SYNTHSING: A SINGING VOICE SYNTHESIZER

James Bunning, Shiyu Mou, Sharada Murali, Mu Yang and Yixin Yang
`{bunning,shiyumou,sharadam,yangmu,yixinyan}@usc.edu`

ABSTRACT

Recent advances in synthesizing singing voice from a music score and lyrics have been achieved using a Neural Parametric Singing Synthesizer (NPSS) based on a modified version of the WaveNet architecture. This model trains on phonetic transcriptions of lyrics and acoustic features, and is able to significantly reduce training and generation times while still achieving the sound quality and naturalness of state-of-the-art concatenative systems. Although NPSS can model a specific singer's voice and style of singing, it cannot generate a new singing performance given new singer data. SynthSing, our WaveNet-based singing synthesizer, expands on the NPSS synthesizer to generate a singer's voice given new singing data and has the potential to transform the timbre of one singer's voice into that of another singer. In addition to implementing SynthSing, we also synthesized singing from lyrics using Tactron, another end-to-end TTS model. Comparing the synthesized singing samples generated by the WaveNet and Tactron architectures, we found that the WaveNet-based model produces superior results in terms of naturalness and sound quality.

1 Introduction

Traditionally, speech and singing synthesizers have used concatenative methods[1, 2] for audio generation because they can naturally reproduce the spectral characteristics of the human voice. Concatenative synthesis, also known as Unit Selection Synthesis (USS), is a technique in which audio clips are synthesized by stringing together short samples of previously recorded sounds (also called units). Since all these units are taken from recordings of a speaker, they preserve the speaker's voice and are of high quality. However, these systems require many hours of recordings (>20 hours) and, consequently, large development times. This method is particularly disadvantageous for singing applications as it cannot generalize beyond a speaker's pre-recorded dataset.

In recent years, considerable breakthroughs have been made in using machine learning approaches for speech synthesis. Such systems have been able to achieve generation quality matching that of concatenative methods, sometimes even surpassing it. TTS models that use neural networks can be trained on data from multiple speakers, and generalize better than concatenative systems. WaveNet[3] is one such example that directly operates on raw audio, generating speech sample-by-sample. It has been shown that WaveNet can be used to model any audio signal, and can successfully synthesize sounds from musical instruments.

A Neural Parametric Singing Synthesizer (NPSS)[4], based on the WaveNet model, has been shown to successfully generate high-quality singing even when trained on relatively small datasets. Unlike WaveNet, it predicts parametric vocoder features rather than raw audio samples, resulting in faster training and generation.

In this work, we present SynthSing, an extension of the timbre model of NPSS, with the ability to train on multiple singers. Like NPSS, this system models vocoder features and employs a multi-stream architecture. We have also facilitated singing synthesis using Tacotron, and the results of our comparative study between the two models are presented in later sections.

2 The SynthSing Model

2.1 Overview

SynthSing, our parametric WaveNet-based singing synthesizer, is based on the timbre model of the Neural Parametric Singing Synthesizer (NPSS)[4]. The network is trained on time-aligned phonemes from song lyrics and acoustic features extracted from the song's audio using the WORLD[5] vocoder. In order to synthesize singing, the network

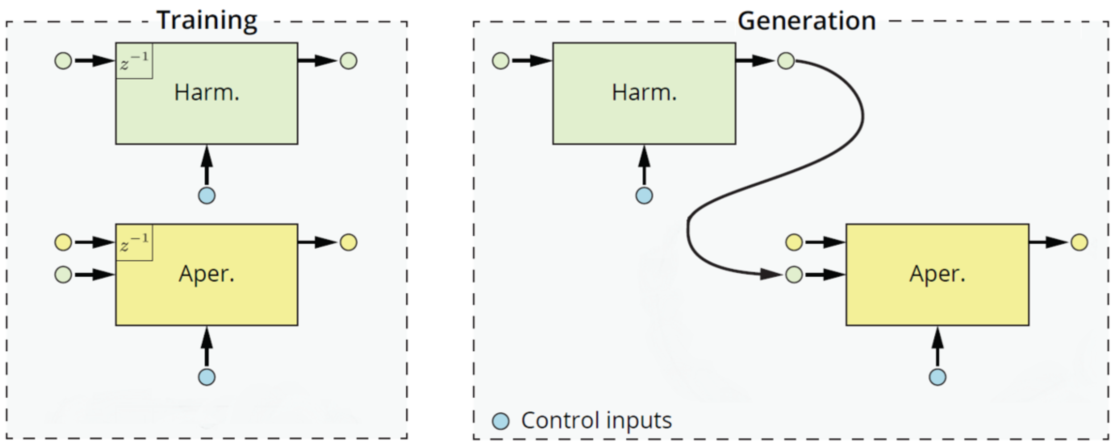

Figure 1: Architecture of the SynthSing Model, adapted from [4].

is given F0 and phonetic timing information to predict 60 log-Mel Spectral Frequency Coefficients (MFSCs) and 4 Band Aperiodicity (AP) coefficients for a frame of audio. The predicted log-MFSCs and AP coefficients along with the provided F0 are then passed as parameters to WORLD for synthesis. Unlike the NPSS synthesizer, which predicts F0 from notes in a music score via a pitch model, we provide ground truth F0 values from a recording to SynthSing and the WORLD vocoder for synthesis.

Similar to the NPSS system, SynthSing consists of two separately trained WaveNet models - a harmonic model and an aperiodic model - detailed in Figure 1. During training, the harmonic model receives 60 log-MFSCs as inputs along with F0 (coarse-coded), the previous, current, and next phoneme (one-hot encoded), and the normalized phoneme position (coarse-coded) as control inputs. The aperiodic model takes 4 AP coefficients as inputs, plus 60 log-MFSCs as additional inputs, along with the same control inputs as the harmonic model. During generation, the harmonic model predicts 60 log-MFSCs given F0 and phonetic timing information. The harmonic model's log-MFSC predictions are then fed into the aperiodic model to predict the 4 AP coefficients.

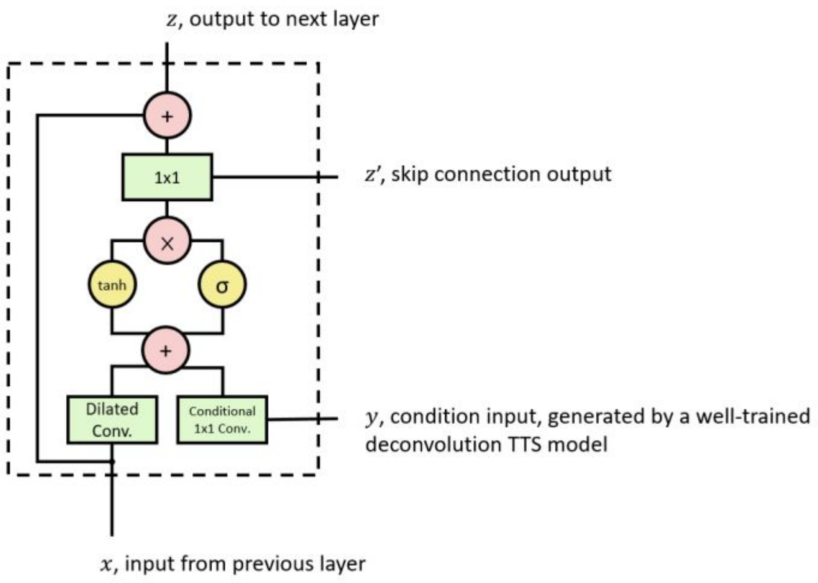
z, output to next layer
+
1x1
z', skip connection output
×
tanh
σ
+
Dilated Conv.
Conditional 1x1 Conv.
y, condition input, generated by a well-trained deconvolution TTS model
x, input from previous layer

Figure 2: Structure of a single layer in WaveNet[3].

2.2 Network Architecture Details - WaveNet

The harmonic and aperiodic submodels that make up the SynthSing model are based on a modified version of the WaveNet architecture. WaveNet[3] is a fully probabilistic generative model, capable of audio generation conditioned on other control inputs. The vanilla implementation of WaveNet directly models the raw waveform of the audio signal sample by sample. While this approach is computationally intensive, it also enables synthesis of more natural-sounding speech. The structure of one layer in WaveNet is shown in Figure 2.

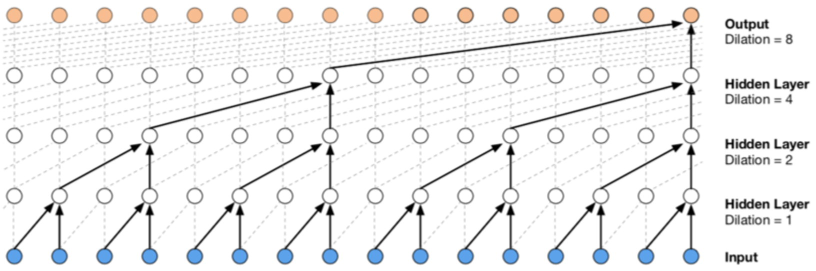

Figure 3: Predictions in WaveNet[3].

The WaveNet model is autoregressive, which means that the predicted distribution of each sample is conditioned on the previous ones, as depicted in Figure 3. By using causal convolutions, the model ensures that the prediction at time t cannot depend on the input at future timesteps.

Some key building blocks of the WaveNet architecture include:

- **Dilated causal convolutions**: These increase the size of the receptive field, allowing the network to predict new frames based on a reasonably long range of previous frames.
- **Gated activation units**:
$$z = \tanh(W_{f,k} * \mathbf{x}) \odot \sigma(W_{g,k} * \mathbf{x}) \tag{1}$$
where $*$ denotes a convolution operator, $\odot$ denotes element-wise multiplication, $\sigma(\cdot)$ is the sigmoid function, and k denotes the layer index. f and g denote filter and gate respectively, while $W_{f,*}$ is a learnable convolutional filter.
- **Conditional input variables**: The conditional input variables facilitate both local and global conditioning. Local conditioning refers to conditioning the network on phonetic and acoustic features whereas global conditioning involves conditioning the network on singer identity.
  - For global conditioning:
$$z = \tanh(W_{f,k} * \mathbf{x} + V_{f,k}^T \mathbf{h}) \odot \sigma(W_{g,k} * \mathbf{x} + V_{g,k}^T \mathbf{h}) \tag{2}$$
The conditional vector $\mathbf{h}$ is a latent representation that influences the output across all frames (e.g. speaker identity). $V_{f,*}$, $V_{g,*}$ are learnable linear projections. Vectors $V_{f,k}^T\mathbf{h}$ and $V_{g,k}^T\mathbf{h}$ are broadcast over the time dimension.
  - For local conditioning:
$$z = \tanh(W_{f,k} * \mathbf{x} + V_{f,k} * \mathbf{y}) \odot \sigma(W_{g,k} * \mathbf{x} + V_{g,k} * \mathbf{y}) \tag{3}$$
where $\mathbf{y}$ is the time series of local control inputs (e.g. phoneme identity) which have the same time resolution as the non-control inputs $\mathbf{x}$. $V_{f,*}$, $V_{g,*}$ are 1-by-1 convolutions.
- **Residual and skip connections**: These enable the training of deeper networks.

An overview of the complete WaveNet-based timbre model architecture is shown in Figure 4. Individual WaveNet layers are stacked to form a deeper network. Skip connections, activation functions, and Constrained Gaussian Mixtures (CGMs) are used to compute the final output. Unlike vanilla WaveNet, which performs sample-to-sample prediction on raw audio, our timbre model makes frame-to-frame predictions of MFSCs and AP coefficients using local control inputs (F0, phoneme identity, and normalized phoneme position) aligned at the frame level, and global control inputs (singer identity).

