The control-synthesis approach for making expressive and controllable neural music synthesizers

Nicolas Jonason[1], Bob L. T. Sturm[1], and Carl Thomé[2]

[1] KTH Royal Institute of Technology
[2] Epidemic Sound
{njona,bobs}@kth.se, carl.thome@epidemicsound.com

Abstract. Deep neural networks have been successfully applied to audio synthesis. Such neural audio generation models can efficiently learn from data how to imitate a variety of instruments, such as piano and violin. However, effective control of these models is difficult. We introduce the "control-synthesis approach" to make neural audio synthesizers more controllable. This approach transforms user input into intermediate features to condition a neural audio synthesis model. We demonstrate this approach by implementing MIDI-controllable neural audio synthesizers and generating several examples for audition.

Keywords: Neural Audio Synthesis, Music Synthesis, Deep Learning

1 Introduction

Recent work introduces *neural synthesizers*: deep neural networks trained to imitate sounds from an audio dataset. These approaches are attractive because they learn how to reproduce timbres and pitches from audio recordings. They suffer from some limitations, however. One system does not incorporate note-to-note timbre dependencies, which are essential for expressively emulating certain instruments like the violin. Other systems circumvent this issue by using continuous user input, such as data manually input with a MIDI controller. But this requires a lot of control data for a user to input. Might such control data be generated by an intermediate model given low-resolution input?

This paper proposes a generic method for turning neural audio synthesis models into musical synthesizers that are expressive and controllable (Jonason, 2020). We demonstrate the *control-synthesis approach* by transforming models from the DDSP library (Engel, Hantrakul, Gu, & Roberts, 2020) into MIDI-controllable synthesizers trained to mimic target instruments from unannotated audio. We demonstrate the approach with two datasets and different MIDI input files. Our results clearly show how the control-synthesis approach effectively controls DDSP models to synthesize expressive music audio, but also show some peculiar artefacts that merit further exploration. The implementation code and audio examples can be found online.[3]

[3] https://erl-j.github.io/controlsynthesis

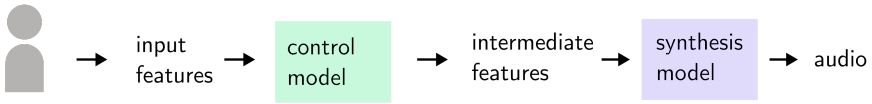

Fig. 1. The control-synthesis approach involves transforming low-resolution user input, e.g., MIDI, into intermediate features, which are then input into a synthesis model.

2 Neural Audio and Music Synthesis

We now briefly review music synthesis using deep neural networks. WaveNet (van den Oord et al., 2016) and WaveRNN (Kalchbrenner et al., 2018) both generate audio signals in the time domain. While WaveNet uses a fully convolutional network with strided convolutions to increase the receptive field for inference, WaveRNN uses a single layer recurrent network and a dual softmax layer. DDSP (Engel et al., 2020) is a different approach, which integrates differentiable classical signal processing modules in the machine learning model. This can yield high-quality audio generation models with fewer parameters than WaveNet and WaveRNN.

NSynth (Engel et al., 2017) uses a WaveNet-based auto-encoder to generate musical notes given a temporal embedding and a pitch value. A unique feature of NSynth is the interpolation between different timbres. However, since NSynth is trained on individual pitches, it does not capture note-to-note timbre dependencies. These can be important for emulating instruments like a violin. Fast and flexible neural audio synthesis (FFNAS) (Hantrakul, Engel, Roberts, & Gu, 2019) uses a WaveRNN-based model for synthesis, but is controlled with continuous loudness and pitch contours, similar to DDSP (Engel et al., 2020). Using these to control a synthesizer results in more dynamic control.

3 The Control-Synthesis Approach

We first discuss the control-synthesis approach in theory, and then describe an implementation of it.

3.1 Control-Synthesis Approach in Theory

The control-synthesis approach, as seen in Figure 1, uses a control model to generate intermediate features to drive a synthesis model. These intermediate features facilitate flexibility in the control of neural audio synthesis. While both the control and synthesis models can be implemented without the use of machine learning, the ability to automatically learn from data makes such approaches very efficient and flexible.

Using intermediate features provides a way to combine different control and synthesis models. First, it allows us to switch between different ways of controlling a synthesizer. For instance, we can extend a control model with additional

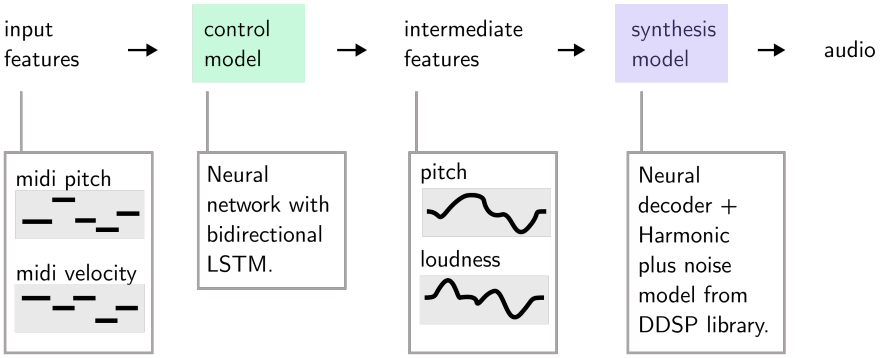

Fig. 2. An implementation of a neural audio synthesizer using a control-synthesis model. The input features MIDI pitch and velocity are converted to continuous pitch and loudness contours by a bidirectional LSTM. A synthesis model then converts these intermediate features into the resulting audio signal.

inputs to modulate performance, e.g., adding vibrato. Secondly, having a predefined intermediate feature representation means we can combine control models with synthesis models fitted by machine learning on different datasets, and vice versa. This allows us to combine different aspects of datasets into one synthesizer. For instance, we can combine the articulations of a certain dataset with the instrument timbre of another. DDSP uses this principle to perform "timbre transfer" (Engel et al., 2020).

3.2 Control-Synthesis Approach in Practice

Figure 2 diagrams an implementation of the control-synthesis approach. In the following, we discuss each of these blocks.

Input features The input features we use are derived from the pitches and velocities of MIDI note-on events over a 20-seconds duration. We create sequences of pitch and velocity values at a sampling rate of 250 Hz. We scale the pitch and velocity values to be within the range $[0, 1]$. Sections where no note events occur are given a MIDI velocity value of 0 and are assigned the pitch value of the preceding note event. All pitch sequences begin with the pitch value of the first note, regardless of when it occurs.

Control Model Figure 3 shows our control model, which transforms the low-resolution MIDI-derived pitch and velocity input features into intermediate features: pitch and loudness contours. The control model is trained with features extracted from audio recordings of a particular instrument. We first cut the recordings into non-overlapping 20-second segments. We randomly select 80% of the segments for training, and the rest we use for testing.

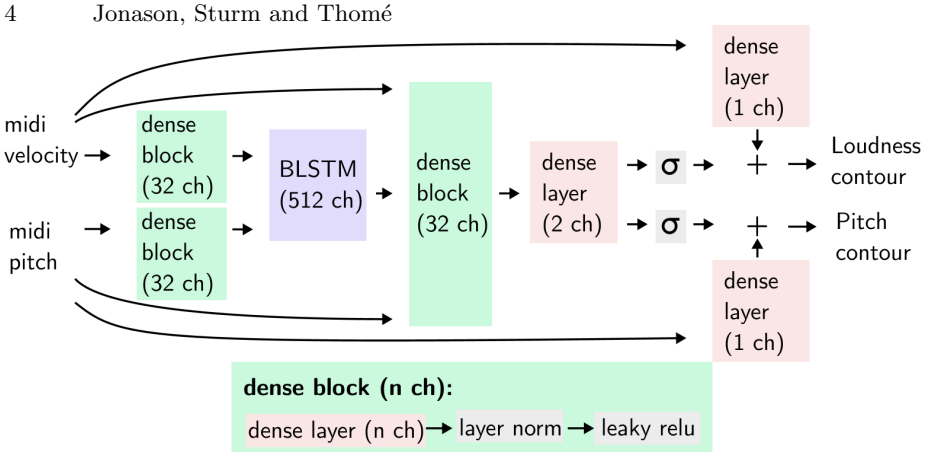

Fig. 3. The control model created pitch and loudness contours from the input features. These features first pass through dense blocks with 32 output channels each. The code for these dense blocks comes from the DDSP library (Engel et al., 2020). Each dense block consists of a dense layer, layer normalization and a leaky ReLU activation. The outputs of the dense blocks are merged and fed to a bidirectional LSTM (BLSTM). The output of the BLSTM is concatenated with the original input and passed to another dense block. Each output channel is summed with weighted and biased versions of pitch and velocity respectively.

We extract input and intermediate pitch features from audio segments in the following way. We extract pitches and a measure of confidence of the pitch estimates using the CREPE algorithm (Kim, Salamon, Li, & Bello, 2018).[4] We also extract the A-weighted loudness contour of the segment with code from the DDSP library (Engel et al., 2020). We then offset the CREPE pitch estimate such that the occurrence of equal-tempered pitches (based on A440 tuning) is maximized. We detect note onsets and offsets by looking for pitch confidence crossing a threshold or significant changes in pitch estimates. The threshold is set by trial and error using the training data. Finally, to determine the MIDI pitch of a note, we assume that the musical notes played belong to the 12-tone equal temperament scale. The MIDI number of each note is determined by the pitch closest to the time-averaged estimated pitch between the start and end of the note event.

We then assign MIDI velocities to each detected note event according to dataset peak A-weighted loudness value R using the following formula:

$$v_{\text{MIDI}}(R;\mu,\sigma) = R\frac{\mu}{\sigma}\sigma_{ref} + \mu_{ref} \tag{1}$$

where μ and σ are the mean and standard deviation of the peak loudness of all detected note events in the training data; and μ_{ref} and σ_{ref} are the mean and standard deviation of MIDI velocity data in a reference MIDI dataset. We use MAESTRO (Hawthorne et al., 2018). Velocity is capped within [0,127]. This formulation is motivated to obtain a plausible distribution of velocity.

[4] Technically speaking, CREPE estimates F0 but in this proof of concept we consider it to be the same as pitch.

Dataset	Pitch RMSE	Loudness RMSE
Violin train	8 cents	0.16 dbA
Violin test	67 cents	1.36 dbA
Flute train	60 cents	0.66 dbA
Flute test	180 cents	1.89 dbA

Table 1. Root mean square error on the pitch and loudness predictions of the control model on training and test data.

We post-process the MIDI data by removing notes shorter than 20 ms in duration, and then recasting the MIDI transcriptions into pitch and velocity sequences as described above. More complex ways to convert audio to MIDI will be explored in future work.

The control model is trained with a loss defined as the mean squared error of the predicted intermediate features. We use mini-batch gradient descent (with a batch size of 4) and ADAM with a learning rate of 3e-4. We stop training when a model has stopped showing an improvement on the test set for about 750 epochs. Since the test set was used for selecting the model, the test error might not reflect the generalization performance. For the violin data, this was around 12k epochs; for the flute data this was around 6k epochs.

Table 1 shows some results of two control models. More detailed examples can be found on the project website. We see a large difference between the errors on the training and test sets, as well as between the two datasets. The large difference between the training and test pitch errors is likely due to the error accumulating in sections where no notes are active. We hypothesise that the difference in error between each dataset is a consequence of the MIDI transcriber being hand-tuned only on the violin dataset. This means it is more likely to make errors on other datasets than on the violin set. Such errors make the task of the control model harder.

Synthesis Model The synthesis model transforms intermediate features into audio. The code for the synthesis model we use comes from the DDSP Python library without modification.[5] We train DDSP models having a harmonic plus noise model (HNM) (Laroche, Stylianou, & Moulines, 1993) with 60 harmonics and 65 noise band magnitudes. The decoder translates the loudness and pitch contours into parameters of the HNM. The HNM then transforms these synthesizer controls into audio. Convolution reverb is applied to the audio to simulate the acoustic properties of the instrument body and recording environment. Similarly to the weights of the decoder, the impulse response of the reverb is trained on target data. More details can be found in (Engel et al., 2020).

To generate training data for the synthesis model, we used the preprocessing code from the DDSP library (Engel et al., 2020). In addition to extracting

[5] `http://github.com/magenta/DDSP/blob/master/DDSP/training/gin/models/solo_instrument.gin`

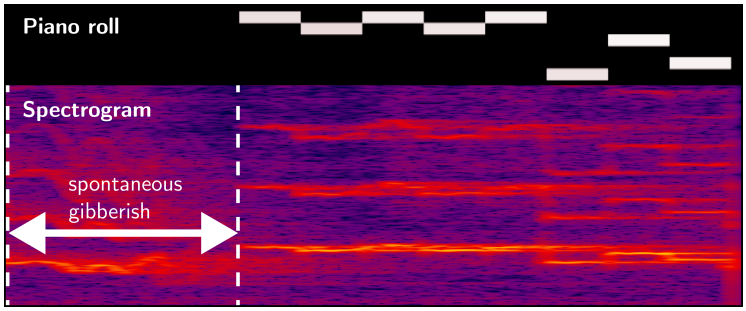

Fig. 4. Time aligned piano-roll and spectrogram of the first three seconds of a neural synthesized flute. Before the first MIDI note occurs the synthesized flute is active.

pitch and loudness contours, it segments the audio files into 4-second clips with a 1-second hop-length. We train each synthesis model with the training code provided by DDSP (Engel et al., 2020). We stop training when the spectral loss reaches around 4.75.

Datasets We train two control-synthesis implementations, each with a different set of audio recordings: one of solo violin (also used in (Engel et al., 2020)) that is about 13 minutes in length,[6] and the other of solo flute that is about 11 minutes in length.[7] We resampled all recordings to 16 kHz and mixed down to mono.

4 Evaluation

We evaluate the implementation presented in Sec. 3.2 using two out-of-training MIDI files: "Für Elise",[8] and "Ode to Joy".[9] We also investigate how the models perform in different parameter regimes by modifying note pitches, durations, velocities, and tempo. Audio examples are available online.

These examples show that the system can synthesize both instruments in realistic and expressive ways. However, we identify four artefacts in the synthesized audio. Sometimes before any note is to be played the synthesizer produces "spontaneous gibberish", and example of which is shown in Fig. 4. During notes with a long duration, one can sometimes hear a "looping" effect, which is similar to sample-based synthesis where the sustain portion of a sound sample is being replayed. We also find that at either low pitch or velocity, or short durations, the sound can have an unrealistic timbre. Finally, the pitches of some synthesized examples can sound a bit off-pitch.

We also synthesize examples with intermediate features created by a naive interpolation of MIDI pitch and velocity input features. The interpolation comes

[6] `https://musopen.org/music/13574-violin-partita-no-1-bwv-1002/`
[7] `https://musopen.org/music/24732-3-fantaisies-for-solo-flute-op-38/`
[8] `www.8notes.com/scores/457.asp?ftype=MIDI`
[9] `www.8notes.com/scores/11477.asp?ftype=MIDI`

from preprocessing the MIDI files in the same way we preprocessed the output of the MIDI transcription when we generated data for the control model. There are clear differences between the synthesized audio using the naive interpolations and the ones using the intermediate features created by the control model we create, particularly in the transitions between notes. The use of the control model results in a performance that more closely matches the sound of the target instrument, particularly in the transitions between notes. This indicates that the model captures some note-to-note timbre dependencies, but this should be verified through more careful testing and listening experiments.

We perform a few other experiments to test the flexibility of the control-synthesis approach. First, we modulate the generated pitch contour with a sinusoid at 6 Hz. We compare this to using the sine wave to modulate the fundamental frequency control of the harmonic synthesizer of the synthesis model. The difference between these two approaches is that the decoder of the synthesis model receives the modulated pitch feature in the first scenario. This means the modulation can, in turn, impact the inference of other controls of the HNM synthesizer besides the fundamental frequency. Although the modulation of pitch does result in a audible vibrato in the synthesis, we find it hard to detect a difference between the two approaches.

In another experiment, we combine control and synthesis models trained on different data. We train a control model on features extracted from recordings of traditional Irish fiddle.[10] We use the same MIDI input data as above, and synthesize the intermediate features with the synthesis model trained on the violin dataset. This audio output can be compared with the output of an instance where both control and synthesis models were trained on the violin dataset. While the outputs sound like a violin, they are noticeably different from each other.

5 Discussion

The results of our experiments show that the control-synthesis approach enables neural music synthesis models to generate expressive performances from low-resolution input like MIDI. This addresses the lack of expressivity in current MIDI-controlled neural music synthesis methods. Furthermore, we demonstrate that the control-synthesis model allows us to creatively modulate the generated performance. We also show that we can combine articulations and instrument sounds from different recordings by training control and synthesis models on different datasets.

Some of the synthesized examples have noticeable artefacts, especially when the model is given input features outside ranges seen in training. We believe that a few concrete modifications to the implementation could mitigate these issues. The gibberish artefact can likely be solved by masking the loudness contour

[10] `www.discogs.com/Tommy-Peoples-An-Exciting-Session-With-One-Of-Irelands-Leading-Traditional-Fiddlers/release/3629547`

when no MIDI notes are present. To address the "looping" artefact, we can introduce "temperature" (Hinton, Vinyals, & Dean, 2015) into the control and/or predictions of the synthesis model. The unrealistic timbre present in some examples could be addressed by using more training data, data augmentation, or more complex methods, e.g., (Arik, Chen, Peng, Ping, & Zhou, 2018).

We made the assumption that the control model needs information on future note events to generate realistic note transitions. This is the motivation for having the LSTM be bidirectional. However, bidirectionality prevents use of the system in a real-time setting. By instead accepting a short delay on the model output, we could remove bidirectionality while still receiving partial information about future note events, to enable the system to render note transitions in streaming applications.

6 Conclusion

We propose and demonstrate the control-synthesis approach to turning neural audio generation models into expressive and controllable music synthesizers. This approach essentially factors music synthesis into two sub-problems. A control model predicts intermediate features from user inputs, and then a synthesis model synthesizes audio from these intermediate features. The idea of using a separate model for controlling a synthesizer is not novel, e.g., (Maestre, Ramírez, Kersten, & Serra, 2009; Arcos, De Mantaras, & Serra, 1998; Sommer & Ralescu, 2014; Donahue, Simon, & Dieleman, 2019). Derenyi and Dannenberg (1998) use two models in series to simulate trumpet performances.

We create a particular implementation of the control-synthesis approach, and perform a variety of experiments. These show we can control a neural audio model to synthesize music with identifiable instrumental timbres. Another interesting observation is that control-synthesis models can be nested. If we consider the structure of the synthesis model used in our implementation, it follows our definition of a control-synthesis model, where the control model is the decoder, the intermediate features are the HNM parameters, and the synthesis model is the HNM synthesizer and convolution reverb. The generic nature of the control-synthesis approach means that it can be easily applied to other types of input features and instrument models. Our results also show some artefacts, which can be addressed in future work.

Acknowledgments

This work was completed as a collaboration between KTH Royal Institute of Technology and Epidemic Sound. Additional guidance was provided by Daniel Klevebring. Sturm is supported by ERC-2019-COG No. 864189 "MUSAiC: Music at the Frontiers of Artificial Creativity and Criticism".

References

Arcos, J. L., De Mantaras, R. L., & Serra, X. (1998). Saxex: A case-based reasoning system for generating expressive musical performances. *Journal of New Music Research*, *27*(3), 194–210.

Arik, S., Chen, J., Peng, K., Ping, W., & Zhou, Y. (2018). Neural voice cloning with a few samples. In *Proc. of NeurIPS* (pp. 10019–10029).

Derenyi, I., & Dannenberg, R. B. (1998). Synthesizing trumpet performances. In *Proc. int. computer music conf.* (pp. 490–496).

Donahue, C., Simon, I., & Dieleman, S. (2019). Piano genie. In *Proc. int. conf. intelligent user interfaces* (pp. 160–164).

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

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., & Norouzi, M. (2017). Neural audio synthesis of musical notes with wavenet autoencoders. *arXiv preprint*, *abs/1704.01279*.

Hantrakul, L., Engel, J. H., Roberts, A., & Gu, C. (2019). Fast and flexible neural audio synthesis. In *Proc. int. soc. music info. retrieval* (pp. 524–530).

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A., Dieleman, S., ... Eck, D. (2018). Enabling factorized piano music modeling and generation with the maestro dataset. *arXiv preprint*, *arXiv:1810.12247*.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Jonason, N. (2020). *The control-synthesis approach for making expressive and controllable neural music synthesizers* (Unpublished master's thesis). KTH Royal Institute of Technology.

Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., ... Kavukcuoglu, K. (2018). Efficient neural audio synthesis. *arXiv preprint*, *abs/1802.08435*.

Kim, J. W., Salamon, J., Li, P., & Bello, J. P. (2018). Crepe: A convolutional representation for pitch estimation. In *Proc. ieee int. conf. acoustics, speech and signal process.* (pp. 161–165).

Laroche, J., Stylianou, Y., & Moulines, E. (1993). Hnm: a simple, efficient harmonic+noise model for speech. In *Proceedings of ieee workshop on applications of signal processing to audio and acoustics* (p. 169-172).

Maestre, E., Ramírez, R., Kersten, S., & Serra, X. (2009). Expressive concatenative synthesis by reusing samples from real performance recordings. *Computer Music Journal*, *33*(4), 23–42.

Sommer, N., & Ralescu, A. (2014). Towards a machine learning based control of musical synthesizers in real-time live performance. In *Proc. modern artificial intell. cognitive sci. conf.* (pp. 61–67).

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*, *abs/1609.03499*.