Nonlinear modeling of the guitar signal chain enabling its real-time emulation

Effects, amplifiers and loudspeakers

Schmitz Thomas

Supervisor: Prof. J.-J. Embrechts

Faculté des Sciences Appliquées
Département d'Électricité, Électronique et Informatique
Institut Montefiore
University of Liège

This dissertation is submitted in fulfillment of the requirements of the degree of
Doctor of Philosophy in Engineering Sciences

University of Liège October 2019

First printing, July 2019

Declaration

The present dissertation has been evaluated by the members of the Jury (sorted by alphabetical order):

- Dr. Stéphane DUPONT, University of Mons, Belgium
- Prof. Jean-Jacques EMBRECHTS (advisor), University of Liège, Belgium
- Prof. Simone ORCIONI, Marche Polytechnic University, Italy
- Prof. Pierre SACRE, University of Liège, Belgium
- Prof. Maarten SCHOUKENS, Eindhoven University of Technology, Netherlands
- Prof. Marc VAN DROOGENBROECK (chair), University of Liège, Belgium

I would like to dedicate this thesis to all musicians suffering while transporting their materials from one place to another.

Acknowledgements

Working on this thesis during six years was a real pleasure. I learned a lot and I had the opportunity to further a subject I care about. I honestly cannot imagine a better job. It was a fantastic experience and I would like to thank you all for making it possible.

Of course, I cannot thank my parent enough. Raising a child is not an easy task but you did it well after only one attempt (just kidding, sis I love you). Anyway thank you for your help, patience and support.

For this thesis, my greatest luck has been to have an awesome supervisor. Aside from your brilliant mind, I always really liked our open discussions and your availability. I have learned a lot from you and I will forever be grateful.

Reading and evaluating a thesis is a task which demands times and energy, I would like to thank you, dear jury members, to have accepted it. I hope this could be the first step of a fruitful collaboration.

I would like to thank the proof readers of my thesis: Jean-Jacques, Pierre and Aubéri. I also received a lot of support from my friends, which are always present to enjoy some time together as well in times of need. I also would like to thank the colleagues and the gardener of the Montéfiore institute who have made of this place a so pleasant place to work at.

I am also grateful to the NVIDIA Corporation for the donation of a Titan Xp GPU to support this research.

Last but not least, a special thanks Aubéri for your support, patience, love and delicious baking.

I wish you all a good reading,

Thomas

Abstract

Nonlinear systems identification and modeling is a central topic in many engineering areas since most real world devices may exhibit a nonlinear behavior. This thesis is devoted to the emulation of the nonlinear devices present in a guitar signal chain. The emulation aims to replace the hardware elements of the guitar signal chain in order to reduce its cost, its size, its weight and to increase its versatility. The challenge consists in enabling an accurate nonlinear emulation of the guitar signal chain while keeping the execution time of the model under the real-time constraint. To do so, we have developed two methods.

The first method developed in this thesis is based on a subclass of the Volterra series where only static nonlinearities are considered: the polynomial parallel cascade of Hammerstein models. The resulting method is called the Hammerstein Kernels Identification by Sine Sweep method (HKISS). According to the tests carried out in this thesis and to the results obtained, the method enables an accurate emulation of nonlinear audio devices unless if the system to model is too far from an ideal Hammerstein one.

The second method, based on neural networks, better generalizes to guitar signals and is well adapted to the emulation of guitar signal chain (e.g., tube and transistor amplifiers). We developed and compared eight models using different performance indexes including listening tests. The accuracy obtained depends on the tested audio device and on the selected model but we have shown that the probability for a listener to be able to hear a difference between the target and the prediction could be less than 1%. This method could still be improved by training the neural networks with an objective function that better corresponds to the objective of this audio application, i.e., minimizing the audible difference between the target and the prediction.

Finally, it is shown that these two methods enable an accurate emulation of a guitar signal chain while keeping a fast execution time which is required for real-time audio applications.

Résumé

L'identification et la modélisation des systèmes non linéaires sont des sujets majeurs dans beaucoup de domaines de l'ingénieur. Cette thèse est dévouée à l'émulation des systèmes non linéaires présents dans la chaîne de traitement du signal de la guitare. L'émulation a pour but de remplacer les éléments matériels de la chaîne par leur équivalent numérique dans le but de réduire son coût, sa taille, son poids et d'améliorer sa polyvalence. Le défi consiste à émuler les éléments de la chaîne en temps réel en tenant compte de leur caractère non linéaire. Pour ce faire, nous avons dévelopé deux méthodes.

