SpiegeLib: An Automatic Synthesizer Programming Library

Jordie Shier, George Tzanetakis, and Kirk McNally

University of Victoria

Introduction

- Automatic synthesizer programming (ASP) is the field of research focused on using algorithmic techniques to generate parameter settings and patch connections for synthesizers.
- Research in ASP began in late the 1970s [1] and since then a body of work has emerged exploring a variety of methods including evolutionary algorithms [2,3] and deep learning techniques [4,5] for automatically programming synthesizers.
- This work presents SpiegeLib, an open source software library designed to support continued development in ASP and to promote reproducible research [6] by providing a platform for sharing implementations.
- The name SpiegeLib pays homage to Laurie Spiegel, an early female pioneer in electronic music. It also stands for Synthesizer Programming with Intelligent Exploration, Generation, and Evaluation Library.
- SpiegeLib is inspired by and builds on work by Yee-King et al. [4]

Design of SpiegeLib

- Written in the Python programming language.
- Object-oriented API with base classes for writing custom implementations.
- Components for constructing research pipelines including classes for integrating with and algorithmically programming VST synthesizers as well as evaluating results using objective and subjective methods.
- Currently included algorithms based on prior work are shown in Table 1. Please refer to PDF handout for more information on these algorithms.
- Packaging and delivery through the Python Packaging Index (PyPI).
- Full library documentation and installation instructions available online[1].

Table 1 - Algorithms currently implemented in SpiegeLib

Audio Features	Deep Learning Estimators	Evolutionary Estimators
FFT	MLP [4]	Basic GA [2]
STFT	LSTM [4]	NSGA III [3]
MFCC	LSTM++ [4]	**Objective Evaluation**
Spectral*	CNN [5]	MFCC Evaluation

*Spectral features include centroid, bandwidth, flatness, roll-off, and contrast

[1] https://spiegelib.github.io/spiegelib/

SpiegeLib Example: Sound Matching Experiment

- In synthesizer sound matching, the goal is to use an algorithm to estimate synthesizer parameters to replicate a target sound as closely as possible.
- In this experiment we attempt to sound match Dexed [7], a VST emulation of the Yamaha DX7 frequency modulation (FM) synthesizer, and compare the different estimator algorithms currently implemented in SpiegeLib.
- For detailed information and to hear results, visit the experiment site via the QR code at the top left corner of this poster, or follow the link blelow[1].

1. Configure VST synthesizer plugin for experiment using SynthVST class

- To reduce the complexity we focus on programming a subset of nine parameters. This turns Dexed into a simple two operator FM synthesizer.

2. Generate datasets and train deep learning models

- Each of the four deep learning estimators is trained over 100 epochs with early stopping if validation loss is stagnant for 10 epochs.

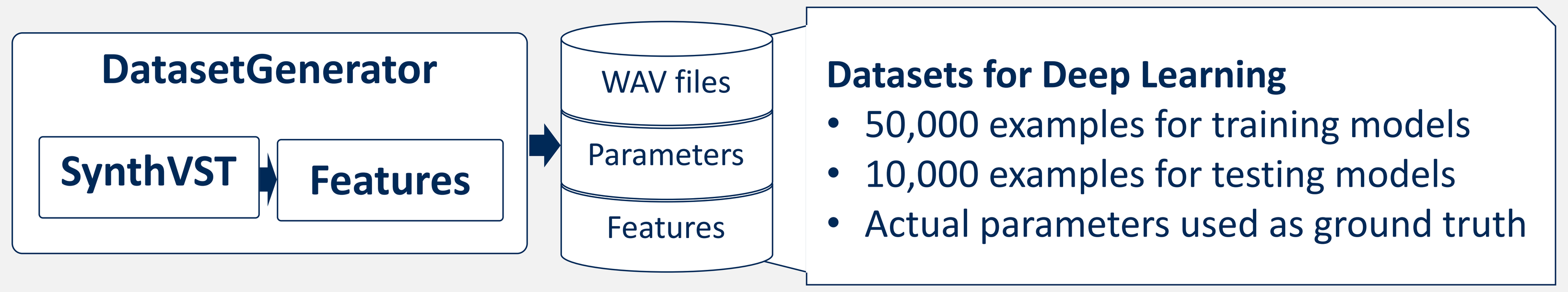


Figure 1 – The DatasetGenerator class generates random patches for a synthesizer and runs audio feature extraction on the resulting audio.

3. Sound matching and evaluation

- A set of 25 target sound files generated from Dexed used for evaluation.
- The mean absolute error (MAE) between MFCCs of the audio target and the results of sound matching is used to evaluate performance.

