Synthesizer Parameter-Matching with Deep Learning *

Sam Lowe, Frank Ockerman, Ishan Shah, Wilfred Wong
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina

May 7, 2019

Abstract: Synthesizers can be used to generate a diverse set of sounds, yet the inverse problem of deducing parameter settings on a synthesizer that best approximates a given sound remains an open problem with large potential ramifications for their commercial use. Our goal was to construct and train a model that could program synthesizer parameters to best match given input audio. We implemented a variety of algorithms to address the parameter-matching problem, including MLP models, two LSTMs, one of which was extended with an autoencoder, and a CNN. These models were trained on a set of generated synthesizer sounds and then tested on both generated sounds and their ability to generalize to real-world audio. We also contributed to the general understanding of the problem's scope with additional analysis. While our results and analysis point to the difficulty of full parameter matching, they provide promising avenues for future development, particularly with our CNN implementation.

1 Introduction

Since the invention and proliferation of synthesis technology in the latter half of the 20th century, musicians have increasingly turned to synthesizers to achieve a palette of musical sounds that would otherwise be inaccessible, but the skills required to program synthesizers presents a barrier to entry into the technology. A typical synthesizer has a large number of controlling parameters with complex interactions affecting the overall sound, making the identification of the settings needed to produce a desired sound difficult.

Applications of machine learning could enable one to identify the settings on a given synthesizer that would best emulate the sound, thus ameliorating this barrier. This would be an example of a tone matching programmer.[1] In such a system, a user would feed an input sound into a network trained for their given synthesizer and receive the best matching parameters for use with the synthesizer, which could then be automatically wrapped with a digital synthesizer program to enable simple automatic programming of a synthesizer and greatly augment the musical range of performers.

2 Previous Work

Many previous studies in automated synthesizer programming have implemented genetic algorithms to match sounds by generating successive patches that eventually converge on a close analog of the given sound.[2,3] Other studies have demonstrated that k-means clustering on Mel Frequency Cepstral Coefficients (MFCCs) can be used to generate a decision tree which matches the given sound to the closest synthesizer patch.[4]

Yee-King et al. explored applications of deep learning to the problem, utilizing a multilayer perceptron network (MLP) and recurrent neural networks (RNN) in the form of bidirectional long short term memory models with highway connections. They found Bi-directional long short-term memory (LSTM) networks with highway layers to be the best deep learning technique as evaluated using the mean values of deviation from reconstructed MFCC values. Using those same metrics, it was found that genetic algorithms performed worse than their LSTM model. However, even with LSTMs, only approximately 25% of all of their samples were considered to be matched after reconstruction using this metric. Furthermore, the Bi-directional LSTM's performance was not distinctive from that of MLP and LSTM models.[5] Their initial metric of 25% as well as weak differences in performance between different deep learning architectures suggests that those architectures could be improved for further performance gains, and that new methodologies could be applied to the problem.

Given the large dimensionality of raw audio data, properly implemented feature engineering is of great importance. Many of the studies above, as well as ours, utilize MFCCs for their audio representation. MFCCs characterize the spectral properties of sounds, and have been shown to be effective for a variety of audio classification problems. Another popular feature for audio data are mel-scaled spectrograms. Mel-scaled spectrograms represent the perceptual frequency composition of sounds over time and have also been successfully utilized for audio classification, particularly as inputs to convolutional neural networks. [6]

3 Methods

In this paper, we extend previous work that has been done on the use of deep learning methodologies to train networks to best match synthesizer parameters. Initial efforts first focused on implementing a multilayer perceptron network in order to determine if more advanced architectures could yield better results, and shallow and deeper networks were explored in order to optimize the results. We then implemented two LSTM models, one of which was extended with a novel application of autoencoders to reduce the total parameter space and enhance the ability of the network to train on the data. In addition, we

*Correspondence to: wong15w@live.unc.edu

developed a novel application of convolutional networks to process spectrogram representations of our data. To further characterize the problem, we evaluated the effects of using different synthesizer for generating the data given that each synthesizer should have a unique function composition which generates the output sound.

