WaVAEtable Synthesis

Jeremy Hyrkas*

University of California San Diego
jhyrkas@ucsd.edu

Abstract. Timbral autoencoders, a class of generative model that learn the timbre distribution of audio data, are a current research focus in music technology; however, despite recent improvements, they have rarely been used in music composition or musical systems due to issues of static musical output, general lack of real-time synthesis and the unwieldiness of synthesis parameters. This project proposes a solution to these issues by combining timbral autoencoder models with a classic computer music synthesis technique in wavetable synthesis. A proof-of-concept implementation in Python, with controllers in Max and SuperCollider, demonstrates the timbral autoencoder's capability as a wavetable generator. This concept is generally architecture agnostic, showing that most existing timbral autoencoders could be adapted for use in real-time music creation today, regardless of their capabilities for real-time synthesis and time-varying timbre.

Keywords: Generative models, neural networks, sound synthesis

1 Introduction

A generative model can be broadly defined as a probabilistic method that learns a distribution based on a corpus of training data such that examples similar to the training data can be generated by sampling from the learned distribution [1]. Recently, the term has been largely associated with deep artificial neural networks that generate images, video, speech, or examples from a variety of other domains. Music researchers have utilized neural network generative models as a technology for sound synthesis in music (for example, the groundbreaking NSynth neural synthesizer [2]). One such approach is the *timbral autoencoder* (i.e. [3][4][5]). In this type of model, networks learn audio representations in the frequency domain, resulting in models that synthesize sounds based on a learned latent space of their training data, usually monophonic instruments. These timbral models target the problem of novel sound generation, particularly in a synthesizer setting [3]. Ideally, musicians can find sounds that interpolate the timbre of multiple instruments, or sounds that do not invoke any recognizable instrument at all. Recently, the variational autoencoder (VAE) [6] has been favored (an overview of VAEs for musical audio can be found here [1]). Once trained, a user may provide latent parameters to the VAE to generate new examples.

There are a number of benefits to training these models. The training data is represented in the frequency domain, which behaves better than time-domain representations with common loss functions that do not account for phase shift. Additionally,

* Special thanks to Karl Yerkes (MAT, University of California Santa Barbara) for his great help with SuperCollider and OSC implementations.

some models render audio in near real-time [3] due to a sufficiently small architecture, as opposed to more complex audio models such as the NSynth autoencoder [2]. More recent efforts have shown that the models can learn the timbre of the training data in a way that sufficiently disentangles the pitch of the training examples, even when the training data does not contain pitch labels [4].

While integration of these models in music systems seems imminent, there are some practical drawbacks that remain unaddressed. Learning problems in the frequency domain largely concern magnitude spectra, meaning that phase reconstruction is necessary before the audio can be rendered in the time domain. Models that learn on magnitude Fourier transforms can use phase reconstruction algorithms that run in real-time [3]; however, models that use alternate spectral representations [5] rely on non-real-time algorithms such as the Griffin-Lim method [7]. Finally, recurrent connections are largely absent in timbral autoencoder models, meaning models are limited to generating a cyclic waveform per selection of latent parameters.

1.1 Motivation and Project Overview

This project aims to integrate a neural network synthesis engine implemented in Python with more general synthesis and composition engines in Cycling '74's Max software and SuperCollider (SC). While many of the aforementioned research efforts focus on improved synthesis in the form of fidelity, realism, or expressiveness, this project takes the philosophy that current synthesis methods are already usable in music creation when combined with existing and well understood computer music methods.

Because most timbral autoencoders produce inherently cyclic audio, we can conceptually treat them as oscillators. Many timbral autoencoders are not conditioned on pitch, precluding them from being used as oscillators in a traditional sense. Additionally, models that do not use real-time phase reconstruction cannot be used to synthesize audio in real-time like a traditional oscillator. Therefore, incorporating these models into real-time engines requires a method that utilizes pre-rendered cyclic audio signals.

