Studio report: sound synthesis with DDSP and network bending techniques

Matthew Yee-King[1] and Louis McCallum[2]

[1] Goldsmiths, University of London
[2] Creative Computing Institute, University of the Arts
m.yee-king@gold.ac.uk

Abstract. This paper reports on our experiences synthesizing sounds and building network bending functionality onto the Differentiable Digital Signal Processing (DDSP) system. DDSP is an extension to the TensorFlow API with which we can embed trainable signal processing nodes in neural networks. Comparing DDSP sound synthesis networks to preset finding networks and sample level synthesis networks, we argue that it offers a third mode of working, providing continuous control in real-time of high fidelity synthesizers using low numbers of control parameters. We describe two phases of our experimentation. Firstly we worked with a composer to explore different training datasets and parameters. Secondly, we extended DDSP models with network bending functionality, which allows us to feed additional control data into the network's hidden layers and achieve new timbral effects. We describe several possible network bending techniques and how they affect the sound.

Keywords: Sound synthesis, neural networks, network bending

1 Introduction

Google Magenta released DDSP in early 2020 (Engel, Hantrakul, Gu, & Roberts, 2020)[3]. It is an extension to the TensorFlow machine learning library, allowing network designers to embed signal processing nodes in a neural network architecture. With DDSP, we can create neural networks with embedded, trainable sound synthesis and sound processing capabilities. In this paper, we will explain why we think DDSP networks represent an exciting creative method for working with sound synthesis, and we will report on our experiences working with and extending on DDSP sound synthesis networks in a composition project.

1.1 Previous work

We shall split previous research using machine learning for sound synthesis into two areas: work focusing on synthesis and work concentrate on sampling. A common problem that researchers address in the synthesis work is eliciting control

[3] https://github.com/magenta/ddsp

data or parameter settings for synthesis algorithms. On the other hand, in the sampling work, a typical problem is how to generate a sequence of audio samples that are statistically similar to a large dataset of input samples. We shall now consider these two areas of study, providing examples and noting limitations.

1.2 Machine learning and sound synthesis: preset finders

Preset finders are a category of machine learning-driven sound synthesis systems. They take as an input an audio signal and produce at their output synthesizer parameter settings which cause a given synthesizer to approximate the input sound. When eliciting parameter settings for sound synthesis algorithms, researchers typically view the algorithms as black-box systems external to the machine learning system. In classic early work in this field, (Horner, Beauchamp, & Haken, 1993) used genetic algorithms to optimize FM synthesis parameters to match an input timbre. A more recent example of a preset finder is (Barkan, Tsiris, Katz, & Koenigstein, 2019), wherein the researchers trained a neural network to output a set of parameter settings for a subtractive synthesizer.

One motivation for this black-box approach is the desire to integrate the systems with the mass of commercially available synthesis software. For example, (Yee-King, Fedden, & d'Inverno, 2018) demonstrated the capability of deep networks to automatically program VST plugins[4], and (Tatar et al., 2016) used a genetic algorithm to elicit presets for a commercial hardware synthesizer. Another motivation for preset finding is enabling a more fulsome or dynamic exploration of the space of possible sounds for a given synthesis engine than is possible using standard synthesizer programming techniques or through the use of presets. In other words, to answer the question 'What is the full timbral range of this synthesizer?'. (Tubb & Dixon, 2014) and (Yee-King, 2011) considered exploration of sound synthesis spaces and more recently, (Esling, Masuda, Bardet, Despres, & Chemla-Romeu-Santos, 2020) applied the concept of latent spaces to the exploration of sound synthesis space.

One advantage of preset finders based on neural networks is that the resulting networks are comparatively small and can potentially run in real-time. For example, (Yee-King et al., 2018) were able to infer parameters for a Yamaha DX-7 emulator in milliseconds once they had trained the network, and (Esling et al., 2020)'s preset finder operated in 'almost real-time'. On the other hand, a limitation of systems that learn to program external synthesizers, especially preset finders, is that they are one-shot systems that do not generally provide continuous control. So preset finders may not unlock the dynamic timbral potential of the synthesizer. DDSP models address this problem by mapping low dimensional but continuous features such as pitch and amplitude to high dimensional synthesizer controls, enabling dynamic and stateful timbre control.