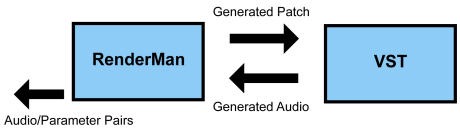
3.1 Synthesizers

The current industry standard for software-based synthesizers is Virtual Studio Technology (VST) format. VST synthesizers are an ideal avenue for the study of automatic synthesizer programming given the relative ease with which they can be interfaced and extended for such applications. In this work, we tested our models on 6 different VST instruments of increasing complexity: ComboF, Obxd, TAL-NoiseMaker, Tunefish4, Dexed, and Synth1. ComboF is a software emulation of an organ and thus not a true synthesizer, but was valuable for providing the opportunity to apply our models to a smaller parameter space. The 5 synthesizer VSTs represent a wide variety of synthesis techniques including FM, additive, and subtractive, which enabled us to study the applicability of our models across techniques. Each synthesizer is controlled by a set of parameters continuously valued between 0 and 1, with each instrument varying in the total number of parameters. The size of the parameter space for each instrument is given in table below.

Synthesizer	Parameters
ComboF	42
Obxd	80
TAL-NoiseMaker	91
Tunefish4	112
Dexed	155
Synth1	255

3.2 Data

We used the RenderMan Python library as our interface for the VST instruments.[7] This library enabled us to generate and set parameter values, as well as generate audio. For each instrument, we generated 1000 samples that consisted of a full set of parameter values drawn from a uniform distribution and 66560 audio frames representing 1.5 seconds of audio sampled at 44100 samples per second. The audio was all generated with a note value of middle C and a constant note velocity.



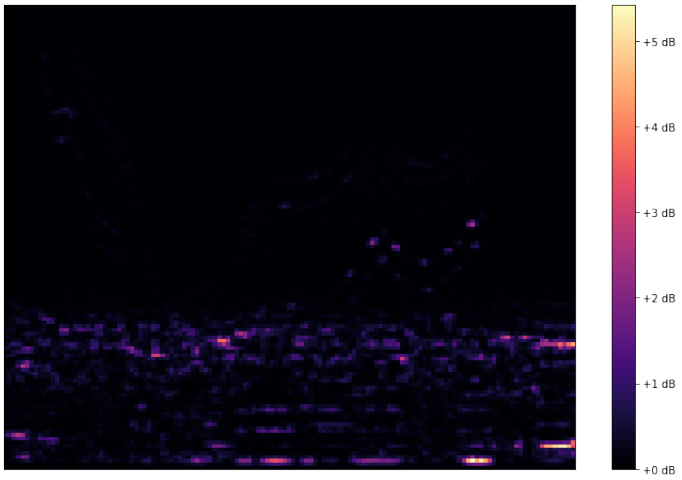
The function of the RenderMan interface.

For each model, the datasets were split into training and test sets of size 600 and 400, respectively.

3.3 Preprocessing

For all data sets, we used LibROSA following sound generation to extract features for training our models.[8] In the case of the MLP and LSTM models, 48 MFCCs were calculated per frame of size 2048 samples and a hop length of 512.

In the case of the CNN, a mel-scaled spectrogram was obtained for each sound sample with 128 mel frequency bins, a frame size of 2048 samples, and a hop length of 1024.



Example spectrogram from our data.

3.4 Model-Multilayer Perceptron

The first variation of a multilayer perceptron that was attempted was a shallow network. Shallow networks should theoretically be able to approximate any function, and would have been thought to over fit as compared to deeper networks. These networks were written in PyTorch. The shallow network consisted of two hidden layers, each with 1000 neurons each using a ReLU activation, and a linear output layer with the number of neurons needed to match the parameter space. The deep network consisted of 20 neurons with ReLU activations for all layers except the third and tenth layer which used a tanh activation, with a similar output layer as the previous one. The learning rate was set to 0.01 and used SGD to train over 10000 and 50000 iterations for the shallow and deep networks, respectively.

