5-2021

Synthesizer Parameter Approximation by Deep Learning

Daniel Faronbi
dfaronbi@unomaha.edu

Alisa Gilmore
University of Nebraska-Lincoln

Synthesizer Parameter Approximation by Deep Learning

University Honors Thesis/Capstone

University of Nebraska Omaha

Daniel Faronbi

May 3, 2021

Faculty Mentor: Alisa Gilmore

Daniel Faronbi Honors Thesis Proposal

Synthesizer Parameter Approximation by Deep Learning

Keywords: Timbral Feature Extraction, Linear Arithmetic Synthesis, Machine Learning

Abstract

Synthesizers have been an essential tool for composers of any style of music including computer generated sound. They allow for an expansion in timbral variety to the orchestration of a piece of music or sound scape. Sound designers are trained to be able to recreate a timbre in their head using a synthesizer. This works well for simple sounds but becomes more difficult as the number of parameters required to produce a specific timbre increase. The goal of this research project is to formulate a method for synthesizers to approximate a timbre given an input audio sample using deep learning. The synthesizer should be able to modify its settings (oscillators, filters, LFOs, effects, etc.) to produce an audio signal as close to the input sample as possible. A cost function will measure the difference between the outputted audio signal from the learned synthesizer parameters and the original audio signal that is being mimicked.

Introduction

Over the years, a variety of machine learning approaches have been used to approximate specific timbres. For the most part, this approach has been used to model acoustic instruments [1-2]. However, this project used these techniques to recreate sounds that were original generated by a computer. The lack of robust research in this area means that there was room to explore multiple aspects of parameterized deep learning for sound synthesizers. InverSynth, a recent project completed in 2019, attempted this same idea [3]. However, only subtractive and frequency modulation synthesis techniques were used. For this project, the goal was to expand the current literature by exploring synthesizer parameter learning with additive, wavetable, and linear arithmetic synthesis. InverSynth used an open source java libray, JSyn [4] to programmatically produce audio. Unfortunately, that software will not work for this project because of the added synthesis methods. As such, a custom synthesizer was created using the JUCE framework [5]. For this semester, the goal was to replicate the InverSynth project, focusing particularly on using parameter learning with subtractive synthesis to learn from an audio signal.

Methods

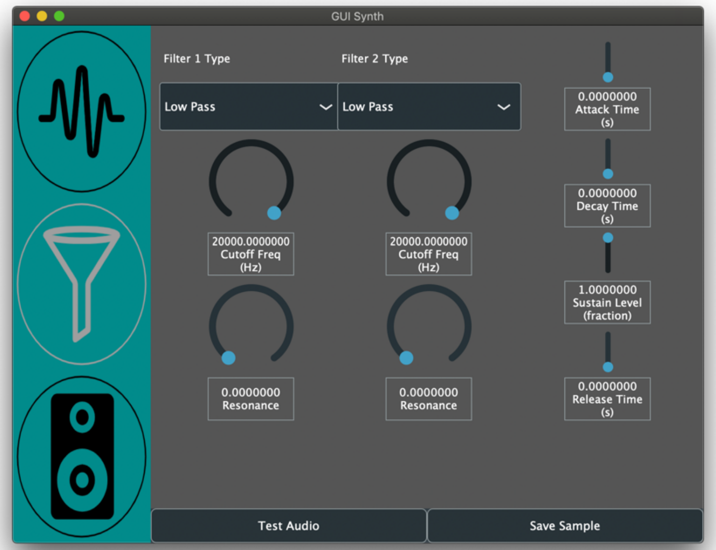
The duration of the project was 5 months. The project was split into three different phases: synthesizer development, data set generation, and deep learning module creation.

Synthesizer Development

JUCE is a very power tool when trying to develop audio tools. It allows for ease of use when juggling between a variety of platforms with strong libraries for digital signal processing and GUI development. It is no surprise that it is the leading framework for multi-platform audio development. I decided to develop two different synthesizer apps, one that used a GUI to tweak parameters and generate sound, and a second that generated a sound from command line parameters. The GUI app was useful when trying to test if the synthesizer was working correctly, the command line app was useful for generating large amounts of audio data very quickly

The first focus for the development stage was understanding both the GUI and audio threads that were foundational to JUCE programming flow. JUCE provided many components that made building a graphical interface fairly simple. Buttons, sliders, and different views were included. The main challenge was to understand how to connect all the different components together. The result can be seen in Figure 1.

The audio thread of the application was separate from the graphics. This was the part of the application that was shared between the command line and GUI application.