[4] https://steinbergmedia.Github.io/vst3_doc/vstsdk/index.html

1.3 Machine learning and sound synthesis: sample generators

The other body of work mentioned above has audio samples as its substrate instead of synthesis algorithms and parameter settings. The aim is to train neural networks to generate raw audio samples, normally with raw audio samples as training input. Wavenet, reported by (van den Oord et al., 2016) is an example of such a system. These systems are capable of generating remarkable results. For example, a collection of example outputs from OpenAI's jukebox system led to a flurry of media attention because they were realistic enough to be considered as 'long lost' tracks by famous artists (Dhariwal et al., 2020). The DadaBots project of Zukowski and Carr also uses the SampleRNN model to generate audio in specific genres, including Black Metal and Math Rock (Zukowski & Carr, 2017).

One major problem with these raw audio synthesis networks is that they tend to be very large. For example, the openAI jukebox system was trained using hundreds of GPUs for weeks, and it has billions of parameters. It takes hours with many GPUs to render a minute of audio, so these systems are not accessible to regular musicians.

In response to this problem of size, researchers have explored other approaches for synthesizing raw audio with smaller networks. For example (Collins, Ruzicka, & Grierson, 2020) generated sequences of spectral frames and (Kiefer, 2019) used conceptor networks to create highly malleable sonic models. The realism of the resynthesis is not comparable to the larger systems, but these systems have much potential for creative exploration. An interesting approach to creative exploration is using network bending techniques to manipulate the network's hidden layers, as described by (Broad, Leymarie, & Grierson, 2020). We build on this work by using network bending techniques to patch into the hidden layers of trained DDSP models.

In summary, we have discussed some machine learning systems that operate on synthesis algorithms and others that operate upon raw audio samples. The former offer limited dynamic synthesizer control, but they integrate with existing technology, and they can potentially run in real-time. On the other hand, low-level sample generators can create impressive results, but they are extremely computationally expensive to train and run and offer little chance any time soon of real-time interactive control. Recent work has shown that smaller networks can generate audio, but their fidelity and long-range coherence is not comparable to the larger networks.

In this paper, we will consider the capabilities of a third class of sound generating system besides preset finders and raw audio synthesizers, which is made possible via the DDSP library. This class of system combines features of the synthesizer preset-finder systems with features of the audio-based systems. The system uses a fixed architecture, additive and noise synthesizer to create realistic instrument sounds, but the model is embedded in a deep, recurrent neural network. We can train the same model to emulate different instruments with fine-grained, continuous control via a stream of pitch, amplitude and feature data. We have run these models in real-time on a current-generation commodity GPU (Geforce 2070).

2 Artistic context

2.1 Artists making use of 'unwanted' artefacts

We position our work as aiming to explore the space of possible sounds creatively, rather than endeavouring to solve the engineering problem of how to perfectly model and resynthesize a given instrument. Consider the example of the Roland TB-303 synthesizer, which according to music technology folklore, Roland originally designed as a guitar practice aid[5]. Once the 303 arrived in the hands of certain musicians, they exploited the artefacts of its resonant filter and oddly designed, accented step sequencer to contribute to the sound of a completely new musical movement, acid house. The idea of taking a musical tool and exploiting its (often unintended) quirks to create interesting new sounds should be familiar to most electronic musicians and sound designers.

2.2 The composition project

The other contextual element of this work is the project we are working on. We have been developing our DDSP models in the context of a creative project working with the human voice. When we gained access to DDSP in early 2020, our first experiments involved incompletely training a model on a Whitney Houston acapella and then using it to resynthesize new vocal inputs. The model generated a fascinating, ghostly noise that was somewhat pitched and subtly dynamic. The results of these initial experiments inspired us to put together a composition project based around DDSP networks. In this project, we are working with a composer to create a through-composed piece of music for human singer and neural network.

3 Experiments with DDSP

3.1 Phase 1: working with the basic models

We trained the model using a Whitney Houston acapella. The recording includes some backing vocals and delays on the voice, which it was not possible to easily remove. In the initial phase of the project, it was only possible to train and run the model at a 16kHz sample rate due to technical constraints in the data processing pipeline. The DDSP library was undergoing rapid development during our project, meaning our code had to be re-written a couple of times. We carried out the training using the DDSP colab notebook from the DDSP GitHub repository with a GPU back-end on the Google infrastructure. We ran the training for a few hours, aiming for the loss value recommended by the instructions in the notebook of around 5.0. We then downloaded the trained model for offline inference (resynthesis).