3.5 Model-CNN

A convolutional network based on VGG-net was used to analyze the spectrogram. A spectrogram was treated as an image and fed into the CNN that was built in Keras. The layers are as follows: Conv2D-64 10 x 4 filters ReLU, Conv2D 64 4 x 4 filters sigmoid, MaxPool2D 2 x 2, Dropout 25 %, Conv2D 64 3 x 4 filters ReLU, Conv2D 64 3 x 4 filters ReLU, MaxPool2D 2 x 2, Dropout 25 %, Flatten, Dense 255 ReLU, Dense 255 tanh. We optimized using RMSProp and MSE and trained for 100 epochs with a batch size of 100.

3.6 Model-LSTM

The long short term memory model was written in Keras using 24 units with ReLU activations, and with a Nadam optimizer using MSE

as its loss function. The data was arranged so that each time point had 3 features as an input sequence for the model. Afterwards, a dense layer with an output size that matched the target size was used to obtain the final values for the results. The model was trained with 40 epochs and a batch size of 10. Initially, Bi-directional, GRU, and stacked versions of LSTM were attempted, but they appeared to provide worse performance in general.

3.7 Autoencoder

With only 1000 samples per synthesizer, we wanted to explore dimension reduction in the synthesizer parameter space, especially for synthesizers with over 200 parameters. Through this, we could predict fewer values at each model run, increasing model efficiency with the potential of increasing model accuracy. Previous research has used Principal Components Analysis to alleviate the problem, but we sought to improve on this by implementing autoencoders with relatively simple encoder and decoder networks. We ran PCA on each of the 6 synthesizers parameter spaces to gain an initial idea on the effectiveness of dimension reduction. We found that the using half of the possible principal components explained anywhere between 64% to 81% of the datas variance (Column 2, below), depending on the synthesizer. This shows dimension reduction is at least slightly effective, and was enough justification to attempt running a simple autoencoder, with the encoded data having half the parameters of the full data. There is one hidden RelU layer in the encoding network, which contains 3/4 of the parameters. The decoder network is essentially the inverse of the encoder network, with the input being the parameter data and the output being the data with all parameters, with a hidden ReLu layer in between.

Synthesizer	Half-Component Variance Explained	Autoencoder Re-Generation RMSE
ComboF	0.810	0.225
Obxd	0.645	0.268
NoiseMaker	0.638	0.270
Tunefish4	0.659	0.269
Dexed	0.702	0.266
Synth1	0.712	0.278

Also, after a 0.6-0.4 train-test split, we ran the full encoder and decoder network on the training set, predicting the test set with an RMSE between 0.225 and 0.278 (Column 3, above), indicating that the data can be effectively regenerated.

3.7.1 LSTM Extension

To test whether reducing dimensions of the synthesizers parameter space improves prediction performance, we ran the encoder network on the parameters for all 6 synthesizers. We then used the aforementioned LSTM, replacing the training and test set labels with the encoded parameters. After obtaining the predictions from the LSTM, we ran them through the decoder network, obtaining the full parameter space. Then, we compared it with the original test data to get a RMSE value for prediction performance, which we compare to other models.

4 Results

The final RMSE achieved by each model on the test set for each synthesizer is given in the table at the end of the paper.

5 Application

Given that our study sought to build a system that would allow for the input of any audio to be turned into a set of synthesizer parameters, we wanted to test the ability of our models to generalize to real-world audio. For this test, we utilized Magenta's NSynth dataset, which consists of a set of single pitch audio samples for a variety of instruments.[9] We generated a subset of this dataset consisting of the examples played at middle C to match our training data, which worked out to 904 samples. Given the promising results presented by our CNN during training and testing on contrived sounds, we used the 6 CNN's trained on the various VSTs to generate parameters given the spectrogram representation of the NSynth sounds. Since perceptual evaluation is most important when working with synthesizer programming, we calculated the average Euclidean distance between 6 MFCC features per each time frame. The results are given in the table below.

