A Differentiable Neural Network Approach To Parameter Estimation Of Reverberation

Lyster, Søren Vøgg; Erkut, Cumhur

Published in:
Proceedings of the 19th Sound and Music Computing Conference, June 5-12th, 2022, Saint-Étienne (France)

DOI (link to publication from Publisher):
10.5281/zenodo.6798156

Creative Commons License
CC BY 4.0

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Lyster, S. V., & Erkut, C. (2022). A Differentiable Neural Network Approach To Parameter Estimation Of Reverberation. In R. Michon, L. Pottier, & Y. Orlarey (Eds.), *Proceedings of the 19th Sound and Music Computing Conference, June 5-12th, 2022, Saint-Étienne (France): SMC/JIM/IFC 2022* (pp. 358-364). Sound and Music Computing Network. Proceedings of the Sound and Music Computing Conference https://doi.org/10.5281/zenodo.6798156

A Differentiable Neural Network Approach To Parameter Estimation Of Reverberation

Søren Vøgg Lyster
Aalborg University Copenhagen
slyste20@student.aau.dk

Cumhur Erkut
Aalborg University Copenhagen
cer@create.aau.dk

ABSTRACT

Differentiable Digital Signal Processing is a library and set of machine learning tools that disentangle the loudness and pitch of an audio signal for timbre transfer or for applying digital audio effects. This paper presents a DDSP-based neural network that incorporates a feedback delay network plugin written in JUCE in an audio processing layer, with the purpose of tuning a large set of reverberator parameters to emulate the reverb of a target audio signal. We first describe the implementation of the proposed network, together with its multiscale loss. We then report two experiments that try to tune the reverberator plugin: a "dark" reverb where the filters are set to cut frequencies in the middle and high range, and a "brighter", more metallic sounding reverb with less damping. We conclude with the observations about advantages and shortcomings of the neural network.

1. INTRODUCTION

Designing a good sounding artificial reverberator is a hard task [1]. An algorithm with multiple filters and delay lines can consist of a high number of adjustable parameters, and the task of tuning these parameters by hand and ears to achieve the desired reverberation can take hours or days, even for a skilled audio engineer. Estimating a large number of parameters to reach a desired target is a use case that fits well into the subject of machine learning and neural networks. We present an adaptation of a neural network model to estimate a large set of parameters of a reverberator with the purpose of tuning that reverberator to emulate a target reverberated audio signal.

Google's Magenta Team recently released the Differentiable Digital Signal Processing (DDSP) [2] and has presented an approach to neural networking that incorporates audio processing in a neural network model and a multiscale spectral loss for audio differentiation. The loss can then be propagated back through the model to update the weights and change the audio generative parameters until the output signal resembles the target signal. The most prominent use of DDSP is to train a recurrent neural network on audio of an instrument for it to be able to infer the timbre information of the given instrument and transfer it to another audio input [1] .

With the introduction of DDSP more research has gone into incorporating different digital signal processing elements in a differentiable neural network. Kuznetsov and his colleagues presented multiple approaches to tuning filters using a differential neural network [3], and they have encountered the issue of training neural networks when a DSP element has a high probability of being unstable, e.g., when selecting coefficients for a biquad filter.

Extending the work of DDSP Ramírez et al. [4] propose using the differential signal processing to estimate parameters of third-party black box audio effect plugins by incorporating them in the neural network. Their work has been done on the LV2 audio plugin framework, and cases was shown for tube amplifier emulation, artifact removal from voiced signals, and music mastering.

Spotify's artificial intelligence research and development department Audio Intelligence [2] has released a utility framework for hosting Virtual Studio Technology audio plugins (VST) in Python called Pedalboard [5]. This utility eases the use of integrating black box audio plugins in a machine learning and neural network environment such as Tensorflow [6].

Our contribution to this body of work are as follows:

- Replacing the biquads of [3] with the state-variable filters (SVFs) for stability,
- Applying the two gradient estimation methods of [4], namely the finite differences and simultaneous perturbation stochastic approximation, to the SVFs and highly recurrent structures of Feedback Delay Networks (FDN) [7], and
- Establishing a building block in differentiable artificial reverberation [8] between black-box and white-box approaches.