The most straightforward candidate for such a synthesis system is wavetable synthesis [8], a scheme in which cyclic waveforms are stored and synthesized by reading and interpolating values at a given frequency. This project recasts timbral autoencoders as wavetable generators and provides methods for sampling and saving wavetables from their output. Proof-of-concept software is provided in Max and SC, demonstrating how timbral autoencoders as wavetable generators can be used in performance and composition, and can be sonically extended using methods such as wavetable interpolation and frequency-modulated playback. We refer to the process of incorporating timbral autoencoders, often VAEs, into a wavetable extraction framework and combining them with music synthesis software as WaVAEtable synthesis.

1.2 Related Methods

NSynth [2] is an early musically focused generative model for audio. NSynth's unique architecture allows it to iteratively create time-domain audio, resulting in audio with time-varying timbre. This result is arguably more musically useful than cyclic waveforms, but the model is expensive to train on most computers and slow to render audio.

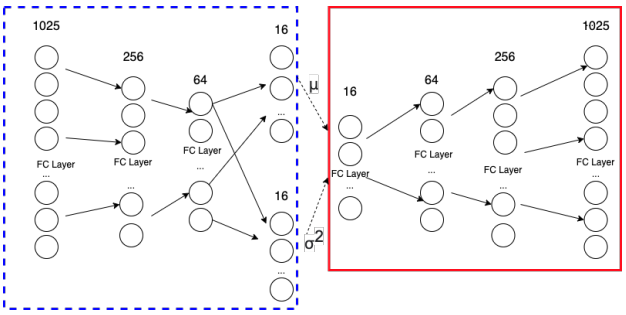

Fig. 1. Architecture of the basic VAE trained for this project. The *dotted blue rectangle* on the left contains the encoder and *solid-lined red rectangle* on the right contains the decoder. The *dotted line* between encoder and decoder represents a sampling operation. The input and output data are magnitude spectra extracted from the STFT of instrumental audio.

Neural Wavetable [9] is perhaps the most similar project to the one presented here. The project uses an autoencoder to learn time-domain wavetables and interpolate between them. Interpolation is performed on the latent encoding of two target wavetables. This is conceptually similar to timbral autoencoders, with the exceptions that Neural Wavetable operates in the time-domain as opposed to the frequency-domain, and that the model is explicitly trained on wavetables. Because the method proposed here extracts wavetables from a broad collection of generative models, it is a more general method than the Neural Wavetable method. Neural Wavetable's underlying model cannot generate audio in real-time, so the associated plug-in uses pre-rendered wavetables for interpolation.

A more thorough survey of generative models for audio can be found here [1].

2 WaVAEtable Synthesis

2.1 Sample neural network architecture and training

This software exploits the architecture of timbral autoencoders, wherein user-provided encodings produce spectral audio that is converted to time-domain audio. To keep the design aimed towards utilizing existing models, this software uses a simple VAE (depicted in Figure 1) implemented in PyTorch [10] that encapsulates the most basic generative capabilities shared by timbral autoencoder models. The data are positive frequency bins from Short-time Fourier Transform (STFT) frames of audio in the NSynth dataset [2]. The architecture model is shown in Figure 1. After training, the 16 latent parameters are used to synthesize the magnitudes of the positive frequencies of an STFT frame. These frames are reflected and time-domain audio is added using the Griffin-Lim algorithm [7]. The decoder-to-audio process is detailed in Figure 2. Adjusting the architecture and hyperparameters of this model could constitute a separate research effort, but are not critical to this project as this method aims to be as architecture- and model-agnostic as possible.

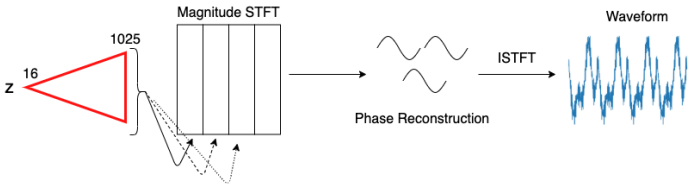

Fig. 2. Decoder to audio process: latent space parameters (z) are provided to the decoder, depicted as a *red triangle*. Output from the fixed z is used to create a *magnitude STFT*. The Griffin-Lim algorithm for *phase reconstruction* then yields in time-domain audio.