[5] https://www.theguardian.com/music/2011/jun/15/tadao-kikumoto-roland

In order to run the network in inference mode in real-time, we reworked the resynthesis colab notebook provided by the DDSP repository into a standalone script. This script reads audio in blocks from the audio input of the machine and feeds the blocks to the model for resynthesis. We have made the script available[6]. The resynthesis script operates at a 16kHz sample rate with a window size of 2048 samples and a hop size of 64 samples. This provides 250 pitch and loudness values per second as an input to the network. We were able to run this system in real-time on a Geforce 2070 GPU-equipped laptop. Further investigation showed that the main bottleneck was actually the CREPE pitch extractor, as opposed to the resynthesis model itself. The model can be run on a regular CPU if the pitch extraction is done beforehand. Figure 1 illustrates the output of our first

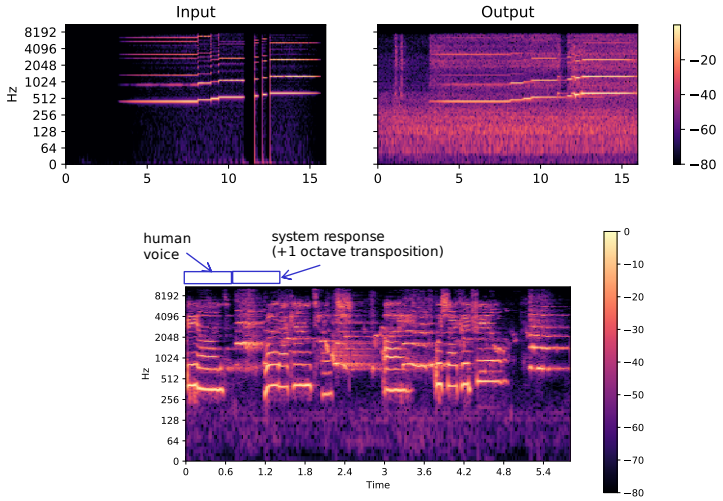

Fig. 1. Top left and right spectrograms are from our first experiment. Firsly the input which is a string-like sound playing an ascending note sequence on an analog Korg MS20 synthesizer, secondly the Whitney Houston model resynthesizing this input. Below that is a spectrogram of a live demonostration on national radio of our realtime DDSP model.

trained model in response to an ascending note sequence played with an analog string-like sound on a Korg MS20. The interesting features of this image are the wavering pitch around the later note transitions and the additional noise that can be seen in the output spectrogram. The wavering pitch is most likely because the pitch tracker is uncertain during the attack and release portions of the notes.

The noise is partly inadequate training data - we were training with only a few minutes of quite noisy audio as opposed to training with about 30 minutes of clean single instrument sound as described in the original DDSP paper. The

[6] https://github.com/yeeking/ddsp-experiments/

noise is also a characteristic of the voice which has a much breathier timbre than the synthesizer sound fed to the input. Around note onsets, the noise was quite dynamic - the sound had the breathy, ghostly quality that first encouraged us to work more with DDSP and voices. We were also able to give a public demonstration of our real-time DDSP system. Figure 1 shows the spectrogram of a DDSP flute model interacting in real-time with a human singer on national radio. We used this impromptu opportunity to test out the real-time performance of the system. The singer was Taryn Southern, who was a guest on the radio show. All the guests on the show were clearly taken aback by the rich timbral quality of the model. We have archived the section of the radio show[7].

3.2 Phase 2: network bending

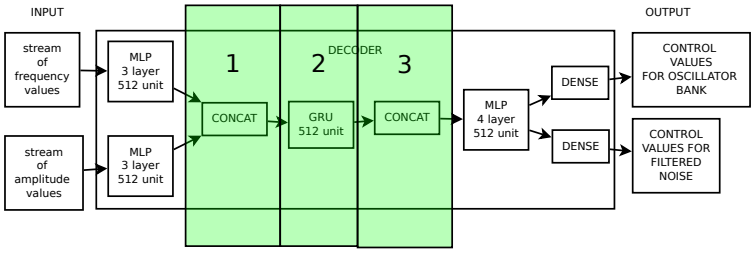

Fig. 2. The DDSP network model. Frequency and amplitude values are fed in, and control values for the 101 harmonic oscillator bank and noise FIR filter are produced. The three network bending points are highlighted as 1, 2 and 3. Transforms and signals are applied to the activation values of nodes in these layers.