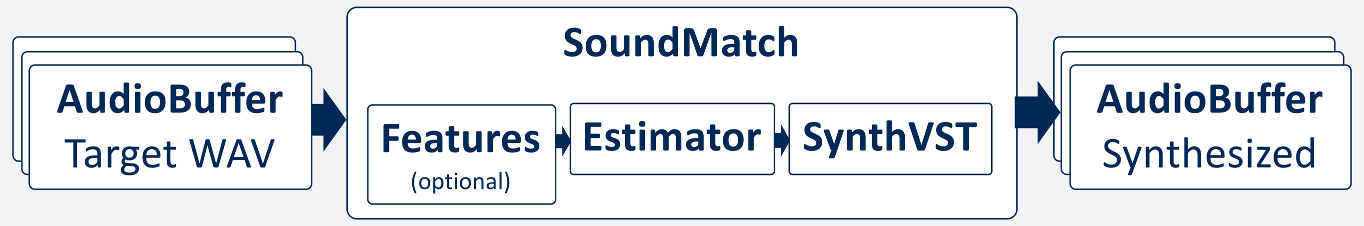


Figure 2 – The SoundMatch class estimates parameters for a given synthesizer using an instance of an Estimator class.

[1] https://spiegelib.github.io/spiegelib/examples/fm_sound_match.html

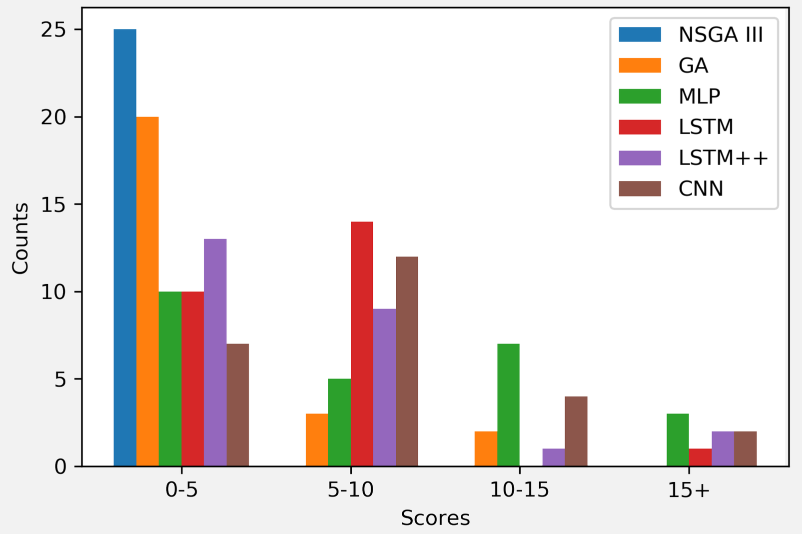
Experimental Results

- Results show that the NSGA III multi-objective genetic algorithm performed the best overall. A summary of all results is shown in Table 2, and a histogram of the results for each estimator is shown in Table 3.
- Results show MAE between MFCCs of the target audio and the synthesized sound match. **Smaller values indicate a closer match.**

Table 2 - Results of MFCC Evaluation

Method	Mean	Stdev	Min	Max
MLP	8.55	6.77	1.92	34.12
LSTM	6.12	3.76	1.20	19.36
LSTM++	4.91	6.50	2.12	21.51
CNN	7.88	4.26	2.68	20.89
Basic GA	2.25	2.58	0.70	11.17
NSGA III	**0.81**	**0.89**	**0.001**	**3.06**

Figure 3 – MFCC results histogram



Future Work

- Planned expansions to the library include adding more deep learning algorithms including different CNN configurations and a generative model.
- Additional programming methods are also planned including interactive approaches and sound matching with vocal imitations.
- Interested researchers and developers are encouraged to contribute.

References

- [1] Justice, James. "Analytic signal processing in music computation." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 27.6 (1979): 670-684.
- [2] Horner, Andrew, James Beauchamp, and Lippold Haken. "Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis." *Computer Music Journal* 17.4 (1993): 17-29.
- [3] Tatar, Kıvanç, Matthieu Macret, and Philippe Pasquier. "Automatic synthesizer preset generation with PresetGen." *Journal of New Music Research* 45.2 (2016): 124-144.
- [4] Yee-King, Matthew John, Leon Fedden, and Mark d'Inverno. "Automatic programming of VST sound synthesizers using deep networks and other techniques." *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.2 (2018): 150-159.
- [5] 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.
- [6] Vandewalle, Patrick, Jelena Kovacevic, and Martin Vetterli. "Reproducible research in signal processing." *IEEE Signal Processing Magazine* 26.3 (2009): 37-47.
- [7] https://asb2m10.github.io/dexed/