DIFFMOOG: A DIFFERENTIABLE MODULAR SYNTHESIZER FOR SOUND MATCHING

Noy Uzrad[⋆] *Oren Barkan*[†] *Almog Elharar*[⋆]
Shlomi Shvartzman[⋆] Moshe Laufer[⋆] Lior Wolf[⋆] Noam Koenigstein[⋆]

[⋆] Tel-Aviv University, [†] The Open University, Israel

ABSTRACT

This paper presents DiffMoog - a differentiable modular synthesizer with a comprehensive set of modules typically found in commercial instruments. Being differentiable, it allows integration into neural networks, enabling automated *sound matching*, to replicate a given audio input. Notably, DiffMoog facilitates modulation capabilities (FM/AM), low-frequency oscillators (LFOs), filters, envelope shapers, and the ability for users to create custom signal chains. We introduce an open-source platform that comprises DiffMoog and an end-to-end sound matching framework. This framework utilizes a novel *signal-chain loss* and an encoder network that self-programs its outputs to predict DiffMoogs parameters based on the user-defined modular architecture. Moreover, we provide insights and lessons learned towards sound matching using differentiable synthesis. Combining robust sound capabilities with a holistic platform, DiffMoog stands as a premier asset for expediting research in audio synthesis and machine learning. Our code is released at: `https://github.com/aisynth/diffmoog`.

Index Terms— differentiable synthesis, sound matching

1. INTRODUCTION

Synthesizers are electronic musical instruments capable of generating a vast spectrum of sounds, from simple tonalities to complex auditory textures, central to many music genres. Composed of modules like oscillators, filters, low frequency oscillators, ADSR envelopes and modulators, they are controlled by an interface to fine-tune the sound output [1]. However, the multitude of interacting parameters makes sound design complex, often demanding deep expertise in sound synthesis and iterative testing.

Neural networks are increasingly used in sound design to replicate input sounds. While initial methods optimized a loss over synthesizer parameters [2–4], it may be beneficial to optimize over the sound directly, since the ultimate goal in sound matching is a high fidelity reproduction. Furthermore, replicating unlabeled out-of-domain sounds not generated by the synth at hand requires unsupervised learning. The implementation of traditional synthesizers does not support automatic differentiation, which is vital for direct sound comparison and optimization via backpropagation of gradients. To address this, several works proposed differentiable synthesizers [5–7]. However they either presented complex [5] or overly simplistic [6] models that deviate from real world synthesizers. For example, both synthesizers from [5, 6] lack modularity and conventional sound modules (e.g., LFOs, FM/AM modulators), leaving a gap for practical applications.

We present DiffMoog - a differentiable synthesizer with a modular architecture. It incorporates modules commonly found in commercial instruments, following an explainable design familiar to those versed in sound synthesis. The modularity enables both the creation of custom signal chains and the ability to isolate modules for research purposes. DiffMoog pairs with an end-to-end platform, facilitating its integration into an automatic sound matching system, along a newly crafted 'signal-chain loss' aimed at guiding the optimization process. The platform supports dataset creation, experiment configuration, model training, and offers comprehensive logging and analysis tools. By releasing DiffMoog as open-source, we aim to propel AI-guided sound synthesis forward. We discuss its implementation, evaluate its potential for sound matching, and share key insights and lessons gained throughout the experimentation process. We note that in the context of sound matching, the sub-task of frequency estimation through gradient descent techniques via minimizing spectrogram-based losses is an intrinsic challenge that remains open [8], as we discovered through our own experimentation. Previous studies [5–7] have relied on alleviating assumptions about the data or employed pitch estimation algorithms like CREPE [9]. However, we did not employ such techniques in our optimization process.

Our main contributions include: (1) The open-sourced DiffMoog synthesizer and sound matching platform, aiming to provide an easy gateway to conduct research in the field of AI sound synthesis and sound matching. (2) The introduction of a novel signal-chain loss. (3) Lessons and insights learned from optimizing DiffMoog; and (4) Demonstrating the superiority of the Wasserstein loss in frequency estimations.