La première méthode proposée est basée sur un sous-modèle de la série de Volterra où seules les non-linéarités statiques sont considérées : la cascade de modèles d'Hammerstein mis en parallèle. La méthode résultante que nous avons appelée *Hammerstein Kernels Identification by Sine Sweep* (HKISS), rend possible l'émulation de systèmes audio non linéaires. Cette méthode atteint ses limites quand le système à modéliser est trop différent du modèle idéal d'Hammerstein. La variabilité des noyaux vis-à-vis de l'amplitude du signal d'entrée empêche alors une émulation précise du signal.

La seconde méthode proposée est basée sur l'utilisation de réseaux de neurones. La méthode est plus appropriée aux signaux de guitare et s'adapte bien aux différents systèmes nonlinéaires testés (circuits de distorsion, amplificateurs à tubes et à transistors). Nous avons développé et comparé huit modèles de réseaux de neurones en utilisant différents indices de performances incluant des tests d'écoutes. La précision obtenue dépend du modèle choisi et de l'élément émulé mais nous avons montré que la probabilité qu'une personne puisse discerner une différence entre le son de l'appareil testé et son émulation pouvait être inférieure à 1%. Selon notre opinion, cette méthode pourrait encore être améliorée en entrainant les réseaux de neurones avec une fonction-objectif qui correspond mieux à l'objectif de cette application audio, à savoir, minimiser la différence audible entre le son de l'appareil testé et son émulation.

Ces deux méthodes permettent l'émulation fidèle d'une chaine d'instrumentation pour guitare, tout en gardant un temps d'exécution suffisamment bas pour respecter une contrainte temps réel acceptable pour ce genre d'application audio.

Contents

List of Figures

List of Tables

Nomenclature

Greek Symbols

π	$\simeq 3.1415\ldots$
θ	Parameters

Indexing

a_i	Element i of vector **a**
$A_{i,j}$	Element i,j of matrix **A**
$A_{i,:}$	Row i of matrix **A**

Neural Networks

C	Number of maps
κ	Activation function
L	Kernel length
N	Number of time steps
N_f	Number of input features
N_h	Number of hidden units
N_out	Number of outputs
σ	Logistic sigmoid function
S	Stride size

Calculus and Operations

$\propto$	Proportional to
A^*	Complex conjugate of matrix **A**
A^T	Transpose of matrix **A**
$\triangleq$	Mathematical definition
$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\otimes$	Element wise multiplication (Hadamard product)
$\int_a^b f(x)dx$	Definite integral with respect to x over the set [a,b]

Signal Processing

$*$	Convolution Operation
δ	Dirac delta function
$\mathscr{F}$	Fourier Transform
$\mathscr{H}$	Hilbert Transform
i or j	unit imaginary number $\sqrt{-1}$

Sets

$\{\cdots\}$	The set containing $\cdots$
$\mathbb{N}^*$	The set of natural numbers without 0
$\mathbb{R}$	The set of real numbers
$\mathbb{Z}$	The set of integer numbers
$[a,b]$	The real interval including a and b
$[a,b[$	The real interval including a but not b

Numbers, Vectors and Arrays

a	A scalar

a A vector

A A matrix or a tensor

Acronyms / Abbreviations

CPU Central Processor Unit

DAW Digital Audio Workstation

DNN Deep Neural Network

ESS Exponential Sine Sweep

FC Fully Connected

FLOPS Floating Point Operation Per Second

GPU Graphics Processing Unit

HKISS Hammerstein Kernels Identification by Sine Sweep

LTI Linear Time Invariant

MSE Mean Square Error

NL NonLinear

NN Neural Network

NRMSE Normalized Root Mean Square Error

ODG Objective Difference Grade

PAD_{corr} Probability of Audible Difference corrected

PAD Probability of Audible Difference

PA Public Addressed

PI Performance Index

RNN Recurrent Neural Network

RT Real Time

SDG Subjective Difference Grade

SPICE Simulation Program with Integrated Circuit Emphasis

SSS Synchronized Sine Sweep

Part I

Introduction and Theoretical Background

Table of Contents

Chapter 1

Introduction

Contents

1.1 Emulation of a guitar chain

1.1.1 Main concept

Guitarists use a lot of *audio effects* to modify the original sound of their guitar.

Audio effect:

An audio effect is a tool or a system that changes the perception that people have of a sound.

These effects can be classified based on their underlying techniques or on their perceptual attributes. Their working principles can be consulted in [Zölzer, 2011; Reiss and McPherson, 2014]. Some of them (compressor, distortion, vacuum tube amplifier, noise gate, loudspeaker) are more difficult to model than others. Indeed, they belong to the class of nonlinear systems and they have retained our attention in this thesis.

These effects create intentional or unintentional *harmonic* or *inharmonic* frequency components that are not present in the original input signal.

Harmonicity:

A harmonic series of sounds is a sequence of pure tones represented by sinusoidal waves named *overtones*, where each wave has a frequency which is an integral multiple of the lowest frequency, the *fundamental*. If the overtones do not line up with a multiple integral of the fundamental frequency, they are called *inharmonic*.