Filter 1 Type: Low Pass
Filter 2 Type: Low Pass
20000.0000000 Cutoff Freq (Hz)
20000.0000000 Cutoff Freq (Hz)
0.0000000 Resonance
0.0000000 Resonance
0.0000000 Attack Time (s)
0.0000000 Decay Time (s)
1.0000000 Sustain Level (fraction)
0.0000000 Release Time (s)

Figure 1 - GUI Application

For the computer to generate audio using subtractive synthesis, it must follow a few prescribed steps. First, an oscillator must generate a periodic waveform. In this case, there are three different oscillators working at once. Many times, in subtractive synthesis, a frequency rich signal is preferred as frequencies can be subtracted later on using filters. For a signal to be frequency rich, it means that Fourier transform [6] of the signal produces a lot of frequency content. The oscillators were capable of producing five different types of waveforms: sinusoidal, sawtooth, square, triangle, and noise. The equations used to generate each of these wave forms can be seen below for the audio signal dependent on time t with a fundamental frequency of f.

$$Sine\ Wave = \sin(2\pi f t)$$

$$Sawtooth\ Wave = \frac{-2}{\pi}\arctan(\tan(\mathrm{ft}))$$

$$Square\ Wave = \operatorname{sgn}(\sin(2\pi f t))$$

$$Triangle\ Wave = \frac{2}{\pi}\arccos(\cos(2\pi\mathrm{ft}))$$

$$White\ Noise = \mathrm{rand}(\mathrm{t})$$

Once the oscillators generate a sound, they are passed through two different filters. Each filter takes away a specific set a frequency depending on the type. A lowpass filter cuts high frequency. A high pass filter cuts low frequency. A band bass filter only lets a small band of frequencies around a certain frequency pass. Finally, a notch filter lets everything but a specific frequency pass.

After the signal is filtered, it is passed to an amplifier. As one would expect, the amplifier can make the sound louder or softer. Attached to the amplifier is something called an ADSR envelope. This is responsible for shaping the amplifier gain over time and it gives the synthesizer a more expressive element to its sound. The A stands for attack. This is the amount of time before the sound will reach its full volume, starting from silence. The D is decay, this is the amount of time before the sound decays to the sustain level after its peak amplitude was reached. The S is the sustain level which is held as long as the note is sustained. The R stands for release and is the amount of time that it takes the sound to get from the sustain level to silence.

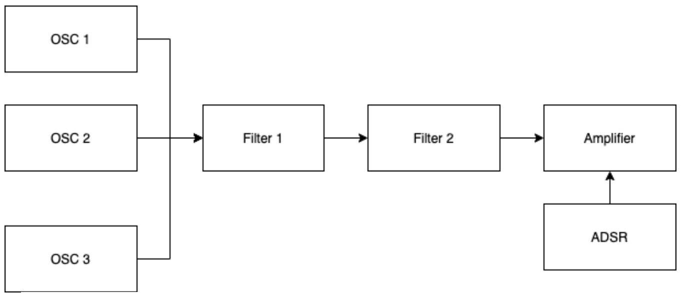


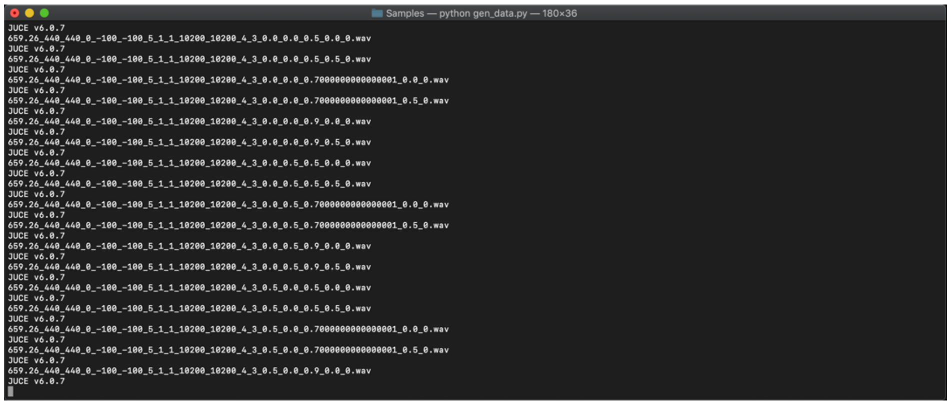
Figure 2 - Subtractive Synthesis Method

All of these components must come together to make a subtractive synthesizer. Figure 2 provides a block diagram of the subtractive synthesis process.

The command line app is very similar to the GUI app. However, sound could not be previewed. All of the synthesizer parameters are passed into the terminal application and a five second audio file is produced in the file directory of your choice.

<u>Data Set Generation</u>

To properly train a neural network, large amount of labelled data is needed. This is called supervised learning because the computer gains knowledge by recognizing patterns and comparing it to correct answers that are supervised by a human. To generate enough data, the command line synthesizer was used. The labels were the synthesizer parameters. These will be the correct answers that we want the deep learning network to



```
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.0_0.5_0.0_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.0_0.5_0.5_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.0_0.7000000000000001_0.0_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.0_0.7000000000000001_0.5_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.0_0.9_0.0_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.0_0.9_0.5_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.5_0.5_0.0_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.5_0.5_0.5_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.5_0.7000000000000001_0.0_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.5_0.7000000000000001_0.5_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.5_0.9_0.0_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.0_0.5_0.9_0.5_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.5_0.0_0.5_0.0_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.5_0.0_0.5_0.5_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.5_0.0_0.7000000000000001_0.0_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.5_0.0_0.7000000000000001_0.5_0.wav
JUCE v6.0.7
659.26_440_440_0_-100_-100_5_1_1_10200_10200_4_3_0.5_0.0_0.9_0.0_0.wav
JUCE v6.0.7
```

Figure 3 - Dataset Generation

understand. A python script was developed that made the synthesizer produce thousands of audio samples. Unfortunately, this method caused my computer to run out of space. The script was modified to generate an audio file, extract the features that will be used in the deep learning network, then delete the audio files. All the data, including labels and features, would be stored in a simple json text file. An example of the data generation process can be seen in Figure 3.

