Universiteit Leiden

Master Computer Science

Automatic Segmentation of Recordings for Text-to-Speech model training

Name: Xin Chen
Student ID: S2425319

Date: 21/08/2021

Specialisation: Computer Science and Science Communication & Society

1st supervisor: Dr. Erwin M. Bakker
2nd supervisor: Prof. dr. Michael S. Lew

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Automatic Segmentation of Recordings for Text-to-Speech Model Training

Xin Chen

Abstract

Text-to-speech (TTS) synthesis models have seen a rapid development and are able to generate human-like speech with a high inference speed. Customized voices in TTS are very desirable for commercial speech platforms. People who currently intend to let the TTS engine speak in their own voice face three challenges: 1) how to segment their speech; 2) how to align their speech with the text they read; 3) how to select the high quality components of self-made speech for training. It is very time-consuming to tackle these three challenges manually, since the training of most TTS engines requires at least 24-hour of recorded and annotated speech. Aligning the text and speech would be a very labor-intensive work. We present an automatic recording segmentation strategy (ARS) to segment speech and align the speech to its corresponding text automatically. The segmentation results of ARS achieve a lower word error rate (0.220) in comparison to the manual segmentation (0.347). Additionally, experimental results demonstrated that a Transformer TTS engine trained with an ARS-processed dataset obtains a word error rate of 0.632 and a mean opinion score of 2.833 after a reasonable training time.

1 Introduction

As Text-to-Speech (TTS) engines become more and more mature and well-developed, people start to shed a light on letting TTS speak in a customized voice, probably their own voice. But ordinary people normally will not have a professional recording studio with recording equipment to capture perfectly a clean speech. Additionally, it will be difficult for them to spend the time and effort to produce a high-quality speech dataset similar to LJSpeech (Ito et al., 2021).

So is it possible to let these TTS systems speak in our own voice? The answer is positive. But there are three major challenges that we have to deal with before entering the TTS training phase.

The first challenge is to segment our recording. As the first approach, we may consider segmenting our recording according to the silences between sentences. A speaker will pause because of a comma, ",", a full stop, ".", etc., information expressions, and having to take a breath. This will make the problem of aligning the speech between silences and the given text challenging.

Szymański et al. (2006) have proposed an automatic segmentation of speech by using HMM-based method, in which dynamic programming was used to correct the boundary points the HMM generated. It is used to process the speech dataset with precise annotations, including the labels of phonemes and the duration of each phoneme, for automatic speech recognizers and speech synthesizers. However, it is rather computationally expensive and its performance is not as good as the manually segmented dataset. Besides, the latest TTS models, such as Tacotron, do not require a training dataset that has a full phonetic level annotation.

The second challenge is how to align the texts we read with our recording. Since a TTS dataset requires in general at least 24-hour speech, which resembles four audiobooks, it is pretty time-consuming and laborious for one person to align all the texts to the corresponding speech one by one.

Recent methods such as forced alignment (McAuliffe et al., 2017) can be used to match each word with the corresponding phoneme forcefully by finding the start time and end time of this word. For instance, there is a speech of "How are you". Forced alignment is able to find out the start time and end time of "How", "are" and "you" respectively according to the transcription of a speech and its phone library indicating the duration of each phone. Even though it is able to align text with speech automatically, it requires a large library of word pronunciation, which still needs lots of human inputs, to search for phones of words so that it can locate each word in the whole speech.

Apart from these two challenges, preliminary TTS training with self-made recording datasets, in general, resulted in bad performance with a mumbling voice. The only difference from the original training procedures is the used dataset, indicating that the quality of speech dataset may have been to blame. It is the third challenge we face.

Although TTS technology made some huge strides coming from requiring a phoneme-based annotated dataset to a sentence-based dataset, it is clear that current TTS engines require a training dataset of a certain quality to be well trained. Thus, a novel preprossing strategy is required to align text and speech automatically.

In this paper, a novel automatic recording segmentation strategy is proposed to help us align text and speech automatically by using an automatic speech recognizer (ASR), which can transform speech to texts. While using ASR as an alignment tool, enables us to get rid of the low-quality parts of our speech.

Specifically, each ASR result is being compared with its corresponding original text so that its highly matching part can be found by calculating its word error rate. Then the low score ones are being selected as our inputs so that we can get our high-quality ones. In this way, we successfully select the high-quality ones as well as aligning our recording with the corresponding texts automatically. Afterward, we fed them into the speech synthesis systems (Tacotron and Transformer) with the corresponding texts input to test out whether this strategy is able to ensure the quality of generated speech by TTS, and even improve its inference performance as well as the speech quality to some extent.

The rest of the paper is organized as follows. In section 2, related work is described. In section 3, basic methods and terminologies are used in the rest of the paper. In section 4, baseline methods are introduced and used in the experiment stage. In section 5, automatic recording segmentation for TTS training is explained in detail. In section 6, experiments and results are displayed. Finally, in section 7, the conclusion is presented.

2 Related Work

In this section, we will introduce the related work of speech segmentation and the background of text-to-speech, including its development and the state of art TTS architectures. Then we will explain why we chose Tacotron and Transformer as our baseline methods.

2.1 Speech Segmentation

Segmenting speech manually is the most common way in terms of segmentation since it is considered as the most accurate way to cut the speech. Furthermore, it provides certain flexibility when it comes to manual segmentation in terms of the semantic meaning of speech. There are several tools that support it, such as PRAAT, a phonetic software. PRAAT enables you to segment the parts you want by setting the boundaries manually, then annotate the parts with the corresponding words. However, manually annotating words one by one is extremely time-consuming and labour-intensive if you have a large dataset, such as the one for TTS training.

To reduce such labour tasks, automatic speech segmentation has been proposed to solve this problem. Segmenting speech based on silences between sentences is widely-used since a speaker will normally pause due to a comma or a full stop. However, when it comes to the annotation, it requires a clear pause without breathings, which are usually unavoidable. This makes the text annotation challenging.

Apart from systematic segmenting based on silences, Ziółko et al. (2006) indicated that the segmentation boundaries could be located based on the analysis of speech signals done by discrete wavelet transform, due to the rapid power changes of phonemes in different frequency subbands. This information enables them to determine the start points and endpoints of phonemes. It is efficient to use this method to do a clean cut on speech at the phoneme level since the whole process can be done without any phoneme recognition and any training. However, without phoneme recognition, it's hard to make sure the quality of audio. Additionally, it also means that it requires other annotation methods to annotate the speech with texts.

Another automatic segmentation for speech synthesis method using Hidden Markov Model (HMM) was proposed by Bansal et al. (2014). This method contains two training models. One model is built for a fixed silence, since it lacks pauses at the end of sentences initially. Another model is designed for aligning words to the most optimal pronunciations. In the end, the Viterbi algorithm is used to compute the location of each phoneme. Even though its segmentation results are closed to the manual segmentation results, the complexity of this method increases the computational time to an extent.

Besides, these speech segmentation strategies are commonly implemented on a standard dataset, such as LJSpeech, which hardly contains any breathings and unclear speeches. In other words, these algorithms in general do not take the "filter" function, which serves for filtering out breathings and unclear speeches, in considerations. However, self-made recordings normally include such low-quality components. There is a necessity to get rid of them so that TTS training won't be affected by them to a great extent.

2.2 Text-to-speech

Text-to-speech synthesis (TTS), a process that transforms text into speech by computer, is getting more and more popular in daily life, especially for the interaction between machines and humans.

TTS is a typical one-to-many problem which one text might correspond with multiple pronunciations. In the earliest TTS engine, it requires massive human inputs of the rules between texts and speech.(Taylor, 2009) This requirement has become an obstacle that stopped TTS from developing at that time. As neural networks came back to the stage of computer science, TTS engines obtained the ability to find the patterns and relationships behind text inputs and speech inputs, imitating the learning process of humans. It succeeded in decreasing the demand of human inputs. Neural networks lead TTS to a whole new world.

During these years, TTS engines with neural networks have been developed a lot. WaveNet (Oord et al., 2016) is a generative model of raw audio waveforms that can predict an output based on previous inputs. It builds a solid foundation for the future vocoder. Deep voice (Arık et al., 2017) succeeded in merging all the necessary TTS components into one system. Tacotron (Y. Wang et al., 2017) realized a real End-to-End TTS systems without training each component independently and extracting linguistic features from texts.

Additionally, latest TTS engines have made a lot of improvements on the inference speed as well as the speed of training. FastSpeech2 (Ren et al., 2020) and Transformer (Li et al., 2019) are able to produce a more human-like and clearer speech and reach a faster inference speed as well as training speed.

2.2.1 the Early State of Art

Wavenet

Wavenet (Oord et al.,2016), a kind of generative model,can be used to generate texts, images and raw audios, etc. In 2017, DeepMind announces that it is able to build a model upon raw audio directly, competing with other state-of-art architectures on TTS tasks. The core of this model is that the prediction of x_{n+1} depends on the given input sequence: $x_1 \sim x_n$. Then we add it into the original sequence generating

a new input: $x_2 \sim x_{n+1}$ for the next prediction. However, while coping with TTS problems, Wavenet still requires linguistic features from text input extracted by the frontend of TTS. Moreover, its inference speed is extremely slow, since it only predicts one audio sample at a time.

Deep Voice

In the same year, Arık et al. (2017) rolls out a novel TTS system called Deep voice, with the replacement of all the components of traditional TTS systems by neural networks. It can learn a kind of voice at one time and need several hours to master each voice with enough datasets. The key of it is that this framework can convert all kinds of linguistic features to various acoustics features, then take these features as the input of its audio synthesis model, which outputs speeches. Deep Voice, "a real End-to-End TTS system", is able to reduce a majority of labor and decrease certain complexity to train the system. However, since Deep Voice is consisted of 5 different models training independently, these errors from different models might compound to a great extent.

Wang

Wavenet and Deep Voice are not "real" end-to-end systems. According to Wang's explanation (W. Wang, Shuang, et al., 2016), "end-to-end" refers to the integration of text analysis and acoustic modeling in the same model. In order to achieve that, an attention-based recurrent network is introduced to the TTS framework, which helps to cope with the intensive labour in text analysis and the complexity of the TTS pipeline at that time. Wang claims that the alignments between linguistic features and acoustic features can be learned automatically with this network.

However, in their experiment, a trained hidden Markov model (HMM) was still employed to align phonemes and their corresponding frames as initial alignments in the training process. Additionally, what it predicts is spectral parameters. A vocoder is still required to help it synthesize speech.

Char2wav

Char2wav (Sotelo et al., 2017)is another end-to-end speech synthesis system integrated the frontend ($texts \rightarrow linguisticfeatures$) and the backend($lingusticfeatures \rightarrow speech$). It is also inspired by the success of attention-based recurrent networks in sequence-to-sequence tasks as Wang's work. One of the differences between them seems to be that attention network is employed in the whole process of speech synthesis in Wang's work, while in Char2wav, it is merely used in the decoder to generate acoustic features. Then, SampleRNN is used to generate a waveform with relatively high quality.

Another difference is that the two components of Char2wav: the reader (an encoder-decoder model) and the neural vocoder, need to be pretrained separately beforehand, while Wang's work only needs to be trained once. Since there is no comparison of the system with other models, it's hard to say how well it performs.

2.2.2 the State of Art TTS

Up till now, all the TTS architectures mentioned above were still struggling with either multiple networks within one model or poor quality of a complete model. In recent years, more efficient, high quality and less complex speech synthesis systems have emerged, such as Tacotron, Tacotron2, Transformer, and Fastspeech2.

Tacotron

Tacotron(W. Wang, Xu, et al.,2016), as one of the earliest TTS model, is a "real" end-to-end TTS model generating speech from text input directly. There are two major differences that made Tacotron stand out.

The first difference is that it integrates text analysis and acoustic modeling in the same model. It got rid of the traditional complex TTS architectures, such as the ones with frontend and backend. Most importantly, it can still produce high-quality human-like speech.

Another major difference is that it does not need to extract linguistic features from texts, thanks to a crucial module: CBHG module, composing of convolutional filters, highway networks, and a bidirectional gated recurrent unit (GRU) recurrent neural net (RNN). It helps Tacotron to extract representations from sequences, instead of manually deriving linguistic features from texts. They use this module in both the encoder and post-processing net, which converts decoder output to the input of synthesizer, which can synthesize input into waveforms. The CBHG module not only enables them to reduce the probability of overfitting and accelerate convergence but also simplifies the architecture by correcting the prediction with contextual information.

Moreover, Tacotron is able to be trained entirely based on given $\langle Text, Audio \rangle$ pairs, which reduces the complexity and, as a consequence, reduces the manual labour of the training dataset. (Y. Wang et al.,2017)

Tacotron2

Tacotron2 (Shen et al., 2018) is an upgraded version of Tacotron. It simplifies the whole architecture of the original Tacotron by replacing the CBHG module with a vanilla LSTM. Apart from that, they also did some adjustments when it comes to the vocoder, which transforms the spectrogram into a waveform. They still use Wavenet as their vocoder but it is a modified version so that they improved the quality of audio

generation. According to their results, they claimed that Tacotron2 outperforms Tacotron to a great extent and the speech produced by Tacotron2 received a high mean opinion score (4.526), which is close to the mean opinion score of the ground truth(4.582).

Transformer

Transformer (Li et al.,2019), is another novel End-to-End TTS model, built upon Tacotron. It is inspired by the success of the Transformer neural network in automatic translation research. It succeeded in improving the efficiency of Tacotron by replacing the recurrent neural networks (RNN) with a Transformer neural network. Since RNN has to process and generate data sequentially, the current hidden state can only be generated based on the previous hidden state. This somehow limits the amount of possible parallelization resulting in slow inference speed. In our early preliminary experiments, we also found that, with the same amount of input (texts and speeches), Tacotron at least took more than twice the time as Transformer on inference as well as training.

FastSpeech2

FastSpeech2, unlike the ones mentioned above, is a non-autoregressive model. It means that its new prediction is less dependent on the previous alignments between text and audio. (Qi et al.,2021) FastSpeech2 is able to deal with the problems the autoregressive model (Tacotron) encountered, such as slow inference speed and word skipping issues. Besides that, it also adds other audio features, such as pitch and energy, as additional inputs in its training phase. Qi et al. indicated that its performance is better than Tacotron2 and Transformer according to their mean opinion score results in which Fastspeech2 received 3.83, Tacotron 2 received 3.70 and Transformer received 3.72. Additionally, its training time, as well as inference speed, are improved to a great extent compared to Tacotron2 and Transformer.

2.2.3 Baseline Method Selection

For our research, we selected these four TTS engines, Tacotron, Tacotron2, Fastspeech, and Transformer as our baseline methods, to test the performance of our automatic segmentation strategy. Here we will explain how and why we chose our baseline methods.

There are two major criteria being used in our selection.

As the first selection criteria, we take the quality of audio generated by these models. Even though all of them claimed that they achieved good performance and the audio they produced is similar to human speech, the results in reality always deviate from the ones they indicated, since the performance, such as the speech quality, is measured by the score of mean opinion score (MOS). However, MOS is a subjective

opinion depending on listeners and the qualities of audio generated by other models it is compared with. To have a better understanding of their audio quality, we tested them by feeding the same text inputs to these four TTS engines, including short texts and long texts. Since they all used the same dataset, LJSpeech, to train, this makes their results comparable.

The second criterion is whether it is feasible to train. Different models require different programming environment features. Some problems are easily solvable by installing a specific version of the library, while some problems are more challenging, such as the conflicts between the conda version and the torch version. We followed the instructions given by their implementation and tested and ran the training.

Baseline Results

Tacotron[1] The quality of its audios is a bit lower than expected. It contains background noise and it sounds somewhat robotic as it speaks. The synthesized speech audio is intelligible. Furthermore, the training script is confirmed to work.

Tacotron2[2] We failed to run its inference script, due to the RuntimeError of PyTorch. Even though its demo audio did sound much clearer than the Tacotron one, we had to leave it out because of the problems we encountered and the limited time we have to try to solve them.

Transformer[3] The sound of its audios is much clearer than Tacotron's and sometimes it sounds like singing. But the major drawback is that it will produce repeating words if the input is too short. For instance, on the input text "hello world", it will produce an audio sounding "hello world hello world hello world". On some occasions, it re-organizes the order of some words by itself. For instance, on the input text "all of them", produces a speech saying "of them all". Running the training scripts was successful, where its speed is at least twice faster than the training of Tacotron with the same speech quality. So it indeed improved a lot when it comes to computational time.

FastSpeech2[4] Its sound quality is the best among four TTS engines. It's much clearer and more human-like, even though it might still mispronounce some words sometimes. But that is probably due to the fact that those particular words don't show up in its training dataset. As for the training, "it requires Montreal Forced Alignment (MFA) to obtain the alignments between the utterances and phoneme sequences", as stated on the instructions page. However, running MFA gave errors that could not be solved due to the limited time of the project. Hence we had to abandon this method also.

[1] https://github.com/keithito/tacotron
[2] https://github.com/NVIDIA/tacotron2
[3] https://github.com/soobinseo/Transformer-TTS
[4] https://github.com/ming024/FastSpeech2

In the end, we selected Tacotron and Transformer as our baseline methods to test out our automatic segmentation strategy, since they are able to generate understandable speech for humans and can be trained by using the new datasets we produced.

3 Fundamentals

In this section, we will introduce the basic methods, different automatic speech recognizers, and metrics, word error rate and mean opinion score, we applied in the later implementation and experiments.

3.1 Automatic Speech Recognizer

Automatic Speech Recognition (ASR) is a reverse process of TTS. It translates speech into texts. In this research, we used ASR to examine the quality and intelligibility of our recordings by verifying how much ASR can recognize. Its transcription would be compared with the original texts so that it can help us to filter those low-quality speech based on a high word error rate.

We considered two high-performance ASR systems: Deep Speech and Google Speech.

3.1.1 Deep Speech

Deep Speech (Hannun et al., 2014) is an End-to-End automatic speech recognition system by using deep learning with two major components: an RNN and a language model. It does not require the manual design of a specific model to be robust against noise or other factors that might affect the transcription capability. It's able to learn directly from the dataset that is composed of 7000-hour clean speech with certain artificial noise. Most importantly, it outperformed other ASR models at the time and reached a word error rate of 16.0% on Switchboard Hub5'00, which is a challenging dataset with telephone conversations between humans and robots as well as conversations between two native English speakers.

RNN. It (Figure 3.1 (b)) is composed of 5 layers of hidden units. Among these five layers, only the fourth layer is the recurrent network. Initially, they first segment an audio clip (with label) into T segments. For each segment, a spectrogram vector is extracted. In the first three layers, it converts each vector into a corresponding character.

Then these characters are fed into the fourth layer, a bi-directional recurrent layer to extract hidden representations from its forward and backward recurrence. The fifth layer is to predict a sequence of character probabilities for each time frame by taking those hidden representations as inputs. By filtering the highest probability of character in each time frame, they can compose a complete transcription.

After each prediction, to minimize the error between prediction and ground truth, the Connectionist temporal classification (CTC) loss is further computed to evaluate

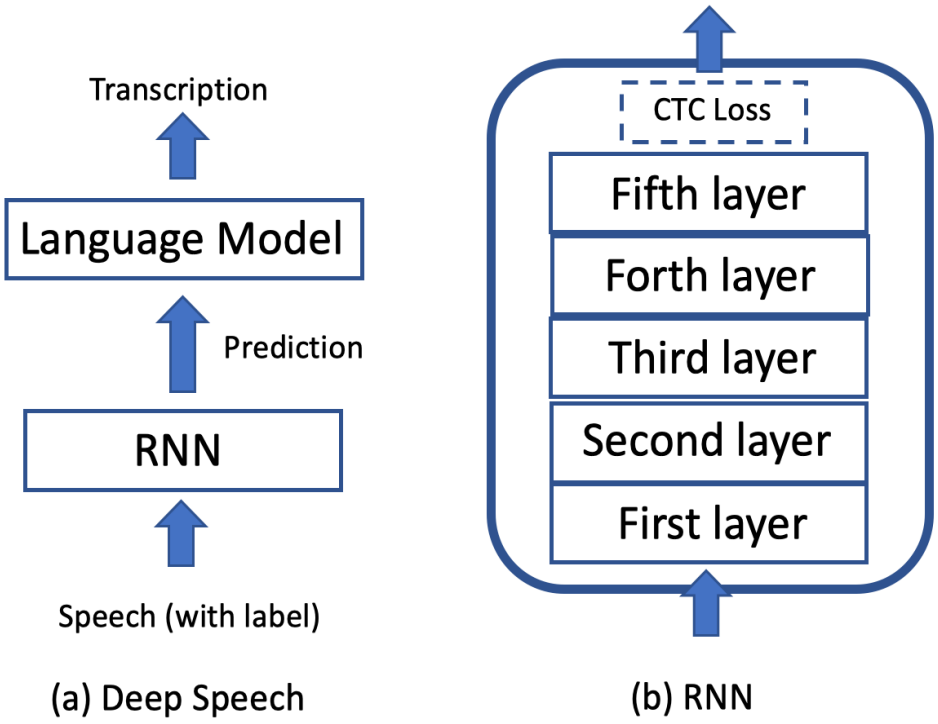


Figure 3.1: (a) The overall architecture for Deep Speech. (b) depicts the structure of RNN.

the edit distance between the sequence of predicted characters and the sequence of corresponding ground truth.

With those speech and corresponding sequences of character, the model is trained by using back-propagation to obtain a predictive model.

Language model. It is a model to cope with the word errors which these words don't exist in our training dataset. The language model takes the prediction result as inputs to compare with the corpora used to train the language model. Combining the result of RNN and the one of the Language model, the final predictive result is obtained as the transcription output.

In this research, we apply deep speech as a tool to measure the quality of our recordings. In our preliminary studies, we discovered that the word error rate of deep speech is relatively low and it is able to recognize all the words in speech.

3.1.2 Google Speech

Google speech is another popular automatic speech recognizer. Its structure is pretty similar to deep speech in the description of Soltau et al. (2016).

It is composed of a deep LSTM RNN neural network and uses the CTC loss to minimize the error between prediction and ground truth. It differs from the deep speech on the "language model". It models words in written form but also maps an

FST verbalization model for words in spoken forms with a larger dataset with written and spoken English vocabulary. For instance, mapping "205" to "two hundred and five" and "two o five".

Both of these two ASRs optimize the transcription by rescoring their prediction in multiple times. However, we didn't use google speech as our tool to examine our audio quality. There are two main reasons behind that.

The first one is that its performance was not good as the deep speech. It either failed to transcribe speech shorter than three seconds most of times or merely recognize one or two words while there are at least five words spoken in the speech. Since we used the same speech to test google speech and deep speech, deep speech outperforms google speech in terms of the accuracy of transcription and the capability to transcribe.

In addition, in our later experiment, we attempted to do a precise cut on the audio according to the transcribed words, (so that we can get rid of the words ASR failed to recognize), where the timestamps of transcribed words were required. It could be something wrong with the environment and the versions. We couldn't run that timestamp python file so we had to choose not to use google speech in this case.

3.2 Signal Preprocessing

Before feeding our recordings into ASR, we preprocessed our speech since we need to feed our speech into ASR to see how much it can recognize. Based on its recognized results, we can compare it with the original texts so as to calculate its WER.

3.2.1 Pre-emphasis

According to previous ASR research, pre-emphasis (Loweimi et al., 2013) is an essential step to preprocess audio signal, since audio signal normally experiences "Spectral Tilt", which is that the amplitude of high frequency signal is relatively lower than the amplitude of low frequency one.

Due to the different energy levels, the low frequency component has higher energy levels than the high frequency one. In that case, with the same amount of noise, the signal-noise ratio (SNR) of high frequency components is much lower than low frequency ones which means that high frequency signal contains more noise relatively giving rise to being less easy to be recognized.

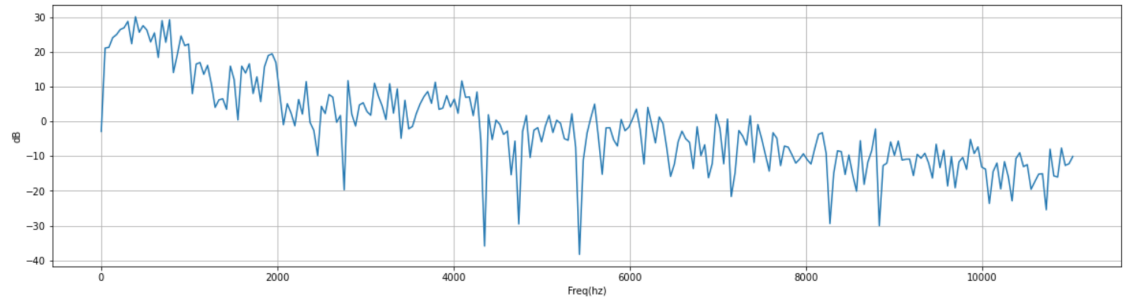
Pre-emphasis ("Emphasis (telecommunications)", 2021) is able to balance the high frequency component and the low frequency component so that the high frequency component can be amplified relatively. It uses the following first-ordered filter to achieve:

$$y(t) = x(t) - ax(t-1), 0.95 < a < 0.99 \tag{3.1}$$

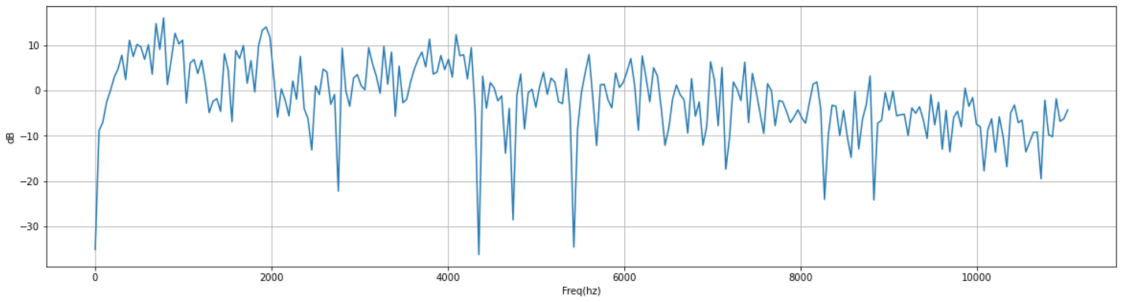
Since the frequency of the signal is determined by the speed of signal level change. After applying this first-ordered differentiation on signal, its differentiation value is high on the high frequency component where signal level changes fast, while its differentiation value is low on low frequency component where signal changes slow so that this filter is able to lower the low frequency component and enhance the high frequency component. In this case, the pre-emphasis process increases the chance of detecting the high frequency component by ASR.

In Figure 3.2 , it presents the comparison between a 3.5-second speech signal and its pre-emphasis results. As we can see, the amplitude of pre-emphasis one in Figure 3.2(b) has been flattened and occupied evenly the whole Spectrum map. If you looked at the vertical axis, you will find out that the range of amplitude has been compressed from $-40dB \sim 30dB$ to $-30dB \sim 18dB$. What results in it is that the amplitude of the low frequency components has been compressed, since they are the ones which contribute to high amplitude, while the amplitude of the high frequency components remains unchanged basically.

In a word, even though the Equation 3.1 didn't really flatten the spectrum map of the signal by directly amplifying the amplitude of the high frequency components but compressing the low frequency ones, it achieves that we can use the same signal-noise ratio (SNR) to analyze the signal, since the original SNR is different in terms of low frequency components (high SNR) and high frequency components (low SNR). Pre-emphasis is able to ensure ASR to better capture the signal info from high frequency components.



(a)



(b)

Figure 3.2: Comparison between original speech signal and the one after pre-emphasis. a): Speech signal before pre-emphasis (Original Signal), b): Speech signal after pre-emphasis

3.3 Word Error Rate

Word error rate (WER) ("Word error rate", 2021) is a common metric to measure the performance of ASR. What it measures is straightforward as its name, the proportion of incorrect words in transcription while comparing to the reference texts (input texts).

There are three situations when incorrect words show up. The first one is the insertion (I). It inserts a word or words in the original texts. It could be because of the misread or because ASR takes another sound, such as the voice of other people, as input. The second one is the deletion (D). It deletes a word or words. It could be due to the fact that ASR fails to recognize it or ASR fails to transcribe it. The last one is the substitution (S). It substitutes a word with another word. The mispronunciation could be blamed.

The equation of WER is defined as in Equation 3.2. N denotes the total number of reference texts. On the other hand, N also equals to the total number of hits (H) and substitution (S). "Hits" refers to that the words in transcription are the same as the words in reference inputs on the location level as well as the spelling level.

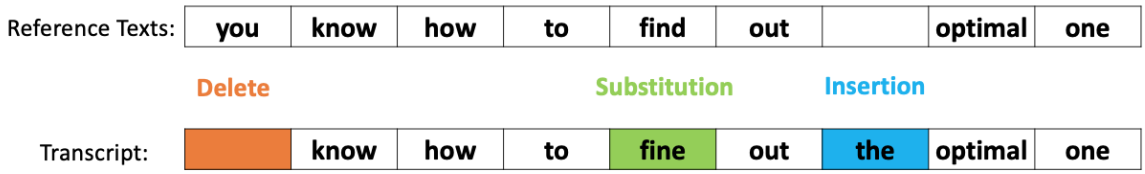


Figure 3.3: WER Example.

See the example in Figure.3.3. Its WER would be $3/8 = 0.375$.

$$WER = \frac{S + D + I}{N} \tag{3.2}$$

3.4 Mean Opinion Score

Mean opinion score (MOS) ("Mean opinion score", 2021) is an evaluation metric that researchers normally use to measure the quality of a product from a human perspective, specifically how they feel and what kind of experience they have while using it. It is a common metric used in the telecommunication area.

Initially, it calculates the arithmetic mean of "values on a predefined scale that a subject assigns to his opinion of the performance of a system quality". Table 3.1 is a regular rating scale for MOS.

In the field of TTS, Tacotron receives a MOS around 4.001 in the comparison among ground truth, Tacotron and Tacotron2 (Shen et al., 2018), while Transformer receives a relatively high MOS around 4.39 in the comparison between Tacotron2 and Transformer (Li et al.,2019). But MOS as a subjective evaluation metric is highly dependent on the object of reference. For instance, if you compare the result of your model with the one of a model that is much worse, it's apparent that you will receive a high MOS for your model. In this case, the object of reference should be carefully selected.

Rating	Label
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

Table 3.1: Rating Scale for MOS

4 Baseline Method

4.1 TTS

Here are the two TTS engines, Tacotron and Transformer, we worked with in our later experiment. In our designing stage, we also have considered other TTS engines, such as Fastspeech2 and Tacotron2. But during our preliminary studies, we found several obstacles that could not be resolved during the limited duration of our project and they stopped us from using them for our research.

4.1.1 Tacotron

Tacotron (Y. Wang et al., 2017) adopts the sequence-to-sequence structure based on encoder and decoder with attention mechanism. It is composed of three main components, including an encoder, a decoder and post-processing, as indicated in Figure.4.1.

Encoder. It (Figure.4.1 (b)) takes character sequences as inputs and each character has been converted into a one-hot vector and a continuous vector before being fed in the encoder. After feeding in the encoder, a series of nonlinear transformations would apply to these inputs in Pre-net, which is able to accelerate the convergence and generalization of the model. Afterward, the output of Pre-net would be fed into CBHG, which also helps to improve the generalization. "This module is able to

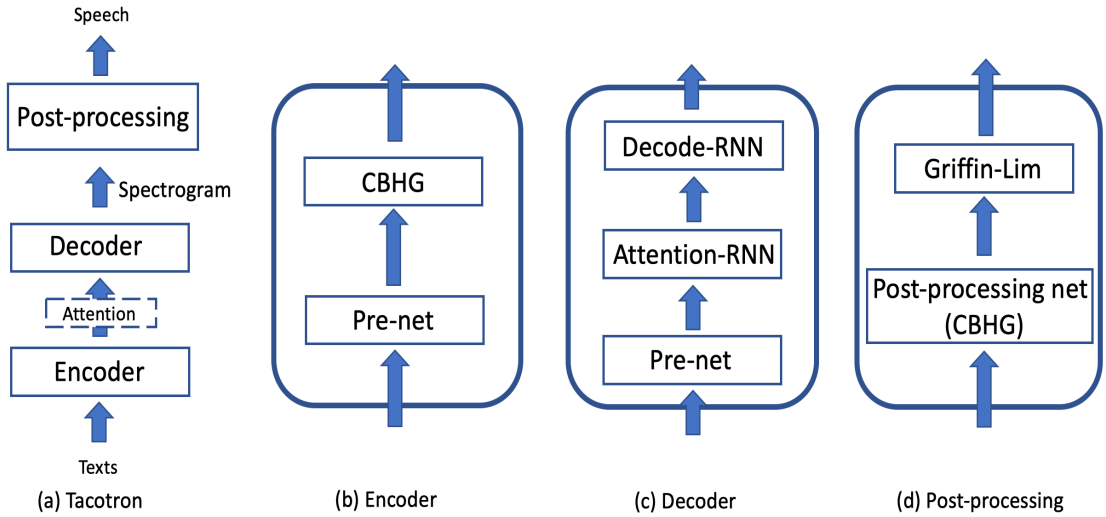


Figure 4.1: (a) The overall architecture of Tacotron. (b) depicts the structure of encoder. (c) depicts the structure of decoder. (d) depicts the structure of post-processing.

extract sequential features from both forward and backward contexts." In a word, this encoder aims to obtain the hidden representations from those character sequences' input.

Decoder. The output of the encoder, hidden representations of context, would first feed in an attention module in subfigure (a) to get the context vectors. The input frames (audio clips) would be processed by the pre-net in the decoder first to do several nonlinear transformations. Then the Attention-RNN takes the output of pre-net and context vectors as its input. Before getting into the inference state, Decode-RNN, the output of Attention-RNN would pass a layer of GRUs first. Each GRU output contains multiple frames of an audio file, since one character might correspond to multiple frames.

While in the inference stage, at each decoding step, it not only predicts one frame corresponding to its input but multiple frames instead, due to the fact that it is able to quicken the convergence of the algorithm. The whole prediction process is highly dependent on the previous output since it will take the last frame of the previous output as the input of the next prediction. In the end, the decoder will generate a mel-scale spectrogram.

Post-processing. Tacotron doesn't generate the speech directly from the result of the decoder, but lets it get through a post-processing net mainly composed of CBHG so that the model can modify the spectral magnitude in each frame before synthesis. After these modifications, the predicted spectrogram would be synthesized by Griffin-Lim into a waveform.

4.1.2 Transformer

Transformer is a TTS model developed upon Tacotron which also comes along with an encoder-decoder structure. It applied the Transformer network, which is commonly used in the translation field, into the TTS area. (Li et al., 2019) It contains four main components, a text-to-phoneme converter, an encoder, a decoder, and a module to synthesize speech and produce stop tokens.

Text-to-phonome converter. Instead of taking texts as inputs, they used phonemes as their inputs due to the limitation of their dataset, which is not able to cover all the regularity of each letter. Here a rule system has been produced to cope with this problem.

Scaled Positional Encoding. Without the recurrence and convolution, Transformer will generate the same output even though the input sequences of the encoder and decoder present in a different order. To avoid that, a triangle positional embedding with a trainable weight has been employed to the outputs of encoder pre-net and the ones of decoder pre-net so that Transformer is able to learn the order info of input sequences.

Pre-net. Pre-net in Encoder is the same as the Pre-net in Tacotron. It can apply several nonlinear transformations to the inputs, while pre-net in the decoder is

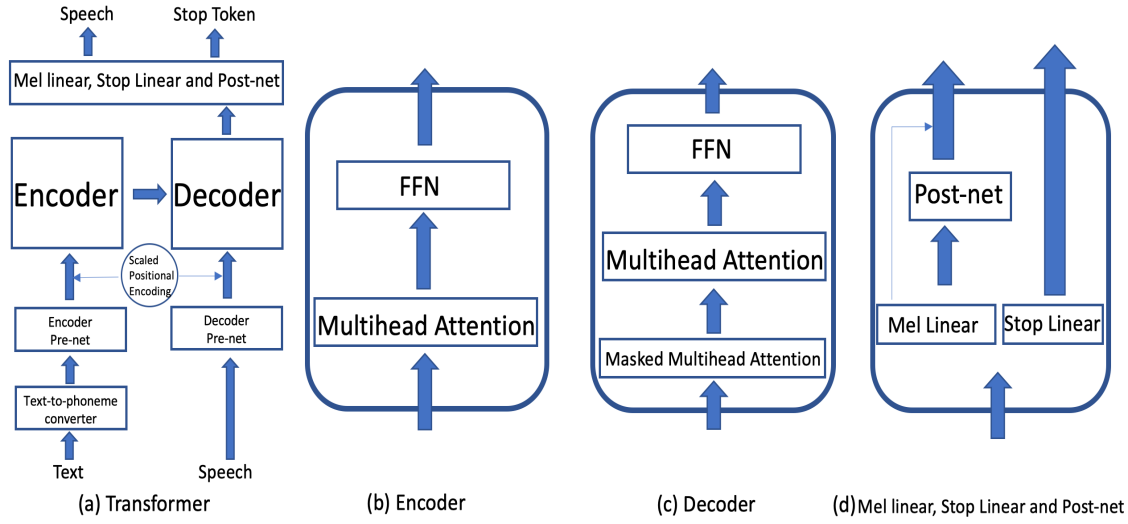


Figure 4.2: (a) The overall architecture of Transformer. (b) depicts the structure of encoder. (c) depicts the structure of decoder. (d) depicts the structure of mel linear, stop linear and post-net.

different from the one in Tacotron, in terms of its structure as well as its functionality. It is composed of a neural network with two fully connected layers, while the one in Tacotron is composed of a 3-layer CNN.

The responsibility of the decoder pre-net is to map mel spectrograms "decoder pre-net is responsible for projecting mel spectrograms into the same subspace as the phoneme embeddings so that the similarity of a $\langle phoneme, melframe \rangle$ pair can be measured, thus the attention mechanism can work".

Encoder. With the multi-head attention network, one attention is being split into several subspaces so that it is able to project the $\langle phoneme, melframe \rangle$ relationship in different perspectives. This replacement of RNN, the multi-head attention network, improved the efficiency since it allows the model to compute those inputs parallel.

Decoder. The decoder of Transformer is to measure the similarity of $\langle phoneme, melframe \rangle$ pairs by using self-attention. Apart from that, the multi-head attention is able to generate better context vectors from the output of the encoder, which is the $\langle phoneme, melframe \rangle$ relationship mentioned above. In addition, it does not have to take previous output into account in its inference stage which also improves the training speed.

Mel Linear, Stop Linear and Post-net. The mel linear and post-net is similar to the post-net and griffin-lim in Tacotron, and are used to predict the mel spectrogram. What makes it different is the stop linear. Since there is an imbalance between positive samples and negative samples, the "stop" is the only positive sample at the end of each input sequence, while others are negative ones. Stop linear produces a stop token which means "stop" in the sentence. Without the stop token, it might give rise to an unstoppable inference.

4.2 LJSpeech

Since we produced our own dataset, we want to compare and validate our dataset against a baseline dataset that is used to train TTS. LJSpeech (Ito et al., 2021) is a well-known widely used for TTS training, including Tacotron and Transformer.

It contains a 24h speech of paragraphs from seven different books read by the same speaker and its corresponding texts. In its "Readme" file, it indicates the features of this dataset. This 24h speech was segmented based on the silences between sentences in the recording. "Each audio clip is a single-channel 16-bit PCM WAV with a sample rate of 22050 Hz." The length of each audio clip varies from 1 second to 10 seconds.

The corresponding texts were aligned to each audio clip manually. While aligning texts to the audio clip, abbreviations should be expanded. For instance, "Mr" should be expanded to "Mister". Besides that, the whole dataset passed the quality assurance test to make sure those texts matched the audio clips accurately word by word.

In the light of LJSpeech, there are several requirements for a dataset for TTS training.

- the total length of speech should be at least 24 hours
- the sample rate of speech should be 22050 Hz
- the format of speech should be a single-channel 16-bit PCM WAV
- the speech should be segmented into multiple audio clips
- the length of each audio clip should not be longer than 10 seconds
- each audio clip should be aligned to the corresponding texts

The first three requirements are being strictly followed while recording. While for the last three requirements, they are the research questions we attempt to resolve automatically in this paper, instead of reaching these requirements with a large amount of labour and time.

On the other hand, to improve the bad performance of preliminary TTS training with recorded speech, speech segmented by silences between sentences should also be abandoned, due to a number of breathings found in self-made recordings. These breathings might prevent TTS systems to learn the alignments between text and speech. They might misinterpret those breathing as words instead of silences.

To achieve that, an automatic speech segmentation is applied to meet the last three requirements. This strategy helps us align our speech with the corresponding texts and get rid of the low-quality clips. The details of this strategy will be discussed in section 5.

5 Automatic Recording Segmentation for TTS Training

In our preliminary studies, a self-made recording dataset used in TTS training normally received a poor performance. It is believed that those low-quality components of the speech we produced should be blamed for that, due to the unavoidable breathings and unclear speeches in our recordings.

While segmenting our speech based on silences between sentences, these breathings and unclear speeches will still be kept in our audio clips. TTS might be misled by considering breathing as a word and aligning unclear speeches to wrong words. Apart from that, aligning the audio clip to the corresponding texts manually is extremely time-consuming and labor-intensive, especially when it comes to a dataset with a 24h speech.

In this paper, an automatic recording segmentation has been proposed to solve such problems. To get rid of the low-quality components of our speeches, breathings, and unclear speeches, ASR is being used to distinguish which word is recognizable for a machine. A speech is cut into multiple audio clips with recognizable words according to the results of ASR.

Furthermore, with the results of ASR, the alignment between an audio clip and the corresponding texts could be established automatically by calculating the WER between the results of ASR and the segments of original texts for each audio clip respectively. The original texts segment with the lowest WER will be saved as the corresponding texts for this audio clip.

5.1 Automatic Recording Segmentation

When it comes to automatic segmentation, the most common way to segment speech is to use silences to segment our speech into audio clips by using the length of silence as criterion.

Before designing our strategy, we concluded several preliminary experiments on the audio segmentation based on silences. It was found that this kind of segmentation can not segment each sentence accurately, due to different lengths of silences caused by the speaker. Apart from that, some segments were too short, e.g. two words, for ASR to recognize it. Therefore, training a TTS model on such a dataset is deemed too challenging.

Alternatively, a novel strategy, automatic recording segmentation for TTS training (ARS), is proposed in which it segments our speech based on the WER of the ASR result of our speech and the original text, as Figure 5.1 indicated.

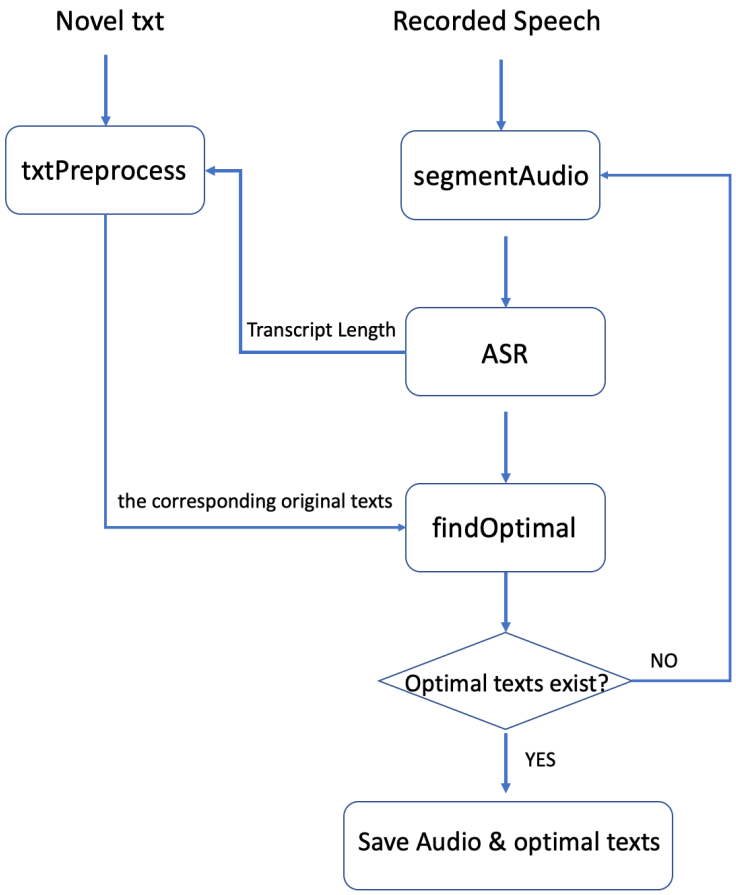


Figure 5.1: Flow Chart of the Automatic segmentation Algorithm.

Initially, audio with a certain length is first segmented, then ASR is being used to transcribe the segmented audio. The original texts are extracted according to the length of the transcript and the length of original texts which is twice as much as the length of the transcript. According to our transcript, the result of ASR, the optimal original texts from our extracted texts is being found by calculating its WER. If its WER is lower than our threshold, it will be kept in our dataset, otherwise, it will be left out.

Due to the fact that ASR is not able to transcribe each word spoken by the speech most of the time, we chose to make some adjustments to the original texts to see which segment of the texts (by moving it back and forth) would be the closest one to our transcripts.

5.1.1 ARS for TTS

Based on the idea explained above, one iteration of finding the optimal original texts for a segmented audio clip is visualized in the Figure 5.1.

Algorithm. 1 is the main function designed for this automatic segmentation. In this

algorithm, the main inputs are the novel texts we read for our recordings and our recordings. The start point (*sPoint*) of our segmented speech and the endpoint (*ePoint*) of it are set as our parameter inputs. These two parameters are used to control the duration of our segmented speech. *times* is a parameter to control the ratio of the length of extracted original texts (*words*), while *tsPoint* is the start point of our extracted original texts. It helps us to locate the beginning of texts. $threshold_{wer}$ is obviously the wer thresold we set for filtering the optimal texts. *var* is a fixed parameter to move segmented speech forward. It also determines the length of segmented speech in our case.

As long as the endpoint of our segmented speech is smaller than or equal to the duration of the whole recorded speech, our finding optimal texts iterations continues. In this algorithm, there are several crucial functions being used, such as "find the optimal word sequence". More details in each function and how those parameters were set and chosen will be explained in the following subsections.

Algorithm 1 Automatic recording segmentation

```
Main Input: $OriginalTexts, recording$
Parameter Input: $sPoint, ePoint, times, threshold_{wer}, tsPoint, var$
Output: $audioclips, transcripts, originaltexts, wer$
Split the $OriginalTexts$ into a $wordList$
while $ePoint <= recording_{duration}$ do
    segment an $audioclip$ with the length of $var$ from the whole $recording$ according to $sPoint$ and $ePoint$
    transcribe the $audioclip$ into a $transcript$
    extract $words$ that are twice ($times$) as long as the the length of $transcript$ from the beginning of the $tsPoint$ and $wordList$
    find the optimal word sequence for the $audioclip$ from the extracted $words$
    if optimal word sequence exits then
        if there are only one optimal word sequence then
            save the optimal word sequence, the $transcript$ and the $audioclip$
        end if
        if there are multiple optimal word sequence then
            select the one with the minimal WER, then save it
        end if
        $sPoint \leftarrow ePoint$
        $ePoint \leftarrow ePoint + var$
        $tsPoint \leftarrow$ index of the beginning of optimal word sequence + the length of transcript
    end if
    if optimal word sequence doesn't exit then
        $sPoint \leftarrow ePoint$
        $ePoint \leftarrow ePoint + var$
        $tsPoint \leftarrow tsPoint +$ the length of transcript
    end if
end while
```

Text Preprocessing

To achieve the flexibility of extraction, the original novel texts are split into a word list so that we can scratch the different lengths of the texts according to our needs. In our previous experiments, it was found that deep speech is able to transcribe the same amount of words in the speech. For instance, the original speech contains 5 words. Deep speech can also transcribe this speech into 5 words, even though they might not all be correct. But at least its transcription ability is strong enough. That's why we set the *times* parameter as 1, in terms of deep speech. It means that we will extract the same amount of words as the result of deep speech while calculating the WER.

Fixed Length Audio Segmentation

A fixed length of speech is segmented for later calculation. As for deep speech, the speech length is set to 3s. Normally, a 3s speech contains 5 words which makes it a suitable length for deep speech. Since the audio clip is obtained from the beginning of the speech, sPoint and ePoint are set as 0 and 6 (segment parameter) respectively, and multiply these two points by the sample rate of speech (Alg.2). As long as the distance between sPoint and ePoint is 6, the segmented audio clip will be always 3 seconds.

Theoretically, it is better to have a relatively short speech rather than the long one for TTS learning so that it doesn't have to learn too many things at one time but only focus on those 5 words on average. In addition, even if the texts are not perfectly aligned to the audio clip. Shorter audio clips could also reduce the possibility of learning incorrect alignments for TTS since they contain fewer words than longer speeches.

Algorithm 2 segmentAudio

1: **function** SEGMENTAUDIO($audioData, sampleRate, sPoint, ePoint$)
2: $audioData$ (is the original speech)
3: segment a fixed length of $audio$ started with $sPoint$ and ended with $ePoint$ according to the $sampleRate$ of $audioData$
4: $audio = audioData[int(sPoint * sampleRate) : int(ePoint * sampleRate)]$
5: **return** $audio$
6: **end function**

Find the Optimal Word Sequence

The strategy to "find the Optimal word sequence" function consists of two parts: speech transcription and the WER calculation. The most optimal original word sequence for the corresponding audio clip can be found by calculating the WER between the transcription of the audio clip and the texts we moved.

There are multiple ways to find the most optimal original texts. Two different ideas have been proposed in the early stage. The first idea is to locate the beginning and ending words of the original texts according to the beginning and ending words of the transcription. There will be four different cases of matching the beginning and ending words, such as both of the beginning words overlap but the ending words do not. In this case, the start point of this speech remains unchanged and the speech will be lengthened by moving the endpoint of the speech a step further to see whether the last words of the updated speech can overlap with the last words of the word sequence.

But we failed since it can only find out the first few sentences, while the later original texts deviate from the transcripts to a great extent. The challenge for this method is to find out the feasible parameters to stop matching the current audio clip and the corresponding word sequence then move to the next pairs. Due to the limited time of our project, this method has been set aside.

The second idea is a simplified version of the first idea. While searching for the optimal $\langle Text, Audio \rangle$ pair, the length, and the location of an audio clip remained unchanged, the location of the word sequence has been moved to find the optimal one with the lowest WER.

First of all, a number of texts are extracted from the original texts which is twice as long as the length of the transcript. In Figure 5.2, if the length of transcript is 5, 10 words are extracted from the original texts. But in the WER calculating phase, the transcript will be compared with the same amount of texts *wordSequence*. After each comparison, the *wordSequence* will be moved one step right (the 14th line in Algorithm.3), then repeat the comparison and calculation till the last word of *wordSequence* is the last word of our extracted texts. It is also our stop criterion. This iteration will be ceased when the length of the *wordSequence* is shorter than the length of transcript in the sixth line of Algorithm.3, since this kind of *wordSequence* is not in our considerations. Then the audio with the lowest WER will be saved among all the comparisons, if the optimal word sequences exist. (the 15th line in Algorithm.1)

After testing our codes with one smaller dataset (a 2-minute speech), it is discovered that 0.625 would be a nice WER threshold and it also works well in a large dataset (an 1-hour speech). Besides, we also attempted to lower the threshold to find a more optimal dataset at one time, otherwise, we need to filter the result again to achieve that. However, due to the mechanism we used for updating *tsPoint* in Alg.1, we discovered that *wordSequence* would deviate from the transcription after certain iterations if the WER threshold is lower than 0.625. In this case, 0.625 is set as our WER threshold.

Once we finish each iteration, our *tsPoint* would move forward. But how the *tsPoint* moves is different, in terms of different situations. (Algorithm.1) If the optimal word sequence exists, the *tsPoint* would first move to the start point of the optimal texts, then continue to move forwards according to the length of the transcript. If not, the *tsPoint* would be updated only with the length of the transcript.

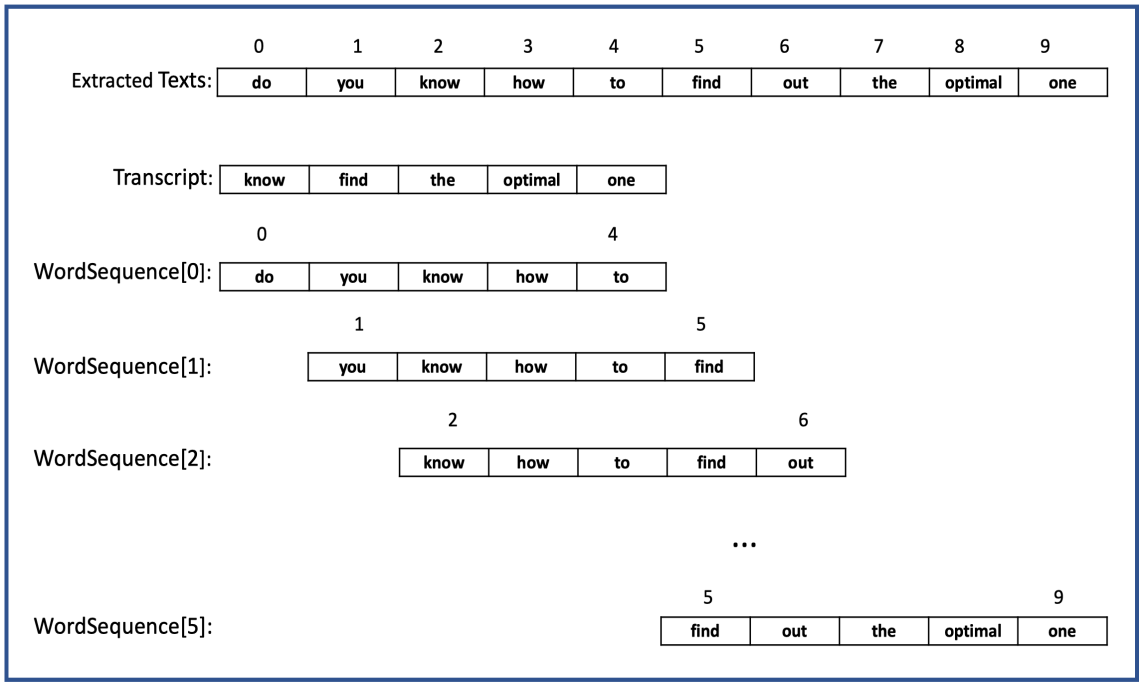


Figure 5.2: Find the optimal *wordSequence* for the corresponding audio clip

Algorithm 3 Find the optimal word sequence

function FIND THE OPTIMAL WORD SEQUENCE($tsPoint, words, transcript, threshold_{wer}$)
  $i \leftarrow 0$
  while $i \leq$ the length of *words* **do**
    $tsPoint \leftarrow tsPoint + i$
    extract *wordSequence* from *words* according to the updated *tsPoint* and the length of *transcript*
    if the length of *wordSequence* <the length of *transcript* **then**
      wordSequence shorter than the *transcript* is not in our consideration
      break
    end if
    calculate the WER between *wordSequence* and *transcript*
    if $WER \leq threshold_{wer}$ **then**
      save this *wordSequence* and the current*tsPoint*
    end if
    move our *tsPoint* a step forward
    $i \leftarrow i + 1$
  end while
end function

```
Metadata(transcripts=[
  CandidateTranscript(confidence=-6.116363525390625, tokens=[
    TokenMetadata(text='a', timestep=41, start_time=0.8199999928474426),
    TokenMetadata(text='l', timestep=43, start_time=0.85999995470047),
    TokenMetadata(text='i', timestep=49, start_time=0.9799999594688416),
    TokenMetadata(text='c', timestep=51, start_time=1.0199999809265137),
    TokenMetadata(text='e', timestep=53, start_time=1.059999942779541),
```

Figure 5.3: Example result of metadata generated by deep speech

Formulation and Organization of the Results

Once the optimal $\langle Text, Audio \rangle$ pair is found, the audio clip and the corresponding *wordSequence* need to be saved. However, while segmenting the audio clip, the end time of the last recognizable word remained unknown. It might result in cutting a word in the middle of its duration, while ASR is not able to recognize and it also might confuse TTS in the later training to some extent.

To cope with it, those incomplete words should be removed. To locate the unrecognizable words or syllables is a challenging task. But to keep the complete ones is much easier by using their timestamp. The timestamp is speech info containing the start time of the word and its duration. It is used to locate where the words show up in the speech. Since our segmented audio clips could be recognized by ASR with a reasonable WER, there must be several alignments being established between the speech and transcripts already. Otherwise, ASR won't be able to recognize them correctly.

Due to the fact that deep speech must have a way to obtain such encoded information with them, we looked into their Github to look for the codes. So this part of code (Algorithm.4) is originated from "mozilla/DeepSpeech" (2021). The metadata is generated by its transcription algorithm containing the start time and duration of each letter and space. (Figure.5.3) In Algorithm.4, it first concatenates letters till meeting a space then adds up their start time to generate the start time and the duration of a word, as Figure.5.3 stated.

Post Processing

Among our speech results, there are some audio files with an unexpected small file size, such as 10 KB. By listening to such audio files, they turn out to contain one word only, the long silence of the original audio clip has been cut out, due to our "Formulation and Organization of the Results" (FOR) procedure described in the previous section.

Additionally, there is another extreme case that some audio files only contain 44 bytes. That might be something wrong with FOR procedure since their transcripts

Algorithm 4 timeStamp

```
1: function TIMESTAMP(metadata)
2:     word = ""
3:     word_list = []
4:     word_start_time = 0
5:     for i, token ∈ enumerate(metadata.tokens) do
6:         if token.text! = "" then
7:             if len(word) == 0 then
8:                 word_start_time = token.start_time
9:             end if
10:            word = word + token.text
11:        end if
12:        if token.text == ""ori == len(metadata.tokens) − 1 then
13:            word_duration = token.start_time − word_start_time
14:            if  thenword_duration < 0
15:                word_duration = 0
16:            end if
17:            each_word = dict()
18:            each_word["word"] = word
19:            each_word["start_time"] = round(word_start_time, 4)
20:            each_word["duration"] = round(word_duration, 4)
21:            word_list.append(each_word)
22:            word = ""
23:            word_start_time = 0
24:        end if
25:    end for
26:    start = word_list[0]["start_time"]
27:    end = word_list[−1]["start_time"] + word_list[−1]["duration"]
28:    return start, end
29: end function
```

and their corresponding *wordSequence* are correct in our result file. It is obvious that it works normally till the FOR part. It still remains unclear why this would happen, since this only happens on a small proportion of the whole dataset, while the majority of the dataset performed regularly as we expected.

Apparently, such small audio files would affect TTS training results, since they do not contain enough useful content. In this case, the result of ARS will be post-processed by simply removing them from our dataset.

6 Experiments & Datasets

6.1 Datasets

There are three datasets that we will use to evaluate the performance of our automatic segmentation.

The first dataset is the original LJSpeech which has been taken as inputs for Tacotron and Transformer. Its speech is segmented by silence automatically and is aligned to texts manually, consisted of 13,100 short audio clips and the corresponding transcripts aligned manually. LJSpeech would be a dataset (a), as a baseline, we used to see if there are any improvements after training with those datasets processed by our automatic speech segmentation.

The second dataset is made from the original speech recordings of LJspeech. We downloaded those original recordings recorded by Linda Johnson on Ito et al. (2021) and their corresponding texts. Then our automatic segmentation is applied on those recordings to segment them into short audio clips and to align these audio clips with the corresponding transcripts. This dataset (b), we called LCSpeech, is composed of 26,344 segmented audio clips and the corresponding transcripts from deep speech as well as the original books.

The third dataset is made from our own recordings. Our recordings are of a single speaker reading from 5 novels that are easy to be understood without any jargon. We also used our automatic segmentation to segment our recordings as well as aligning them with their transcripts. This dataset (c), QCSpeech, also contains 18,044 segmented audio clips and the corresponding transcripts from deep speech as well as the original texts.

6.2 Experiments & Setup

In this section, we will introduce two different experiments to evaluate the performance of our automatic segmentation strategy directly and indirectly.

The first part is the quality assessment of datasets. The second part is the evaluation of the training performance with the input of our processed LCSpeech and QCSpeech by comparing with the training performance of the original LJSpeech, to see whether our automatic segmentation is able to improve its learning performance or not.

WER	0	$0 \sim 0.1$	$0.1 \sim 0.2$	$0.2 \sim 0.3$	$0.3 \sim 0.4$	$0.4 \sim 0.5$	$0.5 \sim 0.6$	≥ 0.6
NUM	12	361	2571	3442	2926	1852	781	1155

Table 6.1: The segmentation result of LJSpeech

WER	0	$0 \sim 0.1$	$0.1 \sim 0.2$	$0.2 \sim 0.3$	$0.3 \sim 0.4$	$0.4 \sim 0.5$	$0.5 \sim 0.6$	≥ 0.6
NUM	7216	153	4960	6380	4073	2540	927	95

Table 6.2: The segmentation result of LCSpeech

6.2.1 Quality Assessment of Datasets

In this part, the quality assessment of datasets has been done by calculating the WER between the transcribed results of audio clips by using ASR (Deep Speech) and aligned texts.

LJSpeech

LJSpeech is the speech dataset commonly used in TTS training. It is segmented and aligned to texts manually. There are 13,100 audio clips in total. Table. 6.1 presents the manual segmentation result of LJSpeech in terms of different WER. Among these audio clips, there are 0.09% of them (12/13,100) that are totally matched with the original texts. Besides that, around 48% of them achieve an accuracy of 70%. $(0\sim 0.3 : 6,386(13,100))$

LCSpeech

The recordings of LCSpeech are the same as those ones for LJSpeech. After processing by automatic recording segmentation, there are 26,344 audio clips with a length range from one second to three seconds that have been segmented with WER threshold of 0.625. Table. 6.2 presents the segmented results of the LCSpeech. Among these audio clips, there are around 27% of them (7,216/26,344) that are totally matched with the original texts. Besides that, around 71% of them achieve an accuracy of 70%. $(0\sim 0.3 : 18,709(26,344))$

QCSpeech

As for QCSpeech, there are 18,044 audio clips in total after segmented by automatic recording segmentation. Table. 6.3 presents the segmented results of the QCSpeech. The accuracy of the segmentation of QCSpeech is dramatically lower than the one of LCSpeech. There are only 10.8% of them (1,949/18,044) that matched with the original texts completely. In addition, there are 41.5% audio clips achieve an accuracy of 70%. $(0\sim 0.3 : 7,487(18,044))$

WER	0	$0 \sim 0.1$	$0.1 \sim 0.2$	$0.2 \sim 0.3$	$0.3 \sim 0.4$	$0.4 \sim 0.5$	$0.5 \sim 0.6$	≥ 0.6
NUM	1949	22	2718	2798	3744	4162	2380	271

Table 6.3: The segmentation result of QCSpeech

Dataset	LJSpeech	LCSpeech	QCSpeech
WER	0.347	0.220	0.340

Table 6.4: The Mean WER of Each dataset

Table. 6.4 presents the mean WER of each dataset. Among these three datasets, LCSpeech got the lowest score.

6.2.2 Performance of TTS Training

Before feeding those three datasets (LJSpeech, LCSpeech, and QCSpeech) into TTS models, the corresponding coding environments have been set up according to the requirements from the implementation of Tacotron[1] and Transformer[2]. Due to the limited time, all models with different datasets have been trained on Quadro RTX 6000 for 2 days respectively.

To evaluate the performance of TTS training, the manually segmented dataset (LJSpeech) and two datasets processed by our method (ARS), LCSpeech and QCSpeech, were fed into Tacotron and Transformer respectively.

The reason why LJSpeech was chosen and LCSpeech was produced is that the only difference between them is how their recordings were being segmented and annotated with the corresponding texts. Both of them are originated from the same recordings from seven different books. The former one is segmented and annotated manually according to the clauses basically. The latter one is segmented and annotated by ARS in terms of their word error rate (WER) results. In this case, the performances of TTS trained with LJSpeech and LCSpeech are able to reflect the performance of ARS indirectly. Besides, training with QCSpeech, which is a completely self-made TTS training dataset, is to test out the performance of a self-made speech dataset.

In this experiment, there are two evaluation metrics that have been used to evaluate the performance of TTS. The first one is an ASR-based evaluation by WER to see how many words of synthesized speech could be recognized by Deep Speech. The other one is a subjective evaluation by mean opinion score (MOS) to reflect the naturalness of synthesized speech from a human's perspective.

Eight samples have been selected, which have not been used in TTS training, for the WER and MOS evaluation. For MOS, there are three participants involved.

[1] https://github.com/keithito/tacotron
[2] https://github.com/soobinseo/Transformer-TTS

Method	WER
Transformer (LJSpeech)	0.349
Transformer (LCSpeech)	0.632
Transformer (QCSpeech)	0.936
Tacotron (LJSpeech)	0.238
Tacotron (LCSpeech)	0.751
Tacotron (QCSpeech)	0.978

Table 6.5: The Mean of WER results

Method	MOS
Transformer (LJSpeech)	4.375
Transformer (LCSpeech)	2.833
Transformer (QCSpeech)	1.583
Tacotron (LJSpeech)	4.791
Tacotron (LCSpeech)	2.667
Tacotron (QCSpeech)	1.292

Table 6.6: MOS results

Word Error Rate. The mean of their WER results is shown in Table. 6.5. The WER results of LJSpeech (Transformer and Tacotron) are lower than the results of other datasets, no matter in which TTS model, which means TTS models trained with LJSpeech receives the highest rates of accuracy. Even though LJSpeech performs better in Tacotron, LCSpeech and QCSpeech perform better in Transformer. It is probably due to the fact that Transformer generally learn faster than Tacotron within the same amount of time. Especially for QCSpeech, in Transformer, $10\% \sim 20\%$ of words were able to uttered while Tacotron trained with QCSpeech hardly utters any words.

Mean Opinion Score. The results of MOS are shown in Table. 6.6. Similar patterns have been observed. LJSpeech ranked the first place. LCSpeech ranked the second place. LJSpeech in Tacotron received a higher score than the one in Transformer, while both LCSpeech and QCSpeech received a higher score in Transformer rather than in Tacotron.

7 Conclusion

In this paper, we proposed an ASR-based automatic recording segmentation strategy for TTS training dataset. This strategy is able to segment speech into audio clips and align audio clips with corresponding texts automatically. In addition, the low-quality components of speech can be gotten rid of by calculating the WER between the transcripts of ASR and the original texts.

The dataset with clean recordings processed by our method can contain 71% alignments with the accuracy of 70%, which is higher than the manual segmentation, while the dataset with self-made recordings can contain 41.5% alignments with the same accuracy. The experimental evaluation results demonstrated that Transformer received the WER of 0.632 and the MOS of 2.833 on the LCSpeech (ARS-processed datasets with clean recordings). While for the dataset with our recordings, it's still challenging for TTS to train.

In future work, we will work on a more accurate alignment with a lower WER and a dynamic segmentation of speech resulting in different lengths of audio clips. Additionally, TTS will be trained for a longer period.

Bibliography

Arık, S. Ö., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., Li, X., Miller, J., Ng, A., & Raiman, J. (2017). Deep voice: Real-time neural text-to-speech. *International Conference on Machine Learning*, 195–204.

Bansal, P., Pradhan, A., Goyal, A., Sharma, A., & Arora, M. (2014). Speech synthesis-automatic segmentation. *International Journal of Computer Applications*, *98*(4), 29–31.

Emphasis (telecommunications). (2021). https://en.wikipedia.org/wiki/Emphasis_(telecommunications)

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., & Coates, A. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Ito, K., & Johnson, L. (2021). The lj speech dataset. 2017. *URL https://keithito.com/LJ-Speech-Dataset*.

Li, N., Liu, S., Liu, Y., Zhao, S., & Liu, M. (2019). Neural speech synthesis with transformer network. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 6706–6713.

Loweimi, E., Ahadi, S. M., Drugman, T., & Loveymi, S. (2013). On the importance of pre-emphasis and window shape in phase-based speech recognition. *International Conference on Nonlinear Speech Processing*, 160–167.

McAuliffe, M., Socolof, M., Mihuc, S., Wagner, M., & Sonderegger, M. (2017). Montreal forced aligner: Trainable text-speech alignment using kaldi. *Interspeech*, *2017*, 498–502.

Mean opinion score. (2021). https://en.wikipedia.org/wiki/Mean_opinion_score

Mozilla/deepspeech. (2021). https://github.com/mozilla/DeepSpeech/blob/5f566f442541d76f320ee081294cdef980c361d1/native_client/python/client.py

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

Qi, W., Gong, Y., Jiao, J., Yan, Y., Chen, W., Liu, D., Tang, K., Li, H., Chen, J., & Zhang, R. (2021). Bang: Bridging autoregressive and non-autoregressive generation with large scale pretraining. *International Conference on Machine Learning*, 8630–8639.

Ren, Y., Hu, C., Tan, X., Qin, T., Zhao, S., Zhao, Z., & Liu, T.-Y. (2020). Fastspeech 2: Fast and high-quality end-to-end text to speech. *arXiv preprint arXiv:2006.04558*.

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., & Skerrv-Ryan, R. (2018). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4779–4783.

Soltau, H., Liao, H., & Sak, H. (2016). Neural speech recognizer: Acoustic-to-word lstm model for large vocabulary speech recognition. *arXiv preprint arXiv:1610.09975*.

Sotelo, J., Mehri, S., Kumar, K., Santos, J. F., Kastner, K., Courville, A., & Bengio, Y. (2017). Char2wav: End-to-end speech synthesis.

Szymański, M., & Grocholewski, S. (2006). Post-processing of automatic segmentation of speech using dynamic programming. *International Conference on Text, Speech and Dialogue*, 523–530.

Taylor, P. (2009). *Text-to-speech synthesis*. Cambridge university press.

Wang, W., Shuang, X., & Bo, X. (2016). First step towards end-to-end parametric tts synthesis: Generating spectral parameters with neural attention. *Interspeech*.

Wang, W., Xu, S., & Xu, B. (2016). First step towards end-to-end parametric tts synthesis: Generating spectral parameters with neural attention. *Interspeech*, 2243–2247.

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

Word error rate. (2021). https://en.wikipedia.org/wiki/Word_error_rate

Ziółko, B., Manandhar, S., Wilson, R. C., & Ziółko, M. (2006). Wavelet method of speech segmentation. *2006 14th European Signal Processing Conference*, 1–5.