Loudspeaker Modelling with Recurrent Neural Networks

Teodors Kerimovs

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.05.2023

Supervisor and advisor

Prof. Sebastian Jiro Schlecht

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Teodors Kerimovs

Title Loudspeaker Modelling with Recurrent Neural Networks

Degree programme Master of Science (Technology)

Major Acoustics and Audio Technology **Code of major** ELEC3030

Supervisor and advisor Prof. Sebastian Jiro Schlecht

Date 31.05.2023 **Number of pages** 57 **Language** English

Abstract

Digital twins of loudspeakers are a useful assets for fine-tuning purposes during the design and the manufacturing phase. They can serve as an alternative to real-time measurement for objective evaluation of adjustments made by digital signal processing. Binaural loudspeaker models could introduce a more repeatable framework for subjective listening and provide flexibility for remote work due to the reduced need for actual physical devices.

Neural Networks are a well-proven tool for system identification of different audio hardware devices. This thesis project will focus on creating a digital twin of a multimedia stereo loudspeaker system by using stereo audio waveform as the *input* and a binaural recording of the system's playback as the *target* waveform for Recurrent Neural Network (RNN) training. The RNN architecture is inspired by the current state-of-the-art method for single channel audio effects modelling, and is adapted for the stereo waveform use case.

Firstly, the RNN model is tested with different synthesized *target* data that simulates the real recorded data. This approach allows us to estimate the properties which are the most challenging for the RNN to learn. Secondly, the experiments are run with a real recorded, time-aligned dataset, and the RNN's performance is objectively evaluated by the Error-To-Signal Ratio (ESR).

In the current state-of-the-art method on single channel audio modelling, the initial hidden state of the RNN is computed by using no-gradient startup inference to accumulate the hidden state over the first few hundred samples of the training sequence. The thesis project proposes a new method called Discontinuous Sequence Training (DISCO). The method prepares the training dataset according to the RNNs architecture's hyper-parameter *sequence length* and the system's impulse response length, such that it allows for correct initialization of the initial hidden state without additional pre-training inference. DISCO reaches the training and inference precision of hidden state initialization in the current state-of-the-art method for black-box modelling with RNNs only by modifying the dataset.

Keywords Loudspeaker modelling, Deep Learning, Stereo modelling, RNN, DISCO sequence training, Digital Twin, System Identification

Preface

This master's thesis project was completed at Aalto University between April 2022 and February 2023.

The project was funded by Microsoft Finland. I am grateful for having Bryn Louise as my colleague for the first few months of the project. I would like to extend my gratitude to Heikki Laine and other Microsoft team members for managing the project. I was very happy to notice the interest not only in the results but also in the process of my research.

I sincerely thank my thesis supervisor, Professor Sebastian Schlecht, for his invaluable guidance and support throughout my master's studies. Even if my ideas did not work out that well most of the time, Sebastian always gave me useful feedback that was encouraging. That is indeed a special skill of a professor. The discussions that we had, both within and outside the scope of my thesis, were an important experience of my studies here at Aalto University and allowed me to learn from his constructive and creative way of thinking.

I would also like to express my gratitude to Doctoral Student Alec Wright and Professor Vesa Välimäki. Their contribution to deep learning in audio and their work on real-time black-box modelling with recurrent Neural Networks are crucial scientific sources for my master's thesis work.

My studies at Aalto started in the autumn of 2021. While majoring in Acoustics and Audio Technology, an important part of my time here has been my dear friends: AV, BS, AG, MP, AJ, and IS. Thank you guys for the nudge when it was necessary and for all the great adventures together!

Otaniemi, 31.05.2023

Teodors Kerimovs

Contents

Symbols and abbreviations

Symbols

f_s	sampling frequency
θ	learnable parameters of the model
z^{-1}	one sample time delay
$\|\|x - \hat{x}\|\|_1$	L1 norm between x and $\hat{x}$
δ	Dirac delta function
σ	sigmoid function

Operators

$(f \star g)(n)$	discrete cross-correlation between f and g, with lag (n)
$(f * g)(n)$	discrete convolution between f and g, with lag (n)
$\triangleq$	definition
$\odot$	element-wise multiplication

Abbreviations

MSE	Mean-Squared-Error
ESR	Error-to-Signal Ratio
SNR	Signal-to-Noise Ratio
MLRS	Multimedia Loudspeaker Recording Simulation
THD	Total Harmonic Distortion
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
HRTF	Head Related Transfer Function

1 Introduction

A loudspeaker is an electroacoustic device that includes one or more electroacoustic transducers that convert electrical signals into sound pressure. Loudspeakers can be used as standalone units or as part of applications where such signal conversion is necessary, such as consumer electronics.

Strong commercial growth in the consumer electronics market appeared in the mid-twentieth century when people and institutions turned their focus towards peacetime uses of technologies [1]. Since then, technical capabilities in audio have grown significantly hand in hand with expectations of the produced sound quality. Small mobile devices such as smartphones, tablets, and laptops are an important part of the modern consumer electronics market. First, cost and size are two major design constraints for the built-in sound system of such mobile device. Loudspeakers used in multimedia devices are miniature compared to HI-FI audio systems. It is a challenging task for engineers and designers to deal with trade-offs between low cost, small size, and the best possible sound quality of the built-in audio systems of mobile devices [2].

Achieving optimal audio system performance for consumer electronics or multimedia devices requires fine-tuning of the audio settings. The process involves using a range of techniques such as equalization, compression, and noise reduction to optimize sound quality for the target audience. To obtain objective measurements of the resulting sound after adjustments, specialized equipment and techniques such as measuring microphones and frequency analyzers may be used, and measurements may be performed in an anechoic chamber to eliminate any unwanted reflections or reverberation. In many cases, the adjustments to the audio playback may be implemented using digital signal processing (DSP) blocks that are integrated into the multimedia device. Collaboration with software developers responsible for implementing the DSP algorithms is necessary to ensure that the adjustments are optimized for the specific hardware and software components of the device. The process of fine-tuning audio playback in multimedia devices requires technical expertise and specialized equipment.

A digital twin of loudspeaker might be a useful asset for fine-tunning purposes during the design and also manufacturing phase. It can serve as a alternative for real-time measurement for objective evaluation of an adjustment made by digital signal processing. Binaural loudspeaker models could introduce a more repeatable environment for subjective listening and flexibility for the remote work due to the reduced need for actual physical device.

1.1 Background

Neural Networks (NNs) have proved to be a useful tool in almost any field of technology, including Acoustics and Audio Technologies. NNs' ability to learn nonlinear functions with a time-dependency from different types of audio signal representations solves many tasks. In the audio effect industry over the last decade, one of the top challenges has been the task of creating a digital twin for analog hardware units. Due to this

demand, there has been a lot of research related to Neural Networks being applied to this issue. An important property for the neural network is the inference time, since the emulation should work in real-time. Recurrent Neural Network (RNN) architectures have proved to be sufficient for real-time tasks.

This Master's Thesis aims to create a system identification model (digital twin) of a stereo multimedia loudspeaker system recorded by a dummy head. The model will be created using RNN. The digital twin will be further used in the fine-tuning process of the stereo multimedia loudspeaker system.

1.2 The Scope of this Research

In the research, an RNN architecture proposed by [3] will be adapted to learn stereo raw waveform data.

An *input* will be played through the loudspeaker system, and a *target* will be recorded. Both the *input* and *target* are 2-channel (stereo) audio.

The model does not have any insight into the loudspeaker system itself, so the modelling could be defined as a black-box problem for learning a non-linear transfer function. A non-linear transfer function is a mathematical relationship between input and output signals that does not satisfy the property of superposition. In other words, the output of the system is not proportional to the input, and the relationship between them can be complex, exhibiting behaviors such as saturation, compression, distortion, and frequency-dependent effects.

The RNN parameters will be optimized by the Error-To-Signal-Ratio (ESR) criterion and will try to replicate the recording (*target*). The model is trained on waveform data.

The loudspeaker system (source) is recorded by a dummy head (receiver) at a distance of 0.30 m in between. The recording is made in an anechoic chamber. The model is dedicated to learning the whole system, i.e., the impulse response of the loudspeaker system and Head-Related Transfer Functions.

Due to the loudspeaker-microphone distance in the measurement setup, a time delay between the *input* and *target* is present, which degrades the model's training performance. A band-passed (40-18000 Hz) impulse is placed at the beginning of the *input* data before the recording. The impulse response is recorded in the *target* data. The *input* and *target* are time-aligned by using cross-correlation between the band-passed impulse in the *input* and its recorded response in the *target*.

A new data management method that allows for the correct initialization of the initial hidden state of the RNN will be presented. Each training sequence starts at the sample when the previous sample's hidden state represents the loudspeaker's *rest state* and can be initialized to a zero vector. The *rest state* refers to the state of the black-box when it is not being excited, which means that there is silence in the input and output of the black-box; no input signal is moving the membrane of the loudspeaker.

1.3 Research Goals and Questions

The goal of the research is to create a system identification model of a multimedia stereo loudspeaker using an RNN architecture. The training dataset consists of stereo music as the *input* and binaural recording of multimedia stereo loudspeaker playback as the *target*. The main research questions are:

- Which properties of the transfer function are challenging for an RNN to learn?
- Can the hidden state of the RNN be initialized better by modifying the training data preparation according to the hyperparameters of the RNN's architecture?

1.4 Structure

Theoretical background of loudspeaker modelling using Recurrent Neural Networks (RNN) is discussed in Section 2. The practical implementation of the model and a new approach to hidden state initialization are presented in Section 3. Additionally, the recording, processing, and properties of data for RNN modelling are described in Section 3. Results based on different datasets or training approaches are shown and organized in Section 4. The Section 5 proposes possible improvements for the implemented method, and the main outcomes are summarized in Section 6.

2 Theoretical Background

Section 2 introduces the reader to the theoretical background of the two main domains necessary for implementation: loudspeaker modelling and raw audio waveform modelling with RNNs.

2.1 Loudspeaker Modelling

Loudspeaker models allow for estimating the effects of adjustments without physically modifying the system. Olive [4] conducted a listening test and concluded that the most preferred loudspeakers among experienced listeners had the smoothest, flattest, and most extended frequency responses, maintained uniformly off the listening axis. To achieve such a design goal, system identification (creation of a digital twin) could be very beneficial. Simultaneously achieving accurate and efficient loudspeaker system identification presents a broad range of challenges [5], [6], [7]. Nonlinear loudspeaker models are valuable for various purposes, such as driver design, loudspeaker equalization, linearization, or virtualization [8, 9, 10].

2.1.1 Mathematical Models of Nonlinear Loudspeaker Modelling

Loudspeakers are characterized by coupled multiphysical phenomena, including mechanical, magnetic, electrical, thermodynamic, and acoustic aspects. The non-ideal transduction process often results in nonlinear distortion, which significantly affects sound quality. Nonlinear distortion is particularly pronounced when a loudspeaker is driven by high-magnitude, low-frequency signals [11, 12]. When small electrodynamic loudspeakers are operated close to their maximum rated power, they exhibit significant objectively measurable nonlinear distortion.

Wiener model It is possible to approximate nonlinear input-output relationship by decomposing it into two or more interconnected elements. In the Wiener modelling approach, the dynamics of nonlinearity can be captured by combining a linear transfer function that passes the signal to a static nonlinear function. A basic memory-less Wiener model is shown in Fig. 1. The $H(z)$ in Eq. (1) represents a linear transfer function in the Z-Transform domain [13]:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-O}}, \tag{1}$$

where the highest value between M and O denotes the transfer function length. The absolute value of the power of z denotes the necessary amount of delay taps for the input sample $x(t)$ before it gets multiplied with corresponding gain coefficients a in recursive path and b in non-recursive path in digital filter implementation. The transfer function length and the values of the coefficients depend on the expected dynamics of the Wiener structure. The choice of N can vary based on implementation, e.g, hyperbolic tangent, sigmoid function, etc., could be used. Most commonly

exploited N in Wiener models are the power series:

$$\hat{y}(t) = N(x'(t)) = \gamma_1 x'(t) + \gamma_2 (x'(t))^2 + \cdots + \gamma_i (x'(t))^i, \tag{2}$$

where γ_i is the gain coefficient for the i-th term in the power series. The value of i is a design choice for the Wiener model.

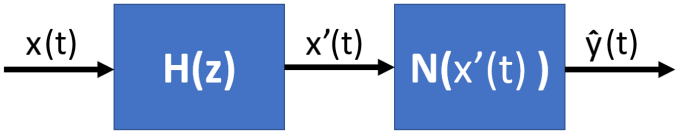

Figure 1: *Block diagram of the Wiener model in discrete time.*

To perform system identification with Wiener model an input signal of system $x(t)$ and corresponding output signal of system $y(t)$ is necessary. Wiener model for optimization can be denoted as:

$$x'(t) = \mathcal{Z}^{-1}(H(\mathcal{Z}(x(t)); \mathbf{b}, \mathbf{a})), \tag{3}$$

$$\hat{y}(t) = N(x'(t); \boldsymbol{\gamma}), \tag{4}$$

where $\mathbf{b}$, $\mathbf{a}$ and $\boldsymbol{\gamma}$ are parameter vectors for optimization. $\mathcal{Z}$ denotes Z-Transform and $\mathcal{Z}^{-1}$ denotes inverse Z-transform. The goal of optimization is to model $\hat{y}(t)$ as close as possible to the real system's output $y(t)$ [14]. These models have a convenient block representation, a transparent relationship to linear systems, and are easier to implement compared to more complex nonlinear models like Volterra models. However, they do not take into account the previous state of the system or the input from previous time steps.

State-space Modelling The loudspeaker signal can be viewed as time-series data. Time-series can be seen as being generated by systems that transform information from the past and present into future observations. In the most general case, such modelling can be described by:

$$z_{t+1} = f(z_t, x_t), \tag{5}$$

where the system's future state z_{t+1} is described by the function f, z_t represents the system's current state, and x_t is the exogenous input to the system. Eq. (5) is an example of a state (space) representation of z_t [15]. State-space modelling can be used for nonlinear loudspeaker modelling by finding a polynomial nonlinear state-space model [16].

Volterra Series Expansion The nonlinear behavior of a loudspeaker mainly depends on the excursion of the voice coil, and the governing differential equations can be written as:

$$u(t) = Ri(t) + L\frac{di}{dt} + Bl\frac{dx}{dt}, \tag{6}$$

$$Bli(t) = m\frac{d^2x}{dt^2} + r\frac{dx}{dt} + kx(t), \tag{7}$$

where $u(t)$ is the input voltage, $i(t)$ is the input current, $x(t)$ is the cone displacement, Bl is the force factor, L is the voice coil inductance, k is the nonlinear function of the displacement x, and R, m, and r are the electromechanical parameters of the loudspeaker. The nonlinear differential equation can be derived by approximating terms from Eq. (6) and (7) with a second-order truncated power series to capture the nonlinear behavior of the voice coil:

$$Bl = bl_0 + b_1x + b_2x^2, \tag{8}$$

$$k = k_0 + k_1x + k_2x^2, \tag{9}$$

$$L_E = L_{E_0} + l_1x + l_2x^2. \tag{10}$$

These nonlinear Differential Eqs. (6) and (7) can be solved using Volterra series expansion [17]. The discrete-time expression for the Volterra series expansion is shown in Eq. (11). In the Volterra series, the output of the nonlinear system depends on the input to the system at all previous time steps in a causal system.

$$y(t) = h_0 + \sum_{p=1}^{P}\sum_{\tau_1=a}^{b}\cdots\sum_{\tau_p=a}^{b} h_p[\tau_1,\ldots,\tau_p]\prod_{j=1}^{p} x[t-\tau_j], \tag{11}$$

The $h_p(\tau_1,\ldots,\tau_p)$ are called discrete-time Volterra kernels. The variable x represents the input of the system at the current discrete time step t, and τ_j allows for looking at previous input values, depending on which Volterra kernel is used. If P is finite, the series operator is said to be truncated. If a, b, and P are finite, the series operator is called a doubly finite Volterra series. If $a \geq 0$, the operator is said to be causal. Volterra kernels are generalized nonlinear impulse response coefficients that hold the information about the "memory" of the system. Solving the Volterra series individually is complicated because the basis functionals of the Volterra series are correlated.

2.1.2 White-box and Black-box Modelling

System modelling methods can be categorized into two classes based on how knowledge of the system's building blocks is incorporated in the modelling process.

***White-box* modelling** This modelling approach utilizes the physical insight of the system. The first attempts at such modelling were made by using lumped electrical equivalent circuits and their descriptive differential equations [18]. Early work on the *white-box* modelling approach was done by Thiele and Small, who discussed the linear modelling of drivers and their enclosures when the loudspeaker is excited by low-amplitude signals [19, 20]. Later, circuital models were extended to describe

the nonlinear loudspeaker behavior under the excitation of high-amplitude signals, where different nonlinearities could be characterized by polynomial functions of the voice coil [11]. The previously mentioned approach with Volterra series expansion also utilizes insights into the system's mechanics [17].

More modern *white-box* modelling approaches for loudspeaker modelling are described in [7], where Wave Digital Filters (WDF) are used. WDF represents an equivalent circuit of a nonlinear loudspeaker in discrete-time and is implemented as a more efficient method of simulating the speaker. The results were compared to SPICE equivalent circuits and showed promising outcomes, supporting the effectiveness of this approach for loudspeaker virtualization.

***Black-box* modelling** In *black-box* modelling, there is no information available about the physical parameters of the system. These types of models solely rely on the *input* signal and the output *target* signal of the device for system identification. There are several approaches to *black-box* identification.

NARMAX modelling [21] utilizes sampled input and output data of the loudspeaker to optimize function parameters based on Akaike's information criterion. This approach aims to learn a mathematical model that accurately reproduces the loudspeaker's output for a given input.

Volterra series can also be utilized in *black-box* modelling. In [8], a second-order Volterra filter approximation is employed. The proposed filter structure, called multi-memory decomposition (MMD), consists of three linear FIR filters and one multiplier. The coefficients of MMD are determined with respect to a second-order reference Volterra kernel. Block-oriented adaptive algorithms are proposed to calculate the MMD weights using input and output measurements of the system. The parameters of the Volterra model are obtained by minimizing the Mean-Squared-Error (MSE).

A separate research area has been dedicated to the modelling of nonlinear audio effects and amplification simulations, which incorporates valuable principles and approaches applicable to the topic of loudspeaker modelling [3, 22, 23, 24].

2.1.3 Receiver and Propagation Path from the Source

By viewing loudspeaker modelling from a system identification method's perspective, then a measurement of the loudspeaker's output (*target*) corresponding to the *input* should be recorded to create a mathematical model, which can replicate the system. The resulting system's impulse response is not only the loudspeaker itself. In addition, the receiver's and the propagation path's response is a part of the system under the modelling process.

In this thesis project, the source is a two channel loudspeaker system. The goal is to create a model, which can replicate binaural recording of source. The source is recorded by a dummy head. Thus, the impulse response consists also of Head-Related Transfer Function for each recording channel.

Another aspect is that the propagation path adds time delay due to the sound propagation through air. This significantly increases the length of total system's impulse response.

Source Directivity Sound sources have a specific radiation pattern that changes over different wavelengths λ. Additionally, the calculation method of the radiation pattern depends on the distance chosen from the radiator.

Let us assume that the radiator is a piston. In close distances, referred to as the source's near-field, the sound field is more complicated. The sound wave is spherical. The particle velocity and pressure have a change in their phase difference over the distance from the piston. At the piston's surface, the phase difference is $\frac{\pi}{2}$ radians. It gradually decreases and reaches 0 radians at the far-field. The distance of the far-field for λ can be estimated by the inequality:

$$r > \frac{\pi a^2}{\lambda}, \tag{12}$$

where r is the distance from the piston, λ is the wavelength, and a is the surface area of the piston. To ensure the far-field, $r >> a$.

In the far-field, r is large enough to approximate the sound wave as a plane wave. The sound pressure at point M within distance r from the piston and θ as the angle between the acoustical center of the piston and the unit vector pointing towards the piston's center from point P could be found by:

$$M_{p(r,\theta)} = A\pi a^2 \frac{2J_1(ka\sin\theta)}{ka\sin\theta}\frac{e^{ikr}}{r}, \tag{13}$$

where A is the pressure amplitude, J_1 is the Bessel function of order 1, k is the wavenumber, and a is the surface area of the piston [25].

In the case of a stereo multimedia loudspeaker, the approximation with a piston might be found too rough [26], and the stereo source (two moving radiators close to each other) adds another degree of complexity for radiation patterns due to inference.

If the sound source is recorded at a single point, the directional characteristics could be modeled with directional filtering. This would require a new equation adjusted for the particular source. Recording at one point suppresses the reality of sound sources emitting sound from different parts of their body [27]. Relatively far away from the source ($r >> a$), complex source geometry is negligible. The estimation of r for the far-field is also dependent on λ. During the transition from the near-field to far-field, the radiation from different body parts could still be important. If the source is recorded with multiple microphones, then the source directivity could be combined with multiple point source interpretations. For stereo source modelling, at least two point source interpretations are necessary, which could be later processed and filtered in the manner of interest.

Head-Related Transfer Functions The human skull, outer ear, and ear canal comprise the final segments of the transmission path before sound reaches the tympanic membrane and enters the middle ear. A head-related transfer function (HRTF) describes the acoustic transfer function between a point sound source in the free-field and a specific position in the listener's ear canal. HRTFs are vital for creating immersive virtual acoustic environments reproduced over headphones or loudspeakers [28]. Although HRTFs are individual-specific, a functional generalized model can still be identified [29].

Dummy Head One of the commonly utilized microphone systems that incorporates HRTFs is known as a dummy head. The dummy head replicates a generalized human torso and head, including the outer ear and ear canal, each with their characteristic resonance, for each canal individually. Recordings made with a dummy head are referred to as binaural recordings. These recordings capture source directivity and serve as the final segment of the transmission path until the sound is captured by the microphone in the binaural recording setup.

Time Delay Alignment with Cross Correlation Cross-correlation also known as the sliding dot product is a measure of similarity of two data series, as a function of the displacement of one relative to the other. For discrete time, the cross-correlation is determined as:

$$(f \star g)(n) \triangleq \sum_{t=-\infty}^{\infty} \overline{f(t)} g(t+n), \tag{14}$$

where $\overline{f(t)}$ denotes the complex conjugate of first function $f(t)$, $g(t+n)$ is the second function and n is the displacement between two functions. For two correlated, but time delayed signals, the cross correlation is a simple method to measure the time delay. The highest function value of cross correlation appears when two signals are most similar and the displacement argument n is equal to the time delay between two signals. The discrete time delay τ_{delay} in samples is found by:

$$\tau_{\text{delay}} = |\arg\max_{t \in \mathbb{Z}}((f(t) \star g(t))(n))|. \tag{15}$$

2.2 Deep Learning in Raw Audio Waveform modelling

It is assumed that the reader is familiar with the basics of Deep Learning. Section 2.2 is dedicated to introducing the reader and providing a high-level explanation of the terminology used. The book "Deep Learning" by Ian Goodfellow [30] will be referenced as a single, relatively comprehensive source for the terminology.

2.2.1 Introduction to Terminology

The quintessential example of a Neural Network is a single-layer feed-forward perceptron, which can be described by:

$$\hat{y} = f(x; \theta), \tag{16}$$

where $\hat{y}$ represents the output, x is the input, and f is the function of the perceptron with its learnable parameters θ.

If the system's response y (also called *target*) is measured using an input x, then the model's *output* $f(x; \theta)$ is denoted as $\hat{y}$. During the training of the network, the parameters θ are optimized to adjust the model's *output* in order to meet the defined criterion, which is determined by the loss function. One of the most commonly used optimization objectives is to minimize the difference between y and $\hat{y}$:

$$\arg\min_{\mathcal{L}_\theta} \mathcal{L}(y, f(x; \theta)), \tag{17}$$

where $\mathcal{L}$ could represent the Mean-Squared-Error between the arguments.

A widely used training method is back-propagation, where the gradient of a loss function with respect to learnable weights θ denotes the adjustment of θ. For a single perceptron, the parameter values for the next forward-propagation, denoted as $\theta_{\text{fp}+1}$, could be calculated as follows:

$$\theta_{\text{fp}+1} = \theta_{\text{fp}} - \eta \mathbf{g}(\theta_{\text{fp}}), \tag{18}$$

where θ_{fp} represents the parameter values during forward pass before current back-propagation, $\mathbf{g}$ is the gradient of the loss function with respect to θ_{fp}, and η is the learning rate, which is a training hyper-parameter. The gradient of the loss function with respect to θ can be computed as:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta}, \tag{19}$$

where $\hat{y} = f(x; \theta)$ is the *output* of the model, used for computing the loss function in Eq. (17).

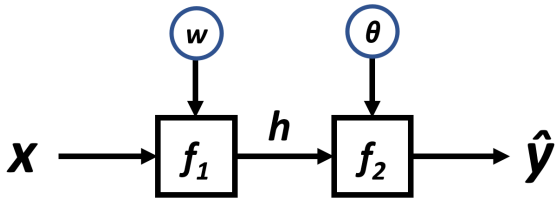

Figure 2: *Two consecutive single-layer perceptrons.* f_1 *and* f_2

It is important to mention that for multiple consecutive single-layer perceptrons, the chain rule can be applied to calculate parameter updates for a particular perceptron. In the case of two consecutive single-layer perceptrons, the chain rule can be applied:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w}, \tag{20}$$

where w are the learnable parameters of the first single-layer perceptron f_1, and h is the output value of it (see Fig. 2).

Mini-batch training Mini-batch training is a technique used in training deep learning models, where the training data is divided into smaller subsets called mini-batches. Instead of updating the model's parameters based on the loss function accumulation over single forward-propagation over the whole dataset, mini-batch training computes the gradients based on smaller subsets of data.

Each mini-batch contains a fixed number of training examples. In case of raw audio, mini-batch contains at least one sequence length of audio samples (see Fig. 3). Sequence length is a training hyper-parameter. The amount of sequences "stacked" on top of each other is called batch size, which is another training hyper-parameter.

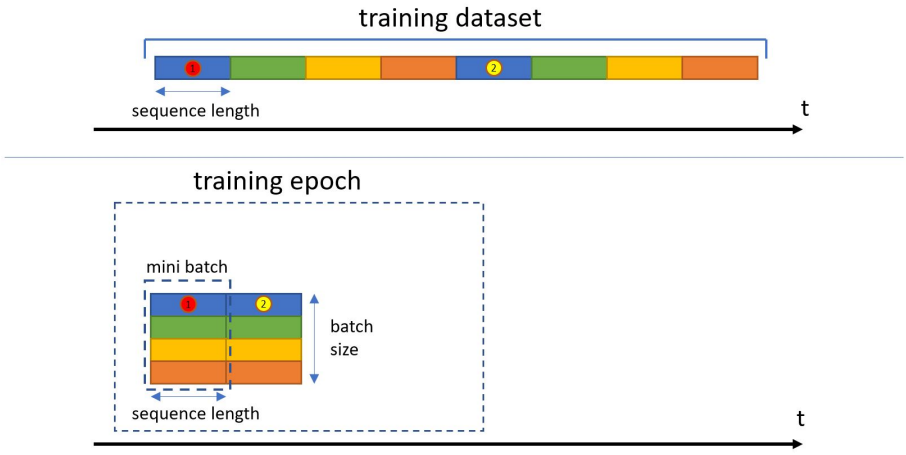

Figure 3: *Mini-batch training.*

For each mini-batch, the model performs forward propagation, where the input data is passed through the network, and predictions are generated. The model's predictions (*output*) are compared to the real system's output (*target*), and loss value is calculated to quantify the discrepancy between the predicted and actual values for mini-batch.

The gradients of the loss function with respect to the model's parameters are computed using back-propagation and indicate the direction of the parameter updates required to minimize the loss. The magnitude of update rule is further controlled by optimization algorithm (such as Stochastic Gradient Descent [31] (SGD) or slightly modified version ADAM optimizer [32]) based on the computed gradients. The update rule adjusts the learnable parameters to reduce loss.

Forward-propagation for different mini-batches, loss calculation, back-propagation and learnable parameter updates are repeated for until all mini-batches in the training data have been processed. This completes one epoch of mini-batch training. Multiple epochs may be performed to further refine the model.

Mini-batch training offers several advantages. It provides a balance between the efficiency of SGD (which updates parameters after each batch) and the stability of single batch gradient descent (which updates parameters after processing the entire dataset). By utilizing different batch sizes, the training process can benefit from parallel processing, memory optimization, and more stable parameter updates. Parallelism is important, especially considering that the audio data has high sampling rates and high resolution, resulting in a large amount of data within a small time frame [33, 30, 34].

2.2.2 Recurrent Neural Networks and Their Architectures

Recurrent Neural Networks (RNNs) are a family of Neural Networks for processing sequential data $x_1, x_2, ..., x_{t-1}, x_t$. RNN is sharing the same learnable parameters between the time steps. This fact introduces the memory mechanism and allows the

network to learn time dependencies over sequential data.

Recurrent Neural Network Let us consider a dynamic system driven by an external force x_t:

$$s_t = f(s_{t-1}, x_t), \tag{21}$$

where s_t denotes the current state of the system, and f takes the system's state from the previous time step s_{t-1}. To learn such a system with an RNN, we use the following equation:

$$h_t = f(h_{t-1}, x_t; \theta), \tag{22}$$

where h_t denotes the current hidden state, f is the RNN that takes the previous hidden state h_{t-1} and the current input x_t, and uses the learnable parameters θ, which are shared across all time steps. The hidden state dimensionality is defined by the RNN hyper-parameter *hidden size*. RNNs perform an embedding from the *input* dimensions to the *hidden size*.

The hidden state is often used as an RNN output. An affine transformation of the hidden state to the *output* dimensions can be performed by a multilayer perceptron (MLP) to calculate the loss value between the *output* and the *target*. An MLP is a type of Neural Network that consists of multiple layers of interconnected single-layer perceptrons (see Section 2.2.1). For this particular affine transformation, at a minimum, a two-layer MLP could be used with input dimensions corresponding to the *hidden size* and output dimensions corresponding to the *target* dimensions per time step.

The Eq. (22) can be visualized as shown in Fig. 4. This RNN processes information from the input x by calculating h. On the left side, when the RNN cell is "unfolded", it represents the training process. On the right side, the RNN cell is "folded" to show the recurrence path during sample-by-sample filtering after training (inference).

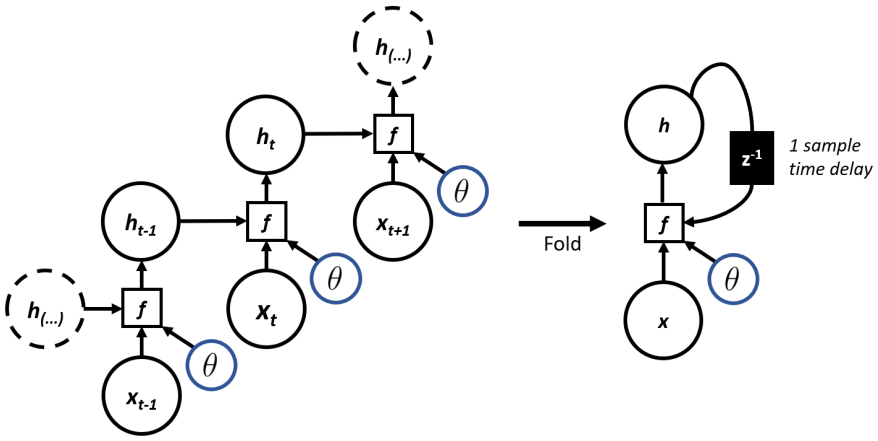

Figure 4: *"Unfolded" and "folded" representation of an RNN.*

The RNN structure introduces two major advantages due to its recursive nature. Firstly, the sequence length of mini-batch can have different lengths because the RNN

can "unfold" as many times as there are time steps. Secondly, it is possible to use the same transition function f with shared parameters across the time steps. These two factors allow for learning a single model f that operates on all time steps for different sequence lengths. The "unfolded" graph illustrates the idea of information flowing forward in time when computing the *output* and loss [30].

Back-propagation through time The process of back-propagation through time (BPTT) is used in RNNs to train the network by "unfolding" it in time (see Fig. 4). During BPTT, starting from the last time step, the gradients of the loss function are propagated backwards through the "unfolded" network with respect to the shared parameters θ at each time step. The gradient calculation for a particular time step is done using the chain rule (see Eq. (20)), and each time step can be interpreted as a new copy of the network when BPTT is performed (the parameters θ remain shared). This leads to increasingly long chains of gradient calculations towards the beginning of the sequence. For long sequences, this is computationally expensive and also leads to the vanishing gradient problem, which is further explained in Section 2.2.2. Another drawback is the infrequent gradient updates, as only one back-propagation is performed per mini-batch. This suggests using short sequences for training, although there is specific advice on choosing the shortest sequence length for modelling a particular system. At the beginning of a new forward propagation with a new sequence, the output (hidden state) of the previous time step is not taken into account. This results in a discontinuity in sequential modelling. The advice is to pick a long enough mini-batch sequence that includes at least one full length of the modeled system's impulse response [3].

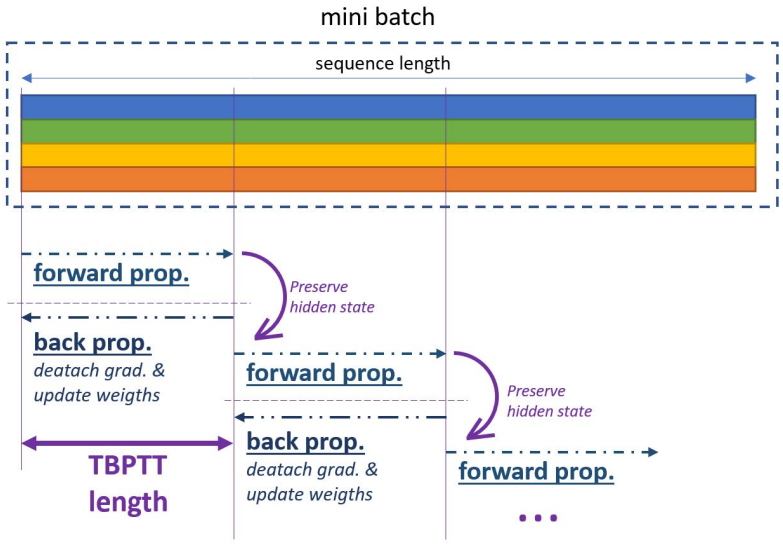

Figure 5: *Truncated Backpropagation Through Time (TBPTT) in minibatch training.*

An alternative approach to BPTT is Truncated Backpropagation Through Time (TBPTT), depicted in Fig. 5 TBPTT breaks the mini-batch sequence into shorter segments or chunks during training. Instead of back-propagating through the entire

sequence, TBPTT back-propagates through a truncated segment of fixed length, which is defined as the network's hyper-parameter. The "unfolded" segment is then treated as a separate subsequence, and the gradients are computed and applied within that subsequence, which can be referred to as the TBPTT length. The hidden state of the RNN is preserved between segments to carry forward information and to avoid discontinuity in sequential data. TBPTT does not provide the precise gradient value for the whole mini-batch, but it gives an approximation that is used to update the network's learnable parameters. Although the hidden state is preserved, it is still advised to choose a TBPTT length longer than the system's impulse response to ensure that the impulse response is not truncated [3].

Problems with RNN training The "unfolded" RNN is trained by applying back-propagation over multiple time steps. This involves calculating the overall error gradient, which is the sum of the error gradients at each individual time step (see Section 2.2.1). In this paragraph the time step notation is put in the subscript for convenience.

If we take a total of T time steps, the gradient update is given by the following equation:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W}, \tag{23}$$

where W is the learnable weight matrix. By utilizing the chain rule, we can determine the overall loss gradient through the following computation:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \overbrace{\frac{\partial h_t}{\partial h_k}}^{\bigstar} \frac{\partial h_k}{\partial W}, \tag{24}$$

where $\bigstar$ marks derivative of the hidden state at time t with respect to the hidden state at time k. This term involves products of Jacobians $\frac{\partial h_i}{\partial h_{i-1}}$ over subsequences linking an event at time t and one at time k given by:

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}. \tag{25}$$

Calculation of RNN hidden state which is also used as the RNN output for loss calculation is stated by:

$$h_t = f_{\text{nonlin}} \left(W_{hx} x_t + W_{hh} h_{t-1} \right), \tag{26}$$

where W_{hx} and W_{hx} are parameter matrices, h_t is the current hidden state, h_{t-1} is the previous hidden state, x_t is the current input and f_{nonlin} is the activation function.

The products of Jacobians in Eq. (25) features the derivative of the term h_t w.r.t h_{t-1}, i.e. $\frac{\partial h_i}{\partial h_{i-1}}$ which when evaluated on Eq. (26) yields to:

$$\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^{\top} \text{diag} \left[f'_{\text{nonlin}} \left(h_{i-1} \right) \right] \tag{27}$$

If eigen-decomposition is performed on the Jacobian matrix $\frac{\partial h_i}{\partial h_{i-1}}$ given by $W^\top \text{diag}\left[f'\left(h_{t-1}\right)\right]$, it leads to eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_t$ and corresponding eigenvectors $v_1, v_2, \cdots, v_n$. Any change on the hidden state between two time steps $\Delta h_t = h_t - h_{t-1}$ in the direction of a vector v_i has the effect of multiplying the change with the eigenvalue associated with this eigenvector $\lambda_i \Delta h_t$. The product of these Jacobians as seen in Eq. (25) implies that subsequent time steps, will result in scaling the change with a factor equivalent to λ_i^t. Looking at the sequence $\lambda_i^1 \Delta h_1, \lambda_i^2 \Delta h_2, \cdots \lambda_i^n \Delta h_n$ it is possible to see that the factor λ_i^t will end up dominating the Δh_t, because this term grows exponentially for increasing t. This means that if the largest eigenvalue $\lambda_1 < 1$ then the gradient will vanish while if the value of $\lambda_1 > 1$ the gradient explodes [35]. Methods for preventing vanishing and exploding gradients are mentioned in [30, 36].

Gated RNNs Most effective RNN architectures have gated memory mechanisms. The two most widely used architectures are Long Short-Term Memory (LSTM) units and gated recurrent units (GRU) units. Gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode. If the input consists of a longer sequence composed of shorter sequences, it might be useful to learn the shorter sequences and forget the old state. The gate mechanisms of LSTM and GRU units are used to learn when to forget the old state.

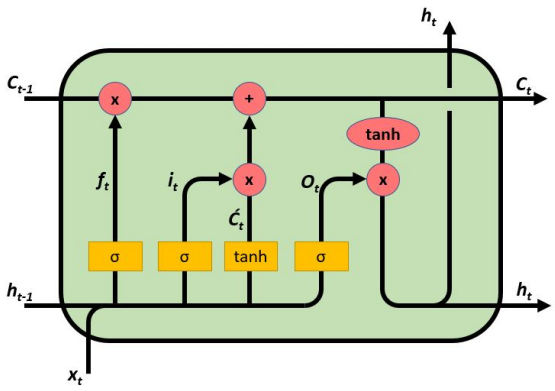

Figure 6: *Architecture of an LSTM cell.*

The LSTM unit's state consists of two vectors: the cell state c and the hidden state h. At each time step, the inputs are the current time step input x_t, the cell state from the previous time step c_{t-1}, and the hidden state from the previous time step h_{t-1}. The LSTM produces two outputs: the updated hidden state h_t and the updated cell state c_t (see Fig. 6). The output of the LSTM can be calculated using the following equations:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}), \tag{28}$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}), \tag{29}$$

$$\tilde{c}_t = \tanh(W_{ic}x_t + b_{ic} + W_{hc}x_{t-1} + b_{hc}), \tag{30}$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}), \tag{31}$$

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t, \tag{32}$$

$$h_t = o_t \tanh(c_t), \tag{33}$$

where i_t is the input gate, f_t is the forget gate, $\tilde{c}_t$ is the candidate cell state, o_t is the output gate, tanh is the hyperbolic tangent function, and σ is the logistic sigmoid function.

The LSTM unit's state is determined by eight weight matrices and eight bias vectors (if bias is used), denoted by W and b respectively. These weights and biases are the learnable parameters of the LSTM unit, which are learned during training. The dimensions of the weight matrices are determined by the channel amount in the input and the hidden size.

The GRU is an alternative gated RNN unit [37]. The GRU unit's state consists of a single hidden state vector, h. At each time step, the inputs are the current time step input x_t and the initial hidden state h_{t-1}. The GRU is depicted in Fig. 7. The hidden state of the GRU is calculated according to the following functions:

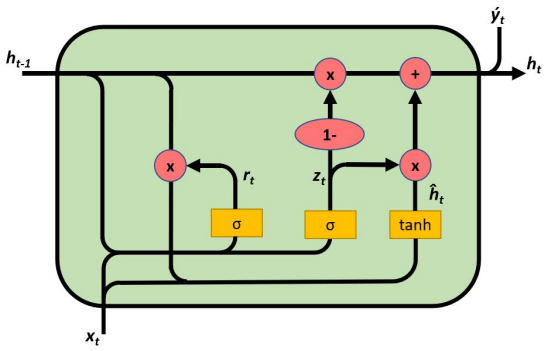

Figure 7: *Architecture of a GRU cell.*

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}), \tag{34}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}), \tag{35}$$

$$\tilde{h}_{t-1} = \tanh(W_{ih}x_t + b_{ih} + r_t(W_{hh}h_{t-1} + b_{hh})), \tag{36}$$

$$h_t = (1 - z_t)\tilde{h}_t + z_t x_{t-1}, \tag{37}$$

where r_t is the reset gate, z_t is the update gate, and $\tilde{h}_t$ is the candidate hidden state. The hidden state vector is determined by six weight matrices and six bias vectors (if used). The weight matrices and bias vectors are learnable during network training. The updated hidden state h_{t-1} is used as the initial state for time step t. The dimensionality of the hidden state is a hyper-parameter of the LSTM and GRU units. The h_t is usually used as the output of these units.

2.2.3 Loss functions

In deep learning, various combinations of optimizers are used. Good results for time series prediction with LSTM RNNs are reported by using the ADAM optimizer with MSE loss [38]. MSE is written in Eq. (38), where y_i is the *target*, $\hat{y}_i$ is the model's *output*, and N is the length of the sequence used for loss calculation:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=0}^{N-1}\left(y_i - \hat{y}_i\right)^2. \tag{38}$$

Wright et al. [39] propose using the Error-to-Signal ratio (ESR) stated in Eq. (39). By algebraic modification, it is possible to see that ESR is MSE normalized over the *target* signal's average energy. The MSE loss value depends only on the difference between the *target* and the *output*, thus it does not depend on the *target* energy. However, in the case of audio, the same MSE loss could be perceived differently depending on the *target* signal's level. In other words, a higher signal level masks the same error better than lower signal levels. Using ESR as the loss function ensures a higher penalty to the network's weights in the case of lower *target* signals for the same error value, as it is normalized over the signal's energy:

$$\mathrm{ESR} = \frac{\sum_{i=0}^{N-1}\left(y_i - \hat{y}_i\right)^2}{\sum_{i=0}^{N-1} y_i^2} = \frac{\frac{1}{N}\sum_{i=0}^{N-1}\left(y_i - \hat{y}_i\right)^2}{\frac{1}{N}\sum_{i=0}^{N-1} y_i^2} = \frac{\mathrm{MSE}}{\frac{1}{N}\sum_{i=0}^{N-1} y_i^2}. \tag{39}$$

To show the previously mentioned statements, a visualization of MSE and ESR calculations was performed. Firstly, white noise sequences with levels ranging from -78 dB to -24 dB with 50 steps were generated to imitate the signal. Secondly, white noise error sequences with levels ranging from -60 dB to -45 dB with 50 steps were generated to imitate the error. ESR values within the given signal and error levels are up to 10^8 higher than MSE values. However, this does not directly translate to 10^8 higher penalties for the model's parameters (weights and/or biases) when the ADAM optimizer is used, due to its loss function's momentum accumulation [32]. The MSE loss is not dependent on the signal level and depends only on the error level. On the other hand, ESR is highest when the error level is highest and the signal level is lowest. This difference does affect the training process, even when the ADAM optimizer's loss momentum accumulation is used, because ESR introduces a different loss function characteristic based on the signal level. In Fig.8, the MSE and ESR are normalized over their maximum values. This type of visualization shows how these two loss functions, depending on the signal-to-error ratio, affect the training with the ADAM optimizer.

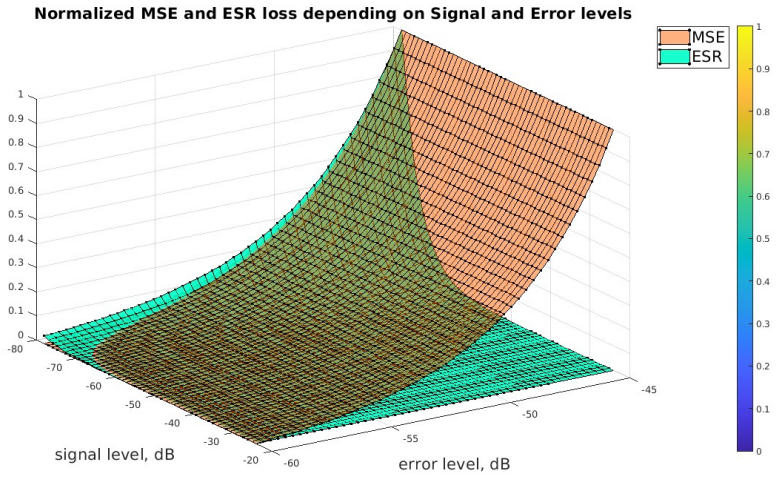

Figure 8: *Normalized MSE and ESR loss functions.*

Wright et al. [39] propose a perceptual loss function. While working with RNNs, they showed that RNNs do not necessarily converge to replicate high-frequency content well, since the signal energy there is much smaller than in low frequencies. Despite the energy difference, high frequencies are audible. By using an emphasis filter, high frequencies were learned better. Furthermore, they introduced an A-weighted ESR loss as the training criterion. This type of loss function puts more emphasis on learning content that is important for the subjective evaluation of the model.

To express the noise-robust loss function described in [40], let us begin with the given equation:

$$\mathcal{L} = \sum_{t} (\hat{y}(t) - [s(t) * h(t) + n(t)])^2, \tag{40}$$

where $s(t)$ represents the source signal, $\hat{y}(t)$ is the measured noisy target signal, $h(t)$ is the ideal impulse response, and $n(t)$ is the uncorrelated noise. By defining the residual $r(t) = \hat{y}(t) - s(t) * h(t)$, we can rewrite the loss as:

$$\mathcal{L} = \sum_{t} (r(t) - n(t))^2 = \sum_{\omega} |(R(\omega) - N(\omega)|^2, \tag{41}$$

where ω represents the angular frequency. We have now expressed the loss function in the frequency domain using Parseval's theorem. If we assume that the phase of the residue $r(t)$ is equivalent to the phase of the noise $n(t)$, we can further simplify the loss function:

$$\mathcal{L} = \sum_{\omega} (|\hat{Y}(\omega) - S(\omega)H(\omega)| - |N(\omega)|)^2, \tag{42}$$

where $\hat{Y}(\omega)$ and $S(\omega)$ represent the Fourier transforms of $\hat{y}(t)$ and $s(t)$, respectively, and $H(\omega)$ represents the Fourier transform of the impulse response $h(t)$. This $\mathcal{L}$ allows us to consider the spectrum of the noise in the loss function.

The multi-scale spectral loss, introduced in [41], addresses the limitations of point-wise loss on raw audio waveforms by considering the spectral characteristics of the signals. Given the model signal's spectrogram S_i and the target signal's spectrum $\hat{S}_i$, with a specific FFT size i, the spectral amplitude distance is defined as follows:

$$L_i = ||S_i - \hat{S}_i||_1 + \alpha||\log S_i - \log \hat{S}_i||_1, \tag{43}$$

where $||\cdot||_1$ represents the L1 norm, and α is a weighting term. The spectral amplitude distance is computed between the model signal's spectrogram and the target signal's spectrum, considering both the magnitude and logarithmic differences. To reconstruct the overall loss, the individual spectral losses are summed:

$$L_{\text{reconstruction}} = \sum_i L_i. \tag{44}$$

The authors of [41] used specific FFT sizes of (2048, 1024, 512, 256, 128, 64) and set the neighboring frames in the Short-Time Fourier Transform (STFT) to overlap by 75%. By using different FFT sizes and considering the spatial-temporal resolutions, the L_i values capture the differences between the original and synthesized audio at multiple scales. This allows for a more perceptually meaningful evaluation and comparison of the synthesized signals.

2.2.4 RNN and other architectures compared for time-series modelling

There have been several approaches to use RNNs and WaveNet architectures in time-series modelling tasks. Both methods have their advantages and disadvantages. Wright et al. [3] state that the WaveNet approach used in [23] and [22] is more precise than RNNs in exceptional cases when comparing the Error-To-Signal ratio. In terms of inference time, RNNs appear to be faster. The inference time comparison was conducted using optimized C++ implementations.

Another comparison between architectures is published in [24]. The comparison was made between Python implementations, and the results showed lower precision for WaveNet. In terms of speed, this review publishes results indicating that WaveNet works faster than RNNs during the inference stage.

From a theoretical perspective, RNNs are hardly parallelizable during training because the output sample of the RNN unit depends on the previous state. In contrast, WaveNet is parallelizable since output samples are not dependent on the previous output and are computed from independent samples of different receptive fields. However, during inference, RNNs consume less time for the task due to the same reason. WaveNet produces one output sample from a large receptive field. The large receptive field is converted to one output sample with several hidden layers, each consisting of three stages: kernel filtering, nonlinear activation, and down-pooling. This means a lot of operations per sample. For RNNs, during inference, one output sample is produced for each time step of the RNN unit operation.

Recently, there has been a publication about learning State-Space Models that can work with both advantages - parallel training and recursive generations [42]. This model operates on raw audio waveforms and is used for generative tasks, similar

to WaveNet [43]. This work also includes a theoretical comparison of RNNs and WaveNet as discussed in this paragraph.

2.2.5 Loudspeaker Modelling with Neural Networks

The dominant part of loudspeaker nonlinear behavior is the voice coil actuator. In [44], the voice coil is modeled with RNNs. 3-D modelling and then the Finite Element Method could also work, but that is computationally inefficient and does not allow for short-time loudspeaker simulations. In this paper, RNNs are incorporated into a 3-D Multiphysics simulation to increase computational efficiency. It is just a part of the actual physical model, but the RNN that simulates the voice coil is trained as a *black box*. The simulation trains the RNN for 14,000 time steps, and for the next 6,000 time steps, the LSTM predicts the transient behavior.

Recently, another *black-box* loudspeaker modelling approach using RNNs was proposed in [45]. The objective is to develop a versatile and generic nonlinear model that can be applied in industrial applications such as distortion cancellation and excursion or power limiters. The proposed model takes a digital audio signal as input and generates output of membrane displacement and voice coil current, resembling a discrete-time single input multiple output system. The training and validation signals employed in the model are explained, and data from a two-inch broadband loudspeaker driver is used for training purposes. The trained LSTM-based model is then compared to a classical state-space model that includes standard displacement-related nonlinearities.

Another domain where nonlinear modelling is often used is guitar effect modelling. In this field, Neural Networks are very often used, and architectures such as RNNs and CNN Wavenets are very popular. Both are used for modelling the *black box* of guitar stomp-boxes or tube amplifiers [3], [24]. The practical part of the Master's thesis is greatly influenced by [3], as there are a lot of principles that could be borrowed from this domain since both are dealing with nonlinear modelling.

3 Method and Data

One of the important aspects of RNN training is the initialization of hidden state. Thesis project proposes new method of hidden state initialization called Discontinuous (DISCO) sequence training (see Section 3.3). To perform system identification the data of the system's *input* and corresponding output (*target*) is necessary. The following Section 3 will also discuss the data used for the training of RNN architecture.

In the Sections 3.5 and 3.6 the two systems that produce the data for the NN to learn are discussed. In the Real Recorded data, one of the most challenging properties for network to learn is the time delay due to sound propagation from source to receiver. If the network is pushed to learn this part of total impulse response, it degrades the later part of learned response.

LSTMs and GRUs are gated RNNs and they prove to be better than Vanilla RNNs for learning time dependencies in sequential data, however it is still under a question mark, what is the upper length of time dependencies that RNN can sufficiently learn. This is one of the reasons why the Multimedia Loudspeaker Recording Simulation (MLRS) system was created (see Section 3.6). Generating data with MLRS for different properties (sample delay, SNR, etc.) and training the NN on it helps to find the optimal NN architecture and its parameters. For an example, in the original paper [3], a skip connection was introduced between input sample and the output sample to allow NN to learn only the difference between *input* and *target*. However, it will not be used in this method due to its diminishing effect when the time delay between the input sample and output sample is present.

3.1 Recurrent Neural Network Architecture in Experiment

For the experimental part of this thesis work, an adaptation of Neural Network proposed in [3] will be used. The architecture of the the network is depicted in Fig. 9.

It consists of one or more stacked RNN units and fully connected layer. In previous work, this particular network was used to learn wave forms of nonlinear guitar effects with black-box modelling approach. Modeled nonlinear guitar effects were single channel audio. In thesis work, the dataset is two channel audio - a stereo recording. Two modifications were introduced to make the previous architecture functional for stereo data. Firstly, the amount of input features were changed from 1 to 2. Secondly, the fully connected layer's output features were changed from 1 to 2, to perform affine transformation from RNN's hidden state to stereo output per time step.

For the network's training, truncated back propagation through time (TBPTT) is used. TBPTT is applied after each 1000 samples of sequence with preservation of hidden state (see Section 2.2.1).

Loss function (criterion) in the experiment is Error-to-Signal Ratio (ESR). Wright [39] was mentioning that simple ESR without using high-emphasis filtering and DC correction term leads into lack of high frequencies and less perceptual quality. In this thesis experiment simple ESR is going to be used (see Section 2.2.3), to evaluate

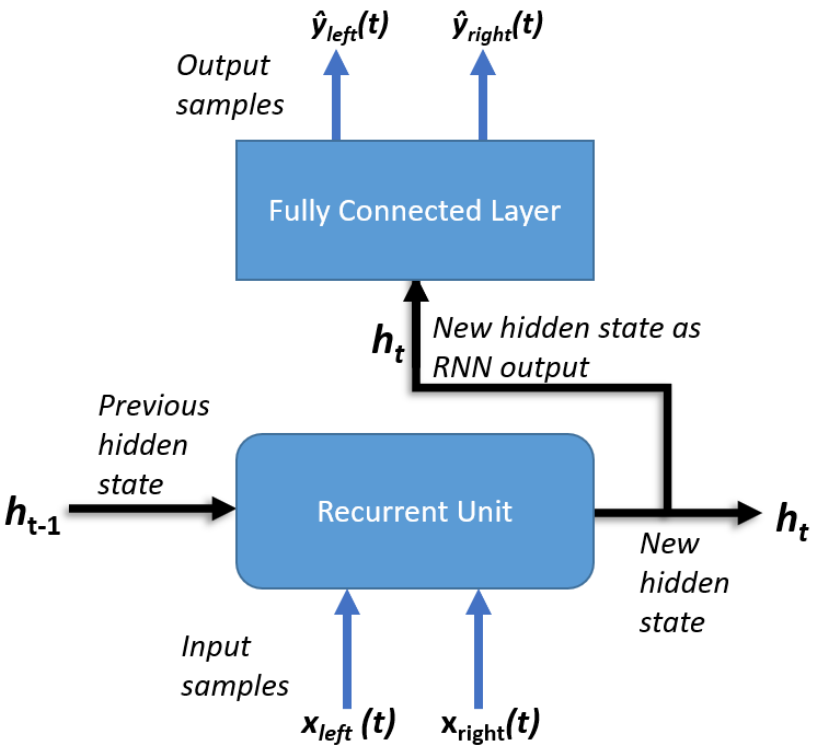

Figure 9: *Neural Network's architecture [3] is adapted for stereo audio, where x_t is the input signal, h_t and h_{t-1} are recurrent unit's hidden states and $\hat{y}_t$ is the Neural Network's predicted output sample.*

networks performance objectively.

One of the first adjustments that were made to adapt the original single channel architecture [3] was the elimination of skip connection between the *input* and *output*, which enabled to learn only the difference between the *input* and *target*. However, the skip connection was degrading the networks performance, in case there was a time delay between *input* and *target* longer than 3 samples in 48 kHz sampling rate. All the further experiments were performed without skip connection, due to the fact that *input* and *target* time-alignment method does not ensure 0 sample delay.

Table 1: Model's and training hyper-parameters

Model	Layers	Skip connection	Channels	Hidden size	Sequence length	Batch size	Epochs	Criterion	TBPTT after n samples
LSTM	2	No	2	128	4800	50	150	ESR	1000
GRU	2	No	2	128	4800	50	150	ESR	1000

Model and training hyper-parameters are shown in Table 1.

3.2 Hidden State Initialization

In the Section 2.2.2 the basic RNN was discussed:

$$h_t = f(h_{t-1}, x_t; \theta), \tag{45}$$

where h_t denotes current hidden state, f is the RNN, which intakes previous hidden state h_{t-1} and current input x_t and uses the learnable parameters θ, which are shared across all the time steps. Important aspect of RNN training is the initialization of very first hidden state h_0. A default method is to initialize it to a vector of zeros, however there are approaches which use no gradient inference (pre-run) for few hundred samples to accumulate the h_0 [3].

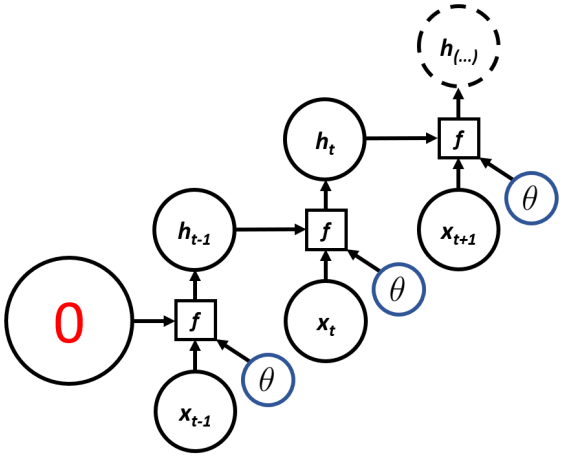

Figure 10: *RNN h_0 initialization with zero vector.*

A naive approach is to set it to a zero vector based on the Neural Network's hyper-parameter called *hidden size*, and then start training with the first sample (see Fig. 10). However, this approach fails to consider the system's state during h_0 initialization, resulting in an incorrect beginning of the training sequence if the system has not been in a *rest state* (see Section 3.3).

Another approach was presented in [3] and the idea is depicted in Fig. 11.

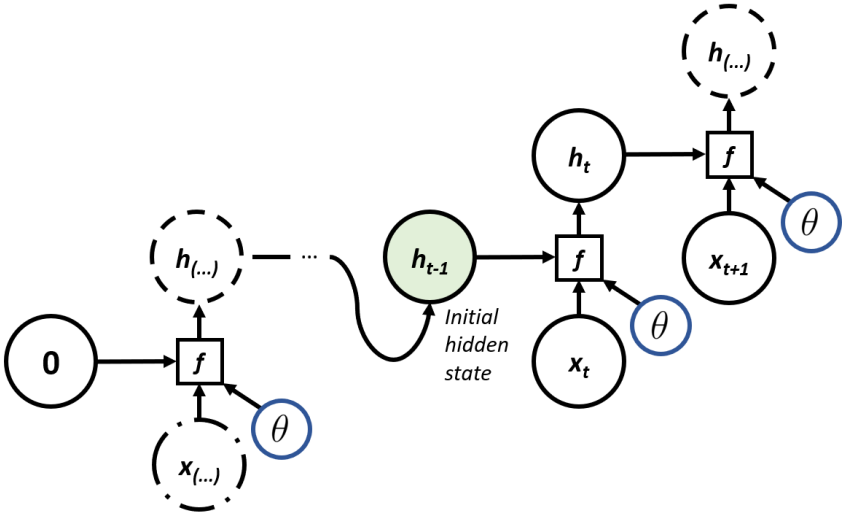

Figure 11: *RNN h_0 initialization with start up inference.*

At the beginning of the training sequence, a no-gradient startup inference is performed for a few hundred samples. This allows the hidden state to initialize properly. Once the hidden state is accumulated during this startup inference, it is referred to as the initial hidden state h_0. This initial hidden state is then used as the previous hidden state for the first sample during training with gradients.

The very first hidden state of the startup inference is also initialized as the zero vector, matching the dimensionality of the *hidden size*. However, because of the startup inference before the actual training, the impact of this initialization mistake is reduced. It is important to note that startup inference is loosing data for gradient updates.

3.3 Discontinuous sequence modelling

Discontinuous sequence modelling (DISCO) is a new method of data management (see Fig. 12) that assures correct hidden state initialization for the first time step for all training sequences (see Eqs. (46) - (57)). Let us take a look at LSTM and GRU Eq. (28) and plug the zero vectors in the input of current time step and hidden state of previous time step. The networks are not using bias parameters.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1}) \tag{46}$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1}), \tag{47}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1}), \tag{48}$$

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1}), \tag{49}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{50}$$

$$h_t = o_t \odot \sigma_c(c_t), \tag{51}$$

where σ_g is sigmoid function and σ_c is hyperbolic tangent. Silence in the input denotes $x_t = 0$. The vectors $h_{t-1} = \{0\}$ and $c_{t-1} = \{0\}$, so the Eqs.(46)-(51) could be rewritten as:

$$f_t = \sigma_g(0) = 0.5, \tag{52}$$

$$i_t = \sigma_g(0) = 0.5, \tag{53}$$

$$o_t = \sigma_g(0) = 0.5, \tag{54}$$

$$\tilde{c}_t = \sigma_c(0) = 0, \tag{55}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t = 0, \tag{56}$$

$$h_t = o_t \odot \sigma_c(c_t) = 0. \tag{57}$$

Assumed *input* values and derivations (eqs. (46) - (57)) show that the hidden state as zero vector are valid during the *rest state* and correct for the LSTM and GRU training as the initial hidden state if the black-box has been in *rest state* before first time step of training. If the black-box is a loudspeaker the *rest state* means that loudspeaker membrane is not under excitation.

A waveform-wise example is depicted in Fig. 13, where the $input'$ and $target'$ of DISCO training is depicted and compared to continuous sequence in context of training and initial hidden state initialization. The silent parts had not been excluded yet in $input'$ and $target'$.

Let us consider that the black-box has a linear impulse response (see Fig. 13). In linear example, the continuous *target* could be generated by using convolution between continuous *input* and black-box impulse response, and DISCO $input'$ and black-box impulse response respectively.

The *input* is modified - an empty vector of audio (silence) is periodically inserted after certain amount of samples corresponding to the NN's hyper parameter *sequence length* - when $input'$ data set is created. The $target'$ is recorded by using modified DISCO $input'$. After the recording, the empty parts are excluded and the $input''$ and $target''$ are used for NN's training. Original audio length is restored after the empty parts are excluded before t_n for more time steps than the impulse response length of the black-box. The *rest state* of black-box is ensured by using silence in the $input'$ while recording $target'$. No excitation means zero signal in the recording microphone, thus no signal in the training data ($input''$ and $target''$). Since the system does not include DC component, the silence in the input does not provide

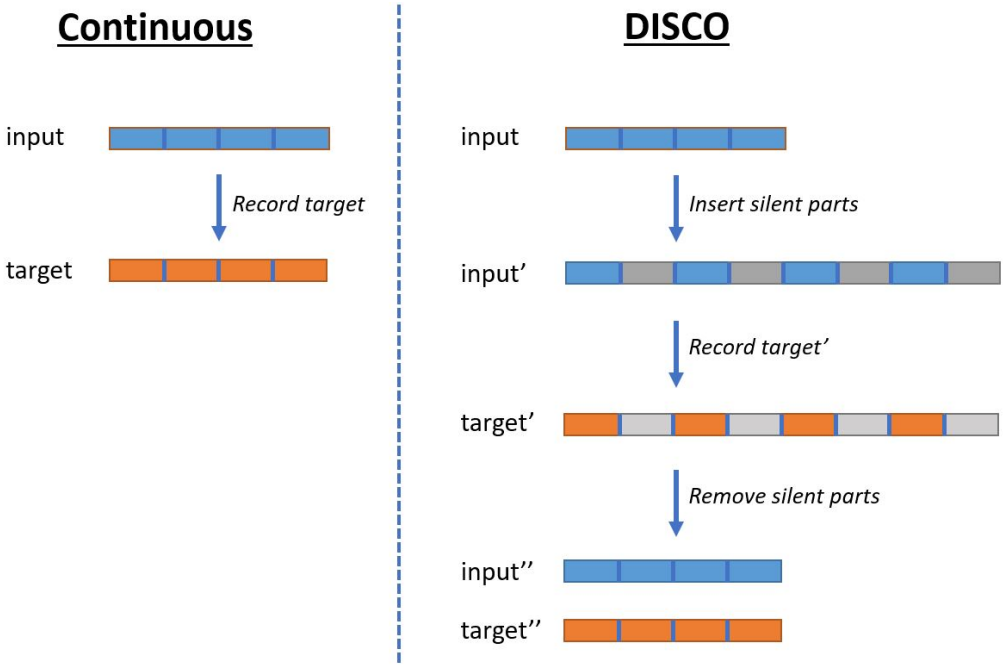

Figure 12: *Continuous and DISCO data management.*

any information about the impulse response of the black-box. Black-box starts to be excited at t_n (see Section 3.1, Fig. 9). During the *rest state* the h_0 or (h_0 and c_0) could be initialized to zero vectors with dimensions corresponding to the NN's parameter *hidden size*.

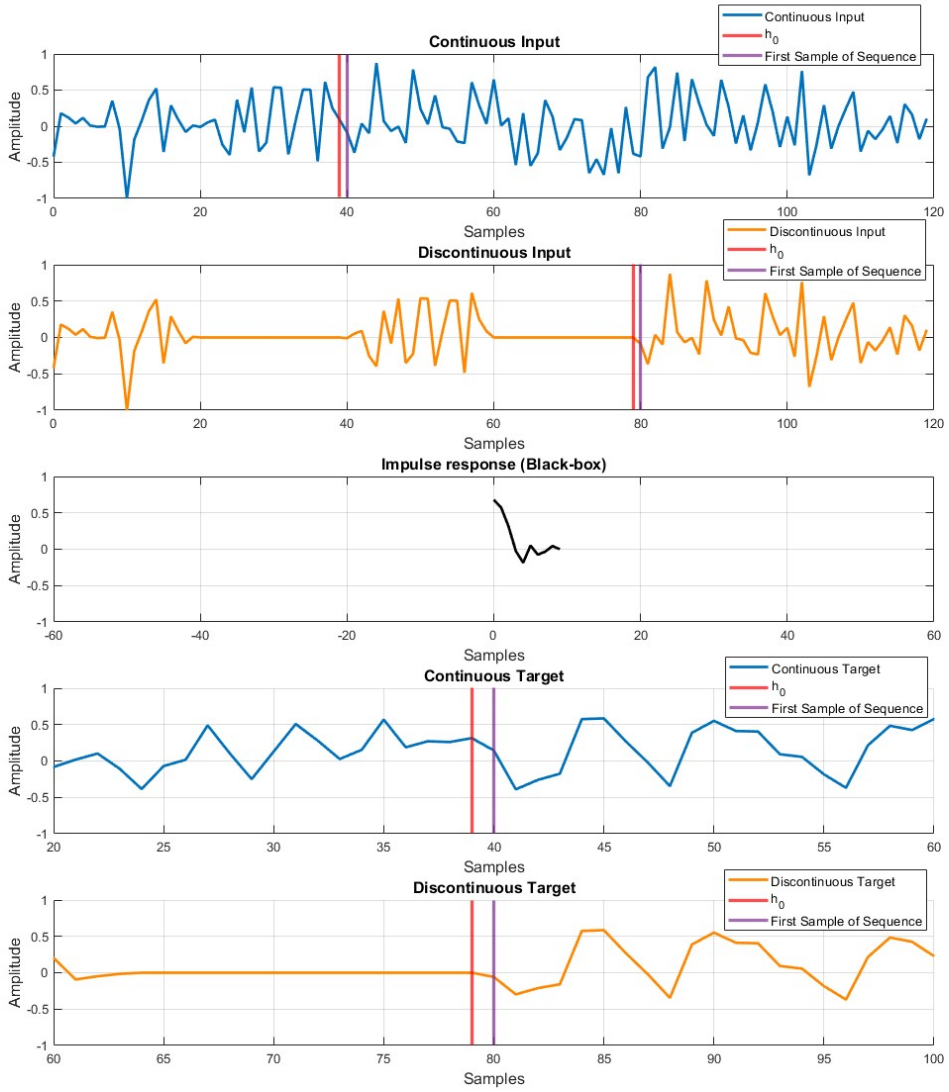

Figure 13: *Discontinuous sequence training and initial hidden state initialization compared to training with continuous sequence.*

The time steps corresponding to the h_0 (t_{39} or t_{79}) are present in $input'$ and $target'$. The signal value corresponding to the first samples of two RNN training sequences are at t_{40} or t_{80}. NN's hyperparameter *sequence length* is taken into account when DISCO $input'$ is generated and when silent parts are excluded from the $input'$ and $target'$ to create $input''$ and $target''$. In the example of Fig.13 the *sequence length* is set to 20 samples. The time steps corresponding to the h_0 is not present in $input''$ and $target''$, so that the previous hidden state can be correctly initialized to the zero vector (see Section 2.2.1 and 3.1). Method proposes that the initial (previous) hidden state vector h_0 at time step t_1 could be correctly initialized

to zero vector if the system has been into *rest state* before first this first training step t_1.

3.4 Content of Data

Two main datasets are going to be used in this experiment. Datasets have sampling frequency of 48000 Hz and are encoded in 32-bit floating - point. This type of data is used for NN's training in thesis experiment. In the training sequence, the data is organized as follows. 90% of data is used for training and validation purposes. At first the training and validation part is split into sequences, e.g. 4800 samples (100 ms). The length of the sequences is training hyper-parameter. These sequences are randomly shuffled before the first training epoch (see Section 2.2.1). After shuffling the training and validation set is divided into a half again, so that during single epoch 50% or data is used for training and 50% is used for the validating of the current epoch.

Figure 14: *Data organization during training, validation and testing.*

The rest 10% of the overall data is used for the test step, which evaluates the trained network's performance for the whole dataset after all epochs (see Fig. 14).

The short *input* dataset consists of 60 seconds of audio from "deadmau5 - Strobe" (4:00 - 5:00) that will be processed by Multimedia Loudspeaker Recording Simulation (MLRS) and will create different types of MLRS processed *target* data. The chosen musical fragment consists of transients that are wide in spectrum. This type of audio content is challenging for the network to learn. Processed *target* is going to be used for experiments with different properties of data and the results are collected in Section 4.1. MLRS is described more in Section 3.6.

The long *input* dataset is 42 minutes in length and covers most of the music genres to ensure that digital twin of loudspeaker could be used for wide variety of music. It will be used to record the multimedia device and is composed as follows:

- Training and Validation part:
  - Ultra Bra - Minä Suojelen Sinua Kaikelta (0:00 - 3:00)
  - Katedrāle - Tas Trakais Kavalieru Gads (0:00 - 3:00)
  - Dons - Izdrāz man zirdziņu (1:15 - 4:15)
  - J.S. Bach - Suite No 3 in C major (Viola) (0:15 - 3:15)
  - Sergei Rachmaninoff - Piano Concerto No 2 (8:15 - 11:15)
  - Sergei Prokofiev - Toccata op. 11 (0:00 - 3:00)
  - deadmau5 - bot (2:15 - 5:15)

  - deadmau5 - FML (0:29 - 3:29)
  - deadmau5 - Ghosts'n'Stuff (2:15 - 5:15)
  - ANÚNA - Fill, Fill a Rún (0:08 - 3:08)
  - ANÚNA - An Raibh Tú ar an gCarraig (0:37 - 3:37)
  - Dead Air - Edwin feat. Joshua Lee Turner (1:05 - 4:05)

- Testing part:
  - Ultra Bra - Minä Suojelen Sinua Kaikelta (3:00 - 3:30)
  - Katedrāle - Tas Trakais Kavalieru Gads (3:00 - 3:30)
  - Dons - Izdrāz man zirdziņu (4:15 - 4:45)
  - J.S. Bach - Suite No 3 in C major (Viola) (3:15 - 3:45)
  - Sergei Rachmaninoff - Piano Concerto No 2 (11:15 - 11:45)
  - Sergei Prokofiev - Toccata op. 11 (3:00 - 3:30)
  - deadmau5 - bot (5:15 - 5:45)
  - deadmau5 - FML (3:29 - 3:59)
  - deadmau5 - Ghosts'n'Stuff (5:15 - 5:45)
  - ANÚNA - Fill, Fill a Rún (3:08 - 3:38)
  - ANÚNA - An Raibh Tú ar an gCarraig (3:37 - 4:07)
  - Dead Air - Edwin feat. Joshua Lee Turner (4:05 - 4:35)

The recording approach for acquiring *target* is more discussed in the following Section 3.5.

3.5 Dummy Head Recording

In the experiment, a multimedia stereo loudspeaker system with two drivers is recorded with GRAS 45BB KEMAR Head & Torso simulator. Recording was done in anechoic chamber. Distance between loudspeaker system and the Dummy Head is 0.30 m. Loudspeaker system is placed at the same height as the dummy head. The acoustical center is parallel to the chamber's floor and aimed towards Dummy Head nose. Loudspeaker system was recorded at L_{Aeq} = 65dB (A-weighted equivalent loudness level) playback, measured with a sound level meter at 0.02 m distance from Dummy Head's left ear. Recording setup is generally depicted in Fig. 15. The resulting linear impulse response of the whole recording setup is about 12 ms long of which 4 ms is the time delay due to the 0.30 m distance between source and receiver. Due to the reason that dummy head does not simulate human skin, the reflections of a rigid mannequin might even more elongate the response. SNR (Signal-To-Noise ratio) of recording is 45 dB.

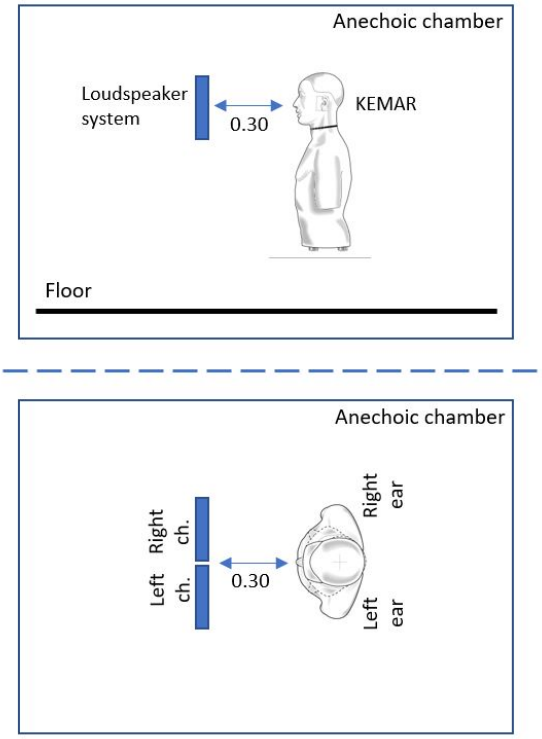

Figure 15: *Recording setup of physical system with dummy head in anechoic chamber.*

3.6 Multimedia Loudspeaker Recording Simulation

Real recorded data has distinguishable properties such as:

- Sample delay due to distance from the microphone to the loudspeaker
- Linear part of speaker's and HRTF's impulse response
- Nonlinear distortion of the drivers
- Signal To Noise Ratio due to audio recording

These properties could be simulated separately or in chosen combination by creating artificial datasets by Multimedia Loudspeaker Recording Simulation (MLRS). This approach will help to look after network's ability to learn single or a combination of real recording properties, e.g. 100 sample time delay with SNR = 30 dB.

MLRS consists of three main blocks - time delay, nonlinear Wiener structure and degradation with noise. Time delay is controllable zero pad length, that is concatenated with an input audio vector, so that after the attachment the zero pad appears in the beginning of input audio vector over the time dimension. Mathematically the concatenation could be replaced with convolution with delayed Dirac function δ (in Fig. 16, the $*$ stands for convolution). Further, the signal path enters nonlinear Wiener structure, where the signal is convolved with Linear Impulse Response (IR)

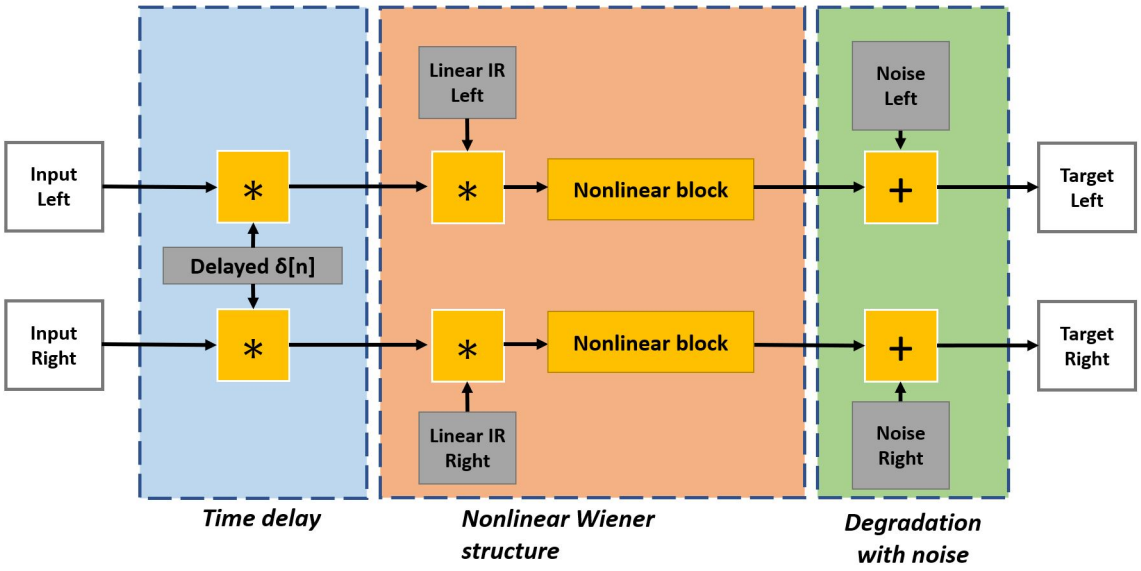

Figure 16: *Multimedia Loudspeaker Recording Simulation processing pipeline.*

which is generated from the temporally filtered white noise. The length of Linear IR is a controllable MLRS parameter. Temporal filter is written in

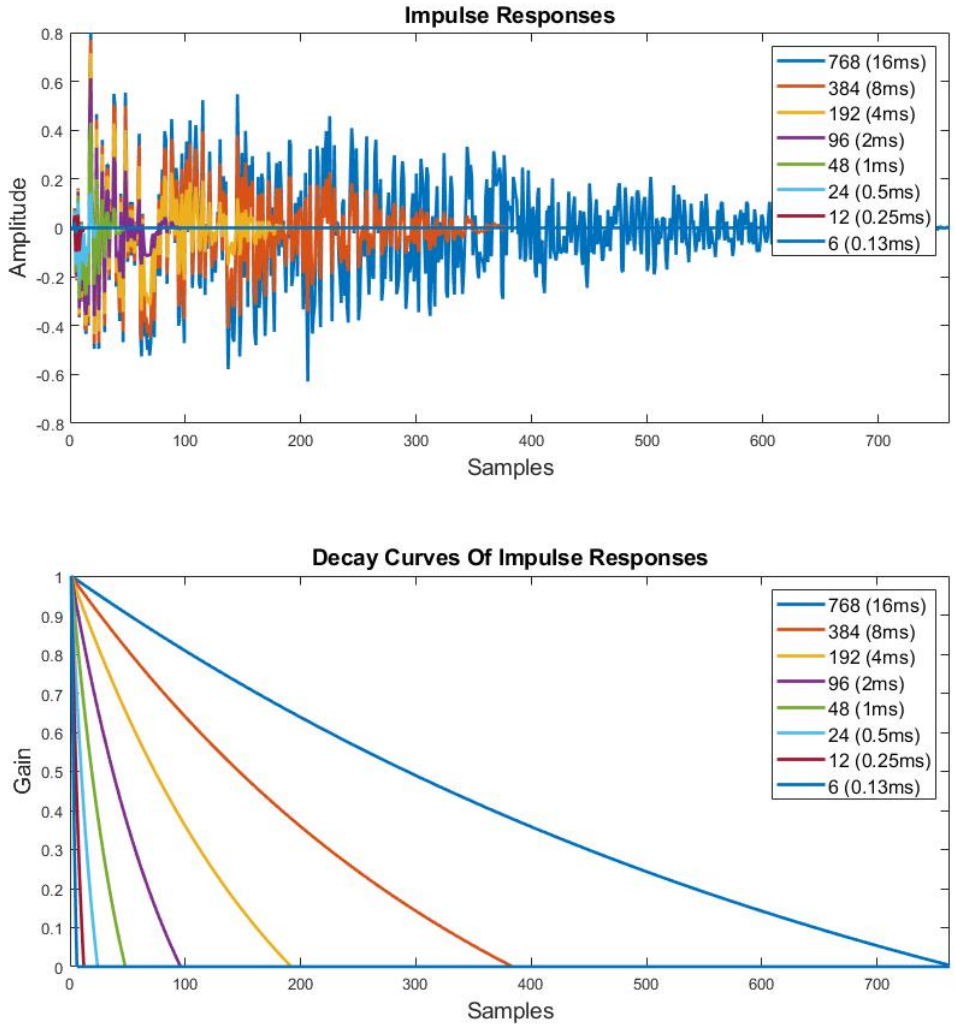

Figure 17: *MLRS linear impulse responses and corresponding decay curves.*

$$s'(t) = s(t) * \delta(t - d), \tag{58}$$

where the $s'(t)$) is the time delayed signal, δ is Dirac delta function, $s(t)$ is the input signal, n is the sample and d is the delay in samples (length of zero pad).

$$s'(t) = \tanh(c_{\tanh} \cdot s(t)), \tag{59}$$

where $s'(t)$ is the nonlinear processed signal, $c_{\tanh}$ is gain coefficient and s is the input signal. Linear IR and nonlinear function forms the Wiener structure.

After the convolution, audio is processed by nonlinear block, which applies tanh nonlinearity with controllable intensity (see Eq. (59)).

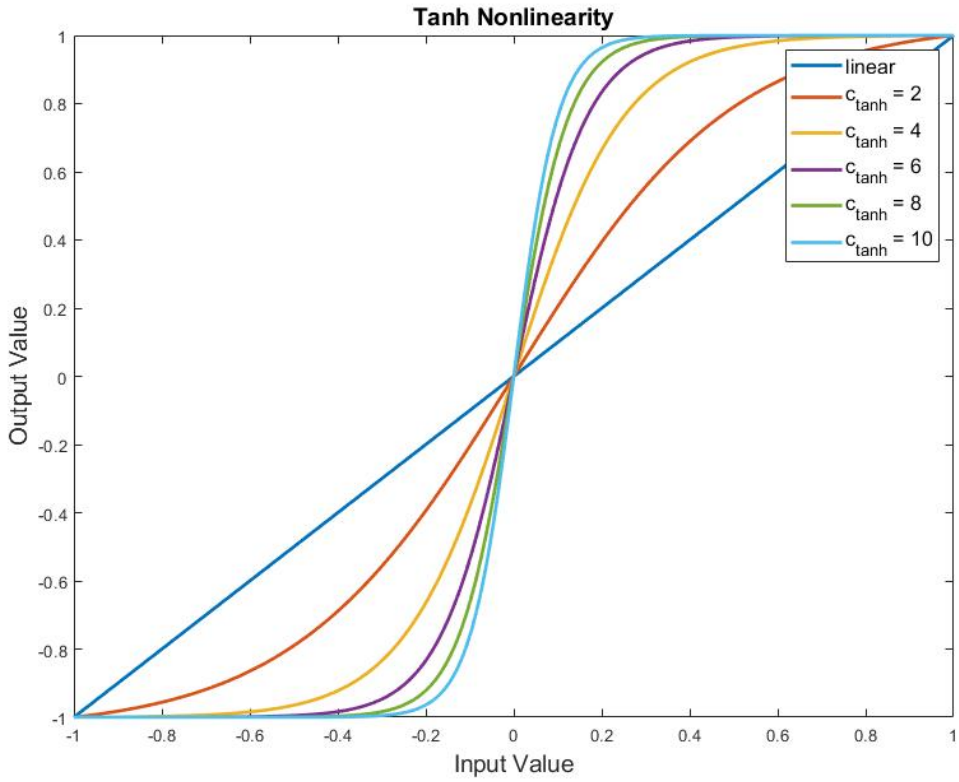

Figure 18: *MLRS tanh nonlinearity curves.*

The last block in MLRS is the degradation with noise, by using signal output of Wiener structure. Signal gets convolved with white noise, which simulates the recording noise and it is controlled by Signal-To-Noise ratio (SNR) as a MLRS parameter. A random *noise* vector is generated by drawing samples from Gaussian distribution and $s(t)$ is the signal, that is under noise contamination.

$$P_{\text{signal, dB}} = 10\log_{10}(s(t)^2_{\text{rms}}), \tag{60}$$

where $P_{\text{signal, dB}}$ is the signal power in dB.

$$P_{\text{noise, dB}} = 10\log_{10}(noise(t)^2_{\text{rms}}), \tag{61}$$

where $P_{\text{noise, dB}}$ is the noise power in dB.

$$g_{\text{noise, dB}} = P_{\text{signal, dB}} - P_{\text{noise, dB}} - \text{SNR}, \tag{62}$$

where $g_{\text{noise, dB}}$ is the necessary gain for noise in dB.

$$g_{\text{noise}} = 10^{\frac{g_{\text{noise, dB}}}{20}}, \tag{63}$$

where g_{noise} is the necessary linear gain.

$$noise_{\text{scaled}} = noise(t) \cdot g_{\text{noise}} \tag{64}$$

$$s(t)' = s(t) + noise(t)_{\text{scaled}}, \tag{65}$$

where $s(t)'$ is the signal with determined SNR. The amplitude of $s(t)'$ should be normalized between [-1; 1].

The signal and noise vectors' average powers are calculated in Eqs.(60), (61) and further used in Eq. (62) to find the g_{noise} which scales the random noise vector, so that the Eq. (65) outputs signal vector with Signal-To-Noise ratio determined by MLRS parameter SNR.

3.7 Input - Target Time Alignment

In the real recorded data a time delay between *input* and recorded *target* is introduced due to the distance between microphone and source. This delay could be eliminated by using simple delay adjustment method, that shortens the time - dependency that NN's need to learn and improves the performance. A discrete, non-fractional delay adjustment will be used.

$$\tau_{\text{delay}} = |\arg\max_{t\in\mathbb{Z}}((input(t) \star target(t))[n])|, \tag{66}$$

where $\star$ denotes cross-correlation, n is the sample delay in discrete time t.

At the beginning of *input*, a band passed (40Hz - 18kHz) impulse is placed. The *target* is recorded and black-box impulse is captured. To estimate the time delay a cross correlation is performed between *input*'s band passed impulse and *target*'s impulse response. The cross correlation maximum appears before t_0 sample. The absolute value of sample number where the cross correlation maximum appears is the time delay for the *target* (see Eq. (66)).

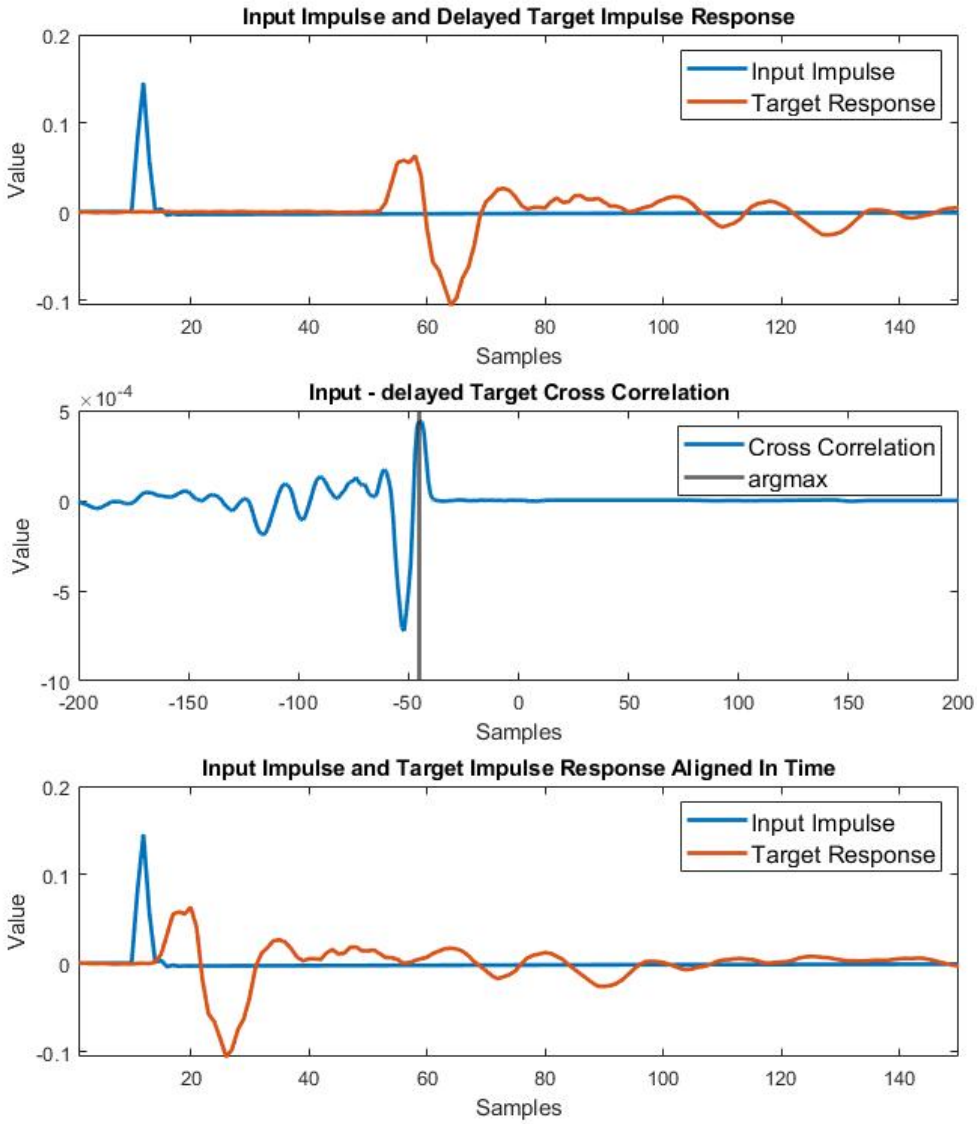

Figure 19: *Input and Target time alignment with cross correlation.*

A 5 sample ($f_s = 48000$) safe addition of delay is kept for the *target* to ensure the systems causality. From experiments, if the delay correction is relaying only on the cross correlation result, the causality might be lost, because the during the playback and propagation through air, the impulse gets low-passed and smeared in time. The $(\tau_{\text{delay}} - 5)$ is the amount of samples that is deleted from the beginning of recorded *target* signal, to ensure the *input* and *target* alignment. A zero-pad with an equivalent length is added at the end of *target* after deletion [46].

4 Results

In the following chapter the results are gathered over two different systems by using two different datasets.

The first system is the Multimedia Loudspeaker Recording Simulation (MLRS) system, which is using the Short Dataset. The results from MLRS are used for determining most challenging properties of a real recording for the network to learn. By analyzing such NN's results on MLRS, it is possible to state possible improvements for recording setup, NN's architecture and existing methods or create new methods, to treat the challenging properties.

The second group of results states the performance of NN's over Real Recorded Data. The network is evaluated by an objective measure Error-To-Signal Ratio (ESR). This is the final and most important evaluation of the network's performance and the new-found methods in this thesis project.

4.1 Neural Network's performance on raw MLRS data

Multimedia Loudspeaker Recording Simulation (MLRS) data sets were used to model speaker recording, to show model's ability to learn different features of data. Data used for MLRS *target* generation is discussed in Section 3.5. Network's architecture from Wright 2019 [3] with no skip connection (more discussed in 3.1) was used in following experiments for Tables 2., 3., 4., 5 combined in Fig. 20. Fixed model hyper-parameters are shown in Table 1. The hyper-parameters for first experiments were found empirically.

For the results the ESR loss evaluated over test part and converted to dB by $10\log_{10}(\text{ESR})$. The *target* signal is generated from input by MLRS with hyper-parameters that are shown in Tables 2., 3., 4., 5.

Table 2: LSTM and GRU performance over different impulse response lengths evaluated by ESR in dB

Model	Impulse response length in samples								
	0	6	12	24	48	96	192	384	768
LSTM	-60	-24	-30	-31	-28	-21	-20	-13	-11
GRU	-52	-24	-30	-27	-23	-21	-19	-15	-12

Other parameters:
linear response; delay = 0; SNR > 90 dB

When LSTM and GRU are trained by dataset with changing impulse response length, both models perform similarly (see Table 2). The longer the impulse response, the higher is the difference and ESR of the test set. When impulse response length is 768 samples (16ms) long the model performs the worst.

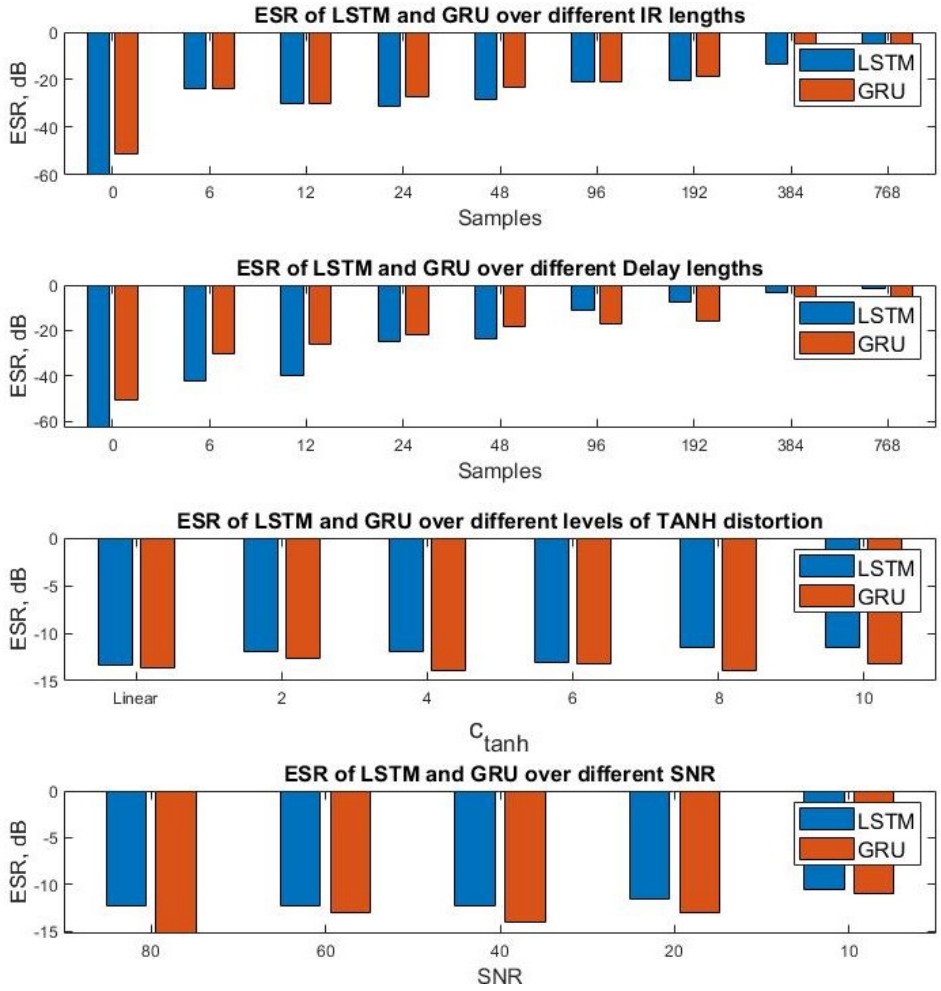

Figure 20: *Network's performance with MLRS data.*

Table 3: LSTM and GRU performance over different delay lengths evaluated by ESR in dB

Model	Delay length in samples								
	0	6	12	24	48	96	192	384	768
LSTM	-63	-42	-40	-25	-24	-11	-8	-3	-1
GRU	-50	-30	-26	-22	-18	-17	-16	-11	-9

Other parameters:
linear response; IR = 0; SNR > 90 dB

LSTM and GRU react differently on the change of delay lengths (Table 3). Up to delay of 24 samples, the LSTM learns with lower ESR, but when the time delay is longer than 48 samples, the GRU shows better ESR. The experiment shows that there is no crucial difference for the NN to learn impulse response compared to the time delay. However, the NN's learn shorter time dependencies better. It is not in experiment's interest to shorten the impulse response, but the delay reduction is possible by *input - target* time alignment and could provide better results. Delay adjustment takes place in Section 4.2.

Table 4: LSTM and GRU performance over different levels of nonlinearity evaluated by ESR in dB

Model	Tanh nonlinearity					
	0 (linear)	2	4	6	8	10
LSTM	-13	-12	-12	-13	-12	-12
GRU	-14	-13	-14	-13	-14	-13

Other parameters:
SNR = 60; IR = 384; delay = 6

MLRS distortion is regulated by c_{tanh}. This fact might bias the results, since the RNN units has the same nonlinear function in the architecture. Never the less, experiment with changing c_{tanh} shows that the network does not loose performance in terms of ESR over different strength of nonlinearity.

Table 5: LSTM and GRU performance over different SNR evaluated by ESR in dB

Model	SNR				
	80	60	40	20	10
LSTM	-12	-12	-12	-12	-10
GRU	-15	-13	-14	-13	-11

Other parameters:
$c_{tanh} = 4$; IR = 384; delay = 6

Different SNR affect LSTM and GRU similarly, with GRU performing with slightly lower ESR over all SNR ratios. For MLRS dataset with changing SNR a 384 samples (8ms) long impulse response and 6 samples (0.25ms) long time delay was applied. It might be that the response was too complicated for network to learn properly, so that the affection of SNR could be slightly mispronounced with this dataset. However, the impulse response of the real recording is expected to be 5 ms if the *input* and the *target* does not include from the time delay.

As it is discussed before, the RNNs do have problems with learning long time dependencies, the performance drops by increasing length of dependency.

4.2 Neural Network's performance on MLRS data with delay correction

Section 3.7 discusses method for input - target time alignment. In this section, data that was generated for the experiment in Section 4.1, Table 3 will be time - aligned. The NN's performance on time-aligned data is shown in Table 6.

Table 6: LSTM and GRU performance over time-aligned delay lengths evaluated by ESR in dB

Model	Delay length in samples								
	0	6	12	24	48	96	192	384	768
LSTM	-63	-62	-63	-63	-61	-62	-61	-61	-61
GRU	-50	-48	-49	-49	-50	-49	-48	-47	-48

Other parameters:
linear response; IR = 0; SNR > 90 dB
* - time aligned

Time-alignment eliminates the time delay, improves the results and maintains the ESR approximately in the same level over different delay lengths. The slightly different values are appearing due to the training with Stochastic Gradient Descent

and non-fixed initial weights and bias terms. The results of Table 6. and 3. are shown in Fig. 21. The time-alignment will be further used in evaluating the DISCO training with MLRS and real data system, due to the significant improvement in results over different time delays.

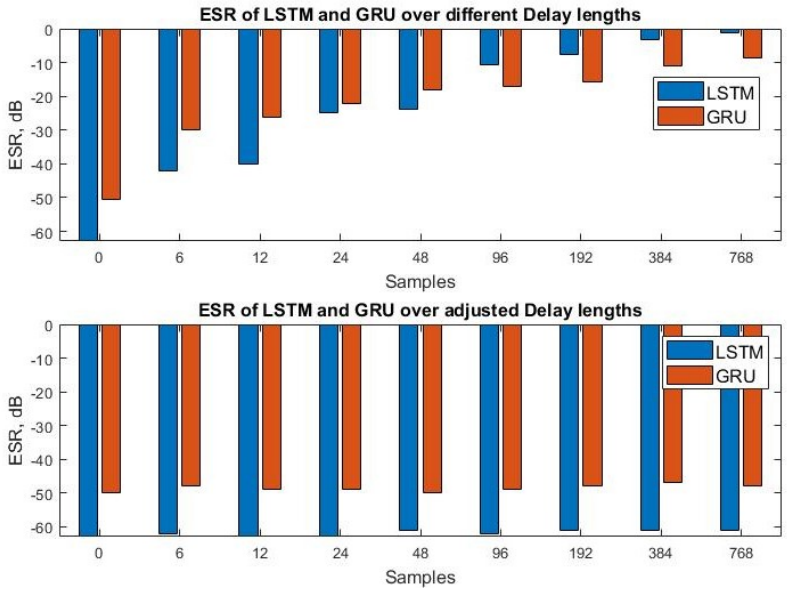

Figure 21: *Network's performance on data with different Delay lengths compared to adjusted Delay lengths.*

Another aspect of the results is the training time. In previous tables and figures the results were gathered by using the hyper-parameters stated in Table 1. The NNs were training for 150 epochs. GRU units are converging faster than the LSTM units due to the less learnable parameters, however, the LSTM's learn more precise, when the time delay is adjusted. The training curves evaluated by ESR over optimizer steps are going to be shown in Section 4.4.

4.3 Neural Network's performance on DISCO MLRS data with delay correction

The current section gathers the results over three different approaches of hidden state initialization. First is the naive approach of "Cold Start", when the hidden state is initialized to zero vector. The second approach is the no gradient startup run for accumulating the hidden state over first few hundred samples of training sequence. The third will be the proposed approached called DISCO sequence training, which is a new method of initializing the hidden state (see Section 3.3). All three approaches are used for LSTM and GRU recurrent units. The results are depicted in Fig. 22. The hyper-parameters of NNs were set to previously used ones (see Table 1). The MLRS parameters were set to following: $IR = 4$ samples, $Delay = 2$ samples, $c_{tanh} = 4$ and SNR > 90 dB. Long dataset was used to produce *input* and

target signals (see Section 3.4). The goal for the particular experiment is determine whether DISCO sequence training functions or not. Its ability to learn longer time dependencies is further tested in Section 4.4.

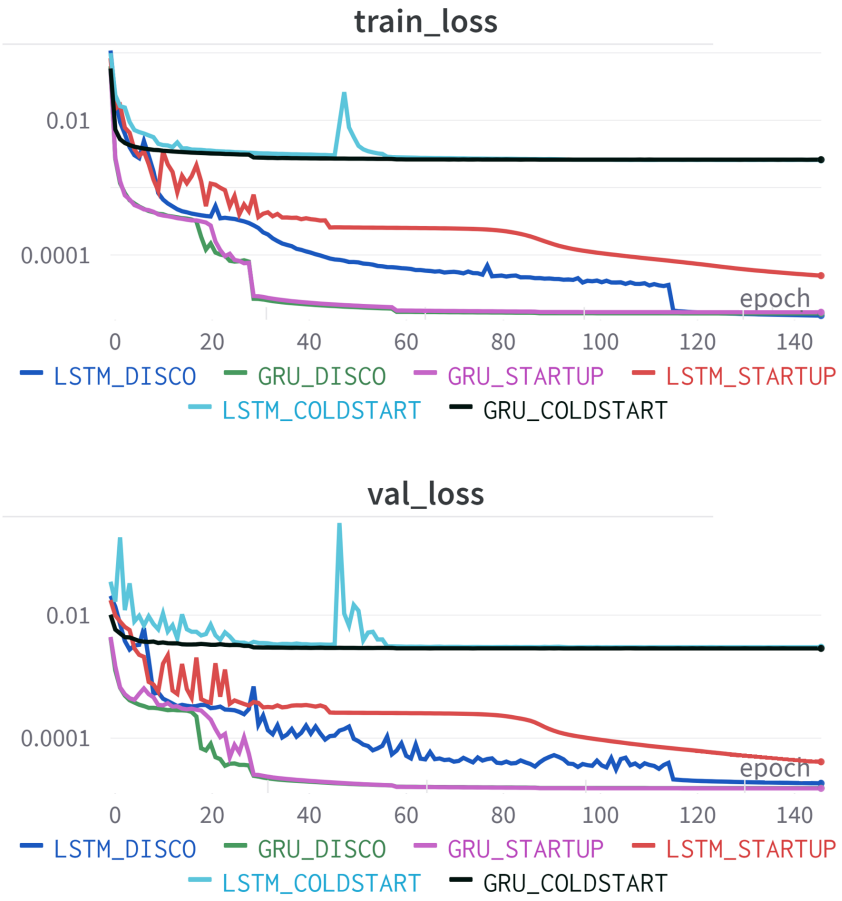

Figure 22: *Training and Validation curves of NN's training with 3 hidden state initialization approaches for 2 recurrent cell types performing on MLRS data. ESR depicted in logarithmic scale over training epochs.*

The DISCO sequence training on MLRS data reaches the precision of no-gradient startup run for hidden state accumulation when using the GRU. The DISCO training with LSTM performs the best in the test inference (see Fig. 23). The performance of LSTM with no-gradient startup run might an unsuccessful training run. This claim is based on the fact that results with Real Recorded data in Section 4.4 shows similar performance between DISCO and no-gradient startup run.

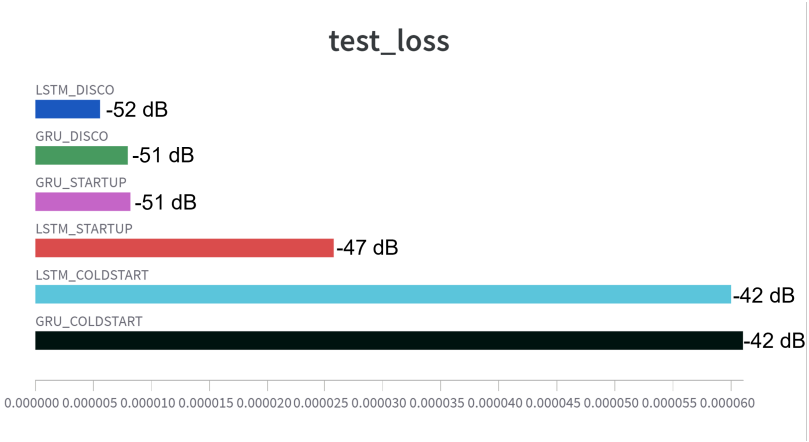

Figure 23: *Test run results of NNs with 3 hidden state initialization approaches for 2 recurrent cell types performing on MLRS data. ESR stated in dB.*

Special treatment for the hidden state initialization is important. Results in Fig. 23 shows that the naive "cold start" approach with initializing the hidden state to zeros reduces the test precision by approximately 10 dB in case of MLRS data.

Training and testing results on MLRS data show that DISCO sequence training is a well-functioning approach of hidden state initialization and shows similar performance to no-gradient startup. The section presents the results on Real Recorded data.

4.4 Neural Network's performance on DISCO real data with delay correction

The following section presents project's results over the real recorded data by using the long dataset stated in Section 3.4. Before the training, the *input* and *target* is time-aligned with non-fractional delay correction (see Section 3.7). Three different approaches on hidden state initialization are used from Section 3. Two different recurrent cells will be tested as a part of the architecture - LSTM and GRU - with hyper-parameters that were utilized in previous results (see Section 1). In total, 6 different training and testing approaches are gathered. The training and validation curves are depicted in Fig. 4.4 and show no evidence of common training problems (see Section 2.2).

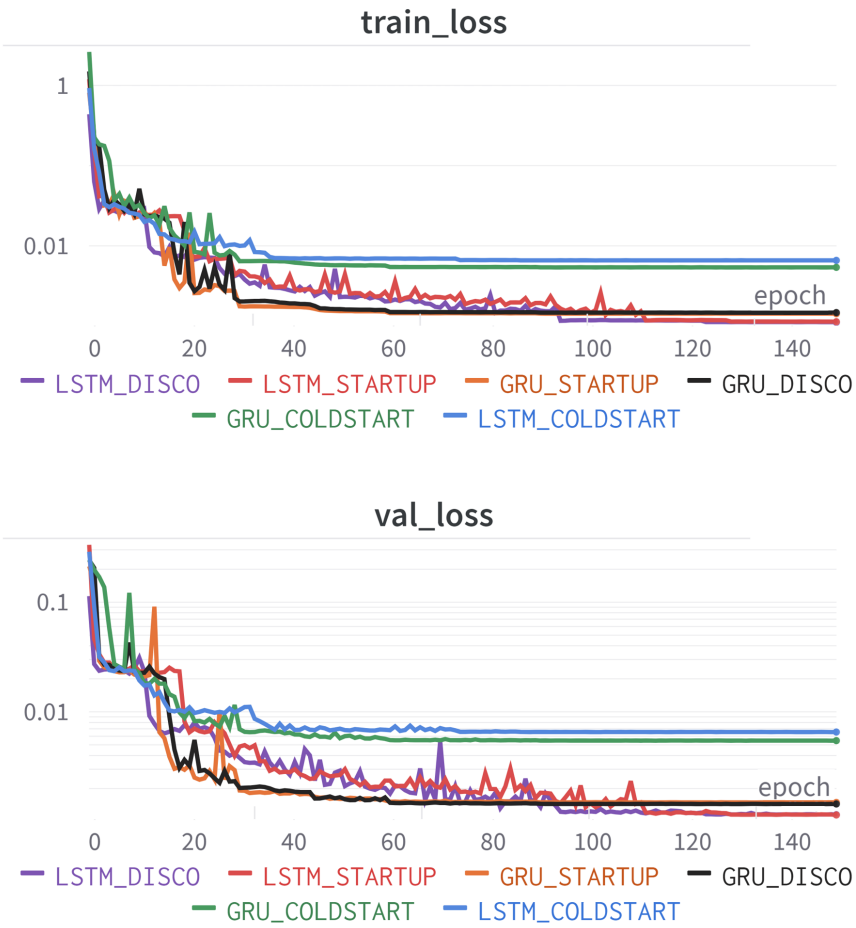

Figure 24: *Training and Validation curves of NN's training with 3 hidden state initialization approaches for 2 recurrent cell types performing on Real Recorded data. ESR depicted in logarithmic scale over training epochs.*

NNs were trained for 150 epochs. The best result is shown by LSTM with the DISCO training (see Fig. 25). The DISCO approach reaches the precision of no gradient startup run, which is the method presented in [3]. Results show how important it is to treat the initial hidden state. The naive approach by just initializing it to zero vector corresponding to *hidden size* gives 6 dB higher Error-To-Signal Ratio.

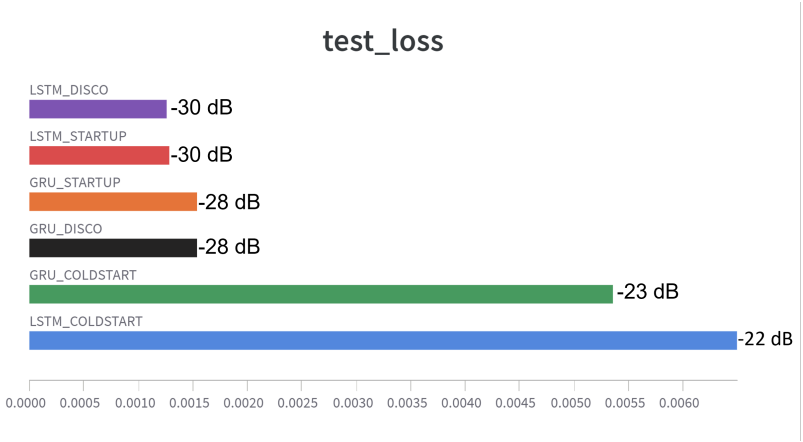

Figure 25: *Test run results of NNs with 3 hidden state initialization approaches for 2 recurrent cell types performing on Real Recorded data. ESR stated in dB.*

The LSTM recurrent cells show better final precision, however the GRU converges faster due to smaller amount of learnable parameters. In training epoch number 55, the GRU shows validation performance of ESR = -28 dB. After this training epoch, the GRU does not show any decrease in ESR in validation or test curves by further training. The LSTM stably shows higher precision than -28 dB only after training epoch number 100 and the final ESR of LSTM is just 2 dB lower of that by being -30 dB (see Fig. 25).

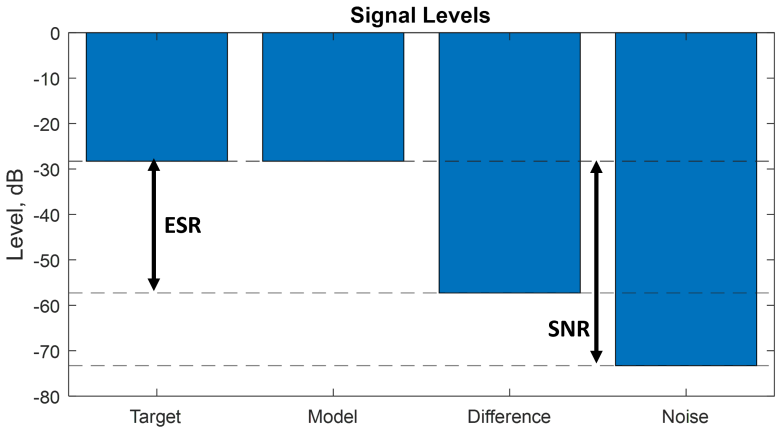

Figure 26: *Signal power levels of target, model's output, their difference and target's SNR.*

Another objective perspective over the best performance by LSTM with DISCO training is depicted in Fig. 26, by comparing 4 different signal power levels of the test run data. The *Target* of the test part is used to evaluate the model's *output* precision during the inference, by in-taking the *input* of the test part of dataset. The *Difference* is acquired by subtracting the wave forms of test part's *target* and *output* resulting into another waveform. Also, the noise during the *target* recording is measured. The levels are further calculated by using Root-Mean-Squared value of

corresponding waveform with $10\log_{10}(\text{RMS}_{\text{test}})^2$. Levels are plotted in bars. The L1 norm between the *Difference* and *Target* results in the ESR loss value during test inference.

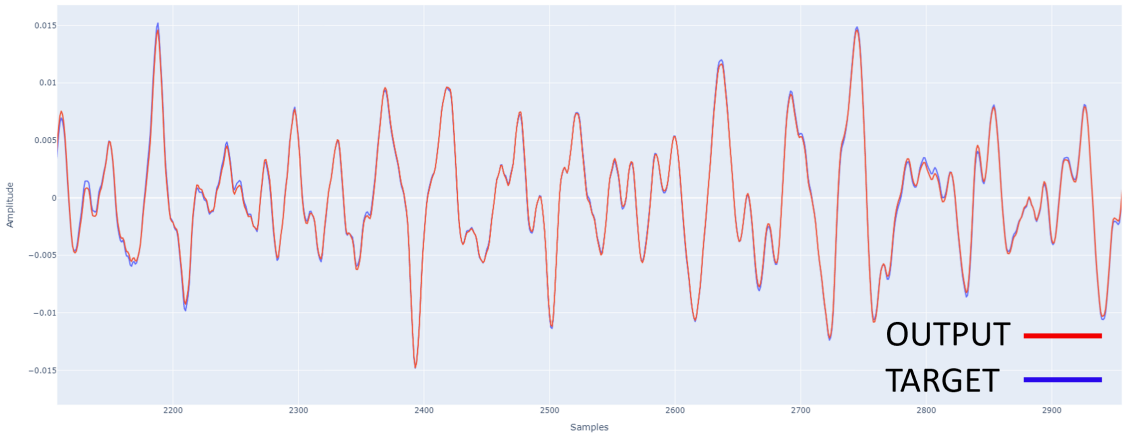

Figure 27: *Target and model's Output waveforms during test run.*

In dB scale the L1 norm between *target* and model's *output* is the same when rounded to two decimal numbers, since the wave form of *target* is replicated by model with precision of ESR = -30 dB (see Fig. 27). The goal of the model is to reach ESR as low as possible. The best possible precision without any noise reduction would be when *Difference* reaches the value of *Noise*, so that there is no more deterministic part between *input* and *target* to learn. The SNR of dataset is 45 dB. The L1 norm between the *Noise* and *Difference* is 14 dB of the best model. Expressed in dB scale, the model has learned 69% of the deterministic part between the *input* and *target*.

5 Discussion

The results (Section 4) of thesis work show that it is possible to create a system identification model of a stereo loudspeaker system by using a relatively simple RNN architecture. The ESR of -30 dB denotes that the inferred audio generated by the model is coherent and close to the target. The State-Of-Art single channel audio modelling [3] was adapted for the stereo use case and functioned well without drastic changes in the model's architecture.

The networks were trained by using an ESR loss. This metric also serves as the main value for objective evaluation of the model. Further evaluation of the model should be made by conducting subjective listening tests, since one of the use cases for the model would be the emulation of the physical system during fine tuning and subjective listening.

In the Section 2.2.3, several approaches on the loss functions were discussed. If the main objective of the model was to perform well in subjective listening tests, the training results would benefit from a perceptual loss function. The multi-scale spectral loss with an emphasis on high frequencies should be tested for such purpose. However, there is an obstacle - the changing size of sequence length in case of TBPTT when whole number of *TBPTT length* does not fit into batch *sequence*, or whole number of *TBPTT length* does not fit into whole training data length. This fact should be taken into account when spectral loss is used, because of multi-scale calculations are chaning the FFT window length. A combination of two loss function could be used - ESR for keeping the wave forms similar and the multi scale spectral loss for learning the spectral content, which is perceptually more descriptive.

The DISCO sequence training has been shown as a functioning method for hidden state initialization and reaches the precision of startup inference stated in [3]. There should be further investigation over DISCO sequence training method for other use cases. For an example it could prepare the data for other Neural Network architecture - such as State Space Models [42].

The time delay in Real Recorded data was manually time-aligned by using the cross-correlation argument at its maximum value. Some methods may allow the NN to learn the delay during training for an example using a Differentiable Digital Signal Processing (DDSP) All-Pass filter with amount of coefficients that correspond to the maximum length of delay in samples [41], but this would lead to slow training for the NN due to the amount of All-Pass filter coefficients.

The used time delay adjustment method does not guarantee a fractional precision for alignment. It might be useful to use the cross-correlation delay adjustment and small DDSP filter in the architecture to further adjust the delay of the system. This might allow the usage of the skip connection that was stated in [3] to further increase the NNs performance.

6 Conclusion

The system identification for stereo loudspeaker system by using an RNN architecture was successful, and the best performing digital twin shows -30 dB of Error-To-Signal ratio. Such an ESR describes a model which can reproduce the physical system in a coherent manner.

Large improvement in the NN performance was made by performing an *input* and *target* time-alignment by a simple, non-fractional but effective method of the cross-correlation between the two signals. A slight manual delay for the *target* was left to ensure causality in data.

For the real recorded data of physical system, in 150 epoch training with 36 minute long dataset, the LSTM performed better than the GRU, but the convergence process took more training epochs. The GRU reached its maximum precision in 35% of the training epochs that are needed for LSTM to outperform the GRU. The total improvement of using LSTM compared to GRU is decrease in ESR of about 2 dB, which is hardly visible in wave forms, but might be noticeable in subjective listening tests - when a perceptual ESR loss is used. It is less computationally expensive to train the GRUs due to less learnable parameters.

Discontinuous sequence (DISCO) training is a new data management method that allows correct initial hidden state (h_0) initialization for RNN training. DISCO reaches the precision of method which accumulates the hidden state by no-gradient startup inference, to use it as the h_0 for the first sample of training for gradient. The DISCO sequence training does not introduce any loss of data for training compared to previous method. The DISCO approach could be used for training other sequential mathematical models, where the previous output of the model is used as the input for the current time step, and the absence of the previous model's output produces error at the very first sample of sequence.

References

[1] Mark R Gander. Fifty years of loudspeaker developments as viewed through the perspective of the audio engineering society. *Journal of the Audio Engineering Society*, 46(1/2):43–58, 1998.

[2] Erik Larsen and Ronald M Aarts. Reproducing low-pitched signals through small loudspeakers. *Journal of the Audio Engineering Society*, 50(3):147–164, 2002.

[3] Alec Wright, Eero-Pekka Damskägg, Vesa Välimäki, et al. Real-time black-box modelling with recurrent neural networks. In *22nd international conference on digital audio effects (DAFx-19)*, 2019.

[4] Sean E Olive. Differences in performance and preference of trained versus untrained listeners in loudspeaker tests: A case study. *Journal of the Audio Engineering Society*, 51(9):806–825, 2003.

[5] Marcelo Soria-Rodríguez, Moncef Gabbouj, Nick Zacharov, Matti S Hamalainen, and Kalle Koivuniemi. Modeling and real-time auralization of electrodynamic loudspeaker non-linearities. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages iv–iv. IEEE, 2004.

[6] Pascal Brunet and Bahram Shafai. New trends in modeling and identification of loudspeaker with nonlinear distortion. In *Proceedings of the International Conference on Modeling, Simulation and Visualization Methods (MSV)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2011.

[7] Alberto Bernardini, Lucio Bianchi, and Augusto Sarti. Loudspeaker virtualization–part i: Digital modeling and implementation of the nonlinear transducer equivalent circuit. *Signal Processing*, 202:108720, 2023.

[8] Walter A Frank. An efficient approximation to the quadratic volterra filter and its application in real-time loudspeaker linearization. *Signal Processing*, 45(1):97–113, 1995.

[9] Bruno Defraene, Toon Van Waterschoot, Moritz Diehl, and Marc Moonen. Embedded-optimization-based loudspeaker precompensation using a hammerstein loudspeaker model. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(11):1648–1659, 2014.

[10] Hocine Merabti, Daniel Massicotte, and Wei-Ping Zhu. Electrodynamic loudspeaker linearization using a low complexity pth-order inverse nonlinear filter. In *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 1–5. IEEE, 2019.

[11] Wolfgang Klippel. Tutorial: Loudspeaker nonlinearities—causes, parameters, symptoms. *Journal of the Audio Engineering Society*, 54(10):907–939, 2006.

[12] Pascal Brunet. *Nonlinear system modeling and identification of loudspeakers*. PhD thesis, Northeastern University, 2014.

[13] Refaat El Attar. *Lecture notes on Z-Transform*, volume 1. Lulu. com, 2006.

[14] Mahdi Kazemi and Mohammad Mehdi Arefi. A fast iterative recursive least squares algorithm for wiener model identification of highly nonlinear systems. *ISA transactions*, 67:382–388, 2017.

[15] Masanao Aoki. *State space modeling of time series*. Springer Science & Business Media, 2013.

[16] Pascal Brunet and Bahram Shafai. State-space modeling and identification of loudspeaker with nonlinear distortion. In *Modelling, Identification, and Simulation, IASTED International Conference on*, volume 755, 2011.

[17] Arie JM Kaizer. Modeling of the nonlinear response of an electrodynamic loudspeaker by a volterra series expansion. *Journal of the Audio Engineering Society*, 35(6):421–433, 1987.

[18] Friedrich Alexander Fischer. *Fundamentals of electroacoustics*. Interscience Publishers, 1955.

[19] Neville Thiele. Loudspeakers in vented boxes: Part 1. *Journal of the Audio Engineering Society*, 19(5):382–392, 1971.

[20] Richard H Small. Closed-box loudspeaker systems-part 1: analysis. *Journal of the Audio Engineering Society*, 20(10):798–808, 1972.

[21] Andrzej B Dobrucki and Piotr Pruchnicki. Application of the narmax method to the modelling of the nonlinearity of dynamic loudspeakers. *Archives of Acoustics*, 26(4), 2001.

[22] Eero-Pekka Damskägg, Lauri Juvela, Vesa Välimäki, et al. Real-time modeling of audio distortion circuits with deep learning. In *Proc. Int. Sound and Music Computing Conf.(SMC-19), Malaga, Spain*, pages 332–339, 2019.

[23] Eero-Pekka Damskägg, Lauri Juvela, Etienne Thuillier, and Vesa Välimäki. Deep learning for tube amplifier emulation. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 471–475. IEEE, 2019.

[24] Tara Vanhatalo, Pierrick Legrand, Myriam Desainte-Catherine, Pierre Hanna, Antoine Brusco, Guillaume Pille, and Yann Bayle. A review of neural network-based emulation of guitar amplifiers. *Applied Sciences*, 12(12):5894, 2022.

[25] David T Blackstock. Fundamentals of physical acoustics, 2001.

[26] Hanieh Khalilian, Ivan V Bajić, and Rodney G Vaughan. 3d sound field reproduction using diverse loudspeaker patterns. In *2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–4. IEEE, 2013.

[27] Tapio Lokki and Lauri Savioja. Virtual acoustics. *Handbook of Signal Processing in Acoustics*, pages 761–771, 2008.

[28] Song Li and Jürgen Peissig. Measurement of head-related transfer functions: A review. *Applied Sciences*, 10(14):5014, 2020.

[29] Birger Kollmeier, Helmut Riedel, Manfred Mauermann, and Stefan Uppenkamp. Physiological measures of auditory function. *Handbook of Signal Processing in Acoustics*, pages 159–173, 2008.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[31] Léon Bottou. Stochastic gradient descent tricks. *Neural Networks: Tricks of the Trade: Second Edition*, pages 421–436, 2012.

[32] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

[33] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

[34] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[35] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[36] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352, 2007.

[37] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[38] SVG Reddy, K Thammi Reddy, and V ValliKumari. Optimization of deep learning using various optimizers, loss functions and dropout. *Int. J. Recent Technol. Eng*, 7:448–455, 2018.

[39] Alec Wright and Vesa Välimäki. Perceptual loss function for neural modeling of audio systems. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 251–255. IEEE, 2020.

[40] Alexander Richard, Peter Dodds, and Vamsi Krishna Ithapu. Deep impulse responses: Estimating and parameterizing filters with deep networks. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3209–3213. IEEE, 2022.

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

[42] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. It's raw! audio generation with state-space models. *arXiv preprint arXiv:2202.09729*, 2022.

[43] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[44] Tianjian Lu, Michael Smedegaard, and Ken Wu. Multiphysics modeling of voice coil actuators with recurrent neural network. *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, 4:12–19, 2019.

[45] Denys Volkov and Lars-Johan Brännmark. Loudspeaker modeling using long/short term memory neural networks. In *Audio Engineering Society Convention 154*. Audio Engineering Society, 2023.

[46] G. Jacovitti and G. Scarano. Discrete time techniques for time delay estimation. *IEEE Transactions on Signal Processing*, 41(2):525–533, 1993.