2. DESIGN

To create a neural network that can estimate reverberator parameters by audio differentiation we need a reverberator plugin that can be incorporated in the network structure. Instead of finding an open-source reverberator we opted to create a custom audio plugin since it would present the easiest way to define exactly what parameters to expose to the neural network.

[1] https://sites.research.google/tonetransfer
[2] https://research.atspotify.com/audio-intelligence/

2.1 Feedback Delay Network

For the reverberator a feedback delay network (FDN) was selected since it generally should be able to emulate a broad selection of different reverberator algorithms [9] [1]. The FDN is an algorithm emulating reverb by having multiple delay lines with feedback through a scattering matrix [1]. A correct selection of delay lengths and scattering matrix should be able to result in a lossless algorithm that will produce an approximation to white noise when applied with an impulse [10].

The equations for the general FDN are shown in the following two equations, where x is the input, y is the output, and s_i is the delayed signal state at i, with i being the index in the number of delay lines N. A is the $N \times N$ scattering feedback matrix with indexes i and j. The variables c and b are gain factors in the system. The delay in samples n for each delay line is defined as M.

$$y(n) = \sum_{i=1}^{N} c_i \, s_i(n) \tag{1}$$

$$s_i(n + M_i) = \sum_{j=1}^{N} A_{ij} s_j(n) + b_i x(n) \tag{2}$$

Many different solutions for implementing a feedback matrix exists [10], but for now the matrix A is selected to be a lossless Householder matrix. The matrix for a 4-channel FDN implementation A_4 can be seen in equation 3, while an extension into a 16 channel implementation A_{16} can be seen in equation 4.

$$A_4 = \frac{1}{2} \begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} \tag{3}$$

$$A_{16} = \frac{1}{2} \begin{bmatrix} A_4 & -A_4 & -A_4 & -A_4 \\ -A_4 & A_4 & -A_4 & -A_4 \\ -A_4 & -A_4 & A_4 & -A_4 \\ -A_4 & -A_4 & -A_4 & A_4 \end{bmatrix} \tag{4}$$

After achieving a lossless reverb, the FDN algorithm can be extended to include frequency dependent dampening with the introduction of filters in the delay lines. Initially these filters were biquads, but they were later changed for state-variable filters (SVF) [11] due to their performance when calculating gradients. State variable filters can be described as a series of bi-quad filters consisting of a low-pass, a band-pass, and a high-pass filter. The benefits of SVF over normal biquads are the easy stability conditions of the contained filters, since they do not directly utilize poles and zeros, but instead have a dampening coefficient R and frequency coefficient g [3]. The stability conditions for the SVF are $R > 0$ and $g > 0$. The equations describing the input x and output y relationship and the internal states h_1 and h_2 (initialized at 0) of the SVF can be seen in equation 5 through 10, where y_{LP}, y_{BP}, and y_{HP} are the output of each filter with c_{LP}, c_{BP}, and c_{HP} being their respective gains.

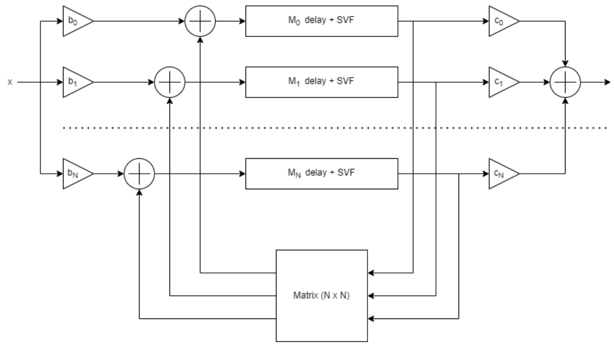

Figure 1: A diagram of the FDN algorithm implemented, showing delay lines $1, 2, \ldots, N$ with M_N samples delay and SVF filters, gain factors b_N and c_N, and the $N \times N$ scattering feedback matrix.