2.2 Wavetable generation

Many synthesis engines use a default wavetable size of 512 samples. If a given timbral autoencoder is conditioned on pitch, the model could generate output at a specific fundamental frequency f_0 such that the period of the waveform is 512 samples given the sampling rate f_s by setting $f_0 = \frac{f_s}{512}$. However, this setup is not always feasible. Many timbral autoencoder models, including the model used here, are not conditioned on pitch, resulting in a generative latent space that changes both timbre and fundamental frequency simultaneously. Even those models that disentangle pitch from timbre may be conditioned on discrete pitches (such as MIDI notes), and in general may not be able to generate the desired f_0. For example, the sampling rate of the NSynth data set is 16 kHz, so a waveform with period 512 samples has $f_0 = 31.25$ Hz, well below reasonable pitches in most music data sets.

Therefore, wavetables are created using a heuristic wavetable extraction algorithm that relies on f_0 estimation and resampling. Given a latent encoding from a user, the decoder is invoked to create a periodic waveform (see Figure 2). We use the pYin [11] algorithm to estimate the fundamental frequency of each frame. The pYin algorithm is probabilistic and determines the likelihood of each frame containing a pitched sound. If a sufficient number of frames are found likely to be pitched, we predict the f_0 of the waveform to be the mean of the f_0 of the voiced frames; if this is not the case, it is likely that the provided encoding is very dissimilar from examples learned in the training data and the resulting sound may be noisy and therefore unpitched.

Given the f_s of a model and the predicted f_0 of the model's output based on the user's inputs, we resample the output to a new sampling frequency $round(f_0 * 512)$, which results in a waveform whose period is very close to 512 samples. Finally, samples are extracted from the new waveform starting from some position that is very near 0 to avoid an impulse at the beginning of the wavetable. Overall, the extraction method is subject to failures in f_0 estimation (usually octave errors) and resampling artifacts, but is architecture agnostic and can be adapted to any timbral autoencoder (or any generative model whose output is sufficiently periodic).

2.3 Synthesizer implementations

Two prototype synthesizers were created as a proof-of-concept to demonstrate multiple musical uses for VAE wavetables. First, a simple polyphonic synth patch was created

in Cycling '74's Max software. This patch assigns incoming MIDI notes to one of five voices. Here, random wavetables are pre-rendered using a Python script and can be regenerated by the user. Communication is done via the file system with wavetables saved and loaded from .WAV files. This patch also combines wavetable synthesis with FM synthesis, with MIDI CC controls controlling the wavetable assignment and FM controls of the wavetable playback speed. This patch, while simple, demonstrates the viability of using timbral autoencoder output in real-time performance.

A more complex system was constructed in SC with the goals of user interactivity with the underlying model, more complex synthesis methods and algorithmic composition. Users control the latent parameters of the VAE described in Section 2.1 using a custom GUI created in SC. The user can listen to a wavetable for a given setting, and if it is interesting for their compositional purposes, save it. All communication between Python and SC is performed locally using OSC, so no file system interaction is required. Figure 3 shows the interface for manipulating and storing wavetables.

Once stored, wavetables are played back using wavetable synthesis and wavetable interpolation. Users can also incorporate other SC generators to create complex synthesizer definitions (SynthDefs) with the generated wavetables at their core. We provide an example SynthDef that can be controlled by a MIDI controller, or used in algorithmic composition. A small etude is included in the provided software to demonstrate this capability. All Max, SC, and Python code, as well as the accompanying VAE model, are available at `https://github.com/jhyrkas/wavaetable`.