Synthesizer	Average Distance
ComboF	22.35
Obxd	20.97
TAL-NoiseMaker	64.71
Tunefish4	30.45
Dexed	65.30
Synth1	20.92

Besides the outliers of TAL-NoiseMaker and Dexed, which can likely be explained by their complex synthesis engines, the algorithm seems to often produce sounds that are perceptually close. Informal listening analysis demonstrates that many of the characteristic timbral qualities of the real world sounds are being approximated by the parameter values.

6 Conclusion and Future Work

Results from the MLPs, LSTM, and LSTM-Autoencoder, indicate that these models are not suited for recovering the parameters for a sound from a synthesizer. Loss functions across all methods appeared to be similar, despite the fact that different synthesizers with a different number of parameters were used.

With respect to the autoencoder, we found that the RMSE for all 6 synthesizers decoder data were comparable or sometimes even greater than the other methods used. This implies reduction of dimensionality in this case can remove too much of the necessary variance captured by the full data, thus negatively impacting prediction performance. In fact, in ComboF, the VST with the fewest dimensions overall, autoencoders have the relative worst performance, suggesting that every dimension carries high importance.

We found in the cases of those three aforementioned methods, the predicted parameter values were typically around 0.5. This would have been expected if the model was averaging all possible parameter

values, as they had been uniformly distributed from 0 to 1. Therefore, these models as they are currently implemented do not appear to be learning the actual parameter values.

However, the CNN on the other hand, despite similar loss values, does appear to be learning features of the data. Analysis of the actual predicted values indicated that the predicted values did not center upon 0.5, indicating that the model was learning values for the model. Furthermore, the spectrogram derived from the predicted values appeared to contain important spectral information, and auditory tests revealed that significant aspects of a sound sample were recovered using this method.

Future work should focus on engineering features which make MLP and LSTM models feasible for this work as well as using different convolutional features and netowkr structures to detect larger scale features present in a spectrogram.

References

[1] Yee-King, M. J. "Automatic sound synthesizer programming: techniques and applications." Thesis, 2011.

[2] A. Horner, J. Beauchamp, and L. Haken, Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis. Comput. Music J., vol. 17, no. 4, pp. 1729, 1993.

[3] T. Mitchell, J. Charles, and W. Sullivan, Frequency modulation tone matching using a fuzzy clustering evolution strategy, in Proc. 118th Conv. Audio Eng. Soc., Barcelona, Spain, 2005, pp. 112.

[4] Caceres, J.-P. Sound Design Learning For Frequency Modulation Synthesis Parameters. http://cs229.stanford.edu/proj2007/caceres-SoundDesignLearningforFrequencyModulationSynthesisParameters.pdf

[5] Yee-King, M. J.; Fedden, L.; Dinverno, M. "Automatic Programming of VST Sound Synthesizers Using Deep Networks and Other Techniques." IEEE Transactions on Emerging Topics in Computational Intelligence 2018, 2(2), 150159.

[6] Sterling A., Wilson J., Lowe S., Lin M.C. (2018) "ISNN: Impact Sound Neural Network for Audio-Visual Object Classification." In: Ferrari V., Hebert M., Sminchisescu C., Weiss Y. (eds) Computer Vision ECCV 2018. ECCV 2018. Lecture Notes in Computer Science, vol 11219. Springer, Cham

[7] Fedden, L. (2018). Renderman

[8] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

[9] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." 2017.

	ComboF	Obxd	TAL-NoiseMaker	Tunefish4	Dexed	Synth1
MLP Shallow	0.538	0.273	0.269	0.437	0.482	0.517
MLP Deep	0.377	0.240	0.240	0.285	0.307	0.278
LSTM	0.296	0.292	0.296	0.297	0.343	0.294
LSTM Autoencoder	0.419	0.332	0.317	0.311	0.328	0.301
CNN	0.242	0.313	0.306	0.307	0.281	0.300