In the case of distortion effects, the creation of a strong harmonic content is intended. In a vacuum tube guitar amplifier, the distortion is also intended and most often preferred by musicians than the distortion of a solid state amplifier (amplifier where all vacuum tubes are replaced by transistors) [Barbour, 1998]. The reason is generally attributed to the dynamic of the vacuum tube and to the harmonic content it produces. Unfortunately such audio effects suffer of a few drawbacks, as they can be:

- Expensive: a typical guitar chain (guitar not included) costs approximately between 600 and 1500 €.
- Bulky: as presented in Fig. 1.1 and Fig. 1.2 where an amplifier and its *cabinet* (guitar loudspeaker and its enclosure) and a rack of effects with its *pedalboard* (the place where the musician can activate or deactivate all the effects with his feet) are respectively shown.
- Heavy: a typical guitar signal chain weights between 15 and 30 kg but professional setups can weight much more than 100 kg.
- Fragile: especially, the tube amplification is an old technology and tubes have to cool down before being transported.

Since musicians have to transport all their materials from their home (to practice), to their rehearsal space (to play with their band), to the recording studio and to the concert stages, it could be advantageous to replace all the guitar signal chain (excepted the guitar) by computer emulations. The Fig. 1.3 presents a typical setup for a guitar signal chain and its equivalent using an emulation system. A typical guitar chain is composed as follows: the guitar is connected to several audio effects, the signal processed is then sent to the amplifier which sends it to the cabinet. On stage, during the rehearsal or the recording session, the acoustic signal is picked by a microphone to be sent to a *mix-table* or to a Digital Audio Workstation (DAW). The mix-table controls the balance between all the present instruments. Finally the sound is sent either to the Public Addressed (PA) loudspeakers, to the *return monitors* (loudspeakers toward musicians to have the final rendered) , or to headphones. The objective of this work is to replace the audio effects, the guitar amplifier and the microphone by a computer model.

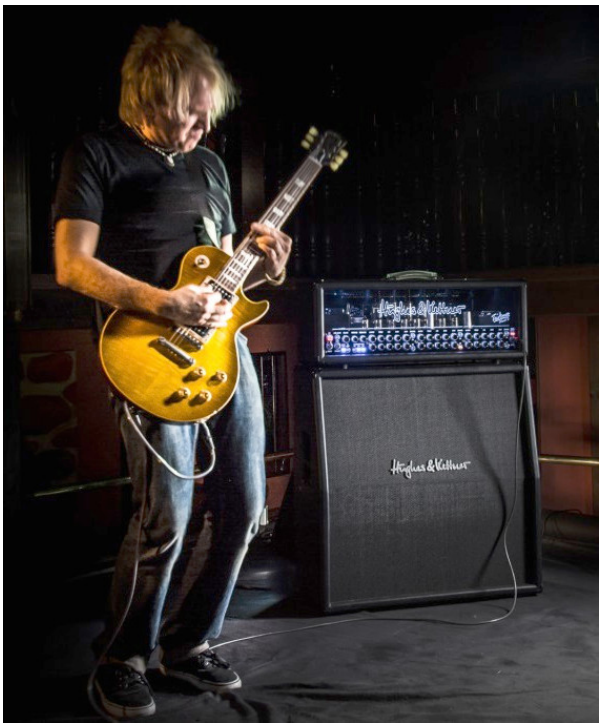


Figure 1.1 Guitar amplifier and its cabinet HUGHES&KETTNER®.

Figure 1.2 JOHN PETRUCCI's guitar rack effects during a concert.

Figure 1.3 Replacement of a typical guitar signal chain by a computer emulation.

1.1.2 Related works

Some fundamental papers have been published on the subject of the emulation of such nonlinear audio devices. In particular, a review of digital techniques for emulation of vacuum tube amplifiers can be found in [Oliveira et al., 2013; Pakarinen and Yeh, 2009]. These techniques can be separated in different classes:

- *Tube emulations by using solid state devices*: historically, the first approach to substitute vacuum tubes in the amplifiers was designed to be directly implemented in the tube sockets as a replacement of the tube triodes [Eby, 1970]. In 1983 Peavy Electronic patented an electronic circuit composed of operational amplifiers in order to reproduce the tube amplifier transfer function [Sondermeyer, 1983]. A more advanced design can be found in [Sondermeyer and Brown Sr, 1997]. These kinds of emulations cannot be described as "accurate", it should be considered as a method to build solid state amplifier so that they sound *like* a tube amplifier.

