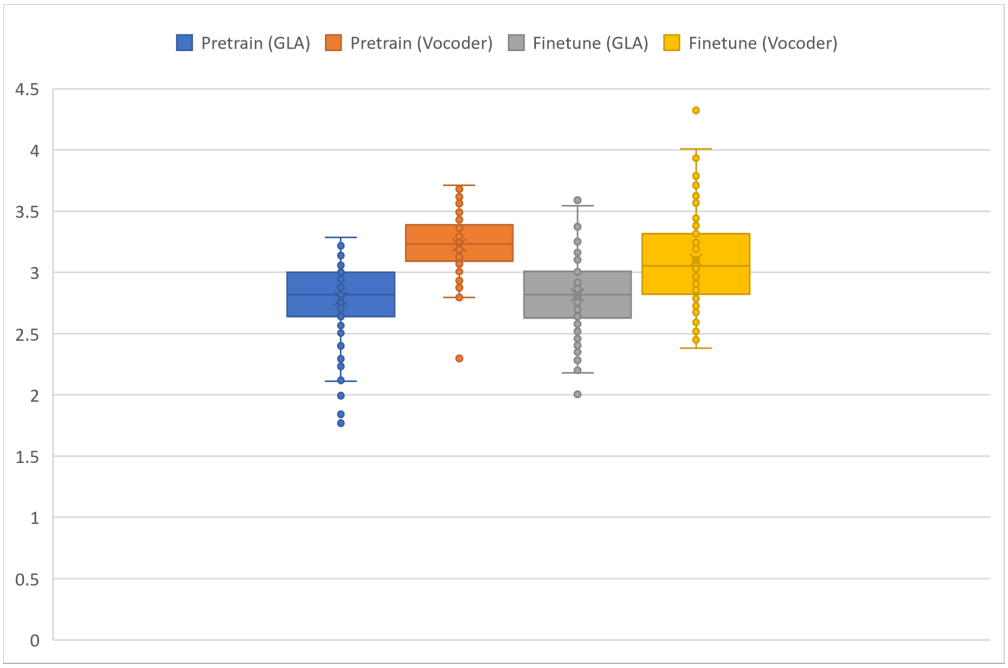


Figure 4.4: NISQA MOS prediction of synthesized audio samples generated by pretrained and finetuned model, using GLA and neural vocoder

After single-speaker finetune, MCD improved from 6.3 to 5.9 when using GLA for waveform construction. MOS predicted by NISQA remained at around 2.8. On the other hand, if neural vocoder is used, MCD is slightly higher than that of GLA, while MOS predicted by NISQA is better than GLA.

4.4 Attention Alignment

In this section, we wish to mention and discuss one more improvement brought by finetuning. From both literature and practical cases, it is known that the alignment generated in attention mechanism is an important performance indicator, in both training and testing time. However, there is no indicator for measuring its learning outcome and thus it can only be used as a brief impression of model convergence.

For speech processing problems, it is often required that the attention alignment should be monotonic, like a diagonal line. That means, the attention mechanism should concentrate on only a small set of key vectors at each decoding step, and it should gradually shift its concentration from the beginning to the end of input sequence. Figure 4.5 is an example of expected attention alignment. It can be observed that high attention weights are distributed to only a few key vectors, and the high weight values are shifting monotonically through the whole decoding process. In contrast, figure 4.6 shows an example of negative example. In the middle of decoding process, roughly between step 50 and 125, it is obvious that the attention mechanism lost its concentration and allocated relatively low attention weights to a large group of key vectors. As a result, a long pause is produced

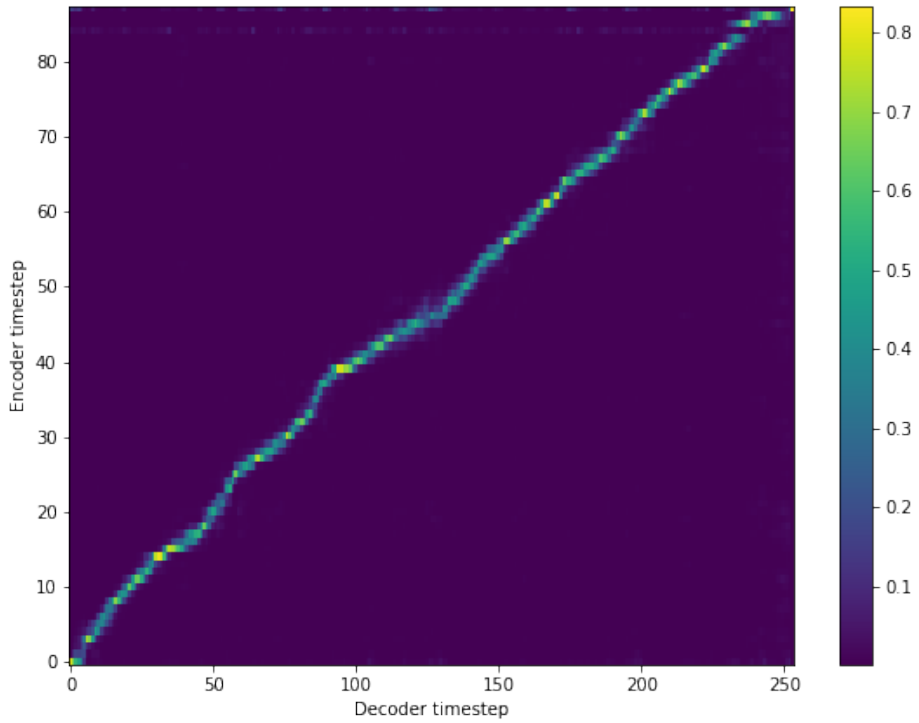


Figure 4.5: Example of a decent attention alignment

between the first and second half of synthesized speech.

It is found that the problematic attention alignment often happened in the pretrained model, and has been improved after the finetuning process. As stated, there is a lack of objective indicators for measuring the condition of attention alignment. Thus, we count the frequency of problematic alignment so that readers can have a brief impression. While synthesizing test data with the pretrained model, about 64% of the attempts faced this blurred alignment problem. On the other hand, the frequency of encountering this problem was reduced to about 19% in the case of finetuned model. It is worth mentioning that the percentage is counted regardless of the length of pause and the severity degree of bad attention. The counting also ignored the necessity of such pauses since there is no objective measure for it.

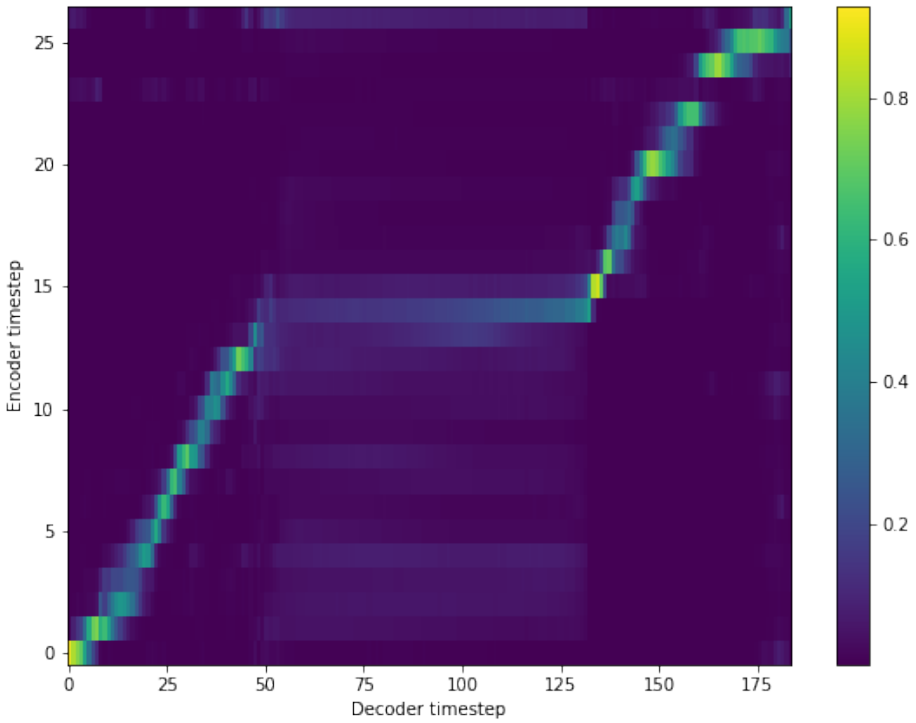


Figure 4.6: Example of a problematic attention alignment

Hence, there is a possibility that not every pause is undesirable.

Chapter 5

Potential Applications of Voice Cloning Systems

For applications of voice cloning, one may easily think of examples such as "deepfake", which refers to a kind of synthetic media that one's looks, facial expression, voice, etc., are replaced by someone else's. Certainly, this kind of application would cause many ethical and legal arguments that should be avoided in our studies.

In this chapter, we would like to propose a potential application of voice cloning for educational purposes, which is Automatic Pronunciation Error Detection (APED). Section 5.1 reviews the recent development in this field and introduces commonly used APED methods. Details of the proposed approach and experiment settings are presented in Section 5.2. Section 5.3 discusses the usefulness of the proposed approach, and its advantages and disadvantages over the existing methods.

5.1 Related Works

APED methods in the field could be divided into two streams: either based on automatic speech recognition (ASR) technology or based on acoustic phonetics. However, the two streams could be complementary to each other. Hence, readers should not be restricted to the boundary of the two streams while reading related literature or developing new APED methods.