$$y_{BP}[n] = \frac{g([n] - h_2[n-1]) + h_1[n-1]}{1 + g(g + 2R)} \tag{5}$$

$$y_{LP}[n] = g \, y_{BP}[n] + h_2[n-1] \tag{6}$$

$$y_{HP}[n] = x[n] - y_{LP}[n] - 2R y_{BP}[n] \tag{7}$$

$$h_1[n] = 2y_{BP} - h_1[n-1] \tag{8}$$

$$h_2[n] = 2y_{LP} - h_2[n-1] \tag{9}$$

$$y[n] = c_{HP} \, y_{HP} + c_{BP} \, y_{BP} + c_{LP} \, y_{LP} \tag{10}$$

The value of the delay lengths M should be mutually prime to encourage denser echoes and avoid artifacts [10]. The delay lengths should be parameterized and exposed to the neural network but that would require that they are handled by some higher-level parameter since the network is unlikely to estimate the values to be mutually prime, as shown in earlier experiments. For this project the delay lengths have been fixed to a set of values given in the implementation of Prawda et al.'s highly flexible feedback delay network [9]. A diagram of the FDN reverberator can be seen in figure 1.

This FDN algorithm can be expressed as having 7 parameters for each delay line i: b_i, c_i, C_{HPi}, C_{BPi}, C_{LPi}, g_i, and R_i. Running a configuration with 16 delay lines results in 112 exposed parameters.

2.2 Neural Network

To achieve the set-out goals we need a neural network that can take a given input tensor containing audio and then output a tensor to compare with the target audio. The network should be able to update its weights by backwards propagation of gradients and loss. To incorporate the instances of the custom reverberator VST in the neural network model a custom audio processing layer needs to be implemented, and to propagate gradients back through the network the custom layer needs to implement a custom gradient function. This section will describe the different elements of the neural network model. A diagram of the neural network model can be seen in figure 2.

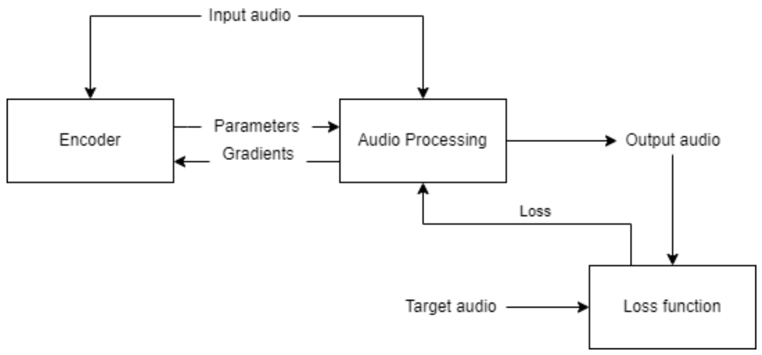

Figure 2: The implemented neural network model consisting of the encoder estimating parameters for the audio processing layer, the audio processing layer hosting the reverberator plugin, and the loss function returning a loss value that is propagated back through the model with the gradients that are calculated for the parameters in the audio processing layer.

2.2.1 *Gradient method*

To be able to do the full back propagation through the model the audio processing layer needs to estimate a gradient vector. Ramírez et al. [4] proposes two different ways of doing this: finite differences and simultaneous perturbation stochastic approximation. Both gradient estimation methods are implemented so they can be evaluated for this project.

The stochastic gradient estimation with finite differences (FD) [12] creates the gradient vector by iterating through all the parameters $\hat{\theta}$ and estimating the signal difference $\widetilde{\nabla}^{FD} f(\hat{\theta})$, where $f(.)$ is the neural network output driven by the dry audio signal. The equation for estimating a single parameter $\hat{\theta_i}$ with a fixed value ϵ and a standard basis of size equal to the number of parameters d^P indexed at the parameter index i is shown in equation 11

$$\widetilde{\nabla}^{FD} f(\hat{\theta})_i = \frac{f(\hat{\theta} + \epsilon d_i^P) - f(\hat{\theta} - \epsilon d_i^P)}{2\epsilon}. \tag{11}$$

This finite difference requires the plugin to generate audio twice for each parameter, with the parameter $\theta_i + \epsilon$ and $\theta_i - \epsilon$, and an additional time to create an output signal. This results in $2 \times P + 1$ computations for each batch item per epoch (with P being the number of parameters). For a reverberator with a long impulse response tail, this approach may be not feasible; it is used here however as a benchmark. Still, it is only used in the training phase and does not have any computational demands on the reverbarator in run-time.

Another approach to gradient estimation is the simultaneous perturbation stochastic approximation (SPSA). SPSA is encouraged when dealing with a high-dimensional problem [13] [12] and is done by creating a P-sized vector of perturbations Δ^P that contains differences for each parameter. In this case the vector contains -1 and 1 from a binomial distribution with p-value 0.5. The SPSA approach only requires $2 + 1$ computations compared to the $2P + 1$ computations of the FD method. The gradient calculation for a parameter with index i can be seen in equation 12.

$$\widetilde{\nabla}^{SPSA} f(\hat{\theta})_i = \frac{f(\hat{\theta} + \epsilon\Delta^P) - f(\hat{\theta} - \epsilon\Delta^P)}{2\epsilon\Delta_i^P} \tag{12}$$

2.2.2 *Encoder*

The task of the encoder is to encode an audio input into a set of values equal the number of parameters exposed by the reverberator plugin. Multiple different encoder models could be utilized for this task, but for simplicity the MobileNetV2 is selected as in the DeepAFx implementation [14] [4]. The MobileNetV2 is a two-dimensional convolutional network, so we use a logarithmic Mel Spectrogram layer to convert the input audio before the MobileNetV2.

2.2.3 *Custom audio processing layer*

The custom audio processing layer should receive an input the size of the number of parameters. During training these parameters will be used to generate gradients using the FD or SPSA method. When building the layer an instance of the VST plugin is created for each item in the given batch. This allows us to utilize the parallelism of Tensorflow and speed up the training. The VST plugin generates the entire audio signal in a single run, and its internal state is reset between each forward run. When training and calculating the vector containing the P number of parameter gradients the plugin is run either $P \times 2 + 1$ times (FD) or $P + 1$ times (SPSA), with the internal state being reset between each run.

2.2.4 *Loss function*

The loss function is an adaptation of the Multi-Scale Spectral Loss implementation from DeepAFx [4]. The short time Fourier transformation S is calculated across the audio signal for each window size i in one of $[1024, 512, 256]$, where y is the target audio and $\hat{y}$ is the output audio. The loss sum for each window size L_i is comprised of a L2 magnitude loss, the logarithmic of the L2 magnitude loss, and a L1 magnitude loss and can be seen in 13. The current construction of the loss is selected by iteratively testing the model training performance.

$$\begin{aligned} L_i = \|S_i(\hat{y}) - S_i(y)\|_2 + \log(\|S_i(\hat{y}) - S_i(y)\|_2) \\ + \|S_i(\hat{y}) - S_i(y)\|_1 \end{aligned} \tag{13}$$

The final loss value L_{tot} is a summation of different losses across the window size range N (equation 14).

$$L_{tot} = \sum_{i=0}^{N} L_i \tag{14}$$

2.2.5 *Optimizer*

As optimizer the Adam algorithm is chosen. The Adam algorithm is a modification of the stochastic gradient descent method that is easy to use and common in many newer neural networks such as DeepAFx [4] and DDSP [2].

2.2.6 *Hyper-parameters*

Selecting appropriate hyper-parameters for the gradient estimation and optimizer has been done by multiple iterations and evaluations of the model. Selecting a correct learning rate has been crucial to achieve a converging model that does not get stuck on local minimums. An initial high learning rate of 1×10^{-3} has been set with a callback for iterative reduction of the learning rate to 1.6×10^{-6} when the loss plateaus over a set number of epochs. For the gradient estimation the value ϵ has been set to 0.05.

3. IMPLEMENTATION

3.1 Reverberator

The FDN reverberator is implemented using JUCE [3] . JUCE is a c++ programming framework that is highly suitable for developing audio processing plugins. The FDN reverberator is compiled as a VST3 plugin using the VST3 SDK from Steinberg [4] . Using GitHub's continuous integration and development tool GitHub Actions an automated workflow has been utilized to compile the plugin for x86_64 Ubuntu 18.04 and publish to the neural network environment. The code is available on GitHub [5] .

The gain variables c and b, and the SVF filter variables C_{LP}, C_{BP}, C_{HP}, g, and R for each delay line are exposed as parameters using the JUCE AudioProcessorValueTreeState class. This allows the neural network to tweak those parameters. The variables M for delay length, and A for the scattering feedback matrix are not exposed. Since the delay lengths benefit greatly from being mutual primes and possibly span a large range of integers, the task of estimating them directly was deemed inefficient for the neural network. The scattering feedback matrix was chosen to be unchangeable since it easily could cause instability in the system. An important step in the implementation has also been to assure that the VST3 plugin could run headless, since we do not want to instantiate a graphical user interface for each forward processing run.

3.2 Neural Network

A repository containing the implemented neural network can also be found at Github [6] .

3.2.1 *Model architecture*

Two models have been implemented in Tensorflow with Keras, where the first model is the decoder model estimating parameter values from an audio input seen in Table 1, and the second model is the full differential model consisting of the decoder model and the custom VST3 layer seen in Table 2.

The log mel spectrogram layer implementation is taken from Keunwoo Choi [7] . The multi-scale spectral loss function is adapted from the DeepAFx Github repository [8] .

3.2.2 *Gradient estimation*

When running the model all 112 initial parameters are set to a value of 0.5. Unfortunately, the SPSA method does not seem to allow the model to converge in the same manner as FD. One reason that FD performs better than SPSA for the FDN plugin might be the result of the SVF parameters being highly dependent on each other, and that the effect of changes in the 16 parallel filters are hard to distinct when calculating gradients using this approach. This effect becomes more apparent when comparing the implementation to a simplified toy model with only one filter, where SPSA performs well.

3.3 Python Integration Using Pedalboard

Spotify's Audio Intelligence Lab has created a Python package called Pedalboard [5] that supports hosting of VST3 and Audio Unit plugins in python. This is done by implementing Python Bindings that allows for C and C++ integration in Python. The package contains functionality to process audio, change parameters, and reset the internal state of plugins (given that the plugin supports these functions). The package also claims to be thread-safe, supporting multiple CPU cores, and compatible with tensor inputs. Utilizing this package has made the implementation of the custom reverberator plugin in a neural network possible.

3.4 Toy Model

A simplified FDN plugin with only one delay line has been implemented to test the neural network model. This simplified version consisted of only 7 parameters, greatly reducing the complexity of the estimation problem. The toy model implementation showed that a higher ϵ value used in the gradient approximation would result in a faster convergence, but with the drawback of possible tuning the SVF parameters quickly towards instability and resulting in exploding gradients. Two short experiments were done to verify the ability of the neural network using two target audio sources generated with the simplified FDN implementation, a signal with high-frequency attenuation (”dark”) and a signal with low frequency attenuation (”bright”). The result is reported on in table 3.

[3] https://juce.com/
[4] https://www.steinberg.net/developers/
[5] https://github.com/VoggLyster/Reverberator/tree/SMC
[6] https://github.com/VoggLyster/ReverberatorEstimator/tree/SMC
[7] https://gist.github.com/keunwoochoi/f4854acb68acf791a49a051893bcd23b
[8] https://github.com/adobe-research/DeepAFx/blob/main/deepafx/losses.py

Layer (type)	**Output Shape**	**Param #**
audio_input (InputLayer)	[(None, 96000)]	0
log_melgram_layer (LogMelgramLayer)	(None, 372, 128, 1)	0
input_norm (BatchNormalizationLayer)	(None, 372, 128, 1)	4
mobilenetv2_1.00_372 (Funtional)	(None, 112)	2400880

Total params: 2,400,884
Trainable params: 2,366,770
Non-trainable params: 34,114

Table 1: Parameter model summary from Tensorflow.

Layer (type)	**Output Shape**	**Param #**	**Connected**
audio_input (InputLayer)	[(None, 96000)]	0	
parameter_model (Functional)	(None, 112)	2400884	audio_input[0][0]
vst_processor (VSTProcessor)	(None)	0	audio_input[0][0] parameter_model[0][0]

Total params: 2,400,884
Trainable params: 2,366,770
Non-trainable params: 34,114

Table 2: Full model summary from Tensorflow.

	loss	mae
1. ”dark”	0.11432	1.9732e-4
2. ”bright”	0.0122	7.834e-5

Table 3: Error values for two toy model experiments

	loss	mae	mse
1. ”dark”	3.12275	0.0111	0.0007
2. ”bright”	3.92548	0.0115	0.0010

Table 4: Error values for the two experiments

4. EXPERIMENTS

To test the model two final experiments were done. Both experiments consisted of a two second audio input and target pair at 48kHz. These pairs where created using the implemented FDN reverberator plugin to ensure that a target solution exists where the algorithms used are able to emulate each-other. The first target is a ”dark” reverb where the filters are set to cut frequencies in the middle and high range. The other target is a ”bright” and more metallic sounding reverb with less damping. For generating the sound an impulse-like finger snap sample has been used.

To utilize the multiple GPU’s available for training a batch size of 8 has been chosen. The batch consists of 8 input audio tensors with the same generated input audio sample, and 8 target audio tensors with the same generated target audio sample.

Both experiments were run in multiple iterations with number of epochs ranging from 500 to 1500. This was done to be able to evaluate the performance of the model simultaneously.

5. EVALUATION

5.1 Qualitative Evaluation

A qualitative evaluation has been done by visual inspection and by ear. The waveforms and spectrograms for the ”dark” and ”bright” experiments can be seen in figures 3 and 4. In the case of estimating the parameters for the ”dark” reverb an audio listening inspection indicates an almost identical reverb. For the ”bright” reverb it is possible to hear a slightly longer decay time in the high frequency spectrum that might be fixed with further training. By using the calculated loss, and by calculating mean-squared-error (mse) and mean-absolute-error (mae) for target audio y and output audio $\hat{y}$ in equations 15 and 16, we get the values reported in table 4.

$$mse = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{15}$$

$$mae = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{16}$$

For both experiments the models were still converging before termination, but the rate of convergence was slow.

5.2 Quantitative Evaluation

The performance of the model can be evaluated by looking at the training time for the two experiments. The ”dark” reverb trained for approximately 20 hours while the ”bright” reverb trained for approximately 56 hours. Both tests where run on a 3 Titan X GPU setup with a 2.4 GHz Intel Xeon E5 processor.

6. CONCLUSION AND FUTURE WORK

A neural network model has been created that is able to incorporate VST3 plugins created in JUCE. This incorporation allows for parameter estimation of a custom FDN reverberator using differentiable audio processing. The parameter estimation model has shown that it is able to converge towards a set of parameters that allows the reverberator to recreate a given input signal. Usable results have been achieved when the reverberation algorithm architecture has been the same for both target and generated audio. But the rate at which the model estimates the parameters is very slow when using the FD gradient estimation.

Two estimation examples have shown the possibilities of the network, but also the current limitations and computational weaknesses. The first experiment gave good initial results with a convergence towards a matching reverb. The second experiment showed considerably more issues with the convergence. A toy model with a simplified reverberation algorithm has shown the ability to quickly estimate parameters utilizing SPSA and a higher epsilon value, but the gap between a single delay line implementation and a 16 delay lines implementation with 16 different filters must be still bridged.

6.1 Future Work

Considering the weaknesses, it is possible to list multiple efforts that should be made in the future to optimize the neural network model and encourage further use. Future work will focus on speed, loss functions, on the parameterization of delay and matrix of the FDN, and on the expansion of the algorithm. Using Finite Differences (FD) as a gradient estimator has a high computational cost. This could be optimized by creating an environment where the SPSA method would prove useful. The convergence of the neural network slows down greatly over time, signaling that we might not have settled on the best hyper-parameters for this problem. Running a hyper-parameter estimation of the neural network is a time-consuming task but should yield better results when moving forward. Further investigation into loss methods might be beneficial for the network. There are many different approaches to utilizing loss functions with regards to audio signals that might warrant

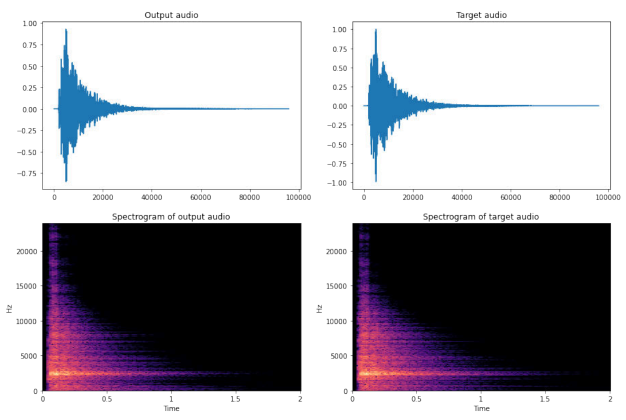

Figure 3: Dark reverb. Upper left figure shows the waveform of the output audio, upper right shows the waveform for the target audio. Lower left shows the spectrogram of the output audio, lower left shows the spectrogram out the target audio.

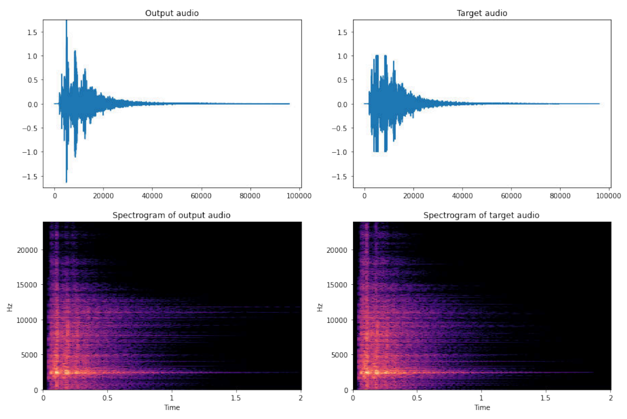

Figure 4: Bright Reverb. Upper left figure shows the waveform of the output audio, upper right shows the waveform for the target audio. Lower left shows the spectrogram of the output audio, lower left shows the spectrogram out the target audio.

an investigation. As the goal of this approach is to emulate different reverberations expansion of the FDN algorithm should be done. By exposing some high-level parameters to the neural network would allow it to be more adaptable to different type of reverberated signals. These high-level parameters should be designed to give the most flexibility while still trying to enforce the rules of losslessness and mutually prime delay lengths. Additionally, delay line modulation would introduce time variance for better physical accuracy. With future work and extension of the project a perceptual evaluation will also be needed to show the viability of technology.

7. REFERENCES

[1] V. Valimaki, J. D. Parker, L. Savioja, J. O. Smith, and J. S. Abel, "Fifty years of artificial reverberation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 5, pp. 1421–1448, 2012.

[2] J. H. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: differentiable digital signal processing," *CoRR*, vol. abs/2001.04643, 2020. [Online]. Available: https://arxiv.org/abs/2001.04643

[3] B. Kuznetsov, J. D. Parker, and F. Esqueda, "Differentiable IIR filters for machine learning applications," in *Proc. Int. Conf. Digital Audio Effects (eDAFx-20)*, 2020, pp. 297–303.

[4] M. A. Martínez Ramírez, O. Wang, P. Smaragdis, and N. J. Bryan, "Differentiable signal processing with black-box audio effects," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, June 2021.

[5] Spotify, "Pedalboard," https://github.com/spotify/pedalboard, 2021.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, and et al, "TensorFlow: Large-scale machine learning

on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[7] J.-M. Jot and A. Chaigne, "Digital delay networks for designing artificial reverberators," in *Proc. AES Convention*, Feb 1991, preprint 3030.

[8] S. Lee, H.-S. Choi, and K. Lee, "Differentiable artificial reverberation," *arXiv*, 2021.

[9] K. Prawda, S. Willemsen, S. Serafin, and V. Välimäki, "Flexible real-time reverberation synthesis with accurate parameter control," in *23rd International Conference on Digital Audio Effects*, 2020, pp. 16–23.

[10] J. O. Smith, *Physical Audio Signal Processing*. `http://ccrma.stanford.edu/~jos/pasp/`, accessed 16/12/2021, online book, 2010 edition.

[11] A. Wishnick, "Time-varying filters for musical applications." in *DAFx*, 2014, pp. 69–76.

[12] M. C. Fu, "Stochastic gradient estimation," *Handbook of simulation optimization*, pp. 105–147, 2015.

[13] J. C. Spall, "An overview of the simultaneous perturbation method for efficient optimization," *Johns Hopkins apl technical digest*, vol. 19, no. 4, pp. 482–492, 1998.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.