2.4 Incorporation of existing timbral autoencoders

The neural network used in this project is not intended to be a standalone model, but acts as a basic stand-in for existing timbral autoencoder models (i.e. [3][4][5]), most of which have more complex architectures and are capable of more pleasing musical audio. To test the viability of the WaVAEtable synthesis approach, the Python script to interface with SC was adapted and added to a fork of the CANNe [3] synthesizer GitHub, available at `https://github.com/jhyrkas/canne_synth`. The only major changes to the script involved reinterpreting the latent space parameters sent from SC, as CANNe's latent space only contains 8 variables and expects a different range of values. With just these minor adjustments, the CANNe model can now be used as a wavetable generator in WaVAEtable synthesis. We posit that other timbral autoencoders can also be easily adapted, so long as they offer an encoding-to-audio synthesis method.

3 Future Work and Conclusion

This work offers a path towards incorporating an existing body of generative models into music systems. The proposed method allows for integrating models regardless of underlying architecture and real-time viability, and allows for a greater reuse of interesting latent parameters, which can be cumbersome to discover. Synth design and model improvement can thus be treated as complementary and orthogonal research avenues.

WaVAEtable synthesis may approach the practical limits of incorporating static generative models for audio in more traditional electronic music synthesis. Future timbral

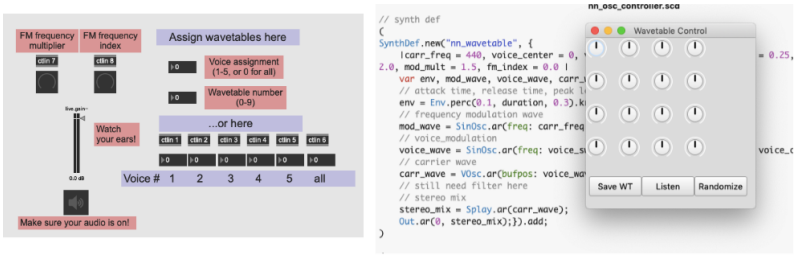

Fig. 3. Left: Max interface to playback wavetables, controlled via MIDI controller. Right: SuperCollider interface to control decoder parameters, listen to and store wavetables for playback.

models that generate audio in real-time and are conditioned on pitch could function as a true oscillator in a synthesis system. Moving beyond these static timbral models to time-varying models allows for new combinations of generative models and synthesis methods, such as neural sample-generation and neural granular synthesis.

References

1. Huzaifah, M., Wyse, L.: Deep Generative Models for Musical Audio Synthesis. In: Handbook of Artificial Intelligence for Music: Foundations, Advanced Approaches, and Developments for Creativity. Springer (2020)
2. Engel, J., Resnick, C., Roberts, A., Dieleman, S., Norouzi, M., Eck, D., Simonyan, K.: Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70 (2017)
3. Colonel, J., Curro, C., Keene, S.: Autoencoding Neural Networks as Musical Audio Synthesizers. In: Proceedings of the 21st International Conference on Digital Audio Effects (2018)
4. Luo, Y. J., Cheuk, K. W., Nakano, T., Goto, M., Herremans, D.: Unsupervised Disentanglement of Pitch and Timbre for Isolated Musical Instrument Sounds. In: Proceedings of the 2020 International Society of Music Information Retrieval Conference (2020)
5. Esling, P., Chemla–Romeu-Santos, A., Bitton, A.: Generative timbre spaces: regularizing variational auto-encoders with perceptual metrics. In: Proceedings of the 21st International Conference on Digital Audio Effects (2018)
6. Kingma, D.P., Welling, M.: Auto-Encoding Variational Bayes. In: arXiv preprint, arXiv:1312.6114 (2013)
7. Griffin, D., Lim, J.: Signal Estimation from Modified Short-Time Fourier Transform. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 32, num. 2, 236–243 (1984)
8. Bristow-Johnson, R.: Wavetable Synthesis 101, A Fundamental Perspective. In: Proceedings 101st Convention of the Audio Engineering Society (1996)
9. Hantrakul, L., Yang, L. C.: Neural Wavetable: A Playable Wavetable Synthesizer Using Neural Networks. In: Workshop on Machine Learning for Creativity and Design (2018)
10. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: Proceedings of the 31st Conference on Neural Information Processing Systems (2017)
11. Mauch, M. Dixon, S.: pYIN: A Fundamental Frequency Estimator Using Probabilistic Threshold Distributions. In: Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (2014)