2. RELATED WORK

Non-Differentiable Synthesizer Sound Matching: With the rise of machine learning methods, works in sound matching, also known as *parameters inference* [3], utilized supervised datasets of sound samples and their parameters, derived from typical non-differentiable synthesizers. Using this data, neural networks were trained to predict sound parameters, a method seen in commercial VST instruments [3, 4, 10]. Sound matching was also explored on custom synthesizers [2, 11]. While supervised methods optimize over synthesizer parameters, the ultimate goal of sound matching is obtaining a high-fidelity sound reconstruction, suggesting a direct audio optimization. These methods are also bound to synthesizer-specific data. DiffMoog bypasses these restrictions, being differentiable, allowing direct input-output sound comparisons beyond parameter reliance, broadening training and evaluation scope to include out-of-domain sounds. Noteworthy, direct sound optimization can also be achieved using genetic algorithms (GA) [4, 12]. However, GAs are inefficient, necessitating a vast number of synthesizer renders for a single match.

Neural Network Based Synthesizers: Advancements in deep learning gave rise to a suite of neural network based synthesizers [13–18]. Unlike traditional methods relying on the superposition of sinusoids or the routing of audio submodules (e.g., Additive, Subtractive, FM, Wavetable synthesis and their variants), these

synthesizers generate sound directly as the output of a neural network. However, they lack the control and explainability that the classic synthesizers offer. Other works explored the power of generative adverserial networks . GANSynth [15, 19] and diffusion models [16, 20]. Setting itself apart, DiffMoog uses traditional synthesizer modules, granting full user control and explainable behavior.

Differentiable DSP: Differentiable digital signal processing (DDSP) [5] integrates signal processing modules as differential operations into neural networks, allowing backpropagation. It uses a flexible additive synthesizer that can produce complex, realistic sounds. Additive synthesis is based on the Fourier theorem of stacking sinusoidal waves to construct a complex sound. A DDSP-inspired wavetable synthesizer was introduced in [21], where arbitrary waveshapes are stacked to generate the final sound. Both methods have limited control due to the large number of parameters with unpredictable effect on the final sound, though DDSP does allow manipulation of fundamental frequency and sound volume. Differentiable methods had also been employed in audio effects applications [22, 23]. Recent advancements include a differentiable mixing console for automatic multitrack mixing [24], transferring audio effects and production style between recordings through differentiable effects [25], and automating DJ transitions with differentiable audio effects [26].

Differentiable Synthesizers: Differentiable synthesizers aim to model commercial synthesizer characteristics, often using subtractive synthesis, in which an initial harmonically rich oscillator is further manipulated to modify its frequency content. A basic synthesizer, built upon two additive oscillators and a filter is used in [6]. Recently, the same authors enhanced their synthesizer to include ADSR envelopes and effects [27]. Yet, they lack certain complexities found in standard instruments, like LFOs, FM modules, and flexible routing options. FM synthesis as advised by Chowning [28], uses sine wave oscillators that are interconnected such that they serve as carriers and modulators, controlled by ADSR envelopes that can create complex harmonics and timbres evolving in time. Caspe et al. [7], manually implemented a few differential patch configurations from the well-known DX7 FM synthesizer [29]. Our proposed DiffMoog, to the best of our knowledge, is the first and most comprehensive modular differentiable synthesizer of its kind. It integrates both FM and subtractive synthesis techniques, offering unparalleled capabilities for sound generation and customization within the domain of differentiable synthesis.

3. THE DIFFMOOG SYNTHESIZER AND SOUND MATCHING PLATFORM

For clarity, we have aligned the nomenclature in this paper with the terms in our codebase. The code is organized as follows:
(1) Core implementation of the synthesizer - `src/synth`.
(2) Functionality of the sound matching system - `src/model`.
(3) Operations related to dataset management - `src/dataset`.
(4) Configurations for training, synthesis, and loss functions - `configs` directory.

3.1. DiffMoog

DiffMoog is a modular synthesizer, inspired by Robert Moog's seminal Moog Synthesizer of 1964, and stands as a differentiable synthesizer, optimized for gradient-based computations. DiffMoog integrates the qualities of subtractive modular synthesis with rich-harmonic oscillators, while also offering the capabilities of FM and additive synthesis. Through its versatile array of modules—oscillators, LFOs, filters, modulators, and shapers — DiffMoog is capable of producing a wide range of soundscapes surpassing previous differentiable synthesizers.

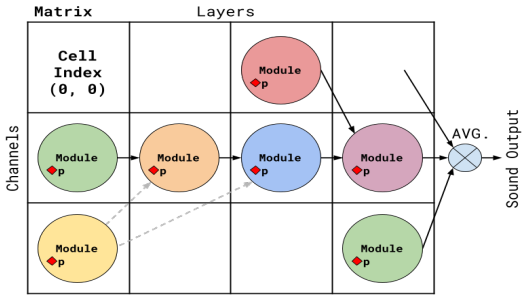


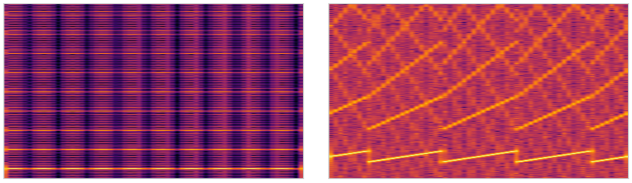
Fig. 1. The DiffMoog synth with an arbitrary *chain*. Shown: *matrix*, *cells*, *modules*, *connections* (arrows) and *parameters* ('p'). Black arrows are *fixed connections*, gray arrows are *optional connections*. Empty cell outputs the 0 signal.

The architecture is built upon a 2D *matrix* of *cells*. These cells act as containers for the synthesizer *modules* and their associated *parameters*, as defined in the `src/synth/synth_architecture.py` file. This matrix is structured by channels (rows) and layers (columns). Each cell, uniquely identified by its channel and layer indices, can host a single module. Signal flow within this structure is managed by: (1) *connections* which are assets of the cells: each cell is allowed to have multiple input and output connections, which dictate how signals traverse the matrix, and (2) using activation parameters to control module behavior, ranging from a simple on/off switch to bypassing an internal procedure. For example, one could opt to use just one oscillator where two are present or neutralize FM in a modulator. In order to simplify the signal flow and precisely define the sound generation process described later, the architecture imposes a rule on cell connections: a cell can only receive input from cells in preceding layers and send outputs to subsequent layers.

DiffMoog allows users to craft signal chains using predefined modules and connections, collectively termed *chains*, detailed in `src/synth/synth_chains.py`. This feature enables versatile synthesizer configurations, but also permits module isolation for research needs. Utilizing a particular chain, one can generate random datasets. Within a chain, connections can be (1) *fixed*, always present, or (2) *optional*, which, during dataset creation, are decided upon randomly for each sound instance in conjunction with the module's activation parameters to finalize the signal flow. Noteworthy, sound modules may possess optional inputs, and these must correspond with connections leading to their respective cells. An arbitrary chain of the DiffMoog synthesizer is depicted in Figure 1.

After populating the synthesizer matrix with modules and defining the cells interconnections (i.e. setting up a chain), the signal chain and system state are solidified. Sound generation proceeds layer-by-layer, from the lowest to the highest order. In each layer, cells compute sound independently, processing their (optional) input signals via their module logic and parameters. The signals from the cells then serve as input for cells in subsequent layers. Upon completion, the final layer's outputs across all channels are averaged to yield the final sound output (See `src/synth/synth_architecture.py::SynthModular.generate_signal()`).

The synthesizer modules, are implemented in `src/synth/synth_modules.py`. The Oscillator generates sine, square, and



(a) AM Square wave using tremolo effect (b) FM square modulated by a sawtooth wave

Fig. 2. Spectrograms sounds synthesized with DiffMoog. While typical, FM/AM sounds cannot be synthesized by prior differentiable synths.

sawtooth signals spanning the human hearing range (20Hz-20KHz), employing closed-form expressions instead of the additive synthesis used in previous works like [6]. The Low-Frequency Oscillator (LFO) acts as a sub-audible control signal, enriching temporal variations in sound, especially in the FM and Tremolo contexts. Furthermore, the system integrates FM Oscillator for frequency modulation, and a Filter typical for subtractive synthesis. Control of amplitude and filter characteristics over time is facilitated by the ADSR modules, while the Mix module blends input sounds, and Tremolo modulates amplitude for a pulsing effects.

Spectrogram plots of several sound examples generated by the DiffMoog synthesizer are shown in Figure 2. The corresponding audio examples are available for listening on the DiffMoog's GitHub repository.

3.2. End-To-End Sound Matching Platform

To train a model, an audio input is fed into a neural network, which predicts parameters for modules in a specified DiffMoog chain. In turn, the predicted parameters are used to synthesize a sound output that is optimized to replicate the input. This process is guided by minimizing both spectral and parameter losses. Since the parameter loss is optional, the system allows unsupervised training of unlabeled sounds, including out-of-domain sounds not made by DiffMoog. At inference time, given a new audio input, the neural network outputs parameter predictions for the corresponding DiffMoog chain that can be used to synthesize the predicted sound. A system diagram is shown in Fig. 3.

The encoder network features a neural backbone (e.g., ResNet, GRU) that produces a latent vector, complemented by a dynamically configured Multi-Layer Perceptron (MLP) head for each parameter, tailored to the selected synth chain. The encoder architecture is illustrated in Fig. 4. Notably, connections are not predicted as they are assets of the cells. Instead, the signal flow is governed exclusively by classified activation parameters. During training, connections deemed as 'optional' within the chain are treated as 'fixed'. Continuous parameters are normalized using the sigmoid function, while the 'Gumbel-Softmax' [30] trick is employed for sampling from categorical parameters. The reader is referred to `src/model/model.py` in our repository for the exact code implementation

The loss function we employed is a combined loss consisting of two terms: a parameters loss and a newly proposed spectral loss we name *signal-chain loss*. The **parameters Loss**, denoted as $\mathcal{L}_{\text{p}}$, is the sum of synthesizer parameters differences between predicted and original values. Regression parameters employ L1 or L2 loss, while categorical parameters use cross-entropy loss. The overall parameters loss is formulated as follows:

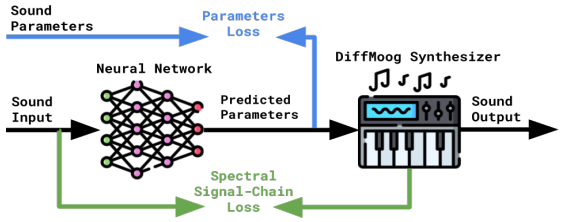


Fig. 3. The end-to-end sound matching system diagram.

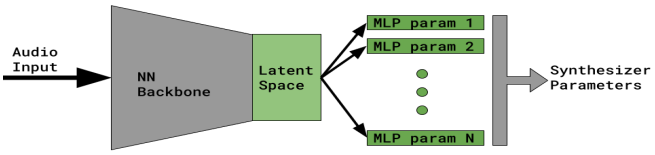


Fig. 4. The neural network architecture with dynamically allocated MLP heads

$$\mathcal{L}_{\text{p}} = \sum_{\text{n}\in\mathcal{N}} L_{\text{reg}}(p_n, \hat{p}_n) + \sum_{\text{m}\in\mathcal{M}} L_{\text{cat}}(c_m, \hat{c}_m) \tag{1}$$

where regression and categorical parameters belong to the sets $\mathcal{N}$ and $\mathcal{M}$ respectively, r_n and c_m represent the genuine values, while $\hat{r}_n$ and $\hat{c}_m$ stand for the predicted values. The loss functions for these parameter types are symbolized by L_{reg} and L_{cat}, respectively. The **signal-chain loss** measures the difference between the ground truth and predicted audio signals. However, unlike prior works, it evaluates the signal at **all stages** within the synthesizer signal-chain, not only at the final output. Given that the synthesizer's output can be seen as a function composition (e.g., $y = f(g(x))$ where y is the output signal, f and g are sound modules and x is an input signal), it aims to guide the optimization by improving early stage predictions. The signal-chain loss $\mathcal{L}_{\text{SC}}$ is defined using Eq. 2 and 3 below :

$$\mathcal{L}_{\text{SC}} = \sum_{\text{i}\in\mathcal{I}} \sum_{\text{j}\in\mathcal{J}} \sum_{\text{k}\in\mathcal{K}} L_{ijk} \tag{2}$$

$$L_{ijk} = \left\| (F_k(S_{ij}(x)) - F_k(S_{ij}(\hat{x}))) \right\|_p \tag{3}$$

where indices i, j, and k indicating cell, FFT window size, and processing type, respectively. The processing function F_{k} can be identity, log, or cumulative sum along time/frequency axes (Wasserstein). The p norm is either 1 or 2, and S denotes a spectrogram or mel-spectrogram transform. Importantly, the sets $\mathcal{I}, \mathcal{J}, \mathcal{K}$ are configurable, allowing various loss configurations e.g. multiresolution comparison, using traditional output-only optimization, etc. In total, the combined loss is defined as:

$$L_{\text{total}} = \mathcal{L}_{\text{p}} + \beta \cdot \mathcal{L}_{\text{SC}} \tag{4}$$

with β as a weighting factor. Please refer to `src/model/loss` for code implementation.

4. SOUND MATCHING EVALUATION

Evaluating DiffMoog for sound matching has been a challenging journey, encountering non-convergence in many experiments. Despite this, we successfully established several findings and techniques, presenting a comprehensive synthesizer and platform as a

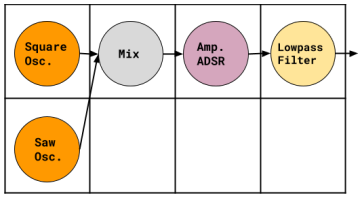


Fig. 5. The chain used for the experiment in Fig. 6, with a sawtooth oscillator, square oscillator, Amplitude ADSR and a Lowpass Filter.

progressive step in the research domain. Our work spanned various synthesizer chains, loss configurations, and neural architectures. While the complexities led to a focus on key findings rather than exhaustive experimental details, we believe this approach illuminates the most promising aspects of our research, fostering further exploration in this field.

Generally, we followed training procedures, hyperparameters and configurations similar to those outlined in [6]: (1)'P-loss': Training with parameters only; (2)'Synth': Training with parameters loss and gradually shifting to spectral (i.e signal-chain or some other configuration) loss over in-domain data; (3)'Real': Same as (2), but continuing training with spectral loss over out-of-domain data (Nsynth [13]).

Using the signal-chain loss solely failed systematically, However, when applied to a relatively basic synthesizer chain (Fig. 5), training with the spectral signal-chain loss after using the parameter loss hinted superior performance over the sole usage of parameters loss on out-of-domain data, which is on par with previous studies [6, 27]. Figure 6 presents an evaluation example using a chain which forms a synth similar to the one presented in [6], with a mix of a square and sawtooth oscillators followed by an ADSR amplitude shaper and a filter.

We also report that more complex chains utilizing FM modulations refused to converge, adding up to the already hard task of frequency estimation using spectral loss [8]. The back-propagated gradient from the spectral distance (Eq. 3) for frequency components and the FM modulation index is very abrupt. This phenomenon is conveyed in Fig. 7.

In another experiment, we examined the effectiveness of various spectral processing functions for frequency estimation, adopting the same experimental setup as described in [8]. A deliberately perturbed signal, more distant from a target than a predicted signal, is expected to have a gradient oriented towards the target. This assumption is compared for different loss configurations. As Table1 shows, using the Wasserstein distance on the time axis significantly enhanced the accuracy of frequency estimation for both square and sawtooth waveforms. This outcome defied our initial hypothesis of seeing more benefits when applying Wasserstein distance on the frequency axis. Notably, this approach varies from earlier studies that relied on spectrograms and their log representations for spectral loss

To conclude, our study indicates that differentiable synthesizers equipped with spectral loss optimization may indeed facilitate sound matching. Yet, achieving high precision in imitating typical sounds remains a formidable challenge. Notably, employing the Wasserstein distance could potentially mitigate the gradient issues encountered in frequency estimation using spectral loss. We anticipate that our platform will catalyze further research in this captivating domain. Moving forward, we propose the exploration of refined audio loss functions, optimization strategies, and alternative neural network architectures to surmount this hurdle.

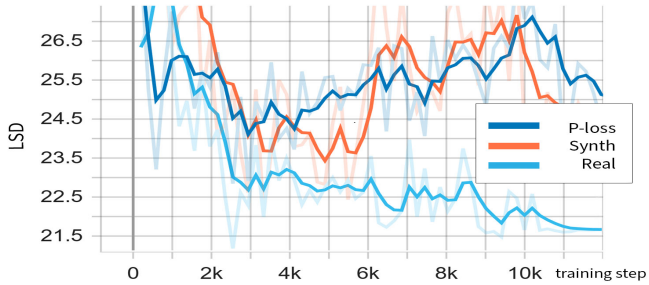
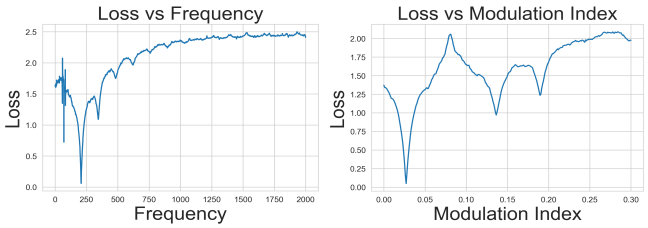


Fig. 6. Training procedures evaluation on NSynth. 'Synth' uses parameter loss until step 2k, then gradually introduces spectral loss (trained on in-domain data) until step 6k. 'Real' mirrors 'Synth' but transitions to out-of-domain data at step 6k. Log-Spectral Distance: $LSD = \|(log(S(x)) - log(S(\hat{x}))), \|_F$, F is the Frobenius norm. The 'Real' procedure shows the efficacy of using out-of-domain data for training, enhancing real-world sound reproductions.



(a) Loss vs. Oscillator Frequency (b) Loss vs. Modulation Index

Fig. 7. Loss surface for frequency and modulation index, illustrating highly non-convex behavior with many local minima, which pose challenges for optimization. Other synth parameters are fixed to their ground truth.

Loss conf. \ Perturbed dist.	**f ± ε**	**f ± 300 cents**	**f ± 600 cents**
Square waves			
Spectrogram, I	0.5	0.676	0.7
Mel, I	0.501	0.498	0.475
Spectrogram, Wasserstein time	0.48	**0.733**	**0.748**
Spectrogram, Wasserstein freq.	0.528	0.605	0.623
Mel, Wasserstein time	0.488	0.466	0.452
Mel, Wasserstein frequency	0.494	0.696	0.645
Sawtooth wave			
Spectrogram, I	0.47	0.642	0.701
Mel, I	0.52	0.506	0.463
Spectrogram, Wasserstein time	0.48	**0.712**	**0.715**
Spectrogram, Wasserstein freq.	0.53	0.571	0.543
Mel, Wasserstein time	0.469	0.487	0.467
Mel, Wasserstein frequency	0.505	0.624	0.619

Table 1. Comparison of different Loss configurations (modified S, F_1, in Eq. 3) and waveshapes across various distances of the pertrubed signal. In a well behaved distance, the prediction should be closer to the target than the perturbation is. This condition gives us a 0/1 error. Repeated and averaged over 1000 trials, high accuracies indicate that gradients usually point in the right direction. f is the predicted wave frequency, ε represents the local gradient, I is the identity function.

5. REFERENCES

[1] M. Russ, *Sound synthesis and sampling*, Taylor & Francis, 2004.

[2] O. Barkan, D.Tsiris, N. Koenigstein, and O. Kats, "Inversynth: Deep estimation of synthesizer parameter configurations from audio signals," *IEEE/ACM Trans. on Audio, Speech, and Language Processing*, vol. 27, no. 11, pp. 2385–2396, 2019.

[3] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos, "Universal audio synthesizer control with normalizing flows," in *International Conference on Digital Audio Effects (DaFX 2019)*, Birmingham, United Kingdom, Sept. 2019, DaFX 2019.

[4] M. J. Yee-King, L. Fedden, and M. D'Inverno, "Automatic programming of vst sound synthesizers using deep networks and other techniques," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 2, pp. 150–159, April 2018.

[5] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts, "Ddsp: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020.

[6] Naotake Masuda and Daisuke Saito, "Synthesizer sound matching with differentiable dsp.," in *ISMIR*, 2021, pp. 428–434.

[7] Franco Caspe, Andrew McPherson, and Mark Sandler, "Ddx7: Differentiable fm synthesis of musical instrument sounds," *arXiv preprint arXiv:2208.06169*, 2022.

[8] J. Turian and M. Henry, "I'm sorry for your loss: spectrally-based audio distances are bad at pitch," *arXiv preprint*, vol. arXiv:2012.04572, December 2020, Published in I Can't Believe It's Not Better! (ICBINB) NeurIPS 2020 Workshop.

[9] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello, "CREPE: A convolutional representation for pitch estimation," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018*, Calgary, AB, Canada, April 2018, IEEE, pp. 161–165.

[10] K. Itoyama and H. G. Okuno, "Parameter estimation of virtual musical instrument synthesizers," in *Proc. 40th Int. Comput. Music Conf.*, 2014, pp. 1426–1431.

[11] Oren Barkan and David Tsiris, "Deep synthesizer parameter estimation," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 3887–3891.

[12] Kıvanç Tatar, Matthieu Macret, and Philippe Pasquier, "Automatic synthesizer preset generation with presetgen," *Journal of New Music Research*, vol. 45, no. 2, pp. 124–144, 2016.

[13] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with wavenet autoencoders," in *Proc. of the 34th Int. Conf. on Machine Learning*, 2017, pp. 1068–1077.

[14] A. Défossez, N. Zeghidour, N. Usunier, L. Bottou, and F. Bach, "SING: Symbol-to-instrument neural generator," in *Advances in Neural Information Processing Systems*, 2018, pp. 9041–9051.

[15] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts, "Gansynth: Adversarial neural audio synthesis," *arXiv preprint arXiv:1902.08710*, 2019.

[16] Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catan, "Diffwave: A versatile diffusion model for audio synthesis," *arXiv preprint arXiv:2009.09761*, 2020.

[17] Antoine Caillon and Philippe Esling, "Rave: A variational autoencoder for fast and high-quality neural audio synthesis," *arXiv preprint arXiv:2111.05011*, 2021.

[18] Gwendal Le Vaillant, Thierry Dutoit, and Sébastien Dekeyser, "Improving synthesizer programming from variational autoencoders latent space," in *2021 24th International Conference on Digital Audio Effects (DAFx)*. IEEE, 2021, pp. 276–283.

[19] Pritish Chandna, Merlijn Blaauw, Jordi Bonada, and Emilia Gómez, "Wgansing: A multi-voice singing voice synthesizer based on the wasserstein-gan," in *2019 27th European signal processing conference (EUSIPCO)*. IEEE, 2019, pp. 1–5.

[20] Q. Huang, D. S. Park, T. Wang, T. I. Denk, A. Ly, N. Chen, Z. Zhang, Z. Zhang, J. Yu, C. Frank, et al., "Noise2music: Text-conditioned music generation with diffusion models," *arXiv preprint arXiv:2302.03917*, 2023.

[21] Siyuan Shan, Lamtharn Hantrakul, Jitong Chen, Matt Avent, and David Trevelyan, "Differentiable wavetable synthesis," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 4598–4602.

[22] Boris Kuznetsov, Julian D Parker, and Fabián Esqueda, "Differentiable iir filters for machine learning applications," in *Proc. Int. Conf. Digital Audio Effects (eDAFx-20)*, 2020, pp. 297–303.

[23] Marco A Martínez Ramírez, Oliver Wang, Paris Smaragdis, and Nicholas J Bryan, "Differentiable signal processing with black-box audio effects," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 66–70.

[24] Christian J Steinmetz, Jordi Pons, Santiago Pascual, and Joan Serrà, "Automatic multitrack mixing with a differentiable mixing console of neural audio effects," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 71–75.

[25] Christian J Steinmetz, Nicholas J Bryan, and Joshua D Reiss, "Style transfer of audio effects with differentiable signal processing," *arXiv preprint arXiv:2207.08759*, 2022.

[26] Bo-Yu Chen, Wei-Han Hsu, Wei-Hsiang Liao, Marco A Martínez Ramírez, Yuki Mitsufuji, and Yi-Hsuan Yang, "Automatic dj transitions with differentiable audio effects and generative adversarial networks," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 466–470.

[27] N. Masuda and D. Saito, "Improving semi-supervised differentiable synthesizer sound matching for practical applications," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 31, pp. 863–875, 2023.

[28] J. M. Chowning, "The synthesis of complex audio spectra by means of frequency modulation," *Journal of the Audio Engineering Society*, vol. 21, no. 7, pp. 526–534, 1973.

[29] M. Lavengood, "What makes it sound '80s? the yamaha dx7 electric piano sound," *Journal of Popular Music Studies*, vol. 31, no. 3, pp. 73–94, 2019.

[30] Eric Jang, Shixiang Gu, and Ben Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.