Having familiarised ourselves with DDSP operations such as training and resynthesis in the first phase, in the second phase, we investigated how we could gain new types of control over the synthesis engine. We took inspiration from the ideas of network bending reported by (Broad et al., 2020), who used it to manipulate image generating networks. In essence, the concept of network bending is to take a pre-trained network and to apply additional signals or transformations to activations between its hidden layers during inference. The technique does not intend to improve the performance of the network in a technical sense. Instead, it aims to expose additional creative capabilities.

We developed several iterations of the software, and this process is described below. We experimented with four types of transformation and signal inserts: sine modulation, signal inversion, signal ablation and thresholding. Table 1 provides details of the transformations, and Figure 2 shows where the insert points are

[7] https://github.com/yeeking/ddsp-experiments/blob/main/radio5-system/call_and_response_radio5_2.mp3

on the network model. We discuss the effects of the transformations below. The software is available on GitHub[8]. We have also provided audio examples of the network bending experiments online[9].

Transform	**Operation**	**Parameters**
Oscillate	A[:,units] + sin(freq) * depth	freq, depth, units
Threshold	A[:,units]<thresh=A.min, A[:,units]>thresh=A,max	thresh, units
Invert	1 - A[:,units]	units
Ablate	A[:,units] = 0	units

Table 1. The network bending transforms we applied to the network

Developing the user interface and workflow The first version of the network bending software used a JSON-style data structure to specify the transformations. The workflow for this system was to write out the JSON specification, run the resynthesis offline, then audition the resulting audio file. When working with the composer, we found that this workflow did not meet their needs as writing JSON was unfamiliar to them, and the offline processing did not allow for rapid parameter exploration. To solve these problems, we implemented a graphical user interface on top of a real-time audio renderer, as shown in Figure 3. This allowed the composer to specify the same parameters but using sliders and drop-downs. It also allowed them to hear the effects of the parameter changes in real-time. This is more aligned with how a typical synthesizer or DAW works.

In this second version, the resynthesis engine operates upon a pre-existing audio file from which the pitch and amplitude inputs for the DDSP model have already been extracted. This allows the resynthesis to run in real-time without GPU acceleration as it only requires the computationally expensive step of extracting pitches with the CREPE model to happen once. In a third iteration of the software, we added MIDI control for the network bending parameters.

Selecting the input point As you might expect, we found the position in the network where we apply the transformation has an impact on the perceptual changes to timbre in the sound generated by the model. The first fully connected layer seemed to be where the most fruitful and interesting timbral changes occurred. The effects of applying changes to the final fully connected layer tended to be more pronounced although with less subtlety, often characterized by a general boosting in terms of amplitude across the whole spectrum. Changes in the recurrent middle layer (GRU) seem to have a limited auditory effect.

Effect of transform type: ablation, oscillation, thresholding and inversion The *ablation* (zeroing) of a proportion of units in the first fully connected

[8] https://github.com/Louismac/network-bending/
[9] http://louismccallum.com/network-bending-audio-examples

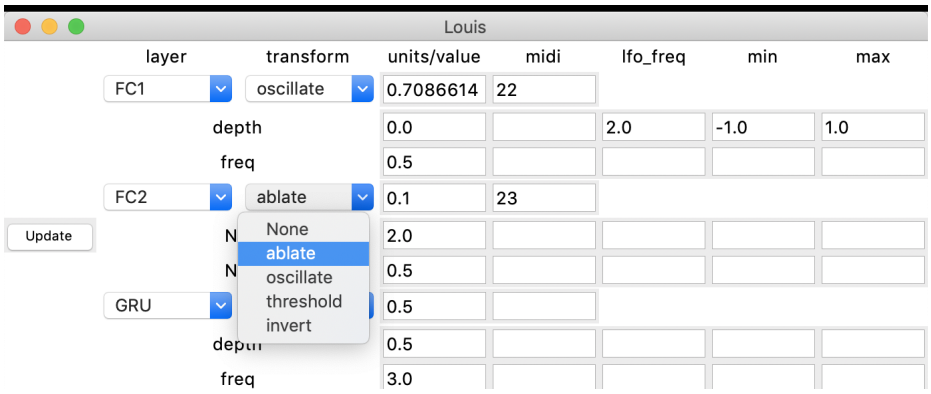

Fig. 3. A screenshot of the graphical interface for applying transformations

layer causes a vibrato effect across all frequencies. The effect ranges from a subtle vibrato at 65% ablation through to a strong effect at 95%. The *oscillation* transformation adds filtering and artefacts to the output at a controllable depth and frequency. In this respect, it has some similar qualities to using an LFO to control a filter parameter on a regular synthesizer. Adding a *threshold* to the first layer adds more noisy artefacts, whilst including it in the final fully connected layer blows up into heavy white noise with only a small proportion of units. We did not find very interesting effects from the *inversion* transformation. It is also possible to chain multiple transformations together to craft a sound quite far from the timbre of the original training audio. Even in these more destructive modes, the input pitch often remains perceptible.

Unit Selection In our original design, we selected units for transformation at random with a fixed proportion. In our second, GUI version we able to change the proportion over time, rather like a gain for the timbral effect of a particular transform. Being able to control this whilst playing creates opportunities to work on structure and variation throughout a longer performance or composition. Beyond this positive outcome, a more informed selection process for the subsets of units is a promising area for further research.

4 Next steps and future work

At the time of writing, we are completing the final composition phase of the project, wherein we will produce the final piece of music. We have commissioned two vocal recordings, each from a different vocalist. We are working closely with the composer to evaluate a range of different configurations of training data, network bending and pre-processing of pitch and amplitude data. A particularly interesting recent development is the release of a PyTorch implementation of DDSP, which allows models to be exported to PureData external format[10]. We are currently enjoying investigating this and will report more in a later paper.

[10] https://github.com/acids-ircam/ddsp_pytorch

References

Barkan, O., Tsiris, D., Katz, O., & Koenigstein, N. (2019). InverSynth: Deep Estimation of Synthesizer Parameter Configurations From Audio Signals. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *27*(12), 2385–2396. (Publisher: IEEE)

Broad, T., Leymarie, F. F., & Grierson, M. (2020). Network Bending: Manipulating The Inner Representations of Deep Generative Models. *arXiv preprint arXiv:2005.12420*.

Collins, N., Ruzicka, V., & Grierson, M. (2020). Remixing AIs: mind swaps, hybrainity, and splicing musical models. In *Proceedings of the 1st Joint Conference on AI Music Creativity*. Sweden.

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., & Sutskever, I. (2020). Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*.

Engel, J., Hantrakul, L., Gu, C., & Roberts, A. (2020). Ddsp: Differentiable digital signal processing. *arXiv preprint arXiv:2001.04643*.

Esling, P., Masuda, N., Bardet, A., Despres, R., & Chemla-Romeu-Santos, A. (2020). Flow synthesizer: Universal audio synthesizer control with normalizing flows. *Applied Sciences*, *10*(1), 302. (Publisher: Multidisciplinary Digital Publishing Institute)

Horner, A., Beauchamp, J., & Haken, L. (1993). Genetic {A}lgorithms and {T}heir {A}pplication to {FM}, {M}atching {S}ynthesis. *Computer Music Journal*, *17*(4), 17–29.

Kiefer, C. (2019). Sample-level sound synthesis with recurrent neural networks and conceptors. *PeerJ Computer Science*, *5*, e205.

Tatar, K., Macret, M., Pasquier, P., Tatar, K., Macret, M., & Pasquier, P. (2016). Automatic Synthesizer Preset Generation with PresetGen Automatic Synthesizer Preset Generation with PresetGen. *Journal of New Music Research*, *45*(2), 124–144. (Publisher: Routledge) doi: 10.1080/09298215.2016.1175481

Tubb, R., & Dixon, S. (2014). The Divergent Interface: Supporting Creative Exploration of Parameter Spaces. In *Proceedings of the International Conference on New Interfaces for Musical Expression* (pp. 227–232). Retrieved from `http://www.nime.org/proceedings/2014/nime2014_415.pdf`

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Yee-King, M. J. (2011). *Automatic sound synthesizer programming: techniques and applications* (PhD Thesis). University of Sussex.

Yee-King, M. J., Fedden, L., & d'Inverno, M. (2018). Automatic programming of VST sound synthesizers using deep networks and other techniques. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*(2), 150–159. (Publisher: IEEE)

Zukowski, Z., & Carr, C. (2017). Generating black metal and math rock: Beyond bach, beethoven, and beatles. *NIPS Workshop on Machine Learning for Creativity and Design*.