ASR-based methods utilized ASR models to predict phonemes from inputting speech. The systems then determine whether there are pronunciation errors or not by comparing the recognized phonemes sequences with the phonemes sequences transformed from the target text that the speakers are expected to pronounce. Recent research has put efforts into training better ASR models. There are studies on the usage of attention-based sequence-to-sequence models such as [55] or transformer-like models such as [56].

Phonetics-based methods directly compare features extracted from students' attempts with reference speech recorded by teachers. The features to be compared in this approach could be acoustic features, perceptual features, etc. One common choice is Mel-frequency cepstral coefficients (MFCCs). After feature extraction, DTW is usually used to align data from students and teachers. An extra classification model is required for determining the existence and location of pronunciation errors. Some studies employed support vector machine (SVM) such as [23] and [24]. Deep belief network is also employed in some studies such as [25].

The advantage of ASR-based methods is that they can transfer learning from the

very active ASR field. Models and techniques of ASR are developed rapidly recently due to the deep learning boom, and many of them are also beneficial to APED. As an example, HMM-based ASR modules are replaced by DNN-based models. However, it is stated that ASR-based methods are slow since they require a large amount of computation, especially for deep neural networks.

In comparison, phonetics-based methods may be lighter, but there are still disadvantages. One of them may be the need of learning a classification model. This classification task may not share knowledge with other research topics, which leads to slower development and lesser resources such as training corpora. Another limitation is that reference speech is required to be compared. Students might only be able to practise the examples in database since no feedback could be given without reference data for comparison.

5.2 Experiments

Speech data of which the pronunciation to be checked are named query data in this chapter. Similarly, data that are known to contain correct pronunciation are named reference data. Pronunciation error detection is done by measuring the distance between query data and reference data. Data are segmented by window functions into small frames. Acoustic features are extracted from the frames and form vector sequences. Distance measures are then done between the two sequences. In our trials, MCD mentioned in equation 3.2 is used as the metric.

While the MCD calculation in equation 3.2 averaged the distortion in both feature

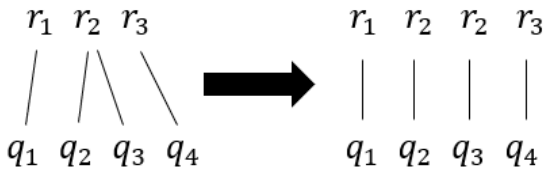
and time axes, in this application we aim to search for potential pronunciation errors by checking the locations where high distance are measured. Thus, MCD is calculated between the pairs of vectors from query and reference sequence and does not take average in the time direction.

It is predictable that in reality, the length of query and reference sequence may not be identical and equation 3.2 could not be applied directly. In Chapter 3, we employed the normalized distance of DTW [40] to avoid the problem. In contrast, in this chapter, we want to warp the reference sequence to fit into the shape of the query sequence. Therefore, asymmetric step pattern is used in DTW calculation so that each vector in the reference sequence is aligned with one vector in the query sequence.

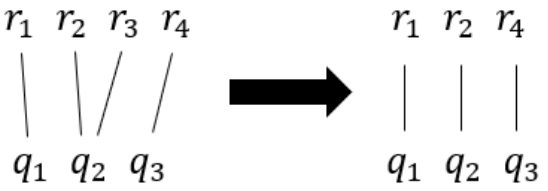
As illustrated in figure 5.1a, if the reference sequence is shorter than the query, one can simply duplicate the vectors in the reference sequence to fit the size of the query sequence.The case of long reference is slightly more complicated. One query vector could be aligned to multiple reference vectors. In our trials, we kept the middle aligned vector and ignore the rest, as illustrated in figure 5.1b, so that the number of vectors left in the reference sequence consists of that in the query.

After rectifying the shape of the reference, MCD is computed for each pair of vectors from the warped reference and query. The purpose of the whole process is to check how likely or unlikely is each vector in the query to exist in reference.

We conduct a series of simulations to support the usefulness of this approach. First, a set of testing sentences, including ten examples, is prepared. To mimic mispronunciation, one word in each testing sentence is replaced by another word



(a) Shorter reference



(b) Longer reference

Figure 5.1: Example of warping

with similar pronunciation. Both versions of the sentences are synthesised by the finetuned model mentioned in Chapter 4.2. Speech data synthesized from testing sentences are named as reference, and that from modified sentences are named as query in the following part.

Considering that there may be random errors in the synthesis process, for example, random initialization of GLA or the random sampling in neural vocoder, reference sentences were synthesized ten times and MCD is computed between each pair of reference and query so that random errors would be averaged out.

It is expected that the highest MCD would appear at the location of the exchanged word in the query since pronunciation errors would cause high acoustic feature errors. We evaluate our proposed approach by accuracy rate of error detection.

5.3 Simulation Results

In this section, we present the result of one example from the testing sentence set. Details of the testing sentence set and results of other examples in the set could be found in appendix.

The testing sentence used in this example is *"I HAVE PUT THE DATE OF THE PARTY DOWN IN MY DIARY"*, the modified version replaced *"DIARY"* by *"DAIRY"*. MCD calculation along query sequence is plotted in figure 5.2. The peak of MCD is recorded at frame index 260. One can check the correctness of this error detection by listening to the query audio around this frame index number. From only the figure, it is difficult to ensure this peak is at the exact location, but readers may briefly compare the location of the MCD peak and the location of exchanged words.

It could be easily imagined that errors in acoustic features might be caused by many factors, including the difference in voice characteristics, accent, intonation, etc. As a control group, we repeated the simulation with query speech synthesized by Google Translate. MCD computed is plotted on figure 5.3. The zeros after frame index 350 come from long silence. Still, the high values of MCD are not located in the changed part of the sentence. We considered this error detection failed.

In our first simulation, the MCD peak is found at the changed word for all 10 examples. In comparison, only 6 out of 10 success is recorded in the second simulation.

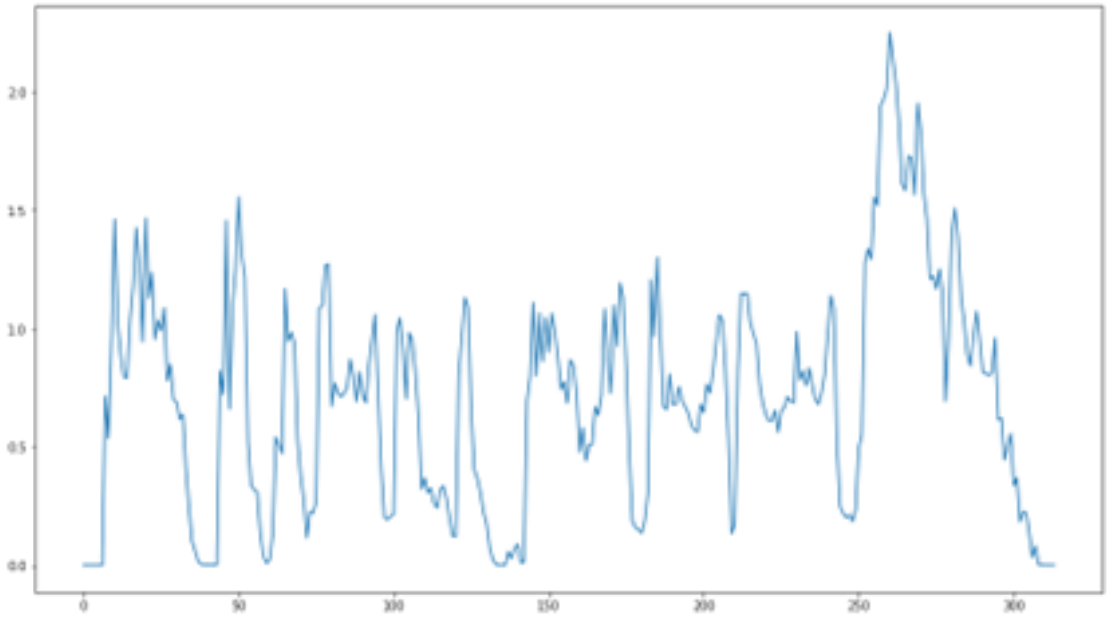


Figure 5.2: Example of pronunciation error detection via MCD

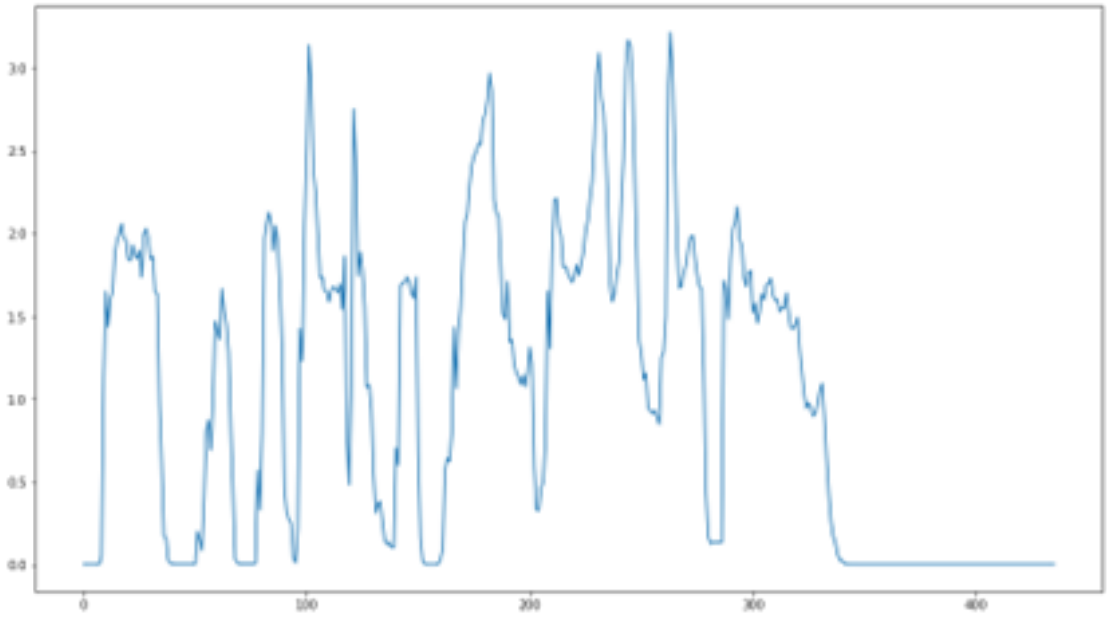


Figure 5.3: Example of pronunciation error detection in different voice

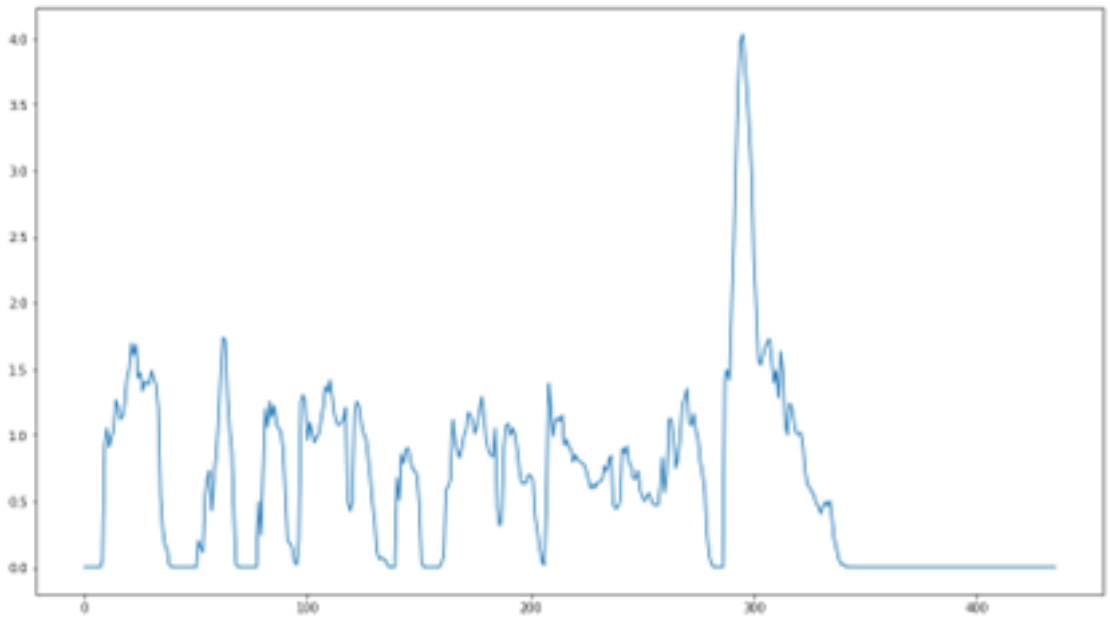


Figure 5.4: Example of pronunciation error detection after voice cloning

Voice Cloning techniques might be helpful in the sense that, by learning the voice of the speaker, errors from the difference in voice characteristics, accent, intonation, etc., would be minimized, and only the error comes from pronunciation differences, and maybe some random errors, would be left.

To justify this claim, we have trained another synthesis model to learn the voice of Google Translate and repeated the simulation again. Figure 5.4 shows the result of MCD matching after learning. It can be observed that the MCD peak is located at the exchanged word, and the value of the peak is relatively high and clearly distinguished from the others. After learning the voice characteristic, the correct rate returned to 10 out of 10. We consider this result shows that voice cloning plays a vital role in this pronunciation error detection.

5.4 Analysis

In the former part of this chapter, it is stated that the voice characteristic would affect the accuracy of pronunciation error detection. Analysis of the MCD values of synthesized speech data is done to support this statement.

As mentioned in the previous section, we have conducted 3 simulations to support our proposed method in total. In the first simulation, we synthesized both the reference and query data set with a random female speaker voice learnt by the finetuned TTS model discussed in chapter 4. In simulation 2, reference data set was the same as that in simulation 1 while the query data set was newly synthesized in another female voice using a service from Google. In the last simulation, we learnt the female voice provided in Google's service with our TTS model, and synthesized a new set of reference data. To summarize, in both simulation 1 and 3, the voice characteristic is expected to be similar in both reference and query data. In contrast, voice characteristic in reference and query set of simulation 2 is different.

MCD mean and variance of the data used in the above simulations can be found in table 5.1. The single speaker finetuned model in Chapter 4.2 is directly reused in simulation 1. Both reference and query data are synthesized speech and there is no ground truth speech for the calculation MCD. Thus, the mean and variance recorded here are repeated from the results in Chapter 4.2. The other two simulations are done between the speech data from Google Translate and our own synthesized speech. MCD is newly calculated using the method introduced in Chapter 3. For simulation 2, in which the voice of reference and query is not the same, the mean

Table 5.1: MCD mean and variance of the data used in simulations

	Simulation 1	Simulation 2	Simulation 3
Mean	6.19523	7.797666	4.559118
Variance	0.396654	0.273756	0.297676

MCD is about 7.8.

The voice of Google Translate is nicely cloned, supported by a mean MCD score of 4.5. This results in a higher distinguishing power in our error detection application. From the plots in appendix, we observed that in simulation 1, the value of the MCD peaks is around 2 to 2.5 in most cases, while the values other than the peak are around 1 to 1.5. On the other hand, peak values in simulation 3 are often higher than 3, even up to 4, resulting in a relatively larger difference between peak and other values.

From all these results, we have an inference that there exists a threshold that speech data with similarity above this threshold would result in detection failure. In our cases, similarity is measured by MCD and the highest mean score recorded without detection failure is 6.2 from the model used in simulation 1. Since the accuracy drops to 60% in simulation 2, where the mean score is 7.8, it is expected that the MCD threshold would be in between.

In our simulations, about 20 mins of data are used for the training of simulation 1 and about 17 mins of data are used for the training of simulation 3. However, considering real-life applications, it might be difficult to collect such an amount of data from users for finetuning models. Hence, we studied the effects of using fewer training data in the hope of finding out the minimum level of training data

Table 5.2: MCD mean and 95% C.I. of models trained from different amount of data

Data (mins)	MCD
0	5.171 ± 0.096
1	5.162 ± 0.092
2	4.967 ± 0.102
4	4.653 ± 0.097
8	4.623 ± 0.104
17	4.559 ± 0.107

necessary.

Reduced amounts of training data were used to finetune new models, and the resulting MCDs are shown in table 5.2. Data of 0 mins shown on the first row of the table refers to the pretrained model without finetuning. Finetuning with more data would result in better MCD, but the experiments on data amount of 4 minutes or more resulted in similar MCD. This trend is plotted in figure 5.5.

It is worth mentioning that, the number of epochs trained for each model in the table is not the same. Especially for small data cases, such as 1 min and 2 mins, a long training would cause side effects like mispronunciation or distortion of attention alignment. The results presented in this section are snapshots of the models before such problems appear.

Although the MCDs from all the models are relatively small, it is suggested to use no less than 4 minutes of data for finetuning. Using the 0, 1 and 2-minute model to repeat the APED simulations would result in large random errors that affect

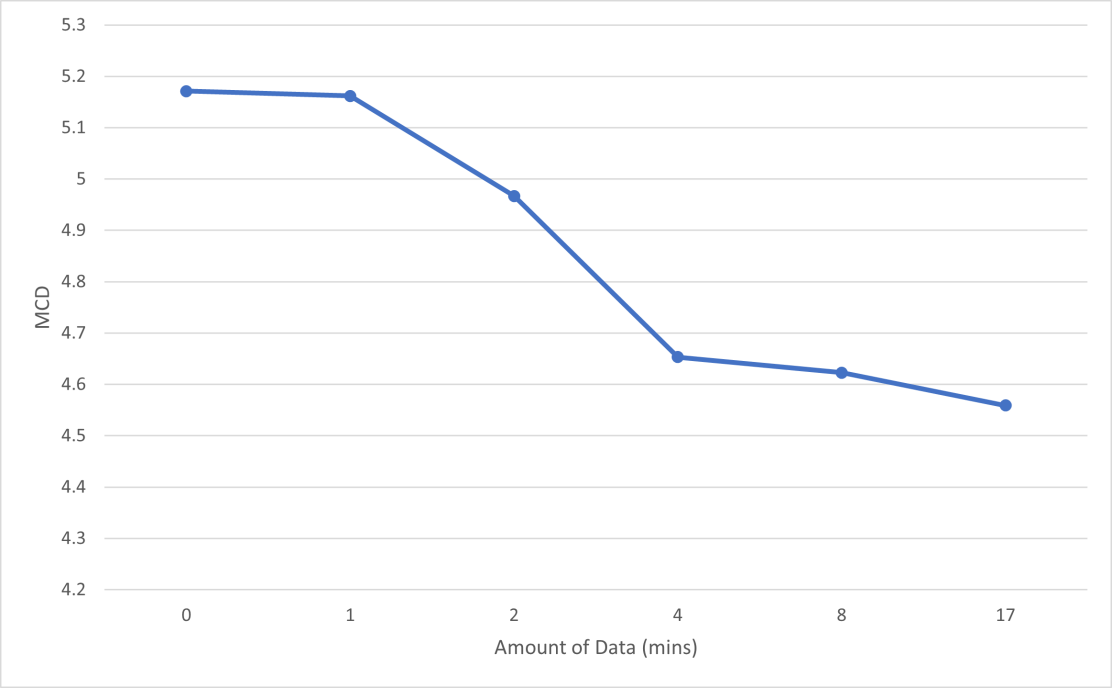


Figure 5.5: MCD of models trained from different amount of data

the accuracy of APED. Still, this may be avoided by using a well generalized and robust TTS model, so that finetuning is not necessary or finetuning with small dataset would not damage the learning outcome in pretrained model.

Compared to phonetics-based methods introduced in Section 5.1, the proposed approach is more flexible in the sense that phonetics-based methods need to record examples from teachers while the proposed approach could synthesize by itself. The feature comparison part is also simpler. Classifiers models, such as SVM, are needed to be trained in phonetics-based methods but distance functions are used instead in the proposed approach.

On the other hand, the proposed approach is easier to use compared to ASR-based methods because it does not require language knowledge like phonemes. The

proposed approach also shares the advantage of rich transfer learning opportunities since TTS is a topic as popular as ASR.

However, there are also limitations to the proposed approach. One of them is the long inference time. In our experiment, we synthesized 10 examples from each transcript and compared the examples to students' attempts. ASR-based methods only require one forward pass from their recognition model, but the proposed approach requires ten forward passes, which is very time-consuming. Another limitation is that a relatively large amount of data is required from the students for finetuning the TTS model. It is shown in our simulation that at least around 4 minutes of data would be necessary. However, this result came from a good initial state that MCD is already 5.2 before finetuning. In general cases, it is expected more data would be required.

Chapter 6

Conclusion

In this thesis, we provided a definition of voice cloning and introduced a popular open-source deep learning-based model. We have explored the history and literature of the neural network modules. We have also reviewed some necessary background knowledge in the signal processing field. We believe that the information provided in the thesis is sufficient for laymen to understand the development in the field.

One of the contributions made by this thesis is that we showed that the performance of the Griffin-Lim Algorithm varies with window function size in Short-Time Fourier Transform. Griffin-Lim Algorithm was a low-budget option for constructing waveforms from acoustic features synthesized by Text-to-Speech models in the past. However, along with the advance of deep learning-based vocoder, neural network-based models have become the major choice nowadays. Neural network-based models are powerful, but it is based on large training data and long training time. Extra training may also be required whenever users want to change the

hyperparameters inside for a new application. Compared to neural network-based models, Griffin-Lim Algorithm does not require training in advance and therefore is universal to spectrograms computed under any STFT setting.

The quality of speech constructed by Griffin-Lim Algorithm is considered to be inferior that that of neural vocoders. In this thesis, the quality of audio from Griffin-Lim Algorithm is improved by optimized the window size used in data preprocessing. Although the quality is still not as good as that of neural vocoder outputs, we have significantly narrowed the gap between. Since we have only optimized the window size in our studies, there is still a possibility that other factors that affect Griffin-Lim Algorithm performance could be found in future studies, so that Griffin-Lim Algorithm could again become a cost-saving alternative to neural vocoders.

Another contribution of this thesis is a newly proposed application of voice cloning techniques for education purposes. We found that pronunciation error detection by calculating the distance between acoustic features of reference speech and that of testing speech is not robust, because the difference in acoustic features could be caused by many factors, and pronunciation error may not be the dominating term.

We have shown that by learning from the speakers' speech samples, synthesis models could minimize the difference in voice characteristics, accent and intonation, and the acoustic feature error made by pronunciation mistakes could therefore dominate the total feature distance. This claim is supported by a series of simulations. The simulations provided evidence in the sense that the accuracy rate of pronunciation error detection is remarkably improved after voice cloning.

There are still limitations in this study and they suggest directions for further studies. First, only one acoustic feature is employed in our simulation. Other types of acoustic features should also be tested to understand their strengths and weaknesses. It may suggest the optimal choice of feature under different scenarios.

Second, we have still not yet found the threshold of similarity level necessary for robust detection, while we have gained some idea about the sufficient level. Knowing this threshold could help to save training resources as it helps to monitor the learning process of voice cloning. Early stop after the minimum amount of training becomes possible with this standard.

Moreover, in our simulation setting, it is known in advance that there is only one pronunciation error in each example so that we can focus on only the location from which the maximum feature distance is measured. In real-life cases, multiple errors may exist in the same audio, or there could be no error. It is necessary to study the value of distance measured to distinguish pronunciation mistakes from other differences.

Last but not least, the voice cloning model itself has room for improvement. The model used in this thesis has a classical encoder-decoder sequence-to-sequence structure. Using a more advanced model may lead to better results. For example, self-attention is a very active research topic in the field and has also been applied to speech synthesis tasks. Limited training resources, including training data and computation resources, may also be a problem. While this thesis is studying a row-resource case, the lack of resources may limit the model performance.

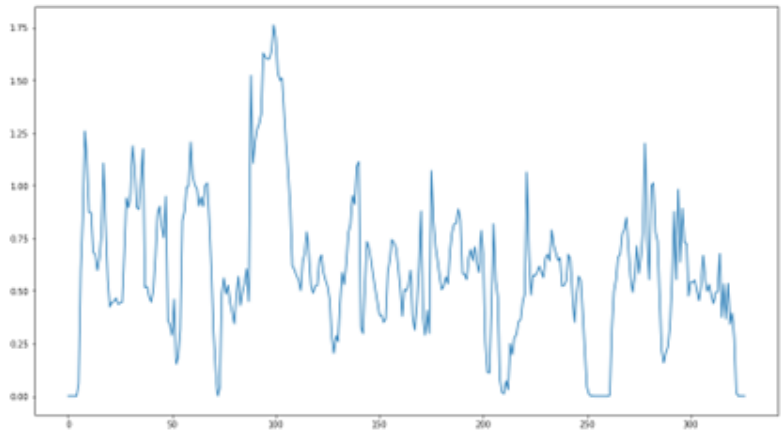
Appendix A

Details of Pronunciation Error Detection Simulations

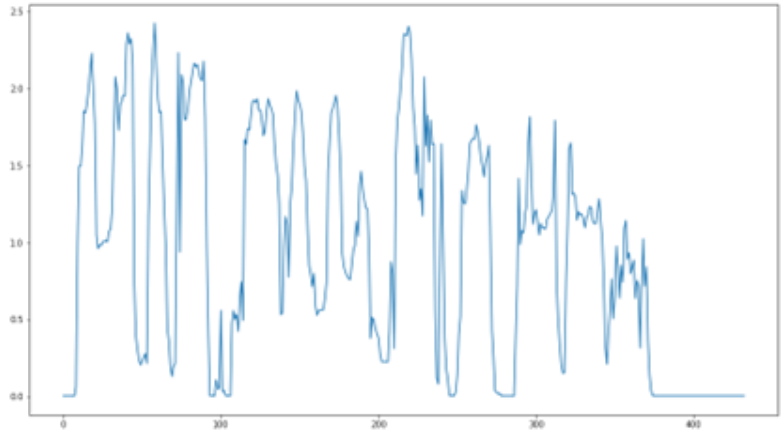
Here is the transcript of the test sentence set mentioned in Chapter 5. For each sentence, one of the words is replaced by the word in blanket to mimic mispronunciation. Results of each example sentence in the three simulations are attached in this chapter.

Table A.1: Transcript of testing sentence set

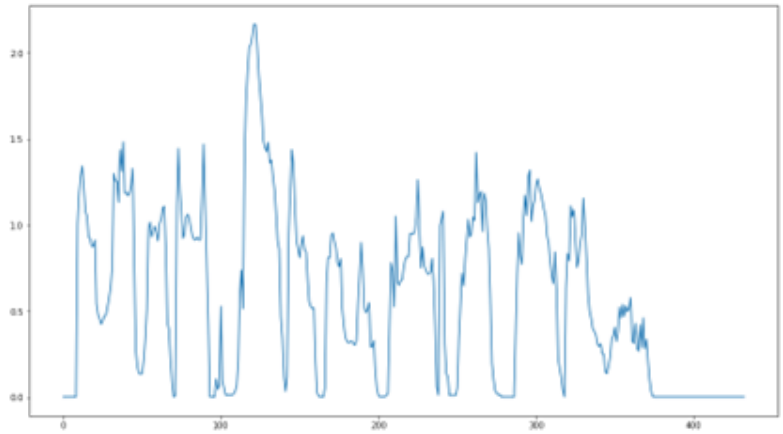
Index	Transcript
1	I NEVER GOT FURTHER (FARTHER) THAN THE FIRST FIVE PAGES
2	THE PRESIDENT HAS BEEN APPRISED (APPRAISED) OF THE SITUATION
3	THE PREFECTS (PERFECT) ARE KEY TO THE RUNNING OF THE SCHOOL
4	I HAD A VERY STRANGE DREAM (GYM) LAST NIGHT
5	HE SAT DOWN AND WIPED THE SWEAT (SWEET) OFF HIS FOREHEAD
6	FRESH OR DRIED FRUIT MAKES AN IDEAL SNACK (SNAKE)
7	I HAVE PUT THE DATE OF THE PARTY DOWN IN MY DIARY (DAIRY)
8	HE LOVES CHILDREN AND HAS A CERTAIN EMPATHY (SYMPATHY) WITH THEM
9	THE ACCENT (ASCENT) FALLS ON THE FINAL SYLLABLE
10	HE HAS BEEN UNDER A LOT OF PRESSURE (PLEASURE) RECENTLY



(a) Simlulation 1

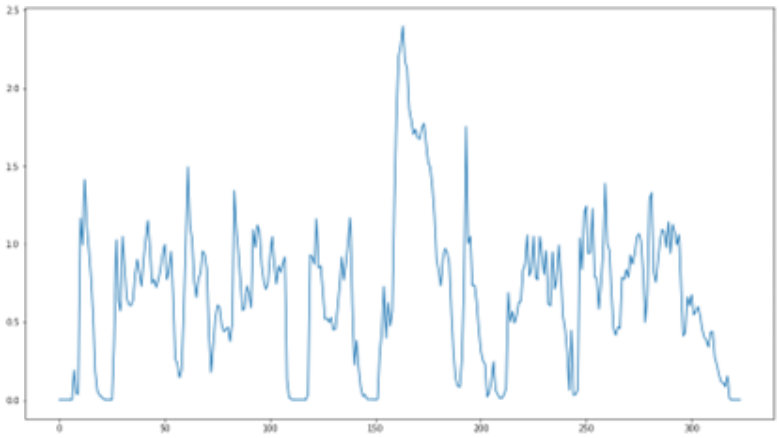


(b) Simlulation 2

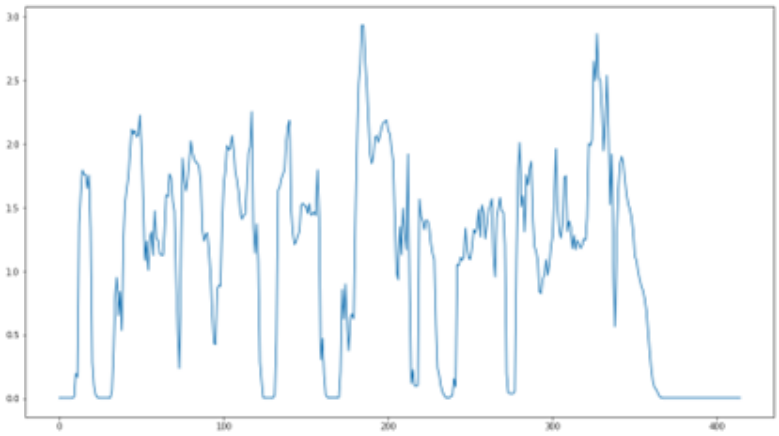


(c) Simlulation 3

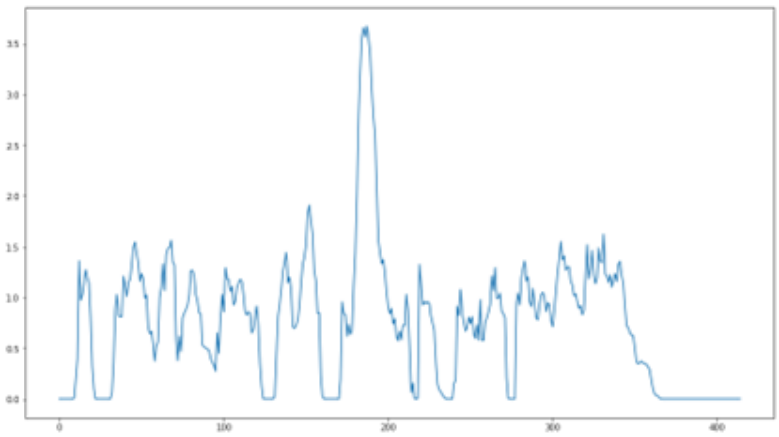
Figure A.1: Results of example index 1



(a) Simlulation 1

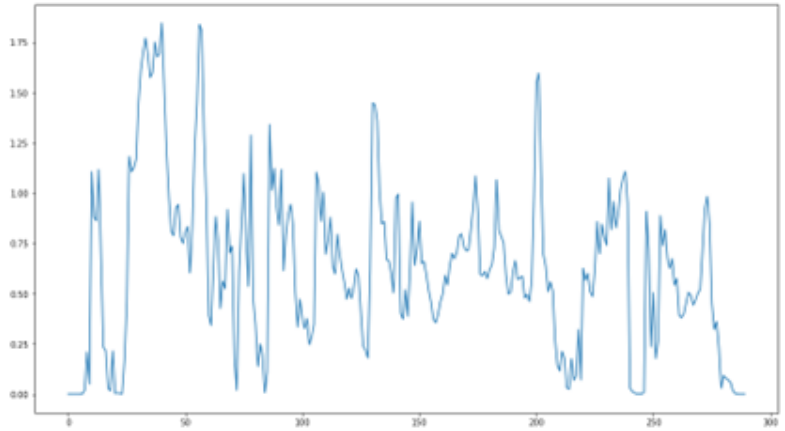


(b) Simlulation 2

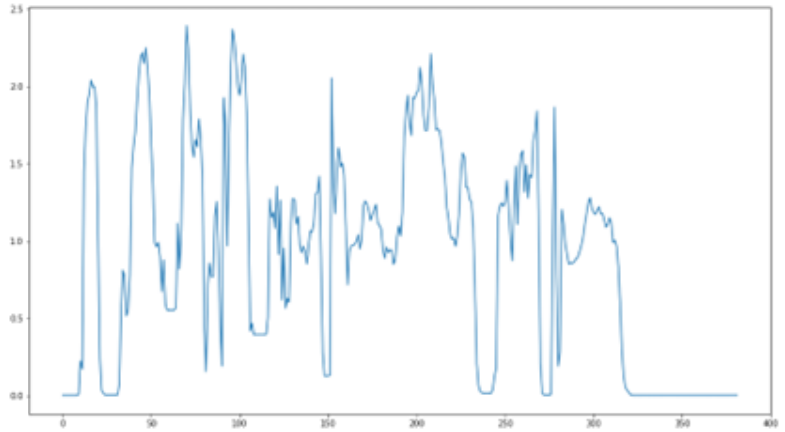


(c) Simlulation 3

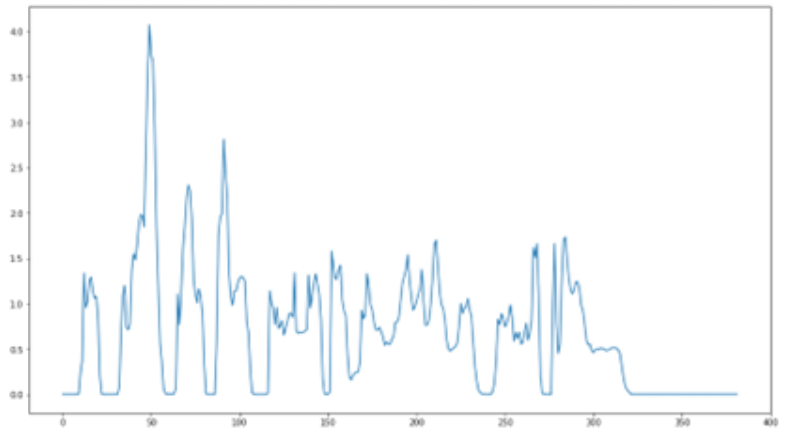
Figure A.2: Results of example index 2



(a) Simlulation 1

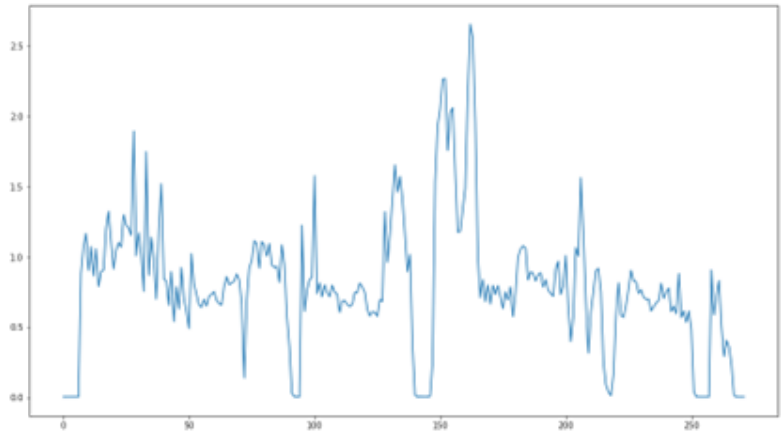


(b) Simlulation 2

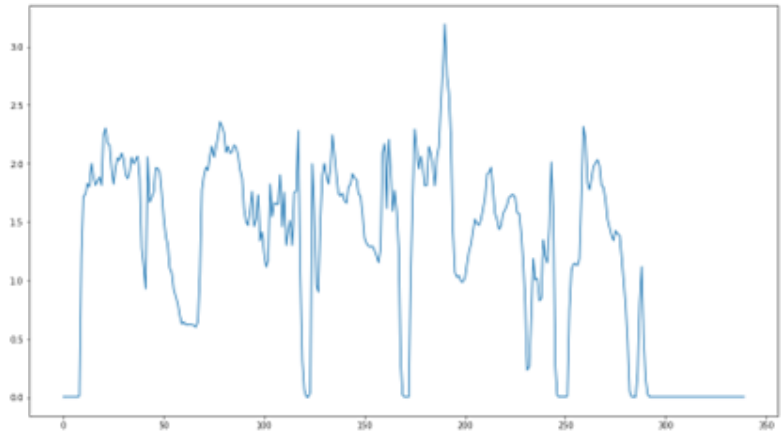


(c) Simlulation 3

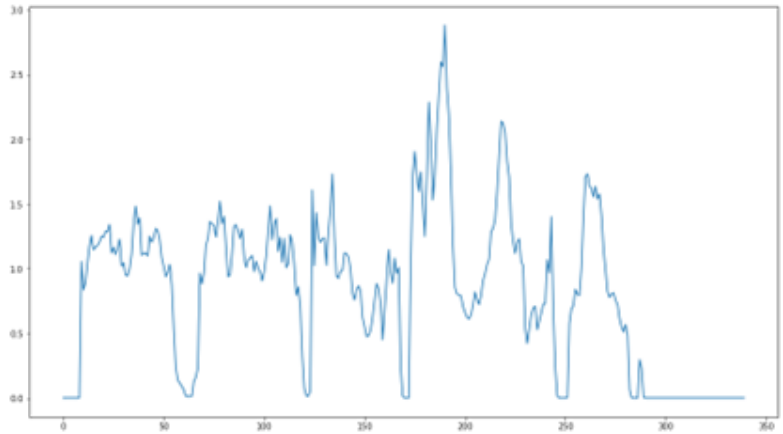
Figure A.3: Results of example index 3



(a) Simlulation 1

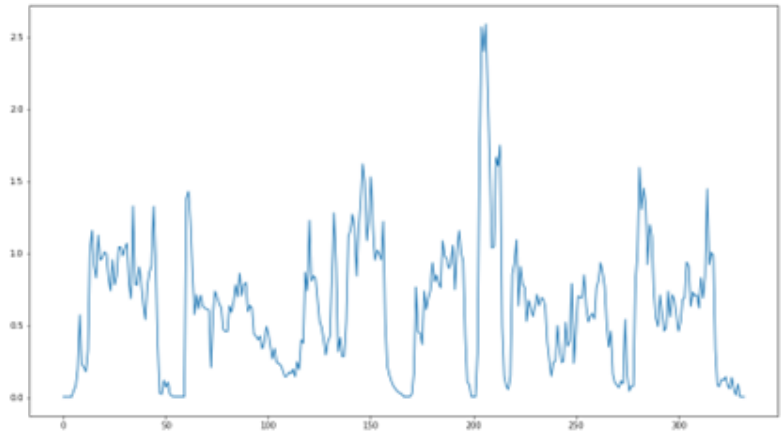


(b) Simlulation 2

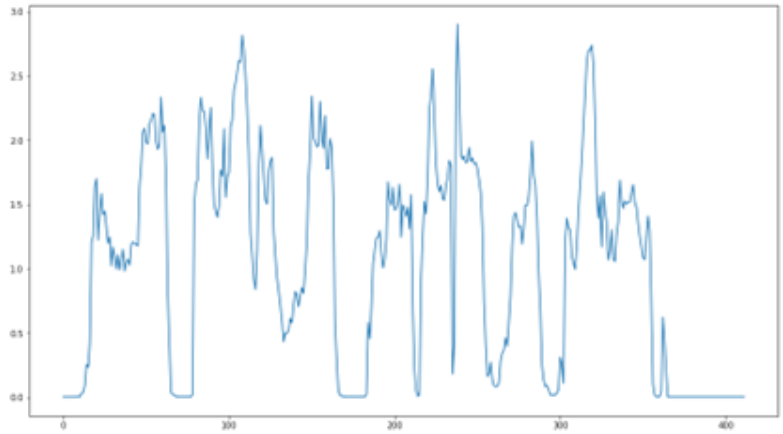


(c) Simlulation 3

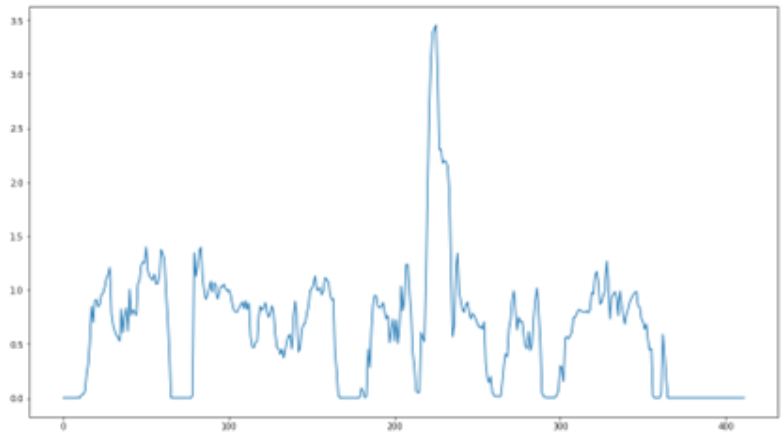
Figure A.4: Results of example index 4



(a) Simlulation 1

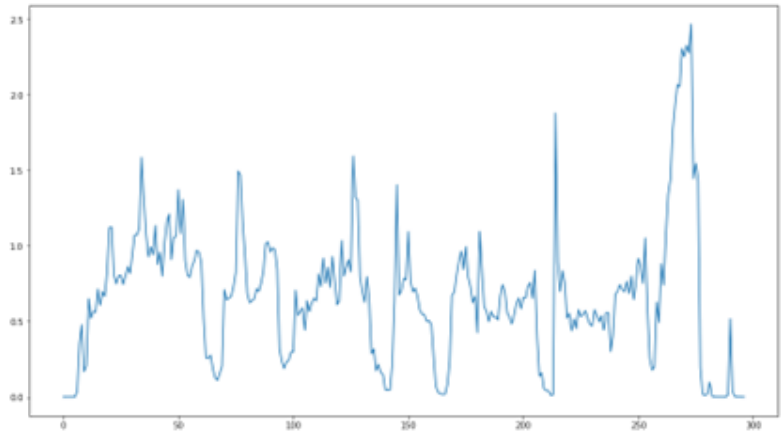


(b) Simlulation 2

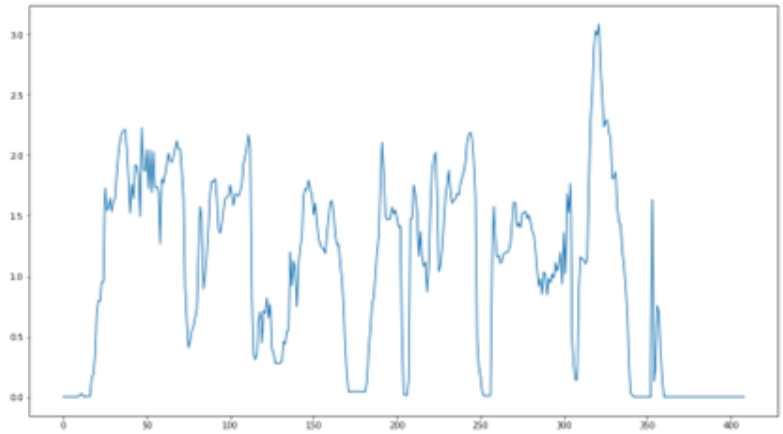


(c) Simlulation 3

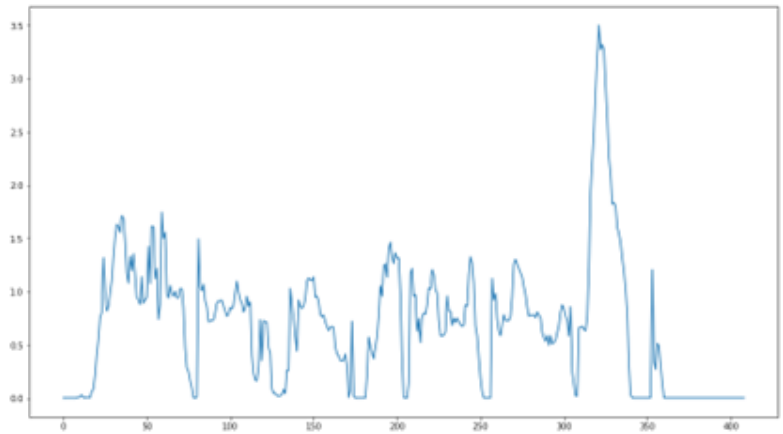
Figure A.5: Results of example index 5



(a) Simlulation 1

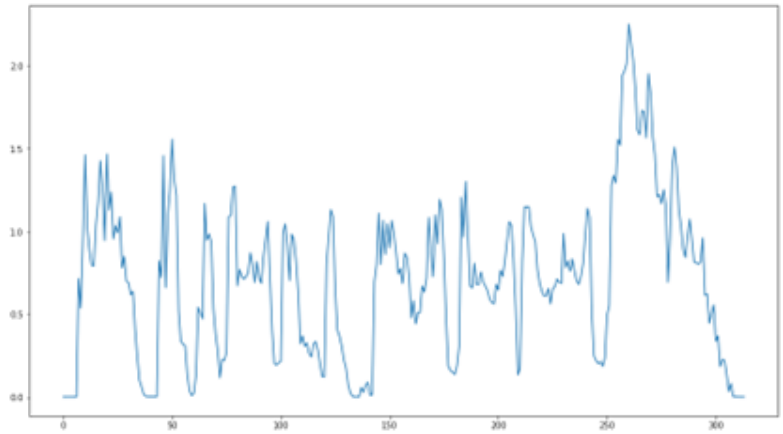


(b) Simlulation 2

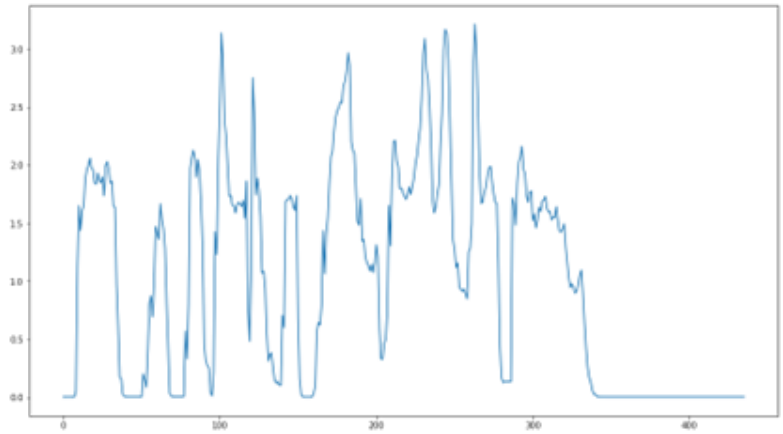


(c) Simlulation 3

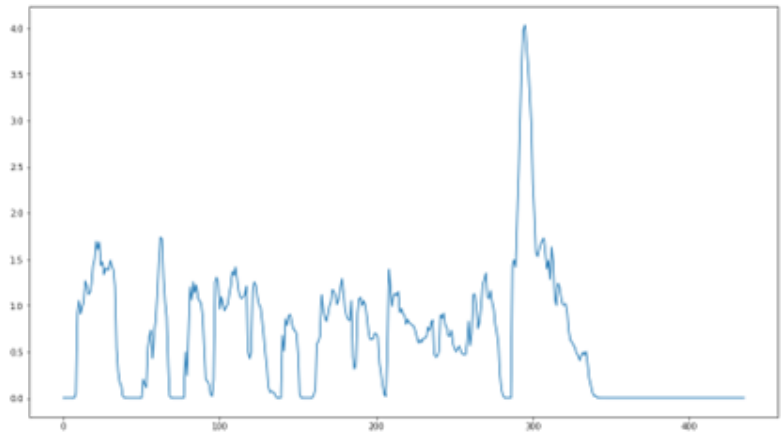
Figure A.6: Results of example index 6



(a) Simlulation 1

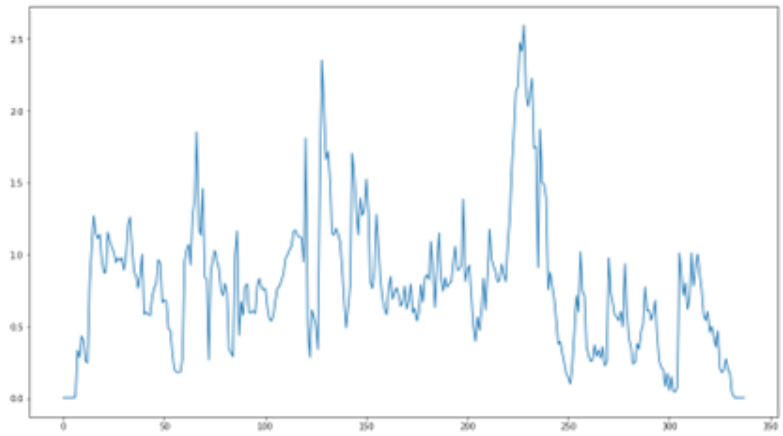


(b) Simlulation 2

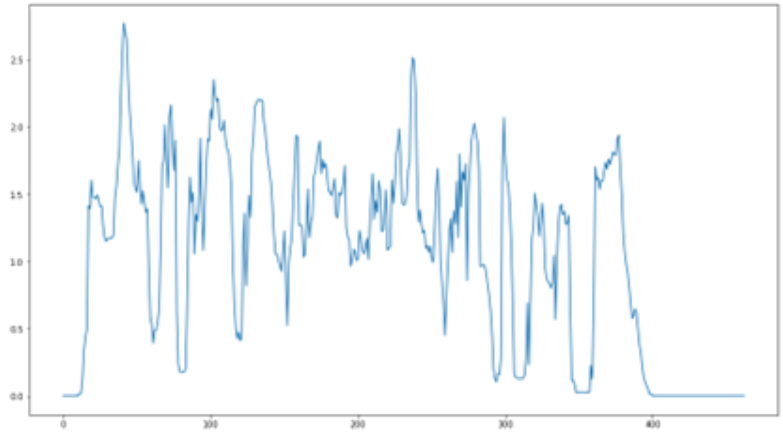


(c) Simlulation 3

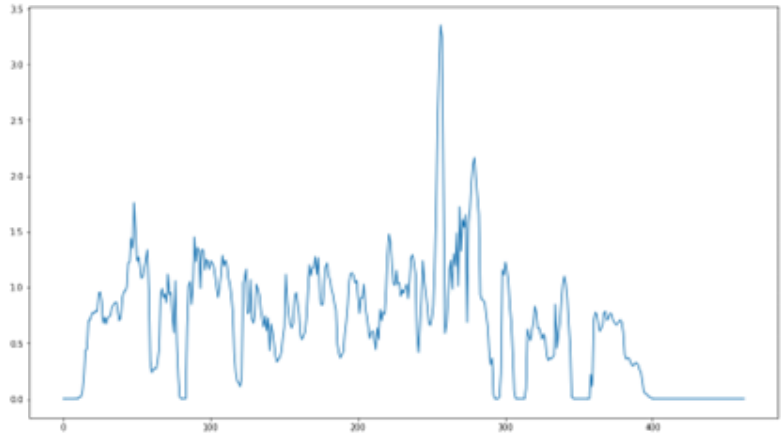
Figure A.7: Results of example index 7



(a) Simlulation 1

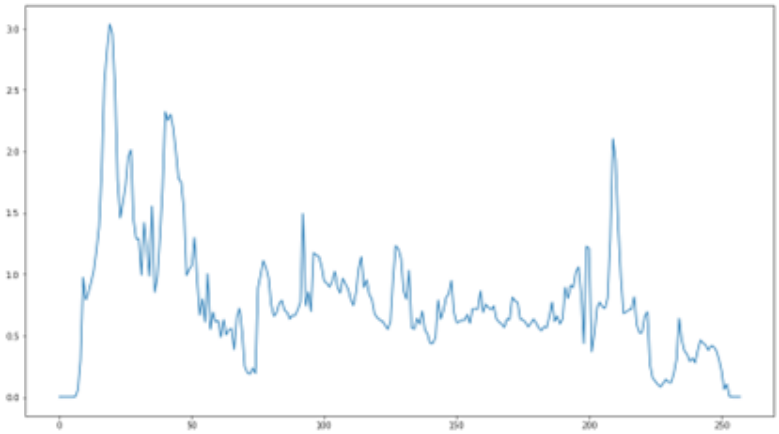


(b) Simlulation 2

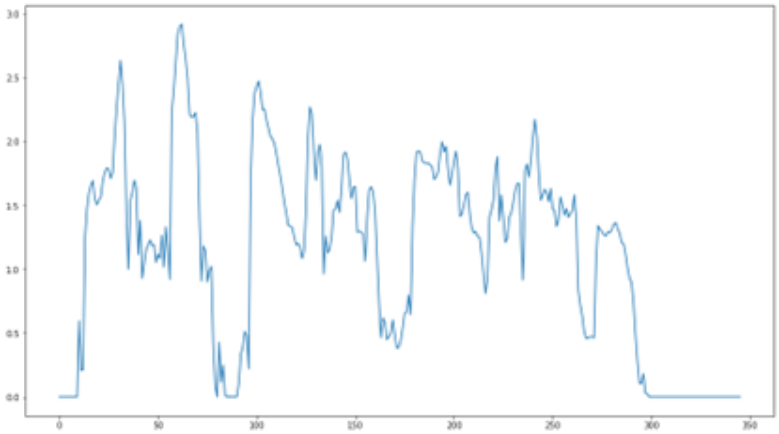


(c) Simlulation 3

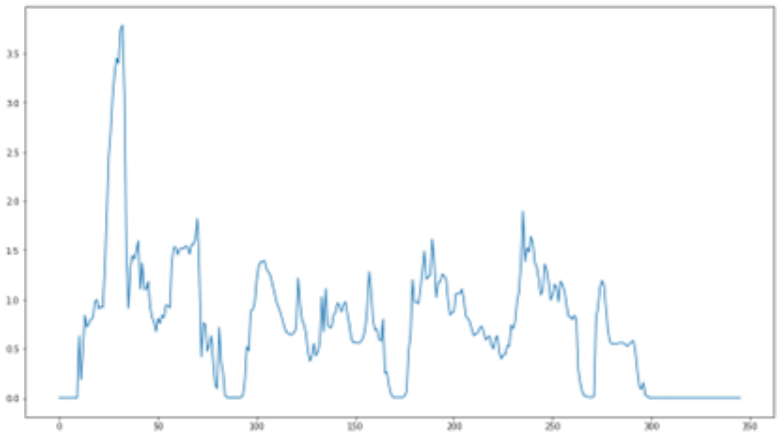
Figure A.8: Results of example index 8



(a) Simlulation 1

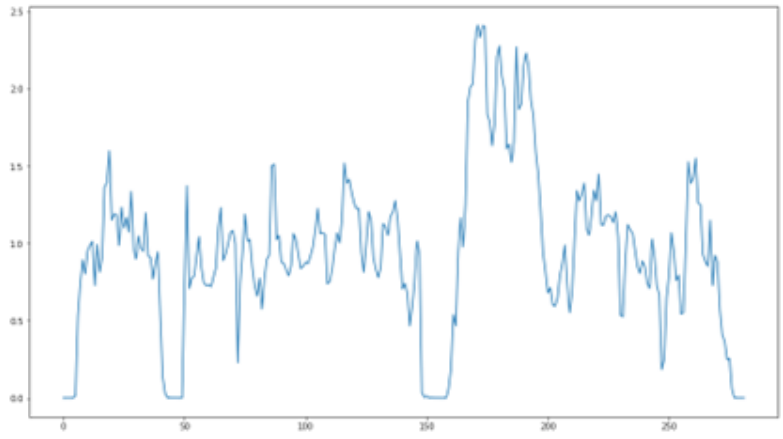


(b) Simlulation 2

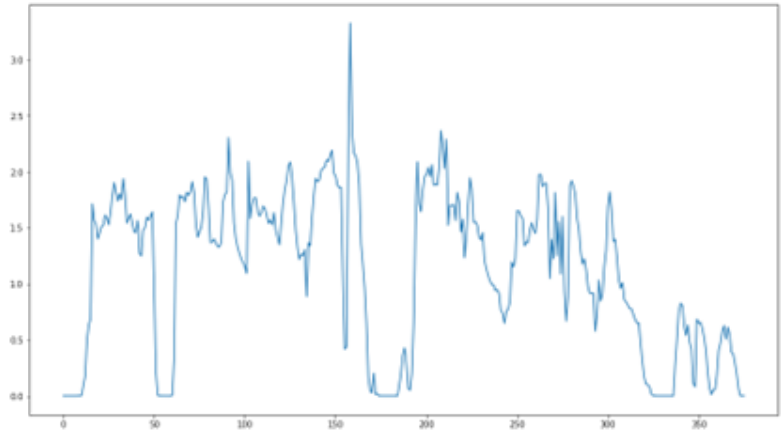


(c) Simlulation 3

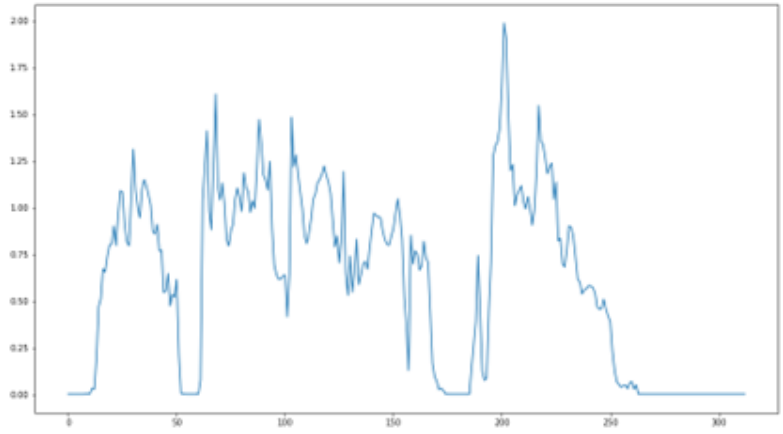
Figure A.9: Results of example index 9



(a) Simlulation 1



(b) Simlulation 2



(c) Simlulation 3, note that the word "recently" is not synthesized in this simulation

Figure A.10: Results of example index 10

References

[1] S. Ö. Arık, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman, et al. Deep voice: Real-time neural text-to-speech. In *International Conference on Machine Learning*, pages 195–204. PMLR, 2017.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[5] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

[6] J.-c. Chou, C.-c. Yeh, and H.-y. Lee. One-shot voice conversion by separat-

ing speaker and content representations with instance normalization. *arXiv preprint arXiv:1904.05742*, 2019.

[7] N. Dehak, R. Dehak, P. Kenny, N. Brümmer, P. Ouellet, and P. Dumouchel. Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification. In *Tenth Annual conference of the international speech communication association*, 2009.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] M. K. Fagan. Why repetition? repetitive babbling, auditory feedback, and cochlear implantation. *Journal of experimental child psychology*, 137:125–136, 2015.

[10] V. Franc, V. Hlaváč, and M. Navara. Sequential coordinate-wise algorithm for the non-negative least squares problem. In *International Conference on Computer Analysis of Images and Patterns*, pages 407–414. Springer, 2005.

[11] T. Fukada, K. Tokuda, T. Kobayashi, and S. Imai. An adaptive algorithm for mel-cepstral analysis of speech. In *icassp*, volume 92, pages 137–140, 1992.

[12] T. Giorgino. Computing and visualizing dynamic time warping alignments in r: the dtw package. *Journal of statistical Software*, 31:1–24, 2009.

[13] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[14] D. Griffin and J. Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on acoustics, speech, and signal processing*, 32(2):236–243, 1984.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[17] C. Jemine et al. Master thesis: Real-time voice cloning. 2019.

[18] Y. Jia, Y. Zhang, R. Weiss, Q. Wang, J. Shen, F. Ren, P. Nguyen, R. Pang, I. Lopez Moreno, Y. Wu, et al. Transfer learning from speaker verification to multispeaker text-to-speech synthesis. *Advances in neural information processing systems*, 31, 2018.

[19] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. Oord, S. Dieleman, and K. Kavukcuoglu. Efficient neural audio synthesis. In *International Conference on Machine Learning*, pages 2410–2419. PMLR, 2018.

[20] J. Kominek, T. Schultz, and A. W. Black. Synthesizer voice quality of new languages calibrated with mean mel cepstral distortion. In *SLTU*, pages 63–68, 2008.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with

deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[22] R. Kubichek. Mel-cepstral distance measure for objective speech quality assessment. In *Proceedings of IEEE pacific rim conference on communications computers and signal processing*, volume 1, pages 125–128. IEEE, 1993.

[23] A. Lee and J. Glass. A comparison-based approach to mispronunciation detection. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 382–387. IEEE, 2012.

[24] A. Lee and J. Glass. Pronunciation assessment via a comparison-based system. In *Speech and Language Technology in Education*, 2013.

[25] A. Lee, Y. Zhang, and J. Glass. Mispronunciation detection via dynamic time warping on deep belief network-based posteriorgrams. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8227–8231. IEEE, 2013.

[26] C.-C. Lo, S.-W. Fu, W.-C. Huang, X. Wang, J. Yamagishi, Y. Tsao, and H.-M. Wang. Mosnet: Deep learning based objective assessment for voice conversion. *arXiv preprint arXiv:1904.08352*, 2019.

[27] R. Mama. Tacotron-2. `https://github.com/Rayhane-mamah/Tacotron-2`, 2019.

[28] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25, 2015.

[29] G. Mittag and S. Möller. Deep learning based assessment of synthetic speech naturalness. *arXiv preprint arXiv:2104.11673*, 2021.

[30] M. Morise. D4c, a band-aperiodicity estimator for high-quality speech synthesis. *Speech Communication*, 84:57–65, 2016.

[31] M. Morise, F. Yokomori, and K. Ozawa. World: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems*, 99(7):1877–1884, 2016.

[32] S. Nawab, T. Quatieri, and J. Lim. Signal reconstruction from short-time fourier transform magnitude. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31(4):986–998, 1983.

[33] D. K. Oller and R. E. Eilers. The role of audition in infant babbling. *Child development*, pages 441–449, 1988.

[34] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[35] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[36] J. Patterson and A. Gibson. *Deep learning: A practitioner's approach.* " O'Reilly Media, Inc.", 2017.

[37] N. Perraudin, P. Balazs, and P. L. Søndergaard. A fast griffin-lim algorithm. In *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 1–4. IEEE, 2013.

[38] R. Prenger, R. Valle, and B. Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019.

[39] A. W. Rix, J. G. Beerends, M. P. Hollier, and A. P. Hekstra. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In *2001 IEEE international conference on acoustics, speech, and signal processing. Proceedings (Cat. No. 01CH37221)*, volume 2, pages 749–752. IEEE, 2001.

[40] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[41] P. Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23):40, 2008.

[42] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4779–4783. IEEE, 2018.

[43] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur. X-vectors: Robust dnn embeddings for speaker recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5329–5333. IEEE, 2018.

[44] J. Sotelo, S. Mehri, K. Kumar, J. F. Santos, K. Kastner, A. Courville, and Y. Bengio. Char2wav: End-to-end speech synthesis. 2017.

[45] C. Stoel-Gammon. Relationships between lexical and phonological development in young children. *Journal of child language*, 38(1):1–34, 2011.

[46] N. Sturmel, L. Daudet, et al. Signal reconstruction from stft magnitude: A state of the art. In *International conference on digital audio effects (DAFx)*, pages 375–386, 2011.

[47] K. Tokuda, T. Kobayashi, S. Imai, and T. Chiba. Spectral estimation of speech by mel-generalized cepstral analysis. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 76(2):30–43, 1993.

[48] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez. Deep neural networks for small footprint text-dependent speaker verification. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4052–4056. IEEE, 2014.

[49] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[50] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[51] L. Wan, Q. Wang, A. Papir, and I. L. Moreno. Generalized end-to-end loss for speaker verification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4879–4883. IEEE, 2018.

[52] W. Wang, S. Xu, B. Xu, et al. First step towards end-to-end parametric tts synthesis: Generating spectral parameters with neural attention. In *Interspeech*, pages 2243–2247, 2016.

[53] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017.

[54] C. Yan, G. Zhang, X. Ji, T. Zhang, T. Zhang, and W. Xu. The feasibility of injecting inaudible voice commands to voice assistants. *IEEE Transactions on Dependable and Secure Computing*, 18(3):1108–1124, 2019.

[55] L. Zhang, Z. Zhao, C. Ma, L. Shan, H. Sun, L. Jiang, S. Deng, and C. Gao. End-to-end automatic pronunciation error detection based on improved hybrid ctc/attention architecture. *Sensors*, 20(7):1809, 2020.

[56] Z. Zhang, Y. Wang, and J. Yang. Text-conditioned transformer for automatic pronunciation error detection. *Speech Communication*, 130:55–63, 2021.