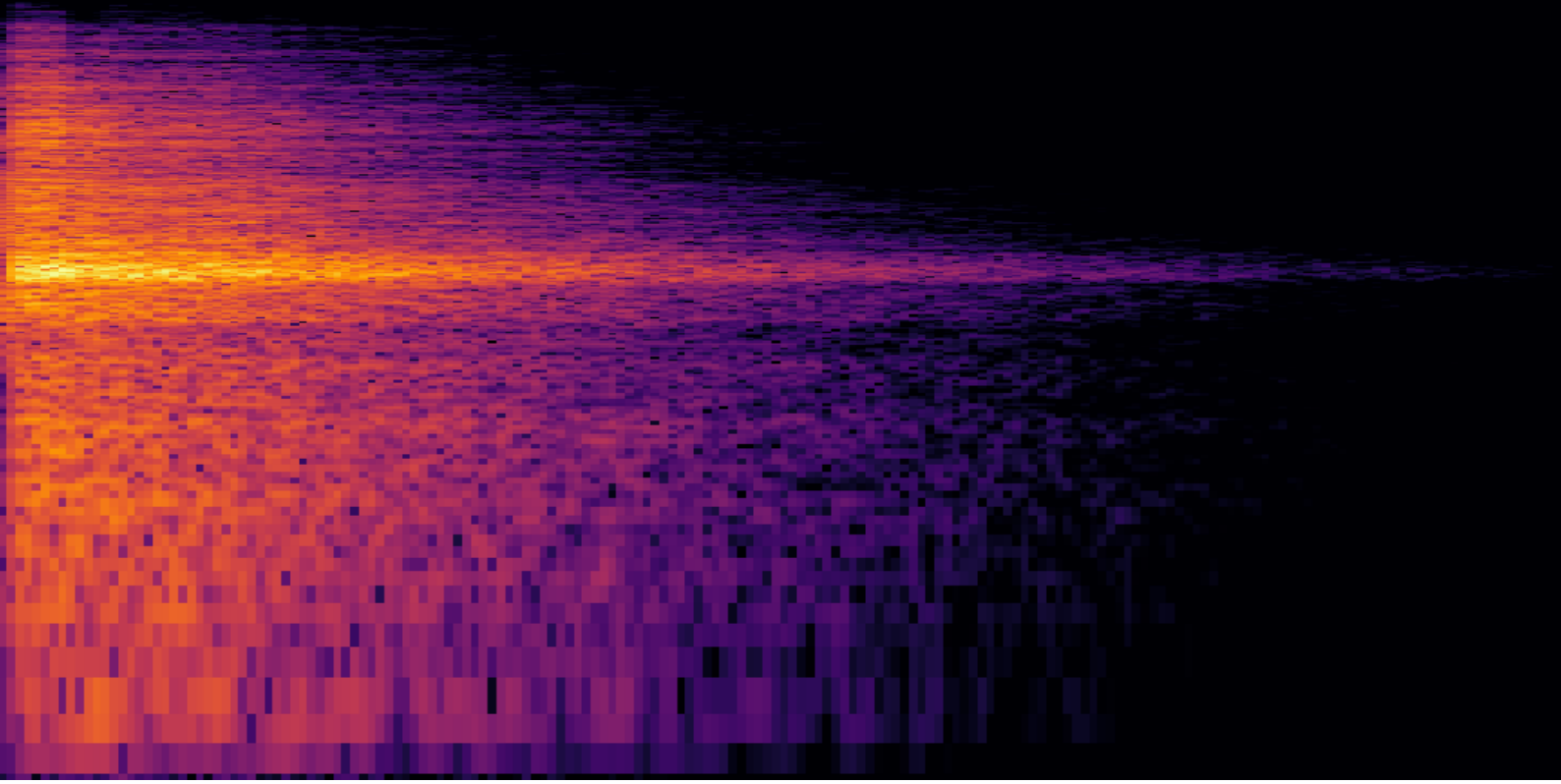


Parametric Tuning of Extended Reverberation Algorithm Using Neural Networks

Søren Vøgg K. Lyster

Sound and Music Computing, 2022-05

Master's Project

Parametric Tuning of Extended Reverberation Algorithm Using Neural Networks

Søren Vøgg. K. Lyster

Sound And Music Computing, 2022-05

Master's Project

Studyboard of Media Technology
Aalborg University Copenhagen
http://smc.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Parametric tuning of extended reverberation algorithm using neural networks

Theme:
Master thesis

Project Period:
Spring Semester 2022

Project Group:
Individual

Participant(s):
Søren Vøgg Krabbe Lyster

Supervisor(s):
Cumhur Erkut

Copies: 1

Page Numbers: 66

Date of Completion:
May 25, 2022

Abstract:

This thesis sets out to implement an extended feedback delay network with a comprehensive set of parameters that can be estimated by a proposed neural network. The goal of the neural network is to use audio differentiation to tune the parameters of the feedback delay network, to allow it to emulate other reverberators. The feedback delay network is implemented as a VST3 audio plugin and embedded in the neural network via a proposed audio processing model. Qualitative evaluation is done on the performance of the plugin and the neural network, and a perceptual listening test is done to evaluate the subjective quality of the reverberated signals created by the estimated reverberation parameters.

Studyboard of Media Technology
Aalborg University Copenhagen
http://smc.aau.dk

AALBORG UNIVERSITET
STUDENTERRAPPORT

Titel:
Parametric tuning of extended reverberation algorithm using neural networks

Tema:
Master thesis

Projektperiode:
Spring Semester 2022

Projektgruppe:
Individuel

Deltager(e):
Søren Vøgg Krabbe Lyster

Vejleder(e):
Cumhur Erkut

Oplagstal: 1

Sidetal: 66

Afleveringsdato:
25. maj 2022

Abstract:

This thesis sets out to implement an extended feedback delay network with a comprehensive set of parameters that can be estimated by a proposed neural network. The goal of the neural network is to use audio differentiation to tune the parameters of the feedback delay network, to allow it to emulate other reverberators. The feedback delay network is implemented as a VST3 audio plugin and embedded in the neural network via a proposed audio processing model. Qualitative evaluation is done on the performance of the plugin and the neural network, and a perceptual listening test is done to evaluate the subjective quality of the reverberated signals created by the estimated reverberation parameters.

Contents

Preface **xi**

1 Introduction **1**
1.1 Thesis Concept . . . 1
1.2 Background . . . 1
1.2.1 Reverberation . . . 1
1.2.2 Neural Networks . . . 2
1.3 State-of-the-art . . . 2
1.4 Black-Box Reverberators . . . 3
1.4.1 Ableton Reverb . . . 3
1.4.2 Hall Of Fame 2 . . . 3
1.4.3 Valhalla Room . . . 4

2 Feedback Delay Network **7**
2.1 Architecture . . . 7
2.1.1 Lossless Prototype Design . . . 8
2.2 Delay-Line Lengths . . . 9
2.3 Graphic Equalizer . . . 10
2.3.1 Interaction-Matrix-Based Design . . . 10
2.3.2 Attenuation Time . . . 12
2.3.3 High Shelf Filter . . . 12
2.4 Feedback Matrices . . . 13
2.4.1 Householder Matrix . . . 13
2.4.2 Hadamard Matrix . . . 13
2.4.3 Matrix Difference . . . 14
2.5 Time Varying Delays . . . 14
2.6 Pre-Delay . . . 15

3 Neural Network **17**
3.1 Parameter Model . . . 17
3.1.1 Mel-Spectrogram . . . 17

3.1.2 Convolutional Network . . . 18
3.2 Audio Processing Model . . . 18
3.2.1 Gradient Approximation Method . . . 18
3.3 Full Model . . . 20
3.4 Losses . . . 20
3.4.1 Multi-Scale Spectral Loss . . . 21
3.4.2 Envelope Loss . . . 22
3.4.3 Echo Density Loss . . . 22
3.4.4 Total Loss . . . 25
3.5 Optimizer . . . 27

4 Implementation **29**
4.1 VST3 in JUCE . . . 29
4.1.1 GUI . . . 30
4.1.2 Sampling Rates . . . 30
4.1.3 Parameters . . . 30
4.1.4 Compatibility . . . 32
4.1.5 Internal State . . . 32
4.1.6 Lossless Prototype . . . 33
4.2 Pedalboard . . . 33
4.2.1 Process and Reset Functions . . . 34
4.2.2 Parameter Control . . . 34
4.3 Neural Network . . . 34
4.3.1 Hyperparameter Estimation . . . 34
4.3.2 Pretraining . . . 35
4.4 Toy Model . . . 35

5 Evaluation **37**
5.1 Perceptual Evaluation . . . 37
5.1.1 Training . . . 38
5.1.2 Test Design . . . 39
5.2 Qualitative Evaluation . . . 40
5.2.1 Plugin Performance . . . 40
5.2.2 Neural Network Performance . . . 40

6 Results and Discussion **43**
6.1 Perceptual Evaluation Results . . . 43
6.1.1 Ableton Reverb . . . 43
6.1.2 Hall Of Fame 2 . . . 43
6.1.3 Valhalla Room . . . 44
6.2 Discussion . . . 44

7 Conclusion **47**
7.1 Conclusion 47
7.2 Future Work 47

Bibliography **49**

A Appendix A - Evaluation Data **53**
A.1 MUSHRA Interface 53
A.2 Test Instructions 53
A.3 Test Results 58

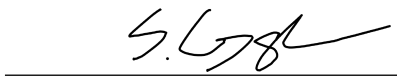
B Appendix B - Neural Network Models **59**
B.1 Summary of the parameter model 59
B.2 Summary of the full model with both parameter model and audio processing model 60

C Appendix C - Jupyter Notebook for Training **61**

Preface

This Master Thesis looks at the often cumbersome task of tuning algorithmic parameters when developing artificial reverberation by implementing a reverberation plugin based on an extended feedback delay network, and tuning it with the help of a proposed neural network. The thesis work has been done for the Sound and Music Computing Master's program at Aalborg University Copenhagen from February 1st to May 25th 2022. I have always had a high interest in audio effects, and the development of them, especially reverberation. I recall a story told at my previous workplace at TC Electronic where multiple audio engineers used months tuning algorithm parameters during their development of one of their world-renowned artificial reverberators. With my introduction to machine learning and work with differential digital signal processing during my masters program, the idea of using artificial intelligence to tune algorithmic parameters of a designed reverberator plugin formed. I would like to extend a thanks to the supervision I have received during the thesis work, and to all the lovely testers that helped me with the perceptual evaluation.

Aalborg University, May 25, 2022



Søren Vøgg Krabbe Lyster
<slyste20@student.aau.dk>

Chapter 1

Introduction

A state-of-the-art artificial reverberator will usually consist of multiple delay-lines with frequency attenuation and time-varying delay modulation, with the result being a complex set of parameters that needs to be estimated for a desired reverberation. This task of estimating parameters is a tedious task but might be enhanced by the use of a neural network. A neural network can be efficient at estimating a large set of parameters when given a suitable design and configuration. By creating training data by recording audio sources processed by a reverberator the neural network model should be able to utilize audio differentiation to tune the parameters of a custom reverberator plugin to recreate a target reverberation.

1.1 Thesis Concept

The goal of this thesis work is in two parts. One is to create a feedback delay network reverberation plugin in the VST3[1] format, using extended feedback delay network techniques. The second part is to create a neural network with an accompanying toolset, that can use the reverberation plugin in a differentiable digital signal processing approach to estimate a set of parameters that will allow the reverberation plugin to emulate other reverberators. Three commercially available black-box reverberators were chosen as the emulation targets for this work.

1.2 Background

1.2.1 Reverberation

In 1961 the first artificial reverberation was introduced by Schroeder and Logan [20] as a digital comb and allpass filter with a negative feedforward path. The

[1] An audio plugin framework that allow audio effects to be run in digital audio workstations.

following years Schroeder expanded on the design to increase the initially rather sparse echo density. In 1979 Moorer added [12] a lowpass filter inside the comb filter to attenuate higher frequencies to reduce the artificial metallic sound that earlier implementations suffered, and to approach a more natural sounding reverb. In 1991 Jot and Chaigne introduced the feedback delay network consisting of multiple delay-lines with different lengths with a feedback matrix [8]. This approach has a high control over the echo density and frequency dependent reverberation times with attenuating filters in the delay-lines and can in many cases be seen as state-of-the-art [25].

1.2.2 Neural Networks

The concept of neural networks began in the 1940's with the first ideas of brain-inspired computers, followed by the first successful perceptron in the 1950's [28]. With the rise in computational power and widely accessible toolsets for working with neural networks, the field of artificial intelligence, machine learning, and deep learning is constantly growing. In 2015 the TensorFlow library was released by Google's Google Brain team [1] with focus on deep learning. On top of TensorFlow is the Keras[2] library for python, an API for creating machine learning models. TensorFlow and Keras allow for quick design and implementation.

1.3 State-of-the-art

When discussing differential digital signal processing it is hard not to mention the Differential Digital Signal Processing (DDSP) library from the Google Magenta team [6]. DDSP uses audio differentiation and a multi-scale spectral loss to generate parameters driving audio synthesizers incorporated into the neural network. One application of the DDSP library is to transfer amplitude and pitch information from an input to various trained instruments, emulating timbre in what they call Timbre Transfer. The DDSP paper has spawned a lot of work based on the concept of differential digital signal processing. Various work have been focused on bringing more digital signal processing and audio generating concepts into the neural network domain (e.g., differentiable wavetable synthesis [21], differentiable IIR filters [9], and differentiable reverberation[10]), though this keeps the audio processing in the domain and is still not suitable for most audio and music workflows, e.g., working in digital audio workstations. We have recently seen the beginning integration of differential digital signal processing moving into these workflows with Mawf[3] and DDSP-VST[4], two plugins both working with timbre transfer. An-

[2] `https://github.com/keras-team/keras`

[3] `https://mawf.io/`

[4] `https://magenta.tensorflow.org/ddsp-vst`

other approach is to make the neural network compatible with a third-party audio processing environment. Ramírez et. al [15] introduce DeepAFx. DeepAFx is a library inspired by the differential digital signal processing done by DDSP, but with utilizing third-party LV2 plugins for Linux as their digital signal processing entity. They propose embedding the LV2 plugin calls in a multi-threaded neural network layer allowing them to train the network to change the parameters depending on the incoming audio. In three experiments they achieve tube amplifier emulation, removal of unwanted artifacts from vocals, and automatic music mastering using open-source plugins. Recent toolchains have made the integration between audio plugins and neural network environments easier. DawDreamer [4] is an extensive Python module that seeks to include many digital audio workstation roles in code that is callable from a neural network, with support for both VST plugins and the FAUST[5] programming language. The goal of DawDreamer is to enable intelligent music production with artificial intelligence. Pedalboard by Spotify's Audio Intelligence Lab [24] is another python library that offers simple support for the control of VST3 and Audio Unit format plugins. The Pedalboard library is directly compatible with TensorFlow.

1.4 Black-Box Reverberators

Three commercial black-box reverberators have been selected for this thesis work. These reverberators can be considered state-of-the-art in terms of reverberation and are highly accessible. These three reverberators will act as our targets for which to estimate parameters.

1.4.1 Ableton Reverb

The digital audio workstation Ableton Live comes with a large set of different audio processing effects. One of these effects is the Reverb unit. Even though many users opt out of using this in favor of third-party reverberators it is still a decent performing reverberation effect, and worth considering as a target for this thesis work. The Reverb unit has an expansive amount of parametric controls as seen in figure 1.1.

1.4.2 Hall Of Fame 2

The Hall Of Fame 2 is the second version of the award winning Hall Of Fame reverb by TC Electronic. This reverb comes in a hardware unit, and is designed for guitar players, even though it performs as well for other instruments. With the

[5] `https://faust.grame.fr/`

Figure 1.1: Ableton Reverb GUI

associated preset editor software there is access to a large amount of parameters. A sub-set of the parameter controls through the editor can be seen in figure 1.2.

1.4.3 Valhalla Room

ValhallaDSP is a rising star in the reverberation plugin scene. The Valhalla Room is one of their reverberation audio plugins with a selection of reverberation modes, and a high level of control over both the early reflections and the late reverberation. The Valhalla Room plugin can be seen in figure 1.3.

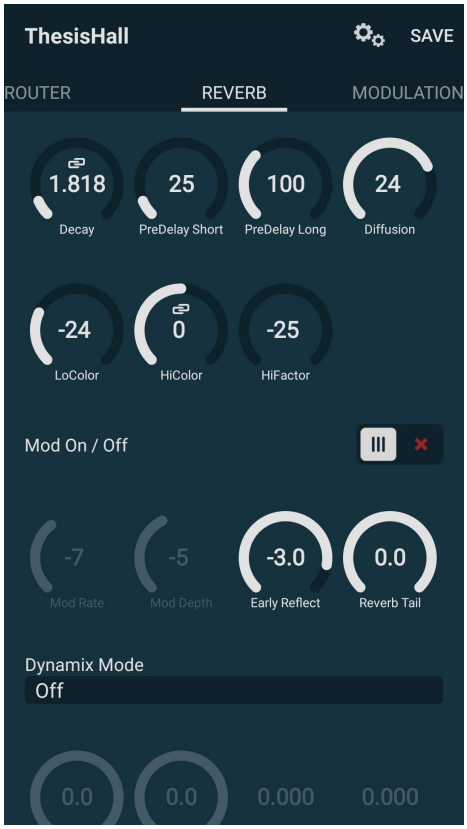


Figure 1.2: Hall Of Fame 2 GUI on Android device

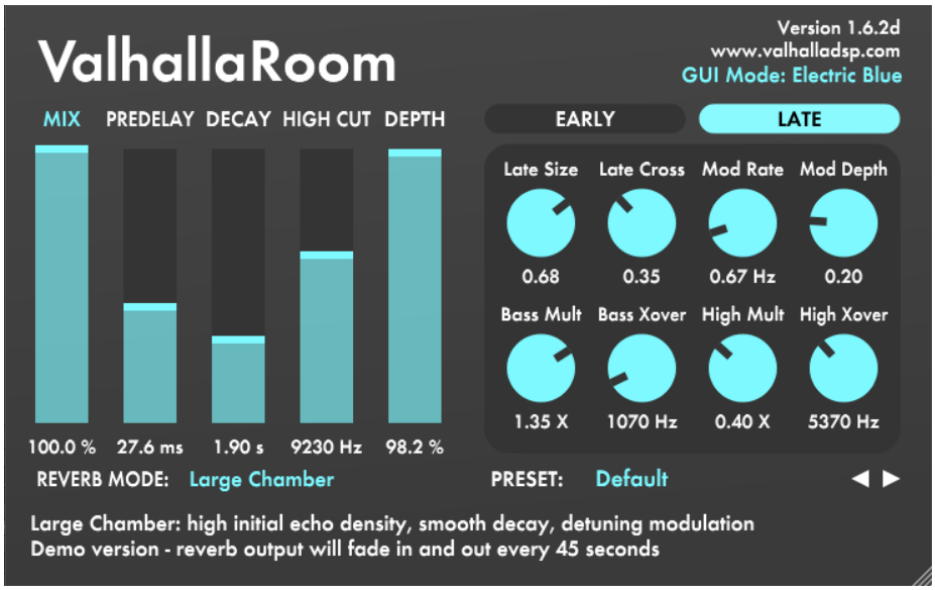


Figure 1.3: Valhalla Room GUI

Chapter 2

Feedback Delay Network

2.1 Architecture

The original feedback delay network (FDN) as proposed by Jot and Chaigne [8] consists of multiple parallel delay-lines with feedback scattered through an orthogonal feedback matrix.

The relationship between the input $x(n)$ and output $y(n)$ for N number of delay-lines can be described in the following two equations:

$$y(n) = \sum_{i=1}^{N} c_i s_i(n) + dx(n) \tag{2.1}$$

$$s_i(n + M_i) = \sum_{j=1}^{N} A_{i,j} s_j(n) + b_i x(n) \tag{2.2}$$

where s is the output of the delay-lines, M_i is the delay-line length at index i, b and c are the input and output gains, d is the dry signal gain, N is the number of delay-lines, and A is the orthogonal feedback matrix of size $N \times N$. For our design of the FDN algorithm we disregard the dry path signal $dx(n)$ in equation 2.1, as we chose to only focus on the fully reverberated signal in this project. The task of reintroducing the dry/wet mixing gain in later designs can be seen as trivial. Since the original FDN design was introduced in 1991 a number of extensions has been proposed for the design [25], and most commercial FDN based reverberators will have multiple of those. One important extension for perceptual natural sounding reverb is the frequency dependent attenuation. This attenuation is achieved by introducing attenuation filters in the delay-lines. If we consider a filter function h_i to be our attenuation filter at delay-line index i, we can update equation 2.2

$$s_i(n + M_i) = \sum_{j=1}^{N} A_{i,j} h_i s_j(n) + b_i x(n) \tag{2.3}$$

A FDN design with attenuation is shown for a single delay-line in figure 2.1. In some cases it is preferable to be able to change the spectral gain of the signal in addition to the frequency dependent attenuation in the delay-lines [22]. For this design a spectral coloration equalizer is added to the output of the FDN algorithm. Many state-of-the-art reverberators also include delay-line length modulation and pre-delay, so these are also added to the design. A diagram of the proposed FDN architecture can be found in figure 2.2.

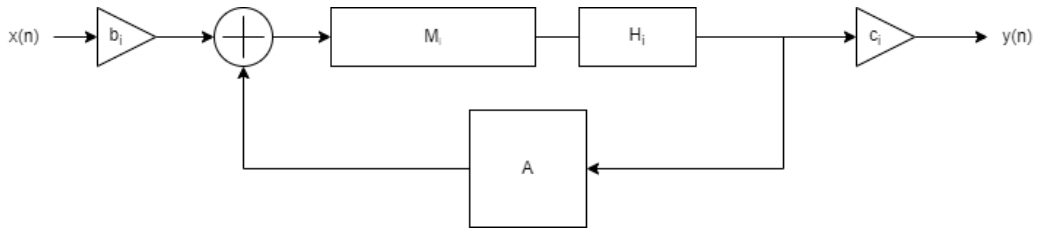


Figure 2.1: Single delay-line at index i with attenuation filter H

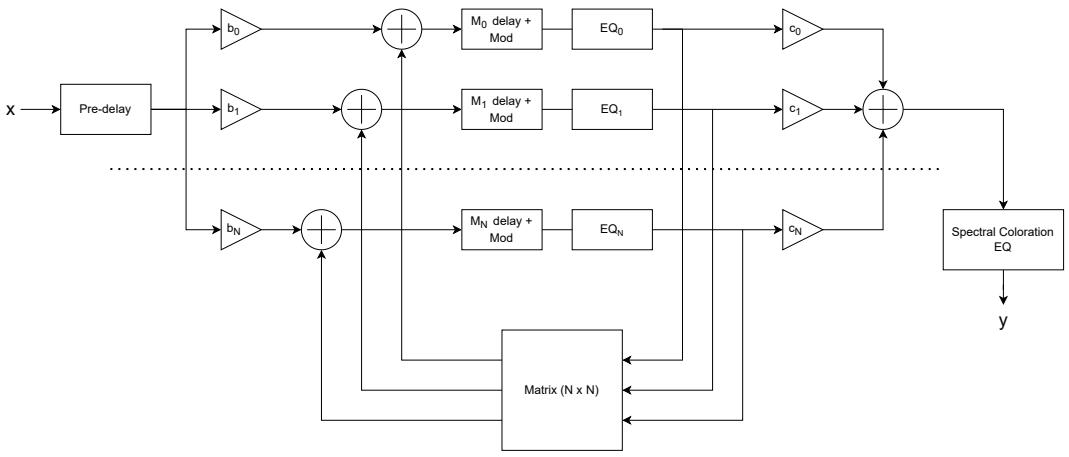


Figure 2.2: Diagram of the proposed FDN architecture for N number of delay-lines.

2.1.1 Lossless Prototype Design

When designing a FDN reverberator it is common to work with the concept of lossless prototypes [7]. The goal of the lossless prototype is to create a unitary feedback system, where there is no internal loss of power. With the introduction of attenuation filters we introduce a potential loss, but if the attenuation gain of all filters in the system is set to 0dB no loss should occur. The lossless prototype design approach is a great help when designing attenuation filters, since errors in the filter design will lead to perceptual attenuation, or in worst case exploding signals where the output of the system quickly rises towards infinity.

2.2 Delay-Line Lengths

The length of the individual delay-lines in a FDN reverberator can greatly influence the perception of reverb, where too high delay values can result in a low echo density, amplitude modulation, and unwelcome ringing [22]. To ensure correct ranged delay times we can look at the mode density of the reverberator. The mode density M can be described as the sum of the delay-line lengths M_i and poles P_i (if using attenuation) in the system

$$M = \sum_{i=1}^{N} M_i + P_i \tag{2.4}$$

Schroeder and Logan [20] notes that a mode density of 150ms per Hz is adequate for a reverberation time of 1 second, so the desired mode density can be described as

$$M \leq 0.15 t_{60} f_s, \tag{2.5}$$

where t_{60} is the reverberation time and f_s is the sampling rate. Though not used in the real-time implementation of the FDN reverberator, the observation of mode density can be beneficial in design and debugging.

Besides mode density it is also important to look at the delay-line length values in reference to each-other, since lengths containing common factors can produce echo cancellation or superposition [19]. The choice of delay-line lengths is often set to a fixed value in reverberator designs, but when the lengths need to be parameterized some considerations must be taken. Since mutually prime delay-line lengths are desired, the different lengths can be chosen as integer powers of primes, as proposed by J. O. Smith [22] and implemented in the Faust Programming Language. If we consider a minimum and maximum desired delay-line length, we can define a range of N delay-line lengths M_i as

$$M_i = L_{min} \left(\frac{L_{max}}{L_{min}} \right)^{i/N}, \tag{2.6}$$

where i is the index of the delay-line length, L_{min} is the minimum desired length, and L_{max} is the maximum desired length. To ensure that these delay-line lengths are mutually prime we calculate them by taking a integer power of a prime number

$$M_{prime,i} = p_i^{m_i}, \tag{2.7}$$

where $M_{prime,i}$ is the delay-line length at index i calculated to be coprime with other lengths, p_i is the prime number indexed at i in the range of naturally occurring primes $p_i \in \{2, 3, 5, 7, 11, 13, 17, ...\}$, and m_i being the integer power calculated from the initial delay-line length M_i in

$$m_i = 0.5 + \left\lfloor \frac{\log M_i}{\log p_i} \right\rfloor. \tag{2.8}$$

2.3 Graphic Equalizer

The expanded feedback delay network has frequency attenuation embedded in the delay-lines. Frequency attenuation can be done in multiple ways, so an objective of this problem can be defined as selecting a frequency attenuation implementation that works well in a neural network structure. An original expanded FDN structure might have simple bi-quad filter structures implemented, but estimating bi-quad filter coefficients leads to a classical machine learning gradient issue of exploding gradients due to their instability. Another approach to this is the state variable filter implementation used in [11], but this tuning of multiple filters in multiple delay-lines with codependent parameters was deemed too hard for the neural network when working with more than one delay-line, and the performance of the attenuation was highly dependent on the selected delay-line lengths. A third approach was investigated by looking at a graphic equalizer design. This design consists of a graphic equalizer bank for each delay-line. The graphic equalizers consist of 10 bandpass filters and a high shelf filter each. By calculating frequency dependent gain values from desired reverberation time values, and known sampling rate and delay-line lengths, this solution has been observed to be much easier for the neural network to interact with. This section describes the graphic equalizer design that is used with attenuation time values in the delay-lines and with dB gain values in the spectral coloration equalizer.

2.3.1 Interaction-Matrix-Based Design

One modern method of creating a graphic equalizer is by cascading second-order filters and solving a set of linear equations for the gains at the center frequency of each filter [26, 27]. The overall frequency response of this graphic EQ design is the product of the filter response for each filter

$$H_C(e^{j\omega T_s}) = G_0 \prod_{m=1}^{M} H_m(e^{j\omega T_s}) \tag{2.9}$$

where M is the number of filters in the graphic EQ, G_0 is the overall gain factor, $H_m(e^{j\omega T_s})$ is the frequency response of filter m, ω is the radial frequency, and T_s is $1/f_s$ with f_s being the sample rate. The amplitude response of $H_C(e^{j\omega T_s})$ can be described as the sum of the amplitude response of each filter

$$A_C(e^{j\omega T_s}) = g_0 + \sum_{m=1}^{M} A_m(e^{j\omega T_s}) \tag{2.10}$$

where $g_0 = 20\log(G_0)$ and $A_m(e^{j\omega T_s}) = 20\log(|H_m(e^{j\omega T_s})|)$ An interaction matrix $\mathbf{A}$ of size $M \times M$ is created to store the normalized amplitude response of all the M filters at M center frequencies

$$\mathbf{A}_{k,m} = A_m(e^{j\omega_k T_s})/g_p \tag{2.11}$$

where m is the filter index, k is the center frequency index, and $g_p = 20\log(G_p)$ is an initial prototype gain for all the filters. An approximation to the gain in dB at each center frequency $\hat{\mathbf{t}}$ can be defined as

$$\hat{\mathbf{t}} = \mathbf{A}\mathbf{g} \tag{2.12}$$

where $\mathbf{g}$ is a vector containing the gain for each filter in dB. Since $\mathbf{A}$ is square and invertible a solution can be found using matrix inversion

$$\mathbf{g}_{opt} = \mathbf{A}^{-1}\mathbf{t} \tag{2.13}$$

where $\mathbf{t}$ is a vector containing the desired gains for each filter, and $\mathbf{g}_{opt}$ is a vector with the optimum gains to reach the target amplitude response of each filter.

Välimäki and Liski proposes an improved method to minimize the amplitude errors that could arise in the cascading graphic equalizer: By adding extra frequency points between the command frequencies and updating the interaction matrix with these new values [26]. The result is a slightly more complex least-squares problem with less potential amplitude error. The updated EQ design has 19 frequency points instead of 10, where the new 9 points are chosen to lie in between the command frequencies. The resulting new interaction matrix $\mathbf{B}$ is now of size $2M-1 \times M$

$$\mathbf{B}_{k,m} = A_m(e^{j\omega_k T_s})/g_p \tag{2.14}$$

where m is the filter index, k is the index of the expanded frequency vector, and g_p still being the initial prototype gain. Since the interaction matrix now is non-square the least-square solution is as follows

$$\mathbf{g} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{t}_1 \tag{2.15}$$

where $\mathbf{t}_1$ is a vector of $2M-1$ elements with the target command gains at odd indices and the linear interpolated values at even indices. To further increase the precision of the resulting gain values a second iteration is done. This time the new interaction matrix is defined as

$$\mathbf{B}_{1,k,m} = A_{1,m}(e^{j\omega_k T_s})/g_m, \tag{2.16}$$

where $A_{1,m}$ is the new amplitude responses with the gain value g_m is the $\mathbf{g}$ from equation 2.15 indexed at each filter m. The least-square solution can then be

$$\mathbf{g}_1 = (\mathbf{B}_1^T\mathbf{B}_1)^{-1}\mathbf{B}_1^T\mathbf{t}_1, \tag{2.17}$$

resulting in a new command gain vector $\mathbf{g}_1$ with an error rate of $<$ 1dB. When designing the interaction matrices, the amplitudes are derived using this following second-order filter equation:

$$H(z) = \frac{1 + G\beta - 2\cos(\omega_c)z^{-1} + (1 - G\beta)z^{-2}}{1 + \beta - 2\cos(\omega_c)z^{-1} + (1 - \beta)z^{-2}} \tag{2.18}$$

where G is the gain at peak, G_B is the gain at the bandwidth B, ω_c is the center frequency, and

$$\beta = \begin{cases} \tan(B/2), & \text{when } G = 1 \\ \sqrt{\frac{|G_B^2 - 1|}{|G^2 - G_B^2|}} \tan(\frac{B}{2}), & \text{otherwise} \end{cases} \tag{2.19}$$

2.3.2 Attenuation Time

To allow for accurate control of the reverberation time T_{60} at the defined control frequencies we must calculate the target magnitude response A_{dB} for the graphic equalizer [13, 17]. First we can calculate the attenuation-per-sample in dB at ω:

$$\gamma_{dB}(\omega) = -60\frac{1}{f_s T_{60}(\omega)}, \tag{2.20}$$

where $\omega = 2\pi f / f_s$ is the normalized angular frequency, f is the selected frequency in Hz, and f_s is the sampling rate in Hz. To calculate the target magnitude response from the attenuation-per-sample value we need to know the length of the delay-line in samples L. Since the lengths of the delay-lines in the FDN are different, we need to calculate A_{dB} for each delay-line M:

$$A_{dB,M}(\omega) = L_M \gamma_{dB}(\omega) \tag{2.21}$$

The desired magnitude response can now be used to calculate the optimum gains for each filter in the graphic equalizer using the method in 2.3.1.

2.3.3 High Shelf Filter

As suggested in [27] and implemented in [14] a high shelf filter is added in succession to the other 10 bandpass filters. The shelf filter is a first order shelf filter and works at the same center frequency as the highest bandpass filter frequency. The purpose of this filter is to attenuate broadly in the highest spectral part of the signal, as most natural sounding reverberation attenuates the signal the most in that frequency range. The transfer function for the high shelf filter H_{HS} is as follows:

$$H_{HS}(z) = \frac{\sqrt{G}\tan(\omega_c/2) + G + [\sqrt{G}\tan(\omega_c/2) - G]z^{-1}}{\sqrt{G}\tan(\omega_c/2) + 1 + [\sqrt{G}\tan(\omega_c/2) - 1]z^{-1}}, \tag{2.22}$$

where G is the gain at the center frequency and ω_c is the normalized center frequency.

2.4 Feedback Matrices

The choice of feedback matrix should ensure the goal of a lossless prototype design, where an input impulse will grow to a smooth noise signal [7]. A feedback matrix can fulfill being lossless if it is a unitary matrix [8], that is when the matrix $\mathbf{A}$ conjugate $\mathbf{A}^*$ is equal to its inverse $\mathbf{A}^{-1}$:

$$\mathbf{A}\mathbf{A}^* = \mathbf{A}\mathbf{A}^{-1} \tag{2.23}$$

Two methods of generating feedback matrices that adheres to this principle are selected for our FDN design: the Householder Matrix and the Hadamard Matrix.

2.4.1 Householder Matrix

The Householder matrix A_N of size $N \times N$ is constructed using a $N \times N$ permutation matrix and a $N \times 1$ column vector of 1's as follows:

$$\mathbf{A}_N = \mathbf{J}_N - \frac{2}{N}\mathbf{u}_N\mathbf{u}_N^T \tag{2.24}$$

where $\mathbf{u}_N^T$ denotes the transposed column vector. In our case the permutation matrix $\mathbf{J}_N$ is the identity matrix $\mathbf{I}_N$. Jot found that the Householder matrix transformation worked best for feedback delay networks with four delay-lines. For systems with 16 delay-lines he proposed a 16×16 matrix $\mathbf{A}_{16}$ with embedded $\mathbf{A}_4$ matrices [22]:

$$\mathbf{A}_{16} = \frac{1}{2}\begin{bmatrix} \mathbf{A}_4 & -\mathbf{A}_4 & -\mathbf{A}_4 & -\mathbf{A}_4 \\ -\mathbf{A}_4 & \mathbf{A}_4 & -\mathbf{A}_4 & -\mathbf{A}_4 \\ -\mathbf{A}_4 & -\mathbf{A}_4 & \mathbf{A}_4 & -\mathbf{A}_4 \\ -\mathbf{A}_4 & -\mathbf{A}_4 & -\mathbf{A}_4 & \mathbf{A}_4 \end{bmatrix} \tag{2.25}$$

2.4.2 Hadamard Matrix

Another commonly used unitary feedback matrix is the Hadamard matrix [22]. The 2×2 Hadamard matrix is defined as:

$$\mathbf{A}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{2.26}$$

where higher order matrices can be created with recursive embedding matrices

$$\mathbf{A}_4 = \frac{1}{\sqrt{2}}\begin{bmatrix} \mathbf{A}_2 & \mathbf{A}_2 \\ \mathbf{A}_2 & -\mathbf{A}_2 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{2.27}$$

2.4.3 Matrix Difference

The difference that the two feedback matrices has on the echo density growth has in the feedback delay network can be subtle but is apparent when the echo density growth is inspected. Figure 2.3 has the echo density plotted for both matrix types in a lossless prototype.

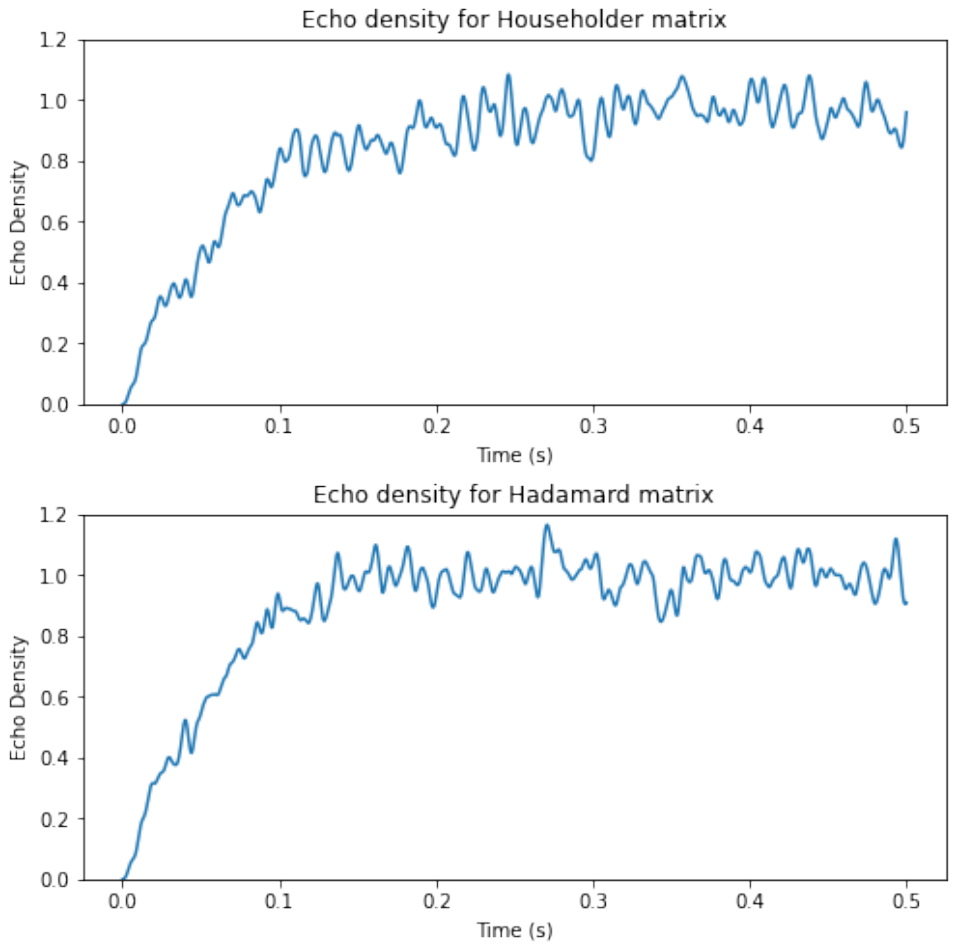


Figure 2.3: Echo density growth over 0.5 seconds plotted for both matrix types in a lossless prototype.

2.5 Time Varying Delays

To emulate the time invariances in natural reverberation caused by, e.g., temperature fluctuations [3] we can make use of delay-line length modulations. A low frequency oscillator can be used to vary the delay-line lengths in real time, so that

a modulated delay-line length L_{mod} could be defined as

$$L_{mod} = L + v(t), \tag{2.28}$$

where L is the initially fixed delay length and $v(t)$ is the oscillating delay-line variation.

2.6 Pre-Delay

Pre-delay is implemented using a simple delay-line delaying the incoming signal before it hits the FDN algorithm, as it exists in most commercial reverberators. The pre-delay can be seen context with the FDN algorithm in figure 2.2.

Chapter 3

Neural Network

The task of the neural network is to converge towards a set of parameters using audio differentiation. With our approach we need to be able to embed a VST3 plugin within the neural network model. The neural network is designed to take a dry signal as input, and a wet signal as the target signal on which to do audio differentiation. This chapter describes the various parts of the proposed neural network model structure, along with a proposed novel combination of losses.

3.1 Parameter Model

The parameter model is a sub-model in the neural network reverberation estimation model. This model takes an audio signal input and returns a set of parameters. Since we want to encode a large number of samples into a relatively small number of parameters, we need to reduce our input audio into smaller features. This is done by converting the audio into logarithmic Mel-spectrogram frames. The resulting frames are then encoded through a 2D convolutional network. The design of the convolutional network has been found empirically by iterative testing and training.

3.1.1 Mel-Spectrogram

By converting the input audio to Mel-Spectrogram frames we reduce the raw input audio data to a number of two-dimensional frames for the 2D convolutional network. Since our parameter model is a problem of compressing the input audio into a neural network with weights and biases that results in an output of parameters, we do not need to be too concerned about the way we convert the input audio into frames. With this reasoning we keep the Mel-Spectrogram implementation used in earlier work by the author [11].

3.1.2 Convolutional Network

Initially we used the MobileNetV2 [16] for encoding in the parameter model, since this encoder has been used in other implementations [11, 15]. But during the development of the neural network we found that a simplified 2D convolutional network performed better for this case. This model was inspired by another encoder model used previously in the DeepAFx implementation [15]. The 2D convolutional model consists of 4 repeats of a configuration of Keras layers: Conv2D $\rightarrow$ MaxPooling2D $\rightarrow$ BatchNormalization. The output is then flattened and reduced into a dense layer, followed by a drop-out layer. A final dense layer is added with a number of output-neurons consistent with the amount of parameters to estimate. The flow of the entire parameter model can be seen in figure 3.1.

3.2 Audio Processing Model

The Audio Processing model is a single-layer model where we embed our FDN plugin in the neural network. The task of this model is to generate an audio signal from an input signal processed through the reverberator with a given parameter set. The functionality of the model can simply be described as

$$y = f(x, \theta), \tag{3.1}$$

where $f()$ is the VST3 processing function, x is the input signal, θ is the set of VST parameters, and y is the output signal.

3.2.1 Gradient Approximation Method

Backpropagation through the model requires that all layers can calculate and/or propagate gradients. Gradients are a part of the off-the-shelf network layers in Keras and TensorFlow, but our introduction of the custom audio processing layer has no such functionality provided, so a custom gradient implementation is required. As the input to the custom audio processing layer consist of an audio signal and a parameter vector, gradients must be calculated for both to enable backpropagation. Looking at the audio input graph through the system we can observe that no weights need to be updated, so we can disregard calculating gradients for this path and just return a gradient vector of same size as the input audio signal filled with zeros. The parameter vector is where we need to estimate the gradients. The problem of estimating gradients for backpropagation for the parameter vector is a problem of estimating the individual parameters gradient. Since we disregard backpropagation with regards to the input signal x in equation 3.1 we only need to calculate the gradients for θ. If we consider our current value of θ as $\hat{\theta}_0$ then the gradient can be defined as the change in $f(x, \hat{\theta}_0)$. As we can ignore the gradients

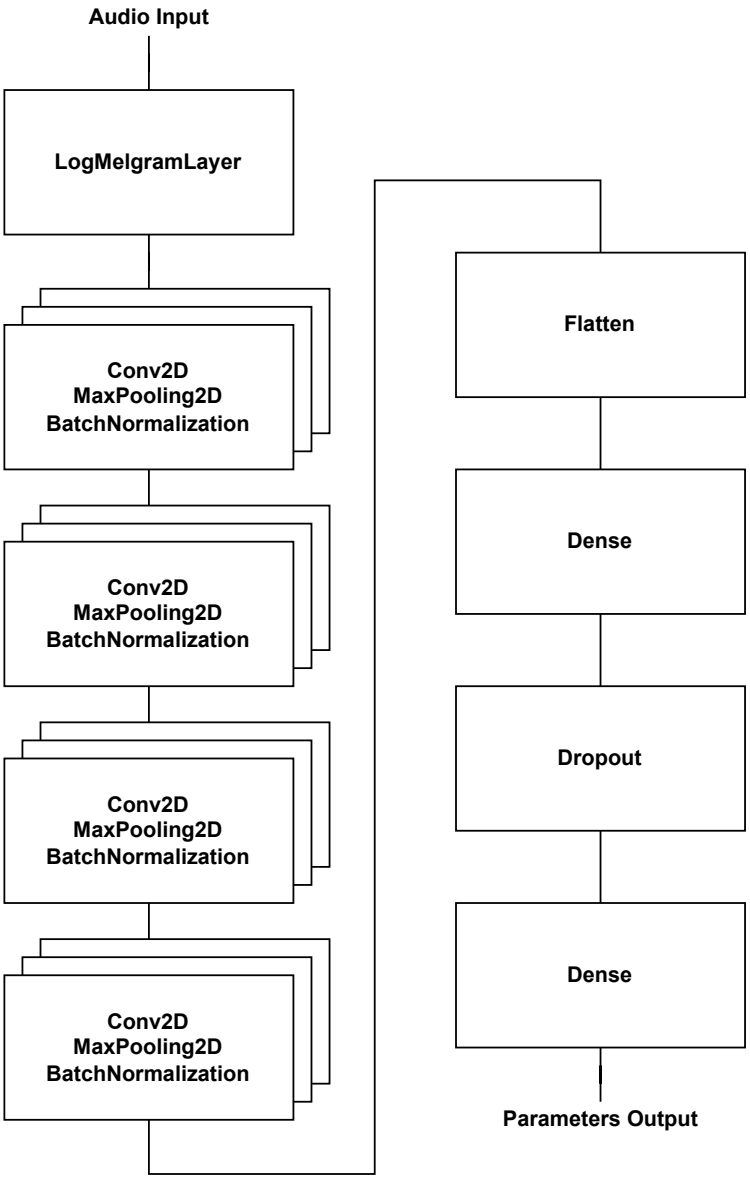


Figure 3.1: The layers of the parameter model.

for x we can define the gradient for a single parameter at index i as

$$\tilde{\nabla} f(\hat{\theta}_0)_i = \frac{\delta f(\hat{\theta}_0)}{\delta \theta_i}, \tag{3.2}$$

where $\tilde{\nabla} f(\hat{\theta}_0)$ is our notation for the gradient at index i, and $\delta f(\hat{\theta}_0)_i / \delta \theta_i$ is the change in output signal at a change in the parameter θ_i. Calculating the gradient for each parameter can be a time-consuming task as we would need to process the input signal x through the audio processor twice for each parameter to get the gradients for all of θ. As we also need one forward pass for each call to this model the amount of processing calls would be $2 \times P + 1$ calls, where P is the amount of parameters in θ. The chosen solution to this problem is an approach called Simultaneous Perturbation Stochastic Approximation.

Simultaneous Perturbation Stochastic Approximation

Simultaneous Perturbation Stochastic Approximation (SPSA) is an optimization method that can be used when estimating gradients in a neural network with high dimensional problems, e.g., a large set of parameters [23]. Estimating the gradient for a parameter $\hat{\theta}_i$ with SPSA can be written as

$$\tilde{\nabla}^{SPSA} f(\hat{\theta}_0)_i = \frac{f(\hat{\theta}_0 + \epsilon \hat{\Delta}^P) - f(\hat{\theta}_0 - \epsilon \hat{\Delta}^P)}{2\epsilon \Delta_i^P}, \tag{3.3}$$

where $\hat{\Delta}^P$ is a vector of perturbations the size of the number of parameters P containing a binomial distribution of -1 and 1, and with ϵ being a small positive number. Using SPSA requires only $2 + 1$ process runs through the VST3 to estimate the gradients for all parameters and do the forward pass. We can therefor utilize SPSA to greatly increase the speed of our neural network.

3.3 Full Model

The full neural network model consists of the parameter model followed by the audio processing model. A diagram of the full model can be seen in figure 3.2. The summary of the parameter model and the full model can be seen in appendix B.

3.4 Losses

For the audio differentiation in our neural network we need a loss value that describes how the signal generated by the network differs from the target signal. The loss calculated should result in meaningful and minimizable values

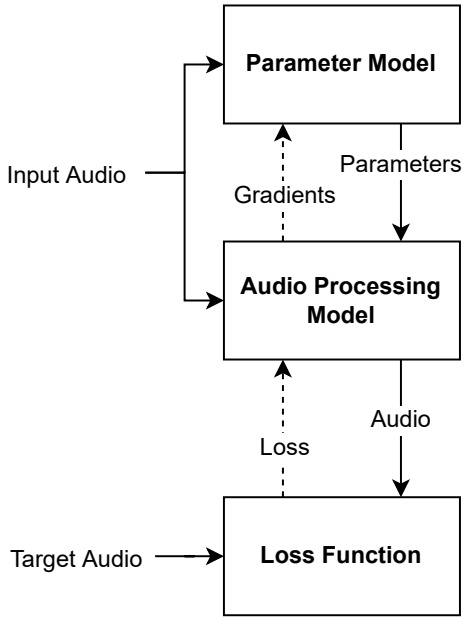


Figure 3.2: A diagram of the full neural network model.

for the backpropagation. There are various different approaches to calculating a loss when working with audio differentiation, with some simply might using mean-absolute-error or mean-squared-error, others look at the spectral components utilizing multi-scale short-time Fourier transformations [6, 15]. The issue with sample-wise losses in mean-absolute-error and mean-squared-error is that various waveforms might be similar but differ perceptually and vice versa [6].

When calculating losses for reverberation where the target signal originates from black-box algorithms mean-absolute-error and multi-scale spectral losses seemed to lack information about key temporal properties. To help with this problem a hybrid-loss approach was investigated with a combination of multi-scale spectral loss, envelope loss, and echo-density loss.

3.4.1 Multi-Scale Spectral Loss

The Differential Digital Signal Processing library from Google Magenta introduced a multi-scale spectral loss approach that compares the magnitude spectrogram of target and synthesized audio at multiple different FFT sizes. This was done to cover the differences at different spatial-temporal resolutions [6]. We can define the partial loss value L_i with an FFT size at index i as the sum of the L1 difference between S and $\hat{S}$ and between $\log S$ and $log\hat{S}$

$$L_i = ||S_i - \hat{S}_i||_1 + ||\log S_i - \log \hat{S}_i||_1, \tag{3.4}$$

where S is the STFT of the target signal, and $\hat{S}$ is the STFT of the signal generated by the neural network. If we do this for a range of N FFT sizes our multi-scale spectral loss will be

$$L_{spectral} = \sum_{i=1}^{N} L_i \tag{3.5}$$

For this project the range of FFT sizes are $\{2048, 1024, 512, 256, 128, 64\}$. The magnitude spectrogram for three different FFT sizes can be seen plotted for the Ableton Reverb impulse response in figure 3.3.

3.4.2 Envelope Loss

Sample-by-sample comparison such as mean-absolute-error and mean-square-error can be difficult to utilize in this black-box approach. An envelope loss function is created to give the model some information about the envelope of the reverberated signal. Since a strong perceptual component of reverberation is decay time [5] this should provide the network with an informed loss component based on reverberation. The envelope of a signal is in this case calculated by splitting the signal into multiple frames and finding the maximum value of each frame. The smoothness of the envelope is dependent on the frame size used. We found that a good frame size for representing a useful signal envelope was 4096 samples at 48kHz. The envelope of the three target case impulse responses can be seen in figure 3.4. The resulting loss is the L1 difference between the envelope of the target signal A and the envelope of the signal generated by the neural network $\hat{A}$

$$L_{env} = ||A - \hat{A}||_1 \tag{3.6}$$

3.4.3 Echo Density Loss

Echo Density Loss is a proposed novel loss function used for estimating the delay-line parameters of the FDN algorithm. The loss function is implemented with a echo density measure algorithm proposed by Abel and Huang [2]. This algorithm is using a weighted sliding window function to estimate the amount of taps that lie outside a standard deviation. The algorithm is as follows:

$$\eta(t) = \frac{1}{\text{erfc}(1/\sqrt{2})} \sum_{\tau=t-\delta}^{t+\delta} w(\tau)\mathbf{1}\{|h(\tau)| > \sigma\} \tag{3.7}$$

where σ is

$$\sigma = \left[\sum_{\tau=t-\delta}^{t+\delta} w(\tau)h^2(\tau) \right]^{\frac{1}{2}}, \tag{3.8}$$

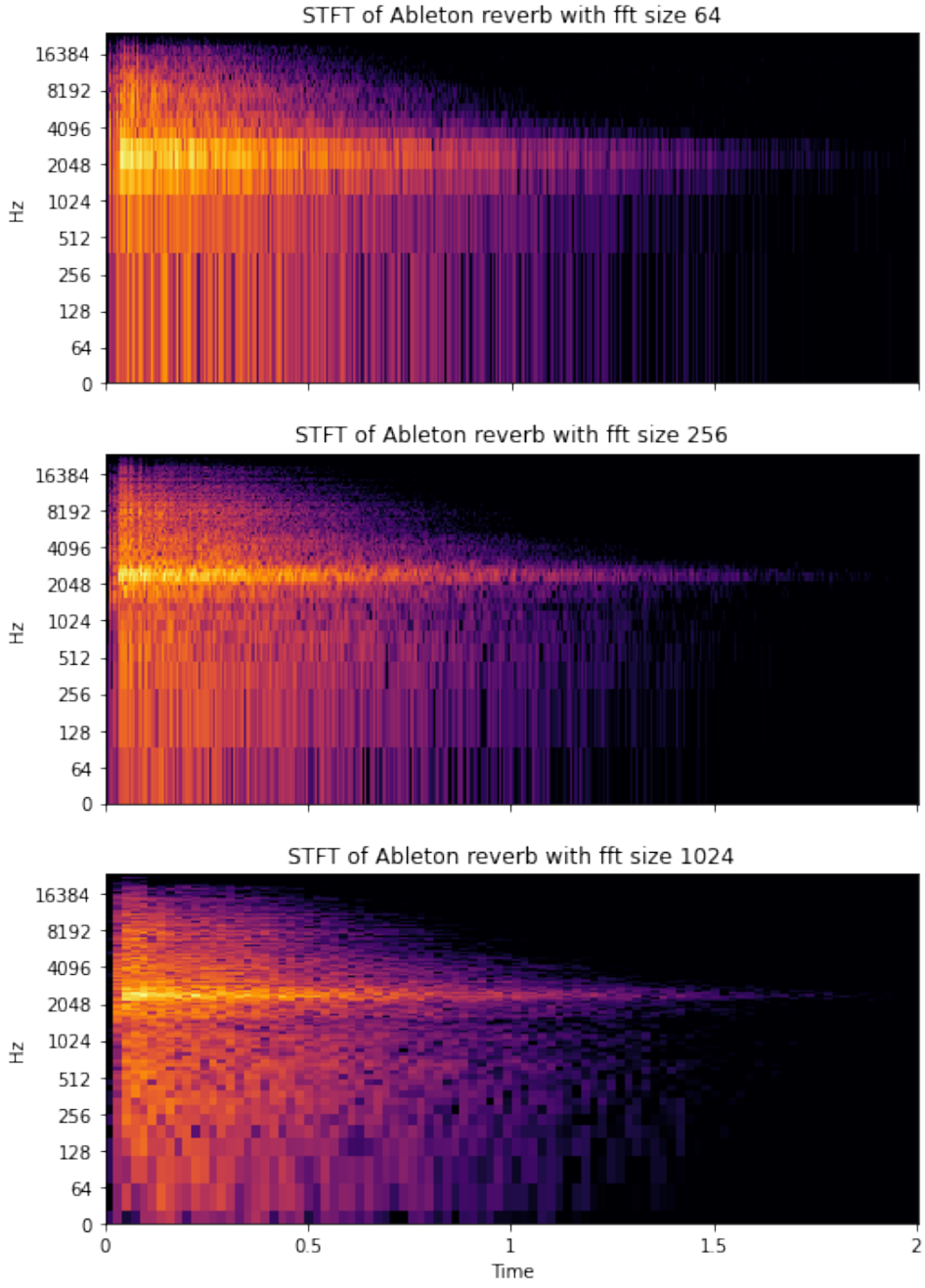


Figure 3.3: STFTs of Ableton Reverb response at three different FFT sizes.

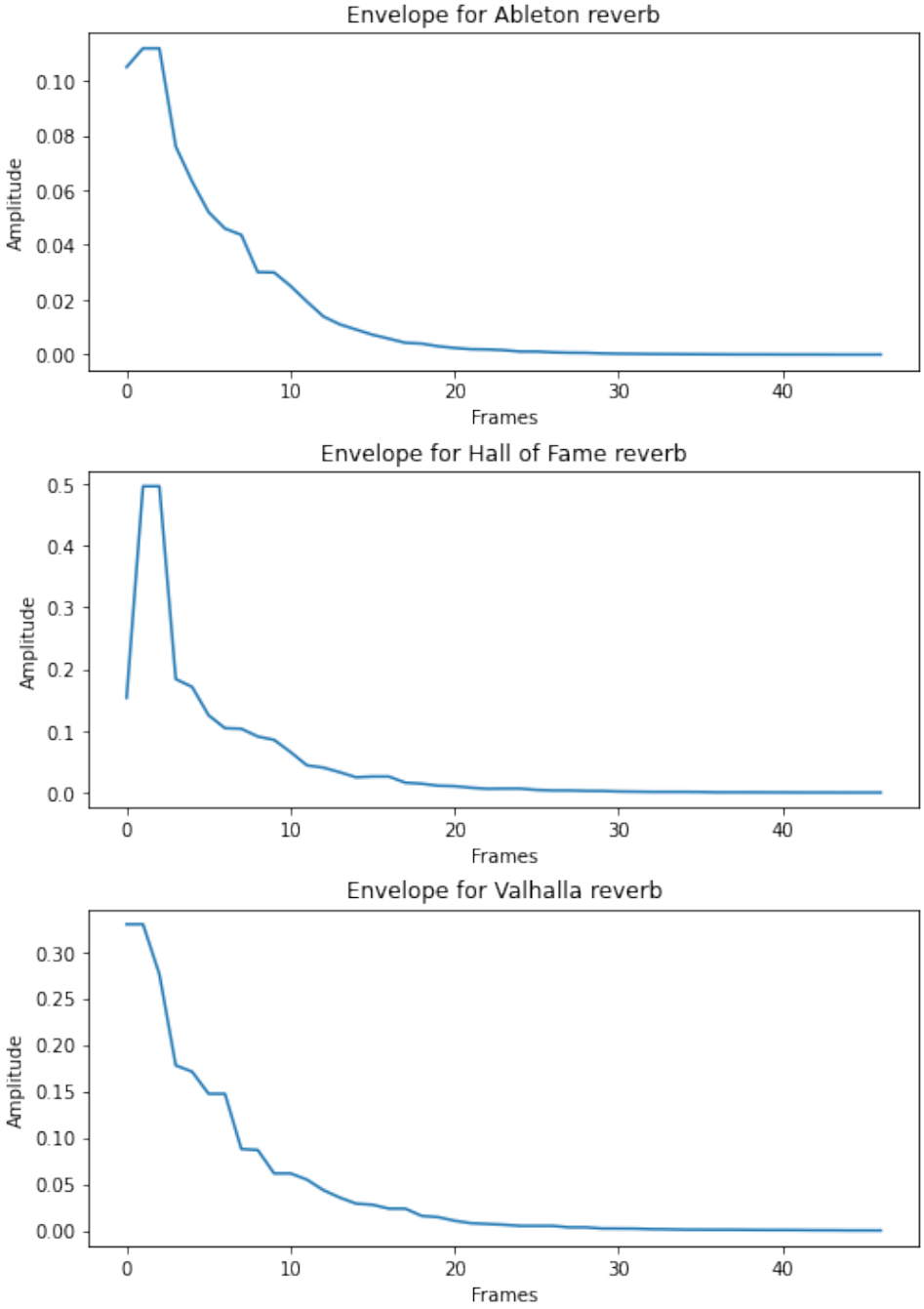


Figure 3.4: Envelope for three different recorded reverb impulses.

the function $w(\tau)$ is a weighted Hanning window, the function $h(\tau)$ is the audio signal, $\text{erfc}(1/\sqrt{2})$ is the complementary error function describing the expected fraction of samples outside the standard deviation of a Gaussian distribution, and $\mathbf{1}\{|h(\tau)| > \sigma\}$ is the indicator function

$$\begin{cases} 1 & \text{if } |h(\tau)| > \sigma \\ 0 & \text{if } |h(\tau)| \leq \sigma \end{cases} \tag{3.9}$$

The loss value is then defined as the L1 difference in echo density for the target signal σ and the signal generated by the neural network $\hat{\sigma}$:

$$L_{echo} = ||\eta - \hat{\eta}||_1 \tag{3.10}$$

The algorithm has been implemented in TensorFlow and optimized for TensorFlow operations. The three figures in figure 3.5 show the echo density for the first 0.25 seconds of the impulse responses of the three black-box reverberators. The light blue dashed lines are the echo densities with the early reflections ignored. A boolean parameter has been implemented in our echo density loss function to allow the removal of what is assumed as being early reflections. Since our FDN design does not include early reflections, we need to ignore these to reduce our echo density loss. Early reflections often exist in the first 100ms of an artificial reverberated signal, before the late reverberation takes over [22]. If we inspect the echo density of the Hall Of Fame 2 and Valhalla impulse responses in figure 3.5, we can see an initial relatively high echo density in the first ~ 60 and ~ 40 milliseconds respectively, with a following steady growth of echo density from a near-zero. Since we do not know the internal structure of the black-box reverberators we can make the assumption that this initial region in the echo density is early reflections. The early reflections are removed from the echo density graph by zeroing echo density values before the near-zero point.

3.4.4 Total Loss

With the different loss function contributions the total loss value L_{total} is then defined as

$$L_{total} = \alpha L_{spectral} + \beta L_{env} + \gamma L_{echo}, \tag{3.11}$$

where α, β, and γ is the weights of the individual losses. A good weight distribution has been found empirically by testing the neural network, with $\alpha = 1.0$, $\beta = 50.0$, and $\gamma = 1.0$.

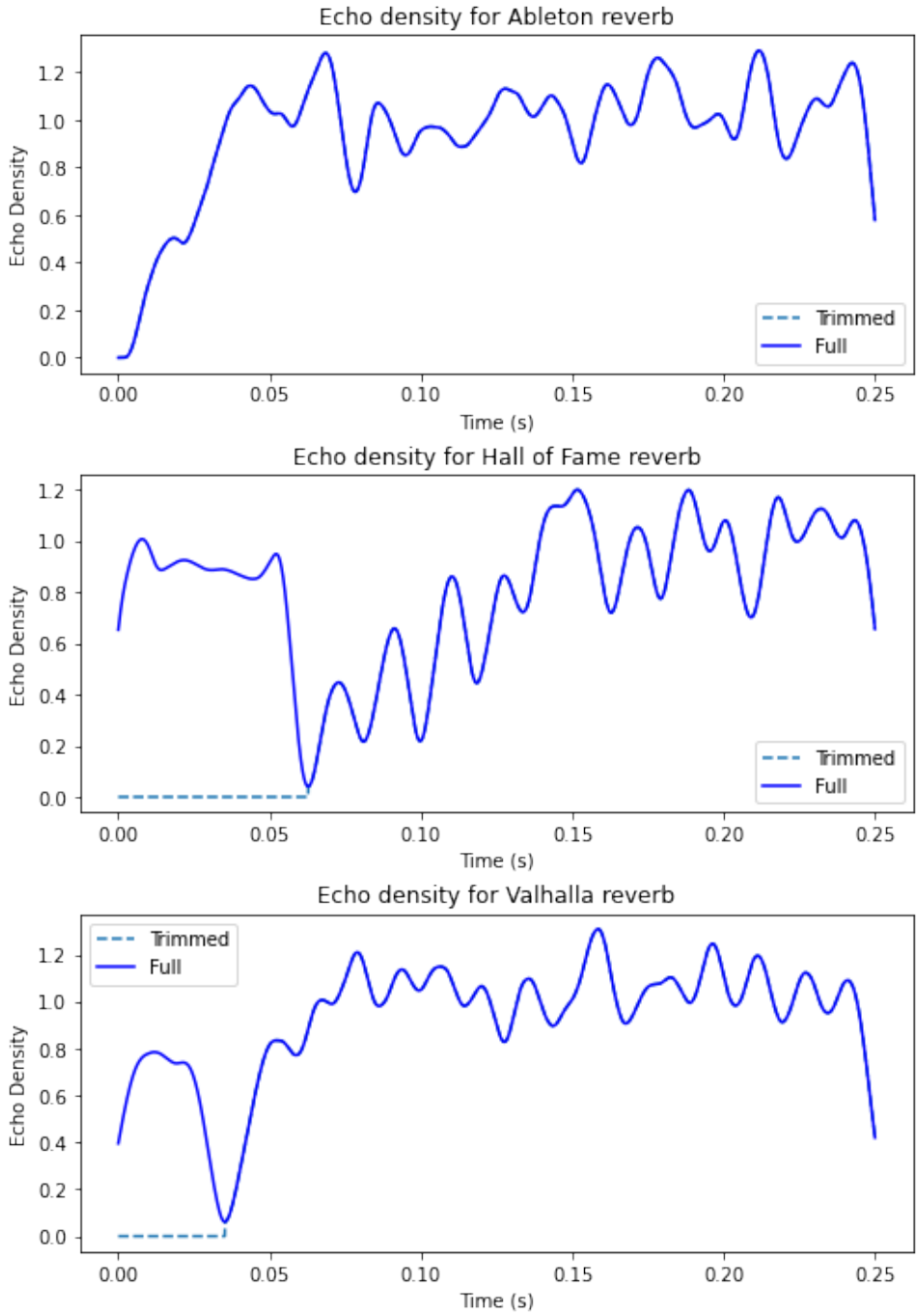


Figure 3.5: Echo density of three reverberated signals.

3.5 Optimizer

As the optimizer in the neural network we use the ADAM optimizer. The ADAM optimizer is an advanced optimizer using gradient descend, and is used in many modern neural networks as well as both the DDSP and DeepAFx implementations [6, 15].

Chapter 4

Implementation

In this chapter we will discuss the various methods and considerations that has been in play when implementing the various parts of this project. The code used for the FDN plugin can be found at `https://github.com/VoggLyster/Reverberator/tree/thesis`. The code used for the neural network can be found at `https://github.com/VoggLyster/ReverberatorEstimator/tree/thesis`.

4.1 VST3 in JUCE

The proposed extended FDN system has been implemented in C++ using the JUCE application framework[1]. JUCE is an open-source application framework that includes a large tool-set for digital signal processing and audio plugin development, as well as GUI and VST3 support. JUCE also includes a build management tool called the Projucer that can create and manage build solutions for multiple platforms. This tool has been used to manage Makefiles for Linux compilation and Visual Studio Solutions for Windows compilation, allowing for easy cross-platform build management. Throughout the implementation and testing it has been beneficial to have different number-of-delay-lines configurations of the plugin, and different configurations with different extended FDN features left out. Instead of having a fragmented codebase and cluttered version control the different configurations were made dynamically available in the implementation using C++ preprocessor directives.

Continuous Delivery

During the development of this project continuous delivery has been managed and maintained using Github Actions workflows[2]. Using the different configura-

[1] `https://juce.com/`

[2] A CI/CD tool available on Github, `https://github.com/features/actions`

tions available via preprocessor definitions multiple VST3 plugins could be compiled continuously in a virtual machine environment when code changes were deployed to version control. The newest version of the FDN plugin can be found at `https://github.com/VoggLyster/Reverberator/releases/latest`, with the current version at writing this thesis being 0.1.61.

4.1.1 GUI

A simple GUI has been implemented. This has been done to make testing easier but does not have any effect on the implementation in the neural network. The GUI has been programmed to dynamically add parameters to conform to the different number of available delay-line builds. The top of section of the GUI for a 16 delay-line configuration can be seen in figure 4.1.

4.1.2 Sampling Rates

The FDN plugin was originally implemented with compatibility for a wide range of sampling rates, but the graphical equalizer design is not yet optimized for lower sampling rates, as it quickly becomes unstable, as shown in figure 4.2.

4.1.3 Parameters

To be easily compatible with the neural network architecture, all parameters are normalized to the range $[0, 100]$. In the reverberator plugin these parameter values are mapped linearly into their desired ranges through these various mappings:

- Delay-line minimum length is in the range $[20, 30]$ meters.
- Delay-line maximum length is $[20, 40]$ meters, offset by the delay line minimum length.
- Pre-delay is in the range $[0, 40]$ ms.
- Reverberation time parameters for the delay-line attenuation graphic equalizer are in the range $[0.3, 2.43]$ seconds.
- Gain values for the spectral coloration graphic equalizer are $[-57, 3]$ dB.
- Gain values for the delay-line input gains are $[-12, 0]$ dB.
- Gain values for the delay-line output gains are $[-12, 0]$ dB.
- Delay-line length modulation frequency $[0, 3]$ Hz.
- Delay-line length modulation depth is $[0, 10]$ samples.

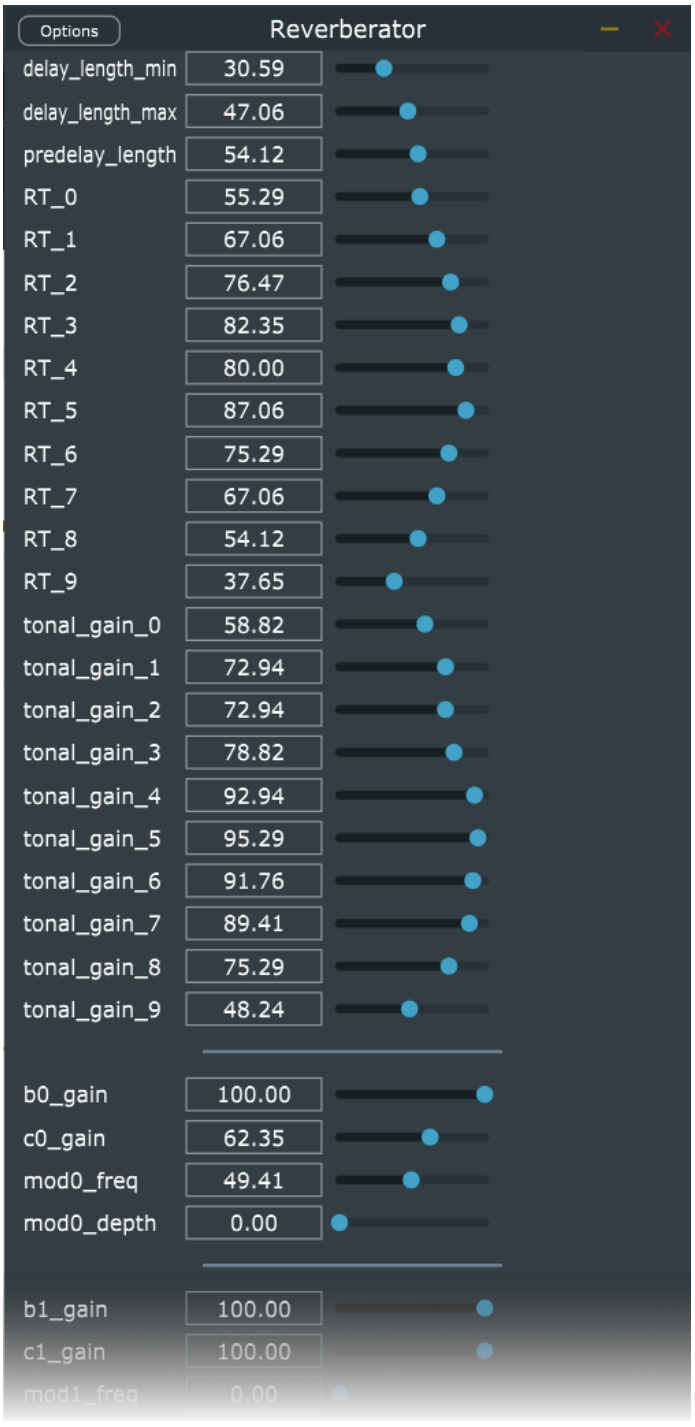


Figure 4.1: Cutout of the dynamic GUI.

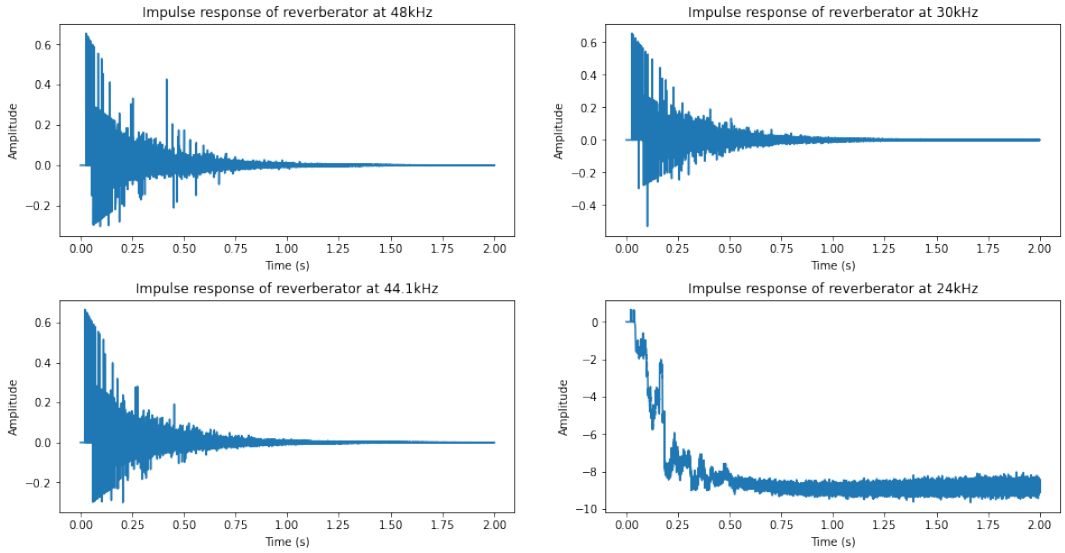


Figure 4.2: Impulse response of the reverberator plugin at different sampling rates, with filter attenuation.

The linear mapping is done as following

$$p = \frac{p_{in}}{100} * (p_{max} - p_{min}) + p_{min} \tag{4.1}$$

An additional parameter has been added for selecting the feedback matrix type. This parameter does not conform with the mapping of the other parameters, since it performs as boolean. Lastly a bypass parameter is available through the default JUCE audio plugin framework - this parameter is not used.

4.1.4 Compatibility

Embedding the VST plugin in a neural network requires some considerations regarding compatibility. The plugin needs to be able to run headless since the process is unlikely to run in an environment with display access. This has been an issue in earlier work [11] that could be fixed by utilizing the headless support included in the JUCE framework versions 6.0+.

4.1.5 Internal State

It is important for the functionality of the neural network that each process call resets the internal state of the plugin. The internal state of the plugin includes the state-values of the second order filters cascaded in the graphic equalizers and the content of the delay-lines. The phase of the oscillators used for delay-line length modulation should also be reset.

4.1.6 Lossless Prototype

During the implementation we still want to adhere to the lossless prototype design concept mentioned in 2.1.1. By setting the control gain values of all filters to 0dB and applying a single one-sample impulse to the input of the reverberator, we should be able to observe an output that settles on a steady noise-like signal. The response of a one-sample impulse can be seen in figure 4.3 for both unitary matrix designs.

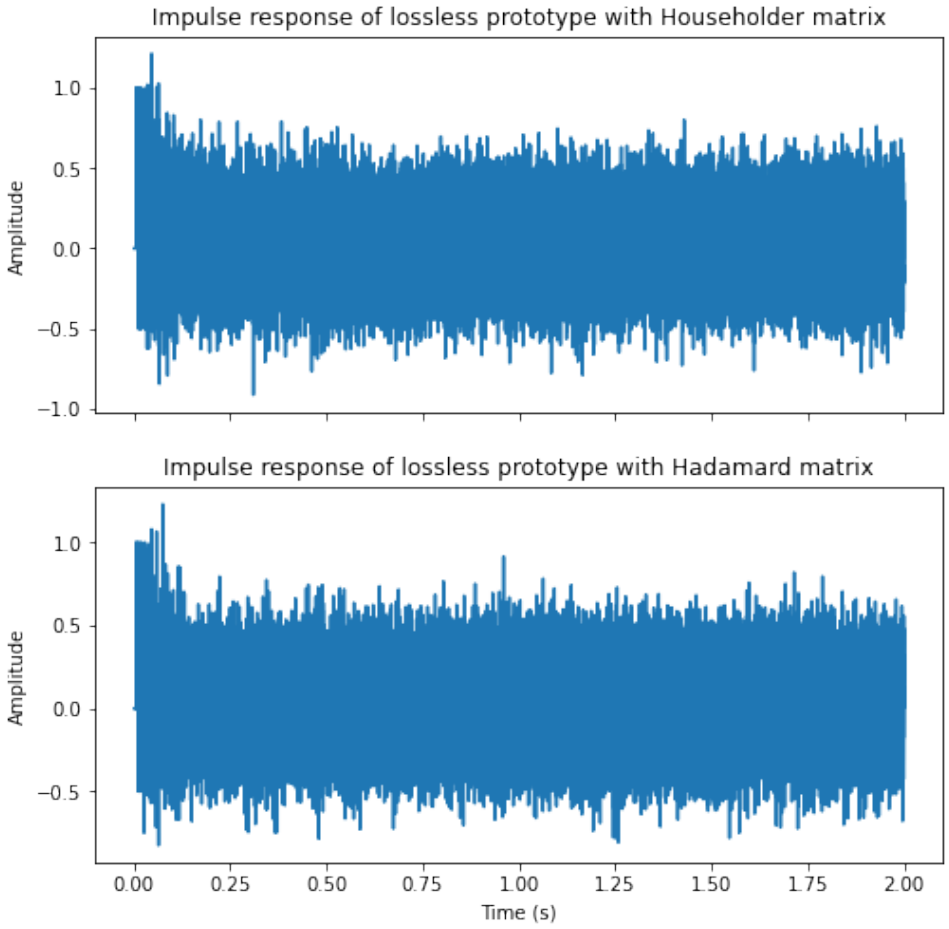


Figure 4.3: Output of the implemented plugin with parameters set to achieve lossless state. The two plots show the lossless signal for both matrix implementations.

4.2 Pedalboard

Pedalboard is a python package created by Spotify's Audio Intelligence Lab, that uses python bindings to allow studio grade audio plugin formats to be used and

controlled via python and TensorFlow [24]. Through the Pedalboard package one can load the VST3-format plugins and access them through function calls. All parameters available in the loaded plugins are available through key-value pairs. Even though the Pedalboard package works out-of-the box it requires some considerations.

4.2.1 Process and Reset Functions

A lot of debugging has gone into checking the process calls from Pedalboard to the VST3 plugin. When processing in mono it is important that you only write to the first channel of the output buffer, since the process would halt otherwise. The virtual reset function available through JUCE needs to be overwritten to clear all internal states of delay-lines and filters.

4.2.2 Parameter Control

As the Pedalboard package allows parameter control through key-value pairs the task of setting the parameters from a parameter vector is trivial. Inspired by the DeepAFx implementation by Ramírez et. al [15] a parameter mapping has been implemented, allowing control of what parameters to set via the neural network, and what parameters to set to an initial fixed value. This has been usefull both to reduce the parameter set when training various toy model problems, to fix the modulation frequencies and depths, and when selecting feedback matrix design between the implemented Householder and Hadamard matrices.

Parameter Ranges

The Pedalboard package has been found to have a high error ratio at $\sim 5\%$ when dealing with values between 0 and 1. To decrease the error ratio all parameters have been scaled from $[0, 1]$ to $[0, 100]$, where the error rate is reported to be at $\sim 0.1\%$. We did not want to do any changes in the parameter model of the neural network, so a factor of 100 is added to the parameter values when setting the values via Pedalboard in the audio processing model.

4.3 Neural Network

4.3.1 Hyperparameter Estimation

Hyperparameters has been estimated empirically throughout the implementation of the models. A well functioning learning rate has been found to be 1e-5, with a callback reducing it when the training plateaus over a set number of epochs. Each of the four losses has been given weight values to weight their contribution to the

overall model loss. The ϵ-value used in the SPSA gradient estimation (equation 3.3) has been found to work best at 0.01. All hyperparameters, along with various values used doing training, has been added to a configuration file for each training case. The configuration file serves as a python dictionary which helps to organize the various training parameters.

4.3.2 Pretraining

Pretraining the weights of the parameter model is beneficial to set an initial starting point for the training. Without pretraining the weights of the parameter model is randomized, and this can lead to unfortunate parameter combinations that the model has a high difficulty moving away from, as well as instability caused by the graphic equalizer exceeding 0dB gain at extreme cases. When pretraining the system the audio processing model is disconnected from the entire model. The training can then be done by creating a tensor of desired initial parameter values as the parameter model output target. Differentiation is done by measuring the mean-absolute-error between the target parameter tensor and system output parameter tensor.

4.4 Toy Model

To test the compatibility of the plugin and neural network a toy model problem was constructed. The idea of this toy model problem is for it to be a simple version of the problem investigated in this thesis. For this a configuration of the FDN plugin was created with only one delay-line, and with a minimized parameter set. The initial toy model problem was to test the integration between the different neural network parts and the FDN plugin, where the only parameters were for the attenuation filters. The result of training this toy model problem can be seen in figure 4.4, where the target audio was created with the same configuration of the FDN plugin that was used under the test, with a specific parameter set. With the toy model problem approach it was possible to test and iterate over different configurations of the neural network. This proved usefull in many different cases, e.g., testing the attenuation filter parameters with spectral loss, and testing delay-line length parameters with echo density loss. The configuration considerations during the plugin implementation mentioned in section 4.1 proved very usefull for quickly generating versions of the FDN plugin with different features.

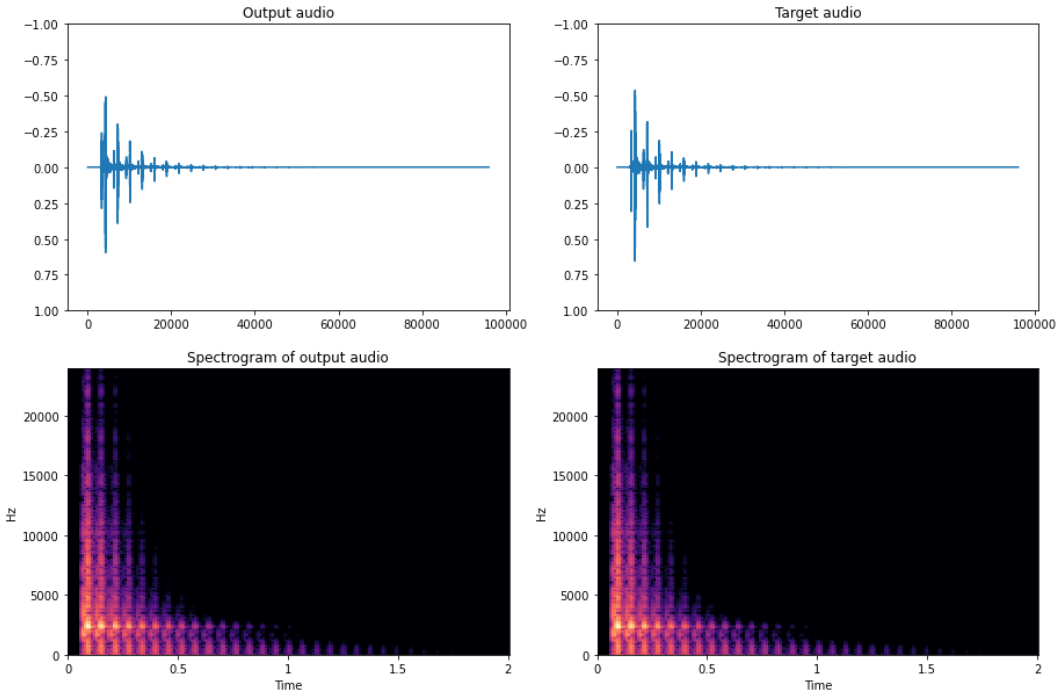


Figure 4.4: Result of the toy model training with one delay-line FDN.

Chapter 5

Evaluation

This chapter contains the multiple evaluations made for this project. A perceptual evaluation is done as a listening test using the MUltiple Stimuli with Hidden Reference and Anchor (MUSHRA) method. A qualitative evaluation is done on the performance of the implemented FDN plugin, and on the performance of the proposed neural network. As the FDN plugin is used in context with the neural network, and not with an end-user in mind, we did not consider a user-evaluation on the plugin.

5.1 Perceptual Evaluation

Reverberation as an audio effect can be hard to compare outside of a perceptual context. Different reverberation algorithms may have vastly different architecture, different attenuation methods, and different interfaces and parameters [5]. A perceptual evaluation is a widely used method of evaluation reverberated signals. This evaluation was done as a listening test using MUSHRA, where the testers were tasked with ranking different audio signals on a scale from 0-100 on their similarity to a reference signal, with 0-20 being **bad** perceived similarity, 20-40 being **poor** perceived similarity, 40-60 being **fair** perceived similarity, 60-80 being **good** perceived similarity, and 80-100 being **excellent** perceived similarity. The audio signals for each test trial consist of following:

Hidden Reference - This signal is expected to be scored at a 100, as it is a copy of the reference signal.

Estimation - This is the signal generated from the FDN plugin with parameters estimated by the neural network. This is the audio under test.

Anchor35 - This is an anchor signal that is the reference signal low-pass filtered at 3.5 kHz. This anchor is expected to be scored at a low score (0-20 points).

Anchor1ch - This is an anchor signal created like the estimation signal with parameters from the neural network, but with a FDN plugin configuration with only one delay-line. Like the other anchor, this is expected to perform badly, as the perception of a one delay-line FDN reverberator is tangent to an echo effect.

5.1.1 Training

To generate audio for the perceptual evaluation three different commercially available black-box reverberators were selected:

Ableton Live 10 Reverb by Ableton. A preset called *Small Room* with an advertised decay time of 1.4 seconds was selected. This preset has no modulation. A screenshot of the parameter settings for the reverb can be seen in figure 1.1.

Hall Of Fame 2 by TC Electronic. A copy of a Hall preset was created through the editor. This new preset has no modulation, and an advertised decay time of 1.818 seconds and some early reflections. A screenshot of the important subset of parameters can be seen in figure 1.2.

Valhalla Room by ValhallaDSP. This reverberator was set to a mode called Large Chamber with an advertised decay at 1.9 seconds. The reverberator has some slight modulation and advertised early reflections. A screenshot of the parameter settings in the GUI can be seen in figure 1.3.

A simple data-set was constructed for each black-box reverberator using a colored impulse-like signal of a finger-snap. The Ableton Reverb and the Valhalla Room are software plugins, so recording these was done entirely through the digital audio workstation Ableton Live[1]. The TC Electronic Hall Of Fame 2 used in this evaluation is a reverberation algorithm embedded in the TC Electronic Plethora X5 hardware unit. The Hall Of Fame 2 was recorded using a M-Audio M-Track Eight audio device. All reverberation impulses were recorded for two seconds at 48kHz. The three datasets was then constructed as a pair of the recorded wet impulse response and the dry impulse-like signal. Training was done for each black-box effect in a Jupyter Notebook[2] environment (see appendix B) for 3000 epochs. For each training the result was a parameter set that could be applied to the FDN plugin. The parameter set selected for training consisted of: delay-line minimum and maximum length, pre-delay value, attenuation graphic equalizer reverberation time values, spectral coloration graphic equalizer control gain values, and input and output gain values. The parameters for time-varying modulation were

[1] `https://www.ableton.com/en/live/`

[2] A web-based environment for coding and creating documents.

omitted. The parameter for feedback matrix selection was set by visual inspection of the echo density graph for the target audio.

MUSHRA-Ready Audio

With the FDN plugin tuned to the parameters given by the neural network training multiple reverberated audio signals could be created for the MUSHRA listening test. Three different dry mono audio signals were gathered for the test. These audio signals were chosen on their different perceptual qualities and musical context:

- One signal should have multiple transients and natural gaps. For this a two-bar drum loop was created with natural sounding drum samples.
- One signal should have varying harmonic content and still keep contain some percussive elements. For this a piano signal was recorded with different chords.
- One signal should have vocal properties. For this a short vocal recording was provided by a singer.

The three audio signals were then processed through the FDN plugin for each estimated parameter set to create the audio signals for the MUSHRA test. All audio signals under test were normalized and converted to pseudo-stereo, where the mono signal was applied to both left and right channel. This was done to conform to the webMUSHRA implementation, and to let testers listen to the audio with stereo-headphones. The MUSHRA-ready reference audio was recorded for each black-box reverberator in similar manner to the training dataset. With the three test audio signals and the three black-box effects the MUSHRA listening test consist of nine trials.

5.1.2 Test Design

An online server-hosted implementation of a MUSHRA evaluation test was set up using webMUSHRA [18]. Having an online server-hosted implementation allowed for offsite evaluation, where the testers were able to do the listening test without supervision. The instructions given to the testers can be found in appendix A. After the initial instructions the testers were shown a page with an example listening test, where they could familiarize themselves with the MUSHRA interface and controls. When advancing they were then prompted that the test was about to begin. The nine listening trials were then presented for the testers in randomized order. After the listening-test the testers were given a post-test questionnaire with following points to answer:

1. Age

2. Years of music experience
3. Years of audio-engineering experience
4. Briefly describe your audio setup used in the test
5. Briefly describe the environment you were in during the test
6. Do you have any hearing impairments?
7. Are there any thoughts or observations you want to share?

The first three points in the questionnaire are for demographic survey data. Points 4-6 in the questionnaire was mainly to be able to investigate if there was any reason for large deviations in the answers, that could be due to hearing impairments, bad equipment, or noisy environments. The last point acts as an interview question where the answers are acknowledged, but not reported on, unless some answers contain something of direct interest for this project.

5.2 Qualitative Evaluation

Qualitative evaluation has been done on the FDN plugin and neural network, where the plugin could be evaluated as a standalone product but is an integral part of the neural network performance.

5.2.1 Plugin Performance

One requirement stated earlier is for the plugin to be able to be set as a lossless prototype. This requirement has been fulfilled as discussed in section 4.1.6, though this required expanded parameter ranges of near-infinite reverberation time values or direct gain control, that is not available in the final implementation used in the neural network. When dealing with VST plugins that should be usable in a digital audio workstation CPU usage percentage is an important metric. The CPU usage has been tested on an Intel Core i5-8400 CPU 2.80GHz processor with 6 cores where it did not exceed 10% and usually stayed around 4%. This seems acceptable for an extended FDN implementation.

5.2.2 Neural Network Performance

An epoch with four VST plugin processes running takes $\sim$ 5 seconds, giving us an average training time for the 3000 epochs used to train the network at around 4 hours and 10 minutes. We can compare multiple qualities of the estimated and target impulse responses to investigate the performance of the neural network

	Estimated T60	Target T60	Difference
Ableton Reverb	1019ms	1458ms	30.1%
Hall of Fame 2	1576ms	1757ms	10.3%
Valhalla Room	1912ms	1815ms	5.3%

Table 5.1: Calculated T60 values from estimated and target signals for each black-box reverberator.

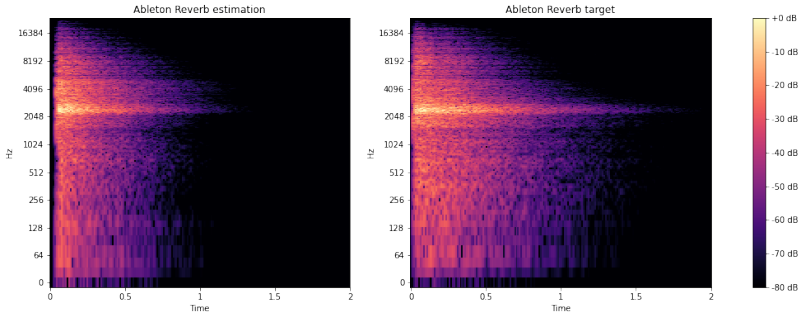


Figure 5.1: STFT of estimated and target signal for Ableton Reverb

training. By calculating the T60 values for each signal we get insight to the reverberation time, and by plotting STFT of the signals we can compare the spectral content.

T60 Values

The T60 values have been calculated using MATLAB[3] and are reported, along with their difference in percent, in table 5.1. It is clear that the estimated T60 values differ a great deal from the target values.

Spectral Content

The spectral content of the estimated and target impulse responses have been plotted in dB using the Librosa[4] STFT function. The power spectrograms for the three cases can be seen in figure 5.1, 5.2, and 5.3.

[3] https://se.mathworks.com/products/matlab.html
[4] `https://librosa.org/`

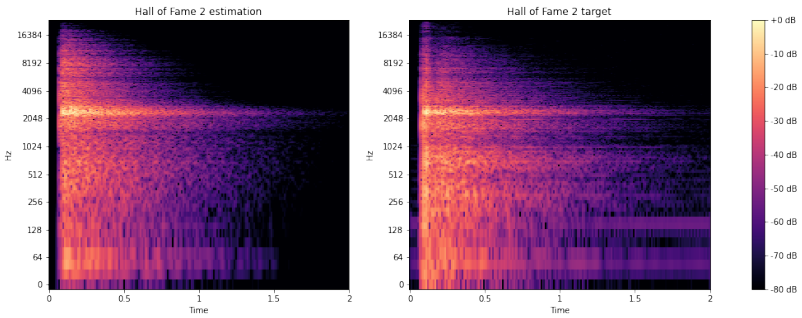


Figure 5.2: STFT of estimated and target signal for Hall Of Fame 2

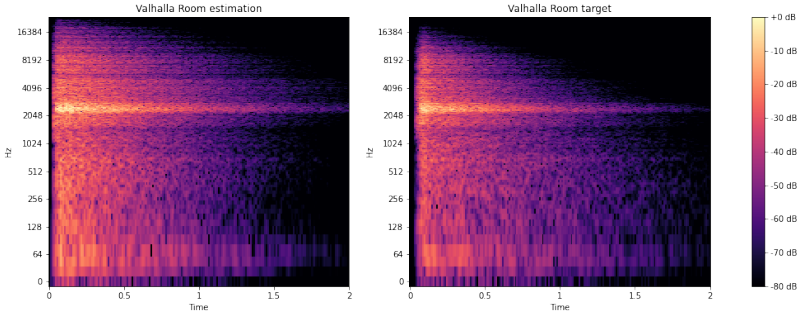


Figure 5.3: STFT of estimated and target signal for Valhalla Room

Chapter 6

Results and Discussion

In this chapter we will discuss the results of the perceptual evaluation. We will report on the mean-value across the test answers for each case and compare the results with the data from the qualitative evaluation.

6.1 Perceptual Evaluation Results

In all trials the two anchors have been correctly identified, with the 3.5kHz filtered anchor performing a bit better than the 1 delay-line FDN implementation. In this section we will look at the three cases, each consisting of their three respective trials (drums, piano, vocals). A one-way ANOVA analysis has been done on the test results but yielded no results worth reporting. 15 testers have participated in the listenening test. Their mean age is 29.6 years, with a mean reported music experience at 15.4 years. 10 of the participants report having audio-engineering experience, with a mean of 10.4 years of experience between them. Three of the testers report that they have mild tinnitus. All testers report that they took the test with a combination of environment and setup hardware deemed acceptable.

6.1.1 Ableton Reverb

The estimated signal for the Ableton Reverb scored a 66.18/100. With 52.5 for the drum signal, 66.4 for the piano signal, and 79.6 for the vocal signal. A violin plot of the results can be seen in figure 6.1

6.1.2 Hall Of Fame 2

The estimated signal for the Hall Of Fame 2 scored a mean rating at 71.02/100. With 62.13 for the drum signal, 81.33 for the piano signal, and 69.6 for the vocal

Figure 6.1: The perceptual evaluation results for the Ableton Reverb test case.

signal. This case has the best overall performance mean score. A violin plot of the results can be seen in figure 6.2

6.1.3 Valhalla Room

The estimated signal for the Valhalla Room scored a 70.42/100. With 66.07 for the drum signal, 65.93 for the piano, and 79.27 for the vocal signal. A violin plot of the results can be seen in figure 6.3

6.2 Discussion

Looking at the results of the perceptual evaluation it is clear that the neural network has not been able to correctly estimate the parameters to emulate the target reverberated signals to satisfaction. We should however not dismiss that the training has yielded good enough results for the mean-value scores of to be within a range that the scale in the MUSHRA listening test labels as **good**. Overall the Hall Of Fame 2 estimation case scores the highest with a 71.02, followed by the Valhalla Room case, and finally the Ableton Reverb case. If we consider the T60 values extracted in qualitative evaluation section 5.2.2 it would make sense that the Ableton Reverb estimation performs the worst, given its T60 value difference at 30.1%, where as the T60 value difference is only 5.3% for the Valhalla Room, which is close to the just-noticeable-difference for reverberation time of 5% [13]. In two of

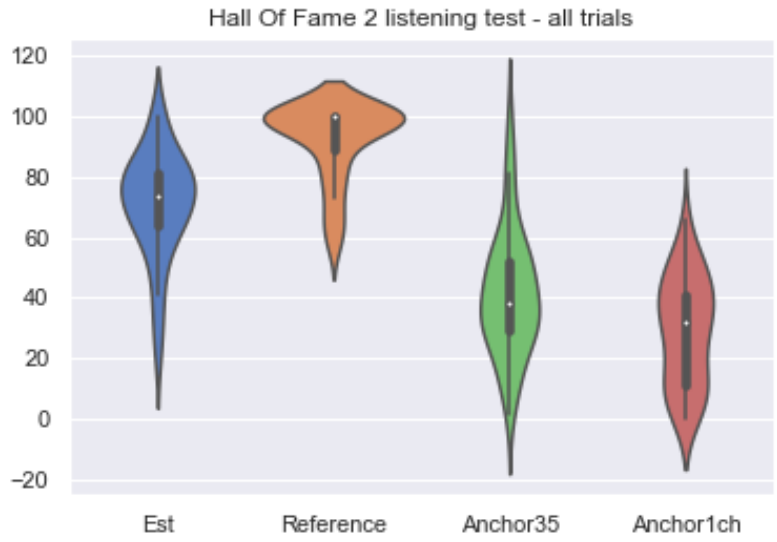


Figure 6.2: The perceptual evaluation results for the Hall Of Fame 2 test case.

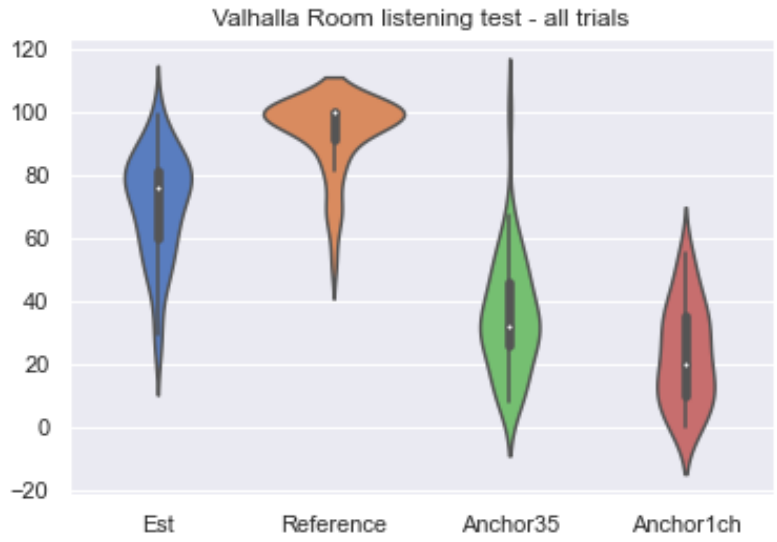


Figure 6.3: The perceptual evaluation results for the Valhalla Room test case.

the three cases the drums performed the worst (see appendix A for individual trial violin plots). This is likely because of its transient and percussive nature, giving room for the reverberation tail to stand out. It is also common to tune reverberation to the sound of a snare, both in studio and live settings. It is interesting to note that for all three test cases the vocal trial scored no less than 69.6, with it being the highest performer in two of the cases.

Chapter 7

Conclusion

7.1 Conclusion

Through the work of this master thesis a well-performing extended feedback delay network has been implemented in C++ and compiled to the VST3 format, along with a neural network structure for estimating parameters of said FDN. Through qualitative and perceptual evaluation, it has been shown that the performance of the neural network has not been optimal for the estimation problem, but it has given initial useful results. Both FDN and neural network have been implemented with compatibility in mind, allowing for easy modification and further development. The FDN is partially compatible with different sampling rates, and the design and implementation of state-of-the-art graphic equalizers and delay-line modulation brings its functionality close to many commercially available reverberators. With the addition of preprocessor configurability and multi-platform continuous delivery workflows further work is encouraged. For the neural network a novel echo density loss function has been implemented and optimized for TensorFlow operations, and the proposed audio processing model with embedded VST3 compatibility is a novel contribution with the possibility of extending it to different VST3 plugins, not just the proposed FDN plugin.

7.2 Future Work

The next step in the development of the neural network will be to look at the performance. With a deeper knowledge in deep learning and neural networks it should be possible to optimize the parameter estimation to allow for further convergence of a fitting parameter set. As mentioned in the conclusion the audio processing model would be compatible with different VST3 plugins, allowing for the reuse in different estimation problems. At the current state the FDN plugin is performing well at 44.1kHz and 48kHz, and is capable of creating high quality

reverberation. Therefore it makes sense to continue work on the plugin to bring it to a state intended for normal use in a digital audio workstation. This requires the ability to process audio in stereo, which could be implemented with, e.g., Inter Aural Cross Correlation. With the introduction of the graphic equalizers the range of functioning sampling rates were reduced, but this could be alleviated with an investigation into the graphic equalizers, and following changes in design and implementation. Furthermore a graphical user interface could be developed, and the exposed parameters of the FDN could be reduced to a number of higher level parameters for better user experience.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Jonathan S Abel and Patty Huang. "A Simple, Robust Measure of Reverberation Echo Density". In: *Audio Engineering Society Convention 121*. Audio Engineering Society. 2006.

[3] Barry A. Blesser. "An Interdisciplinary Synthesis of Reverberation Viewpoints". In: *Journal of the Audio Engineering Society* 49.10 (2001), pp. 867–903.

[4] David Braun. "DawDreamer: Bridging the Gap Between Digital Audio Workstations and Python Interfaces". In: *arXiv preprint arXiv:2111.09931* (2021).

[5] Brecht De Man, Kirk McNally, and Joshua Reiss. "Perceptual Evaluation and Analysis of Reverberation in Multitrack Music Production". In: *Journal of the Audio Engineering Society* 65 (Feb. 2017), pp. 108–116.

[6] Jesse Engel et al. "DDSP: Differentiable Digital Signal Processing". In: *arXiv preprint arXiv:2001.04643* (2020).

[7] William G. Gardner. "Reverberation Algorithms". In: *Applications of Digital Signal Processing to Audio and Acoustics*. Ed. by Mark Kahrs and Karlheinz Brandenburg. Springer US, 2002, pp. 85–131.

[8] Jean-Marc Jot and Antoine Chaigne. "Digital Delay Networks for Designing Artificial Reverberators". In: *Audio Engineering Society Convention 90*. Audio Engineering Society. 1991.

[9] 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.

[10] Sungho Lee, Hyeong-Seok Choi, and Kyogu Lee. "Differentiable Artificial Reverberation". In: *arXiv preprint arXiv:2105.13940* (2021).

[11] Søren V.K. Lyster and Cumhur Erkut. *A Differentiable Neural Network Approach to Parameter Estimation of Reverberation*. Accepted for SMC-22 conference. 2022.

[12] James A Moorer. “About This Reverberation Business”. In: *Computer Music Journal* (1979), pp. 13–28.

[13] Karolina Prawda, Sebastian J Schlecht, and Vesa Välimäki. “Improved Reverberation Time Control for Feedback Delay Networks”. In: *Proc. Int. Conf. Digit. Audio Effects*. 2019, pp. 1–7.

[14] Karolina Prawda et al. “Flexible Real-time Reverberation Synthesis With Accurate Parameter Control”. In: *23rd International Conference on Digital Audio Effects*. 2020, pp. 16–23.

[15] Marco A Martínez Ramírez et al. “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.

[16] Mark Sandler et al. “Mobilenetv2: Inverted Residuals and Linear Bottlenecks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520.

[17] Sebastian J Schlecht and Emanuël AP Habets. “Accurate Reverberation Time Control In Feedback Delay Networks”. In: 2017, pp. 337–344.

[18] Michael Schoeffler et al. “webMUSHRA—A Comprehensive Framework for Web-based Listening Tests”. In: *Journal of Open Research Software* 6.1 (2018).

[19] Manfred R. Schroeder. “Natural Sounding Artificial Reverberation”. In: *Journal of the Audio Engineering Society* 10.3 (1962), pp. 219–223.

[20] Manfred R Schroeder and Benjamin F Logan. “" Colorless" Artificial Reverberation”. In: *IRE Transactions on Audio* 6 (1961), pp. 209–214.

[21] Siyuan Shan et al. “Differentiable Wavetable Synthesis”. In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, pp. 4598–4602.

[22] Julius O. Smith. *Physical Audio Signal Processing*. online book, 2010 edition. `http://ccrma.stanford.edu/~jos/pasp/`, accessed 25/05/2022.

[23] James C Spall. “An Overview of the Simultaneous Perturbation Method for Efficient Optimization”. In: *Johns Hopkins APL Technical Digest* 19.4 (1998), pp. 482–492.

[24] Spotify. *Pedalboard*. `https://github.com/spotify/pedalboard`. accessed 25/05/2022. 2021.

[25] Vesa Valimaki et al. “Fifty Years of Artificial Reverberation”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.5 (2012), pp. 1421–1448.

[26] Vesa Välimäki and Juho Liski. “Accurate Cascade Graphic Equalizer”. In: *IEEE Signal Processing Letters* 24.2 (2017), pp. 176–180.

[27] Vesa Välimäki and Joshua D. Reiss. "All About Audio Equalization: Solutions and Frontiers". In: *Applied Sciences* 6.5 (2016), p. 129.

[28] Neha Yadav, Anupam Yadav, and Manoj Kumar. "History of Neural Networks". eng. In: *An Introduction to Neural Network Methods for Differential Equations*. SpringerBriefs in Applied Sciences and Technology. Dordrecht: Springer Netherlands, 2015, pp. 13–15.

Appendix A

Appendix A - Evaluation Data

A.1 MUSHRA Interface

A screenshot of the webMUSHRA interface can be seen in figure A.10.

A.2 Test Instructions

Following are the test instructions tester were given as part of the MUSHRA listening test.

- **Page 1** *Welcome to the reverberation comparison test for the master thesis by Søren V. K. Lyster at Aalborg University Copenhagen. In this experiment you will be given a reference audio signal and 4 test audio signals that you have to evaluate. The 4 test audio signals need to be judged on their similarity to the reference signal on a moving scale from bad to excellent. You will be shown 9 different cases during the test.*

  All participation in this test is anonymous, and no meta-data will be collected. Participation is voluntary. By continuing you agree on these terms.

- **Page 2** *This first page is just for you to get familiar with the controls. There are 5 buttons below the audio waveform that allow you to play and pause the reference signal and the 4 test signals. The stop button on the left will stop the audio. The horizontal slider below the waveform allows you to control what part of the audio is played. The audio will loop automatically. When you are ready to start the test, press the next button.*

- **Page 3** *The test will begin on the next page.*

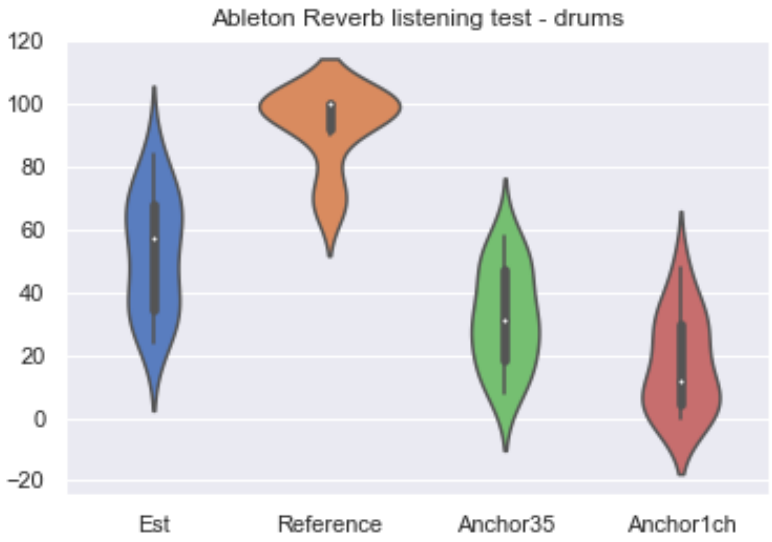


Figure A.1: The perceptual evaluation results for the Ableton Reverb drum signal.

Figure A.2: The perceptual evaluation results for the Ableton Reverb piano signal.

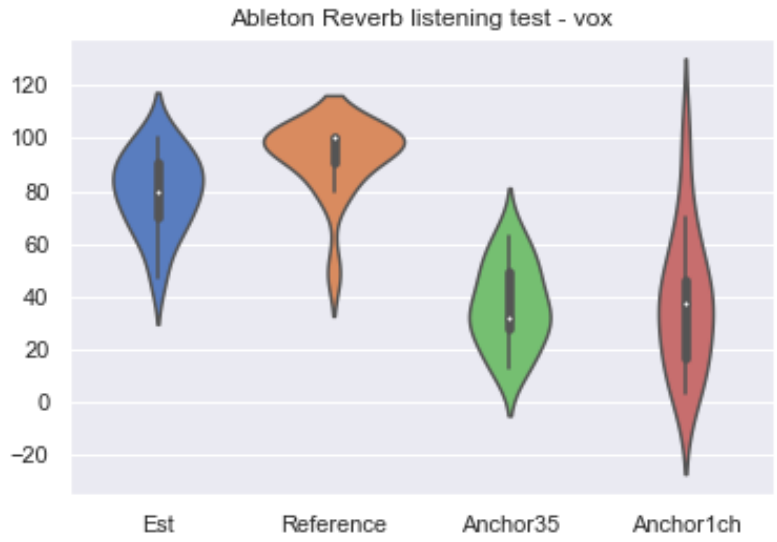


Figure A.3: The perceptual evaluation results for the Ableton Reverb vocal signal.

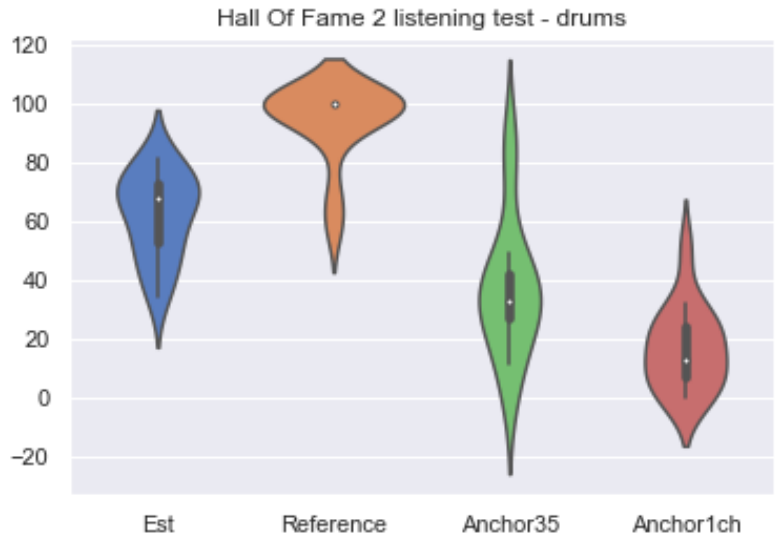


Figure A.4: The perceptual evaluation results for the Hall Of Fame 2 drum signal.

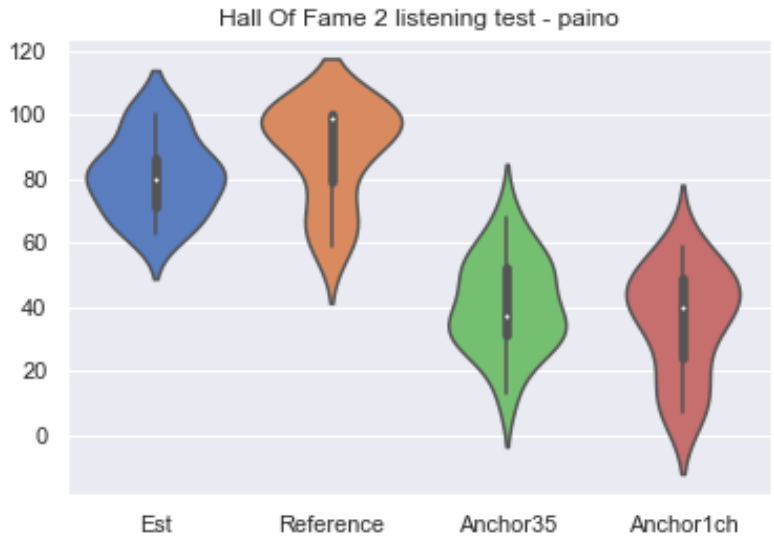


Figure A.5: The perceptual evaluation results for the Hall Of Fame 2 piano signal.

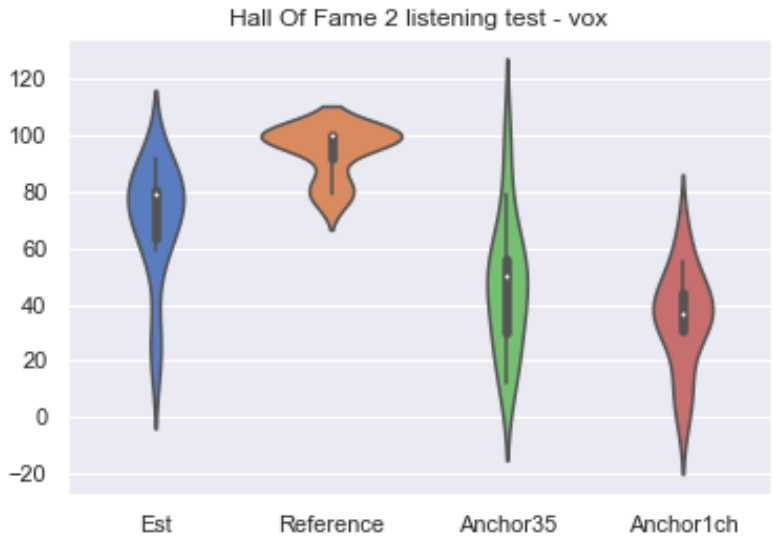


Figure A.6: The perceptual evaluation results for the Hall Of Fame 2 vocal signal.

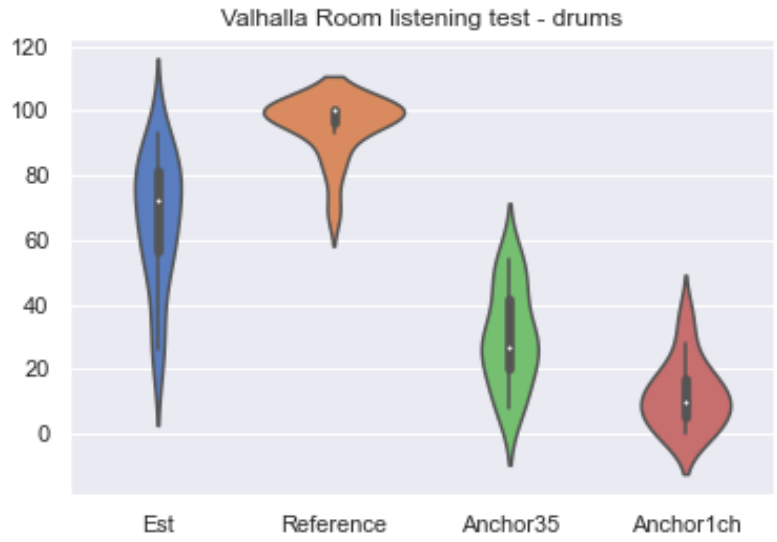


Figure A.7: The perceptual evaluation results for the Valhalla Room drum signal.

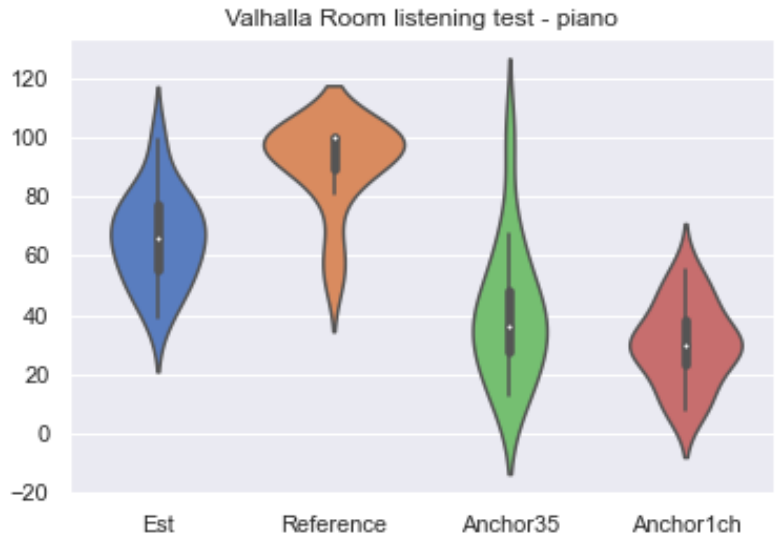


Figure A.8: The perceptual evaluation results for the Valhalla Room piano signal.

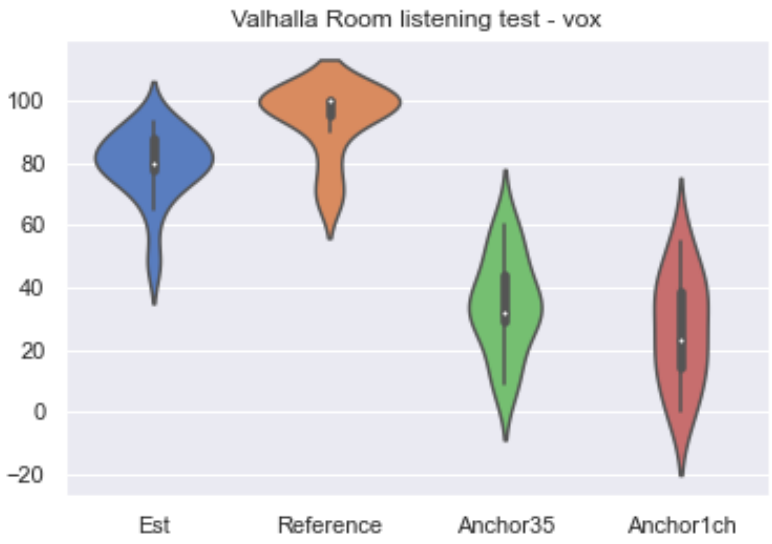


Figure A.9: The perceptual evaluation results for the Valhalla Room vocal signal.

Figure A.10: Screenshot of the webMUSHRA interface

A.3 Test Results

Figure A.1 through A.9 shows the violin plots for the test results for the nine individual trials in the listening test (drums, piano, and vocal for each black-box reverberator).

Appendix B

Appendix B - Neural Network Models

B.1 Summary of the parameter model

Model: "parameter_model"

Layer (type)	Output Shape	Param #
audio_time (InputLayer)	[(None, 96000)]	0
logMelgram (LogMelgramLayer)	(None, 372, 128, 1)	0
conv2d (Conv2D)	(None, 371, 127, 128)	640
max_pooling2d (MaxPooling2D)	(None, 185, 63, 128)	0
batch_normalization (BatchNo	(None, 185, 63, 128)	512
conv2d_1 (Conv2D)	(None, 184, 62, 32)	16416
max_pooling2d_1 (MaxPooling2	(None, 92, 31, 32)	0
batch_normalization_1 (Batch	(None, 92, 31, 32)	128
conv2d_2 (Conv2D)	(None, 91, 30, 16)	2064
max_pooling2d_2 (MaxPooling2	(None, 45, 15, 16)	0
batch_normalization_2 (Batch	(None, 45, 15, 16)	64

```
conv2d_3 (Conv2D)              (None, 44, 14, 32)          2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 22, 7, 32)           0
_________________________________________________________________
batch_normalization_3 (Batch (None, 22, 7, 32)           128
_________________________________________________________________
flatten (Flatten)              (None, 4928)                0
_________________________________________________________________
dense (Dense)                  (None, 32)                  157728
_________________________________________________________________
dropout (Dropout)              (None, 32)                  0
_________________________________________________________________
dense_1 (Dense)                (None, 55)                  1815
=================================================================
Total params: 181,575
Trainable params: 181,159
Non-trainable params: 416
```

B.2 Summary of the full model with both parameter model and audio processing model

```
_________________________________________________________________
Model: "full_model"
_____________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=============================================================================================
audio_time (InputLayer)      [(None, 96000)]           0
_____________________________________________________________________________________________
parameter_model (Functional) (None, 55)                181575    audio_time[0][0]
_____________________________________________________________________________________________
vst_processor (VSTProcessor) (None, 96000)             0         audio_time[0][0]
                                                                 parameter_model[0][0]
=============================================================================================
Total params: 181,575
Trainable params: 181,159
Non-trainable params: 416
_____________________________________________________________________________________________
```

Appendix C

Appendix C - Jupyter Notebook for Training

ReverberatorEstimator

May 23, 2022

1 ReverberatorEstimator notebook

Jupyter Notebook for Parameter Estimation of Reverberation Using a Neural Network project by Søren Lyster

1.1 Import needed packages

[ ]:
```
import matplotlib.pyplot as plt
import IPython
import tensorflow as tf
print(tf.__version__)
import tensorflow.keras as tfk
from ReverberatorEstimator import loss, models, utils, config
import warnings
warnings.filterwarnings('ignore')
import time
import os
import datetime
import IPython
```

1.2 Setup environment

[ ]:
```
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
```

1.3 Setup variables for the notebook

[ ]:
```
k = config.k
sample_rate = k['sample_rate']
sample_length = k['sample_length']
num_epochs = k['epochs']
num_processors = k['n_processors']
steps_per_epoch = k['steps_per_epoch']
batch_size = steps_per_epoch * num_processors
epsilon = k['epsilon']
learning_rate = k['learning_rate']
dry_audio_path = k['dry_audio_path']
wet_audio_path = k['wet_audio_path']
vst_path = k['vst_path']
```

```
time_loss_weight = k['time_loss_weight']
spectral_loss_weight = k['spectral_loss_weight']
envelope_loss_weight = k['envelope_loss_weight']
echo_density_loss_weight = k['echo_density_loss_weight']
use_multiscale = k['use_multiscale']
num_params = k['n_parameters']
parameter_map = k['parameter_map']
non_trainable_parameters = k['non_trainable_parameters']
pretrained_weights = k['pretrained_weights']
checkpoint_path = k['checkpoint_path']

print(parameter_map)
```

1.4 Setup dataset for batch training

[ ]:
```
x_train, y_train = utils.get_dataset(dry_audio_path, wet_audio_path,␣
↪batch_size, resample=True, old_sample_rate=48000,␣
↪new_sample_rate=sample_rate)
```

1.5 Create layers, create partial models, and compile full model

[ ]:
```
model, parameter_model, processor = models.get_models(sample_length,␣
↪sample_rate, num_params, num_processors,
                                                      vst_path, epsilon,␣
↪parameter_map, non_trainable_parameters,
                                                      pretrained_weights)

reverberation_loss = loss.reverberationLoss(sample_rate=sample_rate,
    spectral_loss_weight=spectral_loss_weight,
    spectral_loss_type='L1',
    time_loss_weight=time_loss_weight,
    time_loss_type='L1',
    envelope_loss_weight=envelope_loss_weight,
    envelope_loss_type='L1',
    echo_density_weight=echo_density_loss_weight,
    echo_density_type='L1',
    use_multiscale=use_multiscale,
    )

optimizer = tfk.optimizers.Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss=reverberation_loss, run_eagerly=True)
```

1.6 Print model summaries

[ ]:
```
parameter_model.summary()
model.summary()
```

1.7 Setup checkpoint and callbacks

[ ]:
```
checkpoint_dir = os.path.dirname(checkpoint_path)

model_cp = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                 save_weights_only=True,
                                 monitor='loss',
                                 verbose=1,
                                 save_best_only=True,
                                 mode='min')

lr_callback = tf.keras.callbacks.ReduceLROnPlateau(monitor='loss',
                                  factor=0.5,
                                  patience=500,
                                  cooldown=1,
                                  verbose=1,
                                  mode='auto',
                                  min_lr=1e-10)

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tfk.callbacks.TensorBoard(log_dir=log_dir,␣
 ↪histogram_freq=1)
```

1.8 Restore from previous checkpoint if it exists

[ ]:
```
try:
    model.load_weights(checkpoint_path)
except:
    print("No previous checkpoints found at %s" % checkpoint_path)
```

1.9 Run model and save data before training for analysis and debugging

[ ]:
```
input_audio = tf.reshape(x_train[0], (1, sample_length))
target_audio = tf.reshape(y_train[0], (1, sample_length))

audio_pre = (model.call(input_audio)).numpy()[0]
old_params = parameter_model(input_audio).numpy()[0]
print(old_params)
```

1.10 Run the model.fit to begin training.

[ ]:
```
start_time = time.time()
history = model.fit(x_train, y_train, verbose=1, epochs=num_epochs,␣
 ↪steps_per_epoch=steps_per_epoch,
         callbacks=[model_cp, lr_callback])
print("Training took %d seconds" % (time.time() - start_time))
```

1.11 Plot training loss metrics

[ ]:
```
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(7,5))
ax.plot(history.history['loss'])
ax.set_title('loss')
ax.set_ylabel('loss')
ax.set_xlabel('epoch')
ax.legend(['train', 'test'], loc='upper left')
```

1.12 Run a forward pass and get the output audio of the trained model

[ ]:
```
model.load_weights(checkpoint_path)
output_audio = model(input_audio)
processor.print_current_parameters()
```

1.13 Display the output audio from before training the model

This is done to inspect the changes the training has done

[ ]:
```
utils.plot_single(audio_pre, sample_rate, sample_length)
```

1.14 Plot the output audio against the target audio

[ ]:
```
utils.plot_output_and_target(output_audio, target_audio, sample_rate)
IPython.display.display(IPython.display.Audio(output_audio, rate=sample_rate,␣
 ↪autoplay=True))
IPython.display.display(IPython.display.Audio(target_audio, rate=sample_rate))
```

1.15 Plot the loss function differences

[ ]:
```
utils.plot_differences(output_audio, target_audio, sample_rate,␣
 ↪weights=[time_loss_weight,spectral_loss_weight,envelope_loss_weight,echo_density_loss_weigh
```

[ ]:
```
utils.plot_differences(output_audio, tf.reshape(tf.
 ↪convert_to_tensor(audio_pre), (1,sample_length)), sample_rate)
```

1.16 Print the parameters

These parameters are from the parameter model subpart of the full model. These values are transferable to the FDN reverberator plugin at [https://github.com/VoggLyster/Reverberator]

[ ]:
```
params = parameter_model(input_audio).numpy()[0]
print('New parameter set: ', params)
plt.stem(params)
plt.ylim(0,1)
```

1.17 Plot the parameter differences of before and after training

This shows the movement of the parameters after training and can give a good picture of the momentum of the training loop

[ ]:
```
param_diff = params - old_params
print('Parameter set difference: ', param_diff)
plt.stem(param_diff)
plt.ylim(-1,1)
```

1.18 Generate MUSHRA-ready audio files

[ ]:
```
audio_data, audio_names = utils.generate_MUSHRA_ready_audio(vst_path, params,␣
↪sample_rate)
for i in range(len(audio_data)):
    print(audio_names[i])
    IPython.display.display(IPython.display.Audio(audio_data[i],␣
↪rate=sample_rate))
utils.write_audio_files(audio_data, audio_names, 'MUSHRA_audio/Wet/
↪AbletonReverb', sample_rate)
```