Deciding which features were best to extract took a little bit of time. It was very important that the audio features were able to represent the timbral qualities of the audio sample. In the end, we decided to use Mel-frequency cepstral coefficients, chroma, spectral centroid, and spectral contrast as features.

Mel-frequency cepstral coefficients were used because of their focus on timbral content. These features have often been used in machine learning algorithms to classify song genres or instrument type. These features rely on frequency content that has been log tapered to fit the Mel-frequency scale which is more aligned with how humans hear sound than something like the Fourier series.

Chroma was used because of its strong relation to the western diatonic pitch classes. This made it ideal for trying to have the oscillator learn what frequencies are best to generate. The features are designed to easily categorize sounds that fit into the twelve-tone equal-tempered scale.

Finally, spectral centroid and spectral contrast were used to add more frequency information. In theory, this would help classify timbral content, which was the main purpose of this project.

Deep Learning Module Creation

Tensor flow was used to develop the deep learning module. Keras allows for quick and easy neural networks to be built. First, the json data set generate from the console synthesizer was loaded. Sklearn's training set split algorithim was used to create a training data set and a testing data set.

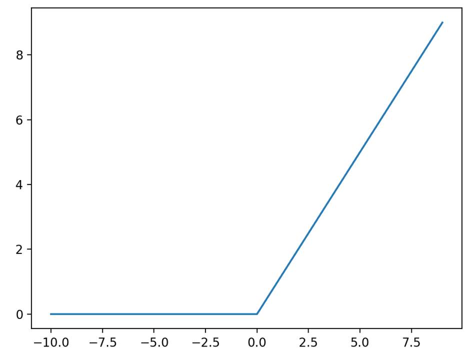


Figure 4 - Rectified Linear Unit

When constructing the deep learning module, it was decided that 2 hidden layers would be serviceable. The first hidden layer contained 1024 neurons and the second hidden layer contained 512 neurons. Rather than use the stand sigmoid function the Rectified linear unit (ReLU) was used instead as seen in Figure 4. This activation function is often preferred in networks with many layers because the sigmoid and hyperbolic tangent functions usually have a vanishing gradient problem. The module had 30 input features and 14 outputs that were compared to labels.

Once the model was built, it was run for 20 Epochs with a learning rate of 0.01.

**Code used in this project is available at https://github.com/dafaronbi/Honors-Thesis*

Results

Unfortunately, the network was not able to classify the signal very accurately. At best, the accuracy was around 40.67%, at worst it was around 25.56%. This is somewhat disappointing. However, there are many improvements that could be made to the model.

The data set could be improved. Generating a wide variety of data took my computer a few days. Because there were so many different permutations of synthesizer parameters, I would never have finished the project in time if I wanted to have the most robust data. To save on time, I lowered the variability of the data. This meant that only one oscillator changed frequency. Also, the amplitudes of all the oscillators never changed. Perhaps if the data set contained more nuance, the trained model would have performed better.

There is also much to be improved with the deep learning model itself. There could be many more layers added and a variety of activation function to test. We could also try implementing dropout during the learning process to make sure that the network is not relying too heavily on specific neurons.

Conclusion

This research project focused on using cutting edge machine learning approaches to music information retrieval. Based on a review of current literature, more research in this area is needed. The knowledge generated from this project would be very beneficial to others involved in music information retrieval and can be applied to other digital signal processing projects. The novel technology developed from this product makes it ideal for a product. This adds economic value to society as well as enjoyment to consumers. The new technology also allows for other products to improve by adding options to developers of similar software. I am proud of my work and I am glad to share it.

[1] Riionheimo, J., Välimäki, V. Parameter Estimation of a Plucked String Synthesis Model Using a Genetic Algorithm with Perceptual Fitness Calculation. EURASIP J. Adv. Signal Process. 2003, 758284 (2003). https://doi.org/10.1155/S1110865703302100
[2] Itoyama, Katsutoshi, and Hiroshi G. Okuno. "Parameter estimation of virtual musical instrument synthesizers." ICMC. 2014.
[3] Barkan, Oren, et al. "InverSynth: Deep Estimation of Synthesizer Parameter Configurations From Audio Signals." IEEE/ACM Transactions on Audio, Speech, and Language Processing 27.12 (2019): 2385-2396.
[4] http://www.softsynth.com/jsyn
[5] https://juce.com/
[6] https://www.thefouriertransform.com/