- *Tube emulations through nonlinear digital models*:

  - Static waveshaping: a straightforward method to obtain a distorted signal is to apply an instantaneous nonlinear mapping between the input signal and its corresponding output [Araya and Suyama, 1996]. This kind of techniques is really limited since the nonlinearity does not depend on the past of the input signal (this type of nonlinearity is called a static nonlinearity or a memory-less nonlinearity). However, in a real circuit, the capacitive elements introduce nonlinear memory effects [Pakarinen and Yeh, 2009]. The type of nonlinearities present in a vacuum tube is described in [Aiken, 2006]. An improvement of the static waveshaping method is proposed by [Gustafsson et al., 2004], where the authors propose the use of Chebychev polynomials to create coefficients and functions that can be altered in real time to model the dynamic nonlinearities of a tube amplifier. However, no demonstration of the accuracy of this method has been found in the literature.

  - *Circuit simulation based techniques*: SPICE (Simulation Program with Integrated Circuit Emphasis) is a software based on transient Modified Nodal Analysis (MNA) and can be used to model vacuum tubes. A SPICE simulation requires a significant amount of Central Processor Unit (CPU) time. Moreover the simulation depends on the accuracy of the SPICE model used. In 1995, Leach [Leach Jr, 1995] introduces some SPICE models for vacuum tube triodes and pentodes, Koren [Koren, 1996] proposes an improvement of vacuum tube models by considering equations obtained

experimentally rather than following the physical laws. A comparison of SPICE models can be found in [Cohen and Helie, 2009; Dempwolf et al., 2011]. The most accurate model from [Dempwolf et al., 2011] shows that the model is accurate for the three first harmonics of the transfer function but the model becomes inaccurate from the fourth harmonic, limiting the use of this method for audio emulation purposes.

- *Techniques for solving Ordinary Differential Equation (ODE)* that describe the behavior of a circuit have also been attempted. [Macak and Schimmel, 2010; Yeh et al., 2008] have worked on the simplification of ODE model resulting in a trade-off between the accuracy and the efficiency of ODE solvers. However, the model fails to emulate high-level or high-frequency input signals [Yeh et al., 2008].
- *Wave Digital Filters (WDF)* are special filters that have a physical interpretation. Each component (resistors, capacitors, diodes, ...) has its own WDF model and by the use of *adapters*, the WDF are connected together as real electrical components are. It has been demonstrated that WDF can efficiently represent some guitar circuits [Yeh and Smith, 2008; Pakarinen and Karjalainen, 2010; Karjalainen and Pakarinen, 2006]. However, finding a general methodology in the WDF framework to model the feedback loops between different parts of the circuit represents a significant challenge.
- *Analytical methods*:
  - Dynamic convolution: [Kemp, 2006] has patented a dynamic convolution method for nonlinear system emulations. The idea is to send a set of impulses at different amplitudes to the device under test and to record their corresponding outputs. During the emulation, the input level is analyzed and compared to the set of impulse responses. The nearest impulse is chosen to evaluate the emulating convolution. Consequently, the coefficients of the convolution change according to the amplitude level. Unfortunately, this method can only model static nonlinearity and fails to emulate *dynamic* nonlinearities (nonlinearities depending of the past of the input signal) [Berners and Abel, 2004] such as those found in a tube amplifier.
  - Volterra series [Boyd, 1985]: is a sum of multidimensional convolutions that can be interpreted as an expansion of Taylor series for nonlinear systems where the polynomial terms are replaced by multidimensional convolutions taking into account the memory associated to the different orders of nonlinearity. Although

the Volterra series model is a valid black-box model for various nonlinearity types, the main challenge is to deal with the numerous coefficients of the Volterra kernels that are growing as the power of the model order. In general this model has been applied to model some soft nonlinearities or to linearize low-order distortion circuits and loudspeakers [Katayama and Serikawa, 1997]. Many papers based on a simplification or a subclass of the Volterra series can be found. In particular [Farina et al., 2001] propose to identify a subclass of the Volterra series using an exponential sine-sweep. This method has been improved to enable the emulation of nonlinear audio effects [Abel and Berners, 2006; Novak et al., 2010b; Tronchin, 2013; Schmitz and Embrechts, 2013, 2014; Novak et al., 2015; Tronchin and Coli, 2015; Schmitz and Embrechts, 2016, 2017] . [Orcioni et al., 2018] present another promising technique based on the identification of Wiener filter coefficients using multi-variance perfect periodic sequences. It is based on the use of different input sequences having different variances to avoid the problem of the locality of the solution that is present in the methods based on exponential sine sweep. However, the use of this method is (for now) limited to low-order models.

– *Neural Networks*: In 2013, [Covert and Livingston, 2013] introduce a simple feed-forward neural network model to emulate a tube amplifier. Despite the inaccurate results (100% of root mean square error), it can be considered as a first attempt to emulate tube amplifiers by neural networks. In 2018, [Schmitz and Embrechts, 2018c] introduce a model based on the Long Short Term Memory Neural Network (LSTM), enabling the emulation of guitