CS230

Towards an End-to-End Deep Singing Synthesizer

Juncen Wang
Department of Computer Science
Stanford University
jwang19@stanford.edu

Abstract

This project considered the first steps in creating an end-to-end deep singing synthesizer, attempting to generate an audio spectrum or waveform of a song directly from input features. I explored using a recent convolutional algorithm called WaveNet, along with an RNN implemented convolutionally. While the outputs did not subjectively sound natural, they were understandable given the input lyrics.

1 Introduction

From the earliest eras of computer music, researchers have tried to get computers to sing - and discovered that it is strikingly difficult to model the human voice in a natural-sounding way. The singing synthesis task itself is not difficult to describe; given an input of lyrical musical notation of a song - a time series of lyric and pitch (for example, in a form similar to a midi) - a waveform should be generated that sounds convincingly like a human's performance of the song. However, under the surface, the problem is quite complex. This criteria of "humanness" is not easy to define objectively, especially as the artistic expression and style of a singing performance comes into the question. Furthermore, the exact timings of the pitch and syllables in a singing performance often varies considerably depending on expression and style, and other timbrical features (such as what is known as "head voice" and "chest voice" by singers) affecting the output audio are not entirely dependent on the input pitch and lyric. Because of this complexity, deep learning may find a better solution than hand-engineering features. This particular project focused on the first steps towards an end-to-end synthesizer - generating sound at one constant pitch and rhythm depending on the input lyrics.

2 Related work

There have been many proposed algorithms for singing synthesis, such as physical modeling of the vocal tract or generating raw audio based off of spectral features. Most of these integrate very little machine learning if at all. The most widespread method, and which is currently the state of the art, is unit selection. These algorithms use a database of human-recorded vocals; at synthesis time, they use signal processing methods to select sections from the database that match the requested lyrics, concatenate them, and do post-processing to ensure the result sounds natural. This has been commercialized in the Vocaloid software, and is also widely used in text-to-speech applications such as the first iterations of Apple's Siri. While these are generally not computationally expensive, the signal processing behind the synthesis and selecting the features is an extremely difficult task, and since unit selection algorithms rely on a recorded database, they are usually not very proficient at generating different types of voices without multiple databases. Other methods include formant synthesis, which encodes and generates audio using hand-engineered spectral features, and using a

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

physical model of the vocal tract, both which had been used in early speech synthesizers. While they are usually more lightweight than unit selection algorithms and generalize to different voices better, they have yet to match the quality of unit selection.

However, there has also been much work applying machine learning to the task. Hidden Markov model-based synthesis uses learned parameters to estimate the maximum likelihood of the next value in time for a set of features, which are then converted to a waveform using a vocoder. Recently, research applying deep learning to estimating similar features has also been successful, and rivals many unit selection algorithms in sounding human.

The Blaauw and Bonada paper utilizes an algorithm based off of DeepMind's WaveNet, a recent advancement in text-to-speech, generating audio sample-by-sample using a convolutional neural network. As previously mentioned, rather than generating the waveform itself, it estimates features that are then converted to audio using a vocoder. In this project, I wanted to build off of this and attempt to build a fully end-to-end convolutional neural network to synthesize singing, bypassing those intermediary features.

3 Dataset and Features

I used freely available databases of raw recordings created for an existing unit selection synthesizer for the Japanese language, available from this website. The database is simply a collection of labeled files, with about six hours of recordings all recorded by one singer. Each recording consists of a string of five to eight sung syllables (of (C)V syllabic structure), and has a constant pitch and rhythm per syllable throughout.

This database can be split into different subsets of recordings using different singing styles; those subsets may be further organized into sections of files at one pitch (for example, out of the 3500 files, 600 are of a "whispery" style, and out of those 600, 150 are at the pitch A4.) In addition, each database comes with a configuration file from which time labels of each phoneme in each file can be easily extracted.

I used a 97/1.5/1.5 train/dev/test split. The audio was sampled at a sampling rate of 22050 Hz, and the labels, originally in a mix of Latin and Japanese characters, was romanized to entirely the Latin alphabet in order to better reflect the pronunciation. For the DC-TTS model, which trained on mel-spectrograms and generated a mel-spectrogram, the data was preprocessed by taking a mel-filtered STFT with 80 mel banks and variable FFT window and hop size depending on hyperparameters. For some experiments on this model, I also transformed the labels from a string into a time series sampled at the same rate as the mel-spectrogram windows, since that is more similar to what would eventually be the use case of the model when rhythm is not consistent.

4 Methods

I tested on two different algorithms, both convolutional - WaveNet and DC-TTS.

WaveNet is a convolutional neural network that learns a conditioning of audio on previous samples, with optional additional local conditions. It doesn't use any pooling layers, and consists of a series of stacks - sequences of layers with increasing dilations (holes between convolved values in the input sequence): The loss function takes both into account the accuracy of the predicted sample, and its predicted classification based on the input local condition. While the original WaveNet paper does not go into detail about the specific loss function, the implementation I developed off of used a discretized mixture loss.

Deep Convolutional Text-to-Speech was designed as a convolutional speedup to RNN text-to-speech systems. It first learns encodings of the input features and of the audio mel-spectrogram using convolutional networks. Like a sequence-to-sequence RNN, it then learns an attention module and a decoding function to produce a mel-spectrogram and a vocoder function transforming mel-spectrograms into audio (the SSRN module) given those encodings. The loss function sums a least-absolute-deviation and a logistic loss between the predicted and the ground truth waveforms:

5 Experiments/Results/Discussion

Because I had limited time and resources to experiment with these extremely expensive algorithms, I focused most of the time on discovering the most efficient configurations of basic properties of the networks.

WaveNet was the more restrictive network. While the original paper did not define the hyperparameters used, the open source implementation I built off of used hyperparameters that caused the network to immediately run out of memory on a 16GB GPU (24 layers/4 stacks, 512 residual channels), and one significantly smaller (24 layers/4 stacks, 32 residual channels) could only handle a batch size of 2 before running out of memory. The smallest network that I tested that had acceptable output quality had 15 layers/3 stacks, and used a batch size of 6. Because I had such small minibatch sizes, I used an Adam optimizer and a constant $\beta 1$ of 0.95.

DC-TTS was easier to train. I was able to test more parameters, including network size and different configurations of the preprocessing STFT. Interestingly enough, even though the train and test losses decreased quite quickly and stayed relatively stable, the outputted audio had a huge difference in subjective quality between, for example, 8000 epochs and 16000 epochs. This suggests that the cost function can be modified to give greater weight to similarity to the ground truth.

The attention module learned by DC-TTS was also interesting. I tested some models passing in the lyrics as one non-time-series string, and some converting the lyrics to a time series array. The models passing in the string had a linear attention plot, as expected, but the time series had a slightly unexpected pattern:

This pattern appeared early in training and only grew stronger over the epochs, so it is unlikely this is simply because the model hadn't converged. This may be attributed to a linguistic phenomenon where the acoustics of phonemes are heavily affected by their contexts.

6 Future Work

In the future, for this particular problem of generating from pitch and lyric alone, I would like to experiment on different cost functions, and perhaps try another architecture which may be less expensive. For the general application of singing synthesis, I would like to find and transfer train on data with varying pitch and rhythm, so as to create a fully functional (if extremely basic) synthesizer. From this, extensions such as conditioning on features of singing style or voice timbre as well as pitch, lyric, and duration could also be worth exploring.

7 Contributions

I did this project alone; source code is available at https://github.com/kavezo/230_project_final.

References

This section should include citations for: (1) Any papers mentioned in the related work section. (2) Papers describing algorithms that you used which were not covered in class. (3) Code or libraries you downloaded and used. This includes libraries such as scikit-learn, Tensorflow, Pytorch, Keras etc. Acceptable formats include: MLA, APA, IEEE. If you do not use one of these formats, each reference entry must include the following (preferably in this order): author(s), title, conference/journal, publisher, year. If you are using TeX, you can use any bibliography format which includes the items mentioned above. We are excluding the references section from the page limit to encourage students to perform a thorough literature review/related work section without being space-penalized if they include more references. Any choice of citation style is acceptable as long as you are consistent.

[1] Kenmochi, Hideki & Ohshita, Hayato, Vocaloid-commercial singing synthesizer based on sample concatenation, Proc. of INTERSPEECH, 2007.

[2] Cook, Perry R., Singing Voice Synthesis: History, Current Work, and Future Directions, Computer Music Journal, 20(3):38-46, 1996.

[3] Nakamura, Kazuhiro; Oura, Keiichiro; Nankaku, Yoshihiko; & Tokuda, Keiichi, HMM-Based singing voice synthesis and its application to Japanese and English, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 265-269, Florence, 2014.

[4] Blaauw, Merlijn & Bonada, Jordi, A Neural Parametric Singing Synthesizer, Proc. of INTERSPEECH, 2017.

[5] van den Oord, Aaron; Dieleman, Sander; Zen, Heiga; Simonyan, Karen; Vinyals, Oriol; Graves, Alex; Kalchbrenner, Nal; Senior, Andrew; & Kavukcuoglu, Koray, WaveNet: A Generative Model for Raw Audio, arXiv:1609.03499 [cs.SD], 2016.

[6] Tachibana, Hideyuki; Uenoyama, Katsuya; & Aihara, Shinsuke, Efficiently Trainable Text-to-Speech System Based on Deep Convolutional Networks with Guided Attention, arXiV:1710.08969 [cs.SD], 2017.

[7] r9y9, wavenet-vocoder (source code), 2017. https://github.com/r9y9/wavenet_vocoder

[8] Kyubyong, dc_tts (source code), 2017. https://github.com/Kyubyong/dc_tts