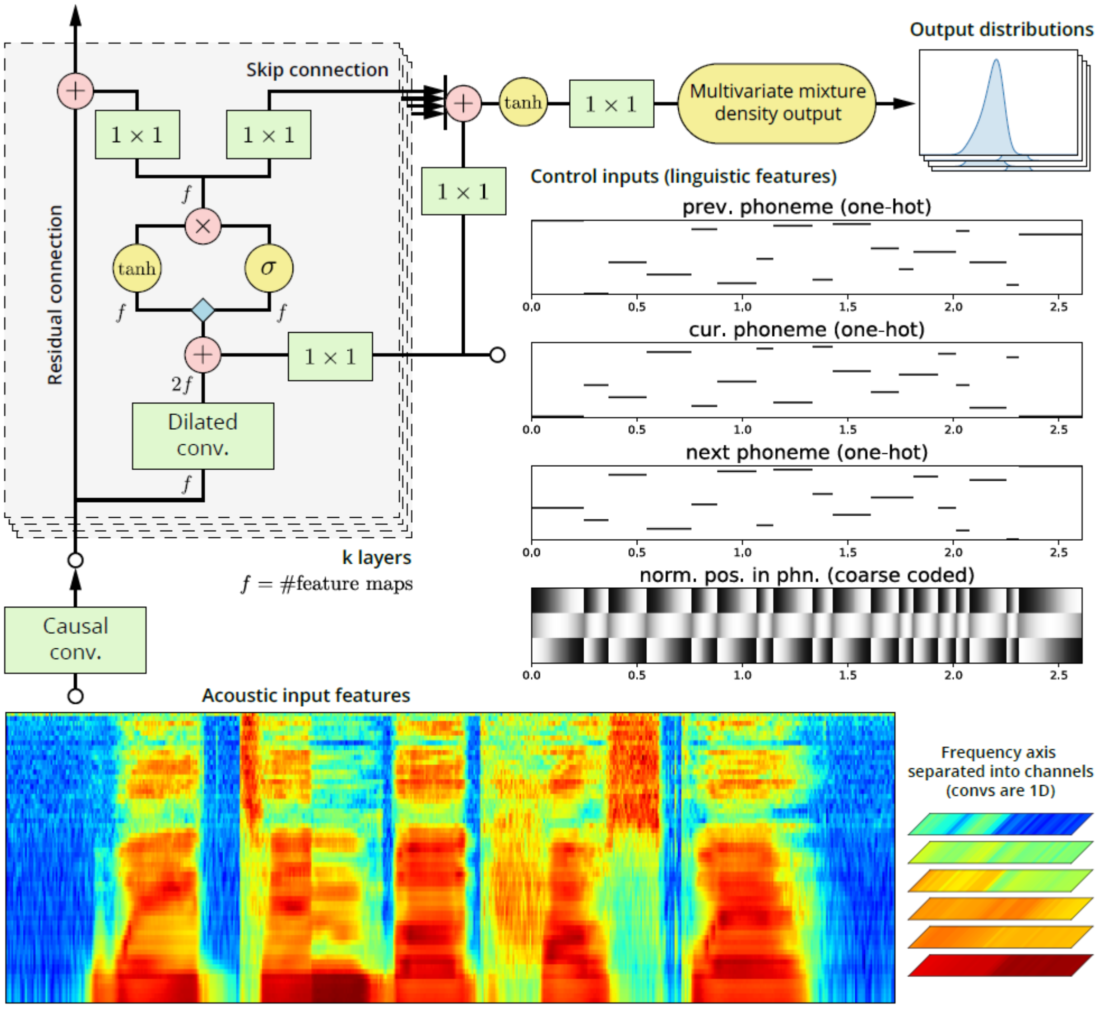

Figure 4: Overview of the Timbre Model from [4].

2.3 Constrained Gaussian Mixture Output

The traditional sample-to-sample WaveNet model performs μ-law encoding on the raw audio input, while the final output is predicted as a 256-way softmax. However, since the vocoder features (log-MFSCs and AP coefficients) predicted by our model vary smoothly, they can be modeled as continuous variables. In this case, a softmax output layer is excessive and will result in a large number of parameters (256 per input feature). Therefore, in our parametric WaveNet model, we replace the softmax output layer with a Gaussian Mixture Model (GMM) prediction with 4 mixtures, resulting in 12 parameters per input feature. These 12 parameters are further reduced to 4 by means of a Constrained Gaussian Mixture (CGM) output instead of the full GMM.

2.3.1 Generation using CGMs

The GMM parameters are predicted from 4 free CGM parameters: location ξ, scale ω, skewness α and shape β. If the outputs predicted by the network are a_0, a_1, a_2 and a_3, some non-linearities can be applied to obtain the CGM parameters, which are in turn used to obtain the GMM paramters. These relations are explicitly defined in Appendix A of [4].

2.4 Regularization

While the generation process in WaveNet is autoregressive, ground truth past samples are used during training rather than using past predictions. However, training our model on ground truth data can lead to the model being overly biased

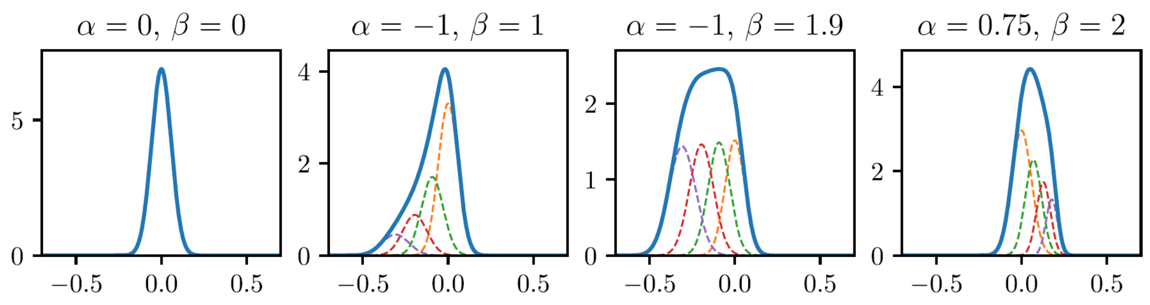

Figure 5: Some example distributions from CGMs from [4]. The dotted lines show the individual Gaussian mixtures.

towards the non-control inputs and ignore the control inputs. This will cause the synthesized singing voice to sound unnatural over time.

To prevent this, we use a denoising objective, as in [4]:

$$\mathcal{L} = -\log p(\mathbf{x}_t | \tilde{\mathbf{x}}_{<t}, \mathbf{c}) \quad \text{with} \quad \tilde{\mathbf{x}}_{<t} \sim p(\tilde{\mathbf{x}}_{<t} | \mathbf{x}_{<t}) \tag{4}$$

$$\text{where} \quad \tilde{\mathbf{x}}_{<t} = \mathcal{N}(\tilde{\mathbf{x}}, \mathbf{x}, \lambda I) \tag{5}$$

is the corrupted Gaussian distribution. Thus, while training, Gaussian noise is added to the input of the network to predict the uncorrupted target. In our model, the corrupted input is min-max normalized while the target is not normalized. The noise level λ is set to 0.4.

2.5 Loss Function

Since our timbre model predicts Gaussian mixture parameters instead of a softmax output, the traditional softmax cross-entropy loss function employed in WaveNet can no longer be used. Instead, we use the negative log-likelihood measure, given in equation 6:

$$\mathcal{L}(y|x) = -\log \left[\sum_{k}^{K} \Pi_k(x) \phi\left(y, \mu_k(x), \sigma_k(x)\right) \right] \tag{6}$$

where y is the target output and Π_k, μ_k and σ_k are the GMM parameters predicted by the model for the k^{th} Gaussian mixture for input x.

3 The Tacotron Model

Tacotron[6] is a fully end-to-end synthesis model, used in Text-to-Speech (TTS) synthesis. Most conventional TTS systems consist of multiple stages such as a frontend acoustic model, synthesis module, etc., and building these components often requires extensive domain expertise and an intensive implementation process. Additionally, as these blocks are trained separately, errors in each component may compound. By contrast, Tacotron is more like a "black box"; as long as the model is well-built, it can be trained completely from scratch with random initializations. Like SynthSing, Tacotron makes predictions at the frame level, rendering a faster generation process than sample-to-sample vanilla WaveNet.

The Tacotron model takes input <text, audio> pairs of aligned word sequences and the corresponding audio data. Unlike most TTS systems, Tacotron does not require phoneme-level alignment, so the model scales well to large amounts of transcribed data. Though this is conventionally used as a TTS system, we implemented this model on singing data to see how well it performs in comparison to SynthSing.

3.1 Model Architecture

Tacotron is a seq2seq model with an attention mechanism. It is composed of an encoder, decoder and post-processing network, as shown in Figure 6. The system takes in characters as input and generates a spectrogram, which is then converted to a waveform by the post-processing network.

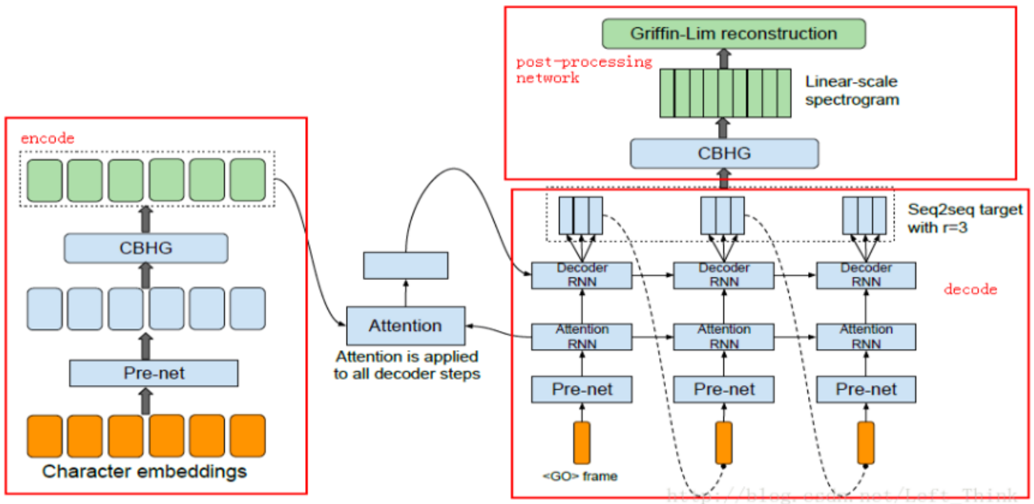

Figure 6: Tacotron Model Architecture, from [6].

3.1.1 Encoder

The goal of the encoder is to extract robust sequential representations of text. Each character input to the encoder is represented as a one-hot vector and embedded into a continuous vector. The data is then fed into the pre-net, which is a bottleneck layer with dropout. This helps convergence and improves generalization.

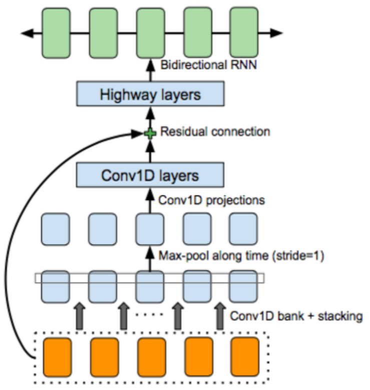

Figure 7: CBHG network, from [6].

CBHG Network: The CBHG network, shown in Figure 7, is used to extract representations from sequences. It takes in the output of the pre-net and produces the final output used by the attention mechanism. The CBHG network consists of a bank of 1-D convolutional filters, followed by highway networks and a bidirectional gated recurrent unit (GRU) recurrent neural net (RNN). The input sequence is first convolved with K sets of 1-D convolutional filters, where the k^{th} set contains C_k filters of width k ($k = 1, 2, \dots, K$). These filters are used to model local and contextual information, similar to modeling unigrams, bigrams, etc. (up to K-grams).

The next stage consists of the highway layers, which are used to extract high-level representations. Here, the input is fed into two fully connected networks at the same time, with ReLu and sigmoid activations respectively. If the output of the ReLu network is $output_1$, and that of the sigmoid is $output_2$, then the output of the highway layer is:

$$output = output_1 \times output_2 + input \times (1 - output_2) \tag{7}$$

The Tacotron model contains 4 highway layers.

The final stage of CBHG is the bidirectional GRU, and the resultant output from the GRU is the output of the encoder. GRU is a variant of an RNN that uses the same threshold mechanism as an LSTM, except that it only has update gates and reset gates.

3.1.2 Decoder

Before introducing the encoder output to the decoder, it is passed through the attention network. In the encoder-decoder structure, the encoder converts input words into vector representations. Each time a word is predicted, the output of the encoder is given different weights. The weighted sum of the encoder output is then input to the decoder. Thus, a position with weight 0 is not given any importance in this forecast, but those locations with large weights are given focus.

3.1.3 Post-processing network

The post-processing network's task is to convert the seq2seq target to a target that can be synthesized into waveforms. Since Griffin-Lim is used for synthesis, the post-processing net learns to predict the spectrogram on a linear frequency scale. In contrast to traditional seq2seq networks, which usually run from left to right, this one has both forward and backward information to reduce the prediction error for each frame. As this network is highly general, it can also be used to predict alternative targets such as vocoder parameters, or even waveform samples, akin to WaveNet.

4 Complexity Analysis

4.1 SynthSing Model

The harmonic submodel of SynthSing has 925,714 trainable parameters and the aperiodic submodel has 54,860 trainable variables. Each submodel was trained for 10 million timesteps, taking about 3-4 hours each. The loss variation during training is shown in Figure 8.

One of the major advantages of using CGM predictions in our model over softmax predictions is the immense reduction in parameter space, which speeds up generation time by a considerable amount. On average, the softmax prediction method in vanilla WaveNet takes over 20 minutes for generating 10s of 16kHz audio data. In comparison, the SynthSing model with CGM predictions takes less than $\sim$2-3 minutes for generating 10s of 32kHz audio data.

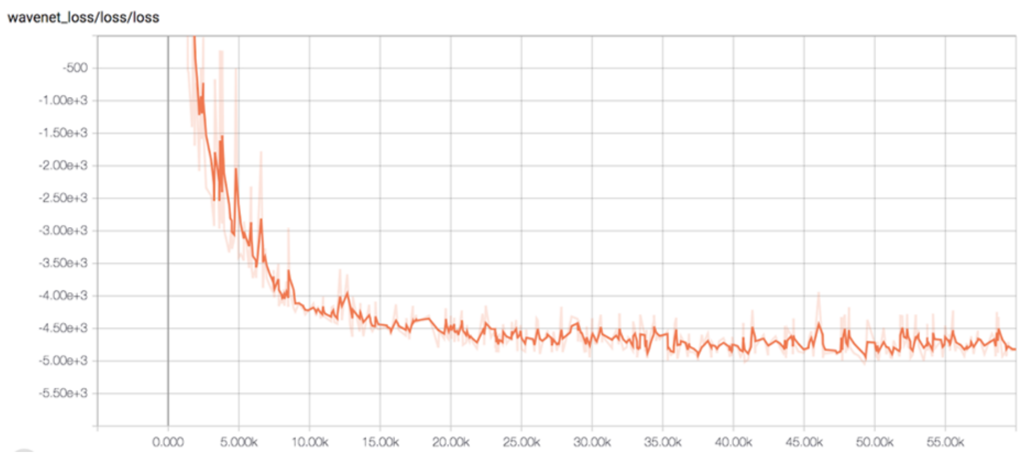

Figure 8: Loss variation during training.

4.2 Tacotron Model

The Tacotron model was trained for 20,000 steps, taking around 6 hours. Since Tacotron is an end-to-end model, it requires a larger dataset and much more training time to attain results on par with SynthSing's output. When generating, 10s of audio is synthesized for each input text sentence (e.g. line of lyrics).

5 Datasets

5.1 SynthSing Dataset

We trained the SynthSing model on two different datasets. The first dataset, NIT-SONG070-F001 (`http://hts.sp.nitech.ac.jp/archives/2.3/HTS-demo_NIT-SONG070-F001.tar.bz2`), provided by the Nagoya Institute of Technology (NITech), is a publicly available dataset containing 31 studio-quality recordings of a Japanese female singer singing nursery rhymes. The Japanese nursery rhyme dataset contains 34 phonemes, including "pau".
The second dataset, created by the authors of this paper, is a collection of 10 Coldplay songs with isolated lead vocal tracks downloaded from YouTube. The Coldplay dataset contains 39 phonemes, including "pau". All audio files in both datasets were resampled to 32 kHz for consistency.

5.1.1 Phoneme Preprocessing and Alignment

The Japanese nursery rhyme dataset provides timestamps (in nanoseconds) for all phonemes contained in the 31 recordings. After extracting acoustic feature frames from these recordings via the WORLD vocoder, we used the timestamps to align each phoneme to its corresponding acoustic feature frame. With the phonemes frame-aligned, we were able to determine the previous, current, and next phoneme for each frame. We also used the frame-aligned phonemes to create a 3-state coarse-coded vector for each frame corresponding to the normalized position of the current frame within the current phoneme (i.e. the probability of the current frame being in the beginning, middle, or end of the phoneme).

Aligning phonemes with acoustic feature frames for songs in the Coldplay dataset proved much trickier because unlike the Japanese nursery rhyme dataset, we did not have access to phonetic timing information. To solve this problem, we used Gentle[7], a robust yet lenient forced-aligner built on Kaldi. We input short Coldplay audio clips along with their accompanying lyrics into Gentle, and used Gentle's output to generate timestamped phonemes similar to the Japanese nursery rhyme dataset. We then followed the procedure described above to one-hot encode and coarse-code the frame-aligned phonemes.

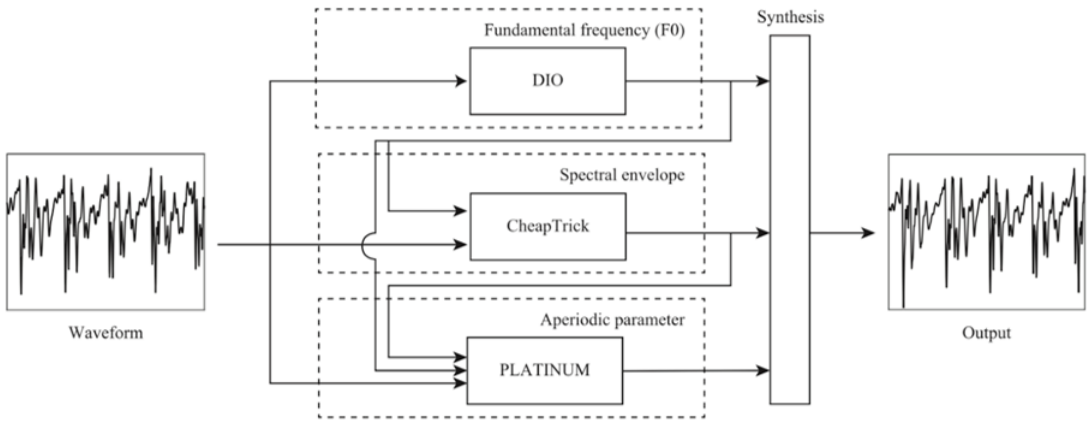

Figure 9: Overview of the WORLD vocoder from [5].

5.1.2 Acoustic Feature Extraction

The acoustic features used for training are directly extracted from each dataset's audio files using WORLD[5]. The three acoustic features required for training and synthesis - F0, harmonic spectral envelope, and aperiodicity envelope - are computed by WORLD via the algorithms depicted in Figure 9 using a 5 ms hoptime and 2048-sample FFT.

1. **F0 Extraction:**
   The WORLD vocoder extracts the fundamental frequency (F0) using the Harvest algorithm[8], which obtains F0 candidates from several band-pass filters and refines them using the instantaneous frequency. The voiced/unvoiced decision determined by Harvest is further refined with WORLD's StoneMask algorithm.

   Coarse-coding F0: During training, the timbre model requires some pitch information to predict acoustic features. However, this has to be independent of the "absolute" F0 value in order to facilitate synthesis in any range. One way to do this is to convert the F0 values from Hz to cents using a reference F0:

$$f_{\text{cents}} = 1200 \times \log_2 \frac{f_{\text{Hz}}}{f_{\text{ref}}} \tag{8}$$

This reference F0 is then used to coarse-code the F0 (in cents) to 3 states, giving an indication of the current F0 value relative to f_{ref}.

2. **Harmonic Feature Extraction:**
The harmonic spectral envelope is estimated from raw audio by WORLD using the CheapTrick algorithm[9]. CheapTrick adaptively windows F0, smooths the frequency domain of the power spectrum, and then lifters in the quefrency domain for spectral recovery. We further reduce the dimension of the harmonic feature from 1024 to 60 by converting the spectral envelope to log-MFSCs. This conversion is performed by the Speech Signal Processing Toolkit (SPTK)[10], by frequency warping in the cepstral domain[11]. During generation, the log-MFSCs predicted by the harmonic model are converted back to the spectral envelope using SPTK for WORLD to synthesize.

3. **Band Aperidocity Estimation:**
The WORLD vocoder determines band aperiodicity coefficients via the D4C algorithm[12]. D4C calculates temporally static parameters on the basis of group delay, computes parameter shaping, and then estimates the band aperiodicity from those shaped parameters. We reduce the dimensionality of the AP coefficients generated by WORLD from 1024 to 4 to reduce the parameter space during training. However, before synthesis, we expand the dimensionality from 4 coefficients back into 1024.

5.2 Tacotron Dataset

The Tacotron model only requires sentence-level audio and sentence-aligned lyrics as input for training. Since no such dataset is publicly available, we generated our own dataset using .lrc files (popular in the .mp3 player days for displaying synced lyrics while a song plays). Each .lrc file is composed of lyric sentences with the corresponding beginning time. We collected 6 Ed Sheeran songs with isolated vocals from YouTube and trimmed these sentence by sentence according to the .lrc file timestamps. In total, 1400 samples of data were used to train the Tacotron model.

6 Existing Codebases

For the WaveNet-based SynthSing model, we started from a partial implementation of WaveNet in Tensorflow[13]. This existing codebase was a vanilla implementation with sample-based softmax predictions. Thus, it had to be retooled to predict acoustic feature frames rather than audio samples, using CGMs. Additionally, this version of WaveNet only implements global conditioning (speaker identity) and does not contain local conditioning, which is essential for training our model on phonetic features. This was a momentous task that needed to be undertaken.

In order to extract acoustic features from our SynthSing dataset audio and synthesize audio from acoustic features predicted by the model, we used WORLD, a freely available C-codebase[14] created for speech analysis and synthesis. Understanding the sparsely documented code was cumbersome, but integrating the code was relatively straightforward thanks to a handy Python wrapper[15].

For the Tacotron model, we used a Tensorflow implementation written by Park[16]. This code is comprehensive and well-documented with several example datasets. We built our Ed Sheeran dataset in the same manner as the example datasets given in this implementation. This data was then directly fed into the preprocessing code and training code. Apart from tweaking hyper-parameters of the networks, there weren't many modifications made to this source code, as we mainly used the Tacotron model for comparison.

7 Experimentation and Results

7.1 Replicating results from the dataset

After training the SynthSing model on the Japanese nursery rhyme dataset, we provided F0 and phonetic timings from the same dataset to generate MFSCs and AP coefficients. The AP coefficients were generated by feeding the predicted MFSCs to the aperiodic model. Then, with the generated acoustic features (MFSCs were converted back to the spectral envelope) and ground truth F0s given to WORLD for resynthesis, we were able to produce audio samples[1] in the timbre

[1]**Original**: `https://soundcloud.com/mu-yang-974011976/hit-004_orignal?in=mu-yang-974011976/sets/results-for-synthsing`, **Synthesized**: `https://soundcloud.com/mu-yang-974011976/hit_004_synthesized?in=mu-yang-974011976/sets/results-for-synthsing`

of the Japanese female singer from the dataset. A comparison of the original and generated MFSCs and AP coefficients are plotted in Figures 10 and 11 respectively. For visualization purposes, only the first 10 log-MFSCs are included.

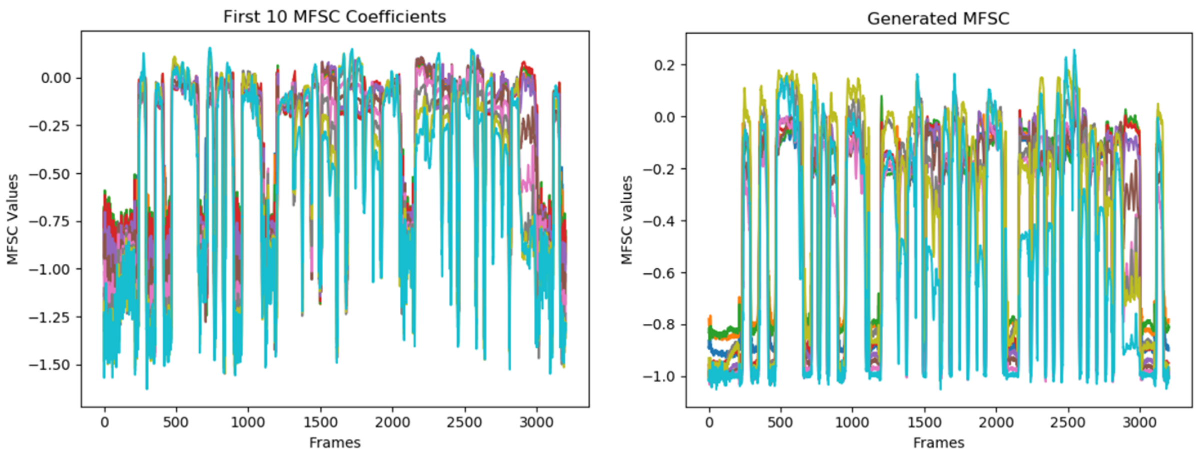

Figure 10: Original and generated log-MFSC values (first 10 coefficients).

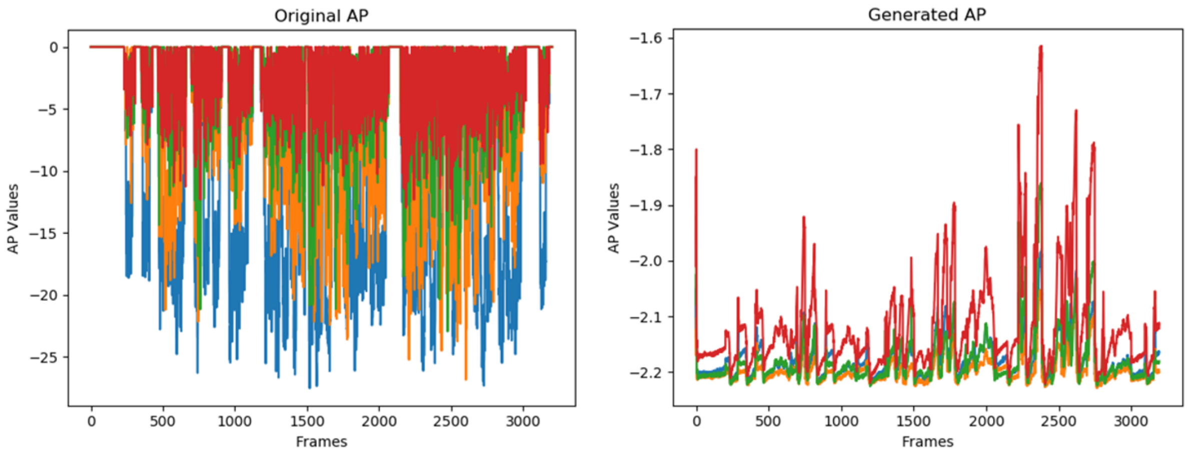

Figure 11: Original and generated AP coefficients.

Comparing the original MFSCs and AP coefficients to the generated ones, we can see that the predicted MFSCs are close to the original coefficients in both absolute values and the variations, whereas the generated AP coefficients diverge from their ground truth values. This is an expected result: due to the autoregressive nature of the model, errors compound over time. Thus, any errors during synthesis of the MFSCs will compound during generation in the aperiodic model.

Despite the divergence of AP values, the quality of our generated audio was not adversely affected. The timbre of the re-synthesized audio sounds very close to the original with only a few mispronounced phonemes. In most cases, the mispronounced phonemes were caused by the previous frames' phoneme bleeding into the current frame. This result is a negative side effect of overfitting; due to our limited data size, the model is overly biased towards the non-control inputs and thus, ignores the control inputs.

7.2 Generating previously unseen sequences

We also experimented with synthesizing previously unseen phonetic sequences[2] to demonstrate the generalization capability of our model. By splicing together random clips from the Japanese nursery rhyme recordings and doing a

[2]**Original**: `https://soundcloud.com/mu-yang-974011976/hit_scramble_original?in=mu-yang-974011976/sets/results-for-synthsing`, **Synthesized**: `https://soundcloud.com/mu-yang-974011976/hit_scramble_synthesized?in=mu-yang-974011976/sets/results-for-synthsing`

similar concatenation of the corresponding F0 and phonemes for each audio clip, we created control input sequences that the model had not seen before. Using the grafted control inputs, we generated MFSCs and AP coefficients from the coarse-coded F0s and phonetic timings and then re-synthesized the audio via WORLD. Again, the generated audio sounded realistic and natural, although some phonemes are indistinguishable due to the same reasons mentioned in the explanation of the previous experiment. However, by generating previously unseen sequences, we demonstrate that our model is able to generalize to new control inputs and generate corresponding MFSCs and AP coefficients.

7.3 Synthesis from self-created English-singer dataset

The SynthSing model was also trained on our self-created Coldplay dataset. Similar to the first experiment, we resynthesized recordings in the Coldplay dataset using true F0 and AP, and MFSCs generated by the harmonic submodel. The quality and intelligibility of the synthesized audio[3] is significantly worse than the samples generated in the previous two experiments. However, the timbre of lead singer Chris Martin's voice is clearly preserved. The inferior result is reasonable considering the smaller size of the Coldplay dataset and the lower quality of audio when compared to the Japanese nursery rhyme dataset.

7.4 Synthesis from Tacotron

We trained the Tacotron model on our homemade Ed Sheeran dataset containing studio acapella recordings aligned with lyrics. The audio from the dataset was further chopped into segments of 1-2s and sent to the Tacotron network. We found that the quality of synthesized audio for a larger dataset was considerably better than that of a smaller one. When trained with 30 minutes of audio, the singer's voice is clearly distinguishable, but does not sound very natural[4].

The Tacotron model performed very well for the few words seen in the training data with minimal phoneme distortion. However, for unseen words, the quality of synthesis was very poor even if the phonemes were previously seen by the network. This also suggests that Tacotron needs huge quantities of training data in order to perform well when synthesizing audio.

7.5 Comparing SynthSing with Tacotron

The clear advantage of Tacotron compared to SynthSing is that Tacotron requires far less data preparation and processing (lyric data only needs to be aligned at the sentence level) and the network architecture is much simpler. However, the disadvantage of Tacotron's sentence-level alignment is that we lose control over all pitch output. Additionally, Tacotron requires a much larger dataset than SynthSing for comparable performance. Tacotron also takes longer to train than SynthSing because Tacotron takes audio samples as input whereas SynthSing trains on vocoder parameters. Lastly, the samples synthesized by SynthSing sound much more natural than those produced by Tactron, even though both models were trained on similar-sized datasets. The Griffin-Lim reconstruction process in Tacotron may be a large contributing factor to the artificial sound of Tacotron's singing.

8 Major Challenges

8.1 Preparing a Training Dataset

Our most significant challenge was finding a dataset with high quality acapella recordings of a single singer with lyrics annotated at the phoneme level. The NITech Japanese nursery rhyme dataset is the only such publicly available dataset. In order to train on another dataset with an English-speaking singer, we needed to manually collect isolated vocal tracks. Preparing a dataset from scratch proved tedious for several reasons.

First, many of the "isolated" vocal tracks available online had audible phase issues or other artifacts resulting from the backing track being removed from the lead vocal. In addition, each song had to be edited into several smaller clips to ensure that the included singing samples did not contain appreciable delay effects, stacked vocal harmonies, or instrument bleed. Furthermore, each song's lyrics needed to be meticulously checked before being input into Gentle, as even minute departures like typos or extraneous words often resulted in annotation errors.

[3]**Original**: `https://soundcloud.com/mu-yang-974011976/coldplay-song02-01-007?in=mu-yang-974011976/sets/results-for-synthsing`, **Synthesized**: `https://soundcloud.com/mu-yang-974011976/coldplay_007_synthesized?in=mu-yang-974011976/sets/results-for-synthsing`

[4]**Sample 1**: `https://soundcloud.com/mu-yang-974011976/4-1?in=mu-yang-974011976/sets/tacotron`, **Sample 2**: `https://soundcloud.com/mu-yang-974011976/1-1?in=mu-yang-974011976/sets/tacotron`

8.2 Coarse Coding Implementation

While preparing conditional input data for training the SynthSing model, we found that certain features, while essential as conditional inputs, could not be given to the network as-is. For example, the WaveNet submodels need to be conditioned on pitch input (F0) to generate meaningful results. However, using the absolute value of F0 during training will prevent the models from generalizing to different pitch ranges of different singers. Thus, we need to represent these features in relative, rather than in absolute terms. One way of doing this is by coarse coding, also implemented in the NPSS system. However, very few details of the implementation were available, and existing literature on coarse coding is very generic. Adapting these concepts to our specific input features proved to be a challenge.

8.3 Implementing Local Conditioning in WaveNet

One of our biggest challenges when implementing the harmonic and aperiodic models was implementing local conditioning. An open-source implementation for sample-to-sample WaveNet with global conditioning for speaker ID is available. However, there is no existing implementation of local conditioning. Implementing local conditioning from scratch turned out to be a tough task - most notably, we had to deal with shape mismatch issues between the control inputs and non-control inputs. In order to address these issues, we needed to dive deep into every aspect of the WaveNet implementation to fully understand what happens in each convolution step of each layer, which was very time consuming.

8.4 Implementing CGM output layer

Another major bottleneck when executing the SynthSing model was our implementation of the Constrained Gaussian Mixture output layer. Unlike vanilla WaveNet, which operates directly on audio samples, SynthSing predicts continuous vocoder features. However, there were no well-defined details of the CGM implementation in the NPSS paper, and it required a lot of effort to figure them out.

In order to limit variations in the output, the GMM features predicted for a frame are shifted towards a "global" average (calculated for the current input batch), which is controlled by the temperature parameter τ. Since the SynthSing submodels take in noisy input during training but need to predict clean audio, the value of τ needs to be tweaked differently for the training and generation phases. This process proved to be quite tricky.

Our first implementation of the CGM layer resulted in very slow generation times ($\sim$ 2 hours for 100 frames). By only considering a fixed number of samples from the GMM for generation, we were able to greatly improve generation speeds (100 frames in a few seconds), with no degradation in output quality.

9 Adaptations to original plans

The system we originally intended to implement contained all three models of NPSS - the timbre model, pitch model and phonetic timing model. However, due to the challenges of implementing a WaveNet model with local conditioning and constructing a dataset from scratch, we were only able to build and train the timbre model and its two submodels - the harmonic model and aperiodic model. As we used F0 values from a recording, we did not need to implement the voiced/unvoiced submodel.

We were still able to successfully synthesize singing without the pitch model by generating audio with ground truth F0 values instead of predicting F0 values from the music score. Since the goal of our project was to synthesize the timbre of a singer's voice, we decided that constructing the timbre model was a more fruitful endeavor than simply predicting F0 pitch values from notes.

A second adaptation from our original plan was that we were unable to complete our proposed transfer learning task of generating singing in the voice of different singers - e.g. having "Taylor Swift" sing like "Ed Sheeran" given a few lyrics. However, because we incorporated global conditioning in our WaveNet architecture, we are confident that transfer learning for multi-speaker singing synthesis is possible with our current model.

Lastly, although we only intended to train and implement the WaveNet-based singing synthesizer, we also built a second one based on Tactron. Anticipating that we might not be able to implement our WaveNet-based timbre model within the given time constraints, half of our team switched to working on Tacotron as a "Plan B". As an unintended but happy coincidence, both models came to fruition, allowing us to do a comparative study between the two neural singing synthesizers.

10 Individual Contributions

- Data Preparation
  - WORLD Vocoder - James
  - Dataset creation - James
  - F0 coarse coding - Sharada
  - Spectral Envelope to log-MFSC conversion - Sharada
  - Gentle phoneme alignment - James
  - Phoneme extraction and encoding - Shiyu
  - Normalized frame position coarse coding - Shiyu
- WaveNet Model
  - Implementing harmonic and aperiodic models - Mu
  - Constrained Gaussian Mixture implementation - Sharada
  - Generation - Mu, Sharada
  - Fast generation from CGMs - Sharada
- Tacotron Model
  - Data preparation - Shiyu, Yixin
  - Model architecture - Yixin
  - Attention Mechanism optimization - Yixin
  - Training and generation - Shiyu

11 Conclusions

We have presented SynthSing, a WaveNet-based synthesizer that can produce singing in a given person's style from phonetic and pitch information. The generated singing audio from different datasets sounds natural to the ear and preserves the singer's timbre. From our experiments with generating novel phonetic sequences, we have demonstrated SynthSing's ability to generalize to new control inputs. With the combination of global and local conditioning, this system is able to train on multiple speakers, and has the potential to transform the timbre of one singer's voice into another. We have also implemented the Tacotron model for singing synthesis using aligned lyrical input. Our comparative study of the two synthesis techniques showed that the quality and naturalness of the SynthSing output is far superior to that of Tacotron's, and also requires less time for training and generation. The SynthSing system can be extended to include a pitch model to predict the fundamental frequency from a given note for a true end-to-end singing synthesizer. Another future avenue of work would be to implement transfer learning between singers, similar to implementations in state-of-the-art TTS systems.

References

[1] Vocaloid. `https://www.vocaloid.com/en/`. Accessed: 2018-11-30.

[2] J. Bonada and X. Serra. Synthesis of the singing voice by performance sampling and spectral models. *IEEE Signal Processing Magazine*, 24(2):67–79, March 2007.

[3] Aaron van den Oord et al. WaveNet: A Generative Model for Raw Audio, 2016.

[4] Merlijn Blaauw and Jordi Bonada. A Neural Parametric Singing Synthesizer Modeling Timbre and Expression from Natural Songs. *Applied Sciences*, 7:1313, 12 2017.

[5] Masanori Morise, Fumiya Yokomori, and Kenji Ozawa. WORLD: A Vocoder-Based High-Quality Speech Synthesis System for Real-Time Applications. *IEICE Transactions*, 99-D:1877–1884, 2016.

[6] Yuxuan Wang et al. Tacotron: Towards End-to-End Speech Synthesis, 2017.

[7] R. M. Ochshorn and M. Hawkins. Gentle, a forced aligner built on Kaldi. `http://lowerquality.com/gentle`. Accessed: 2018-11-30.

[8] Masanori Morise. Harvest: A High-Performance Fundamental Frequency Estimator from Speech Signals. In *INTERSPEECH*, 2017.

[9] Masanori Morise. CheapTrick, a Spectral Envelope Estimator for High-Quality Speech Synthesis. *Speech Communication*, 67, 01 2014.

[10] Speech Signal Processing Toolkit (SPTK). `http://sp-tk.sourceforge.net`. Accessed: 2018-11-30.

[11] Keiichi Tokuda, Takao Kobayashi, Takashi Masuko, and Satoshi Imai. Mel-generalized cepstral analysis - a unified approach to speech spectral estimation. In *Proc. ICSLP*, 01 1994.

[12] Masanori Morise. D4C, a Band-Aperiodicity Estimator for High-Quality Speech Synthesis. *Speech Communication*, 84, 09 2016.

[13] A TensorFlow implementation of DeepMind's WaveNet paper. `https://github.com/ibab/tensorflow-wavenet`. Accessed: 2018-11-30.

[14] WORLD - a high-quality speech analysis, manipulation and synthesis system. `https://github.com/mmorise/World`. Accessed: 2018-11-30.

[15] PyWorldVocoder - A Python wrapper for World Vocoder. `https://github.com/JeremyCCHsu/Python-Wrapper-for-World-Vocoder`. Accessed: 2018-11-30.

[16] Kyubyong Park. A (Heavily Documented) TensorFlow Implementation of Tacotron: A Fully End-to-End Text-To-Speech Synthesis Model. `https://github.com/Kyubyong/tacotron`. Accessed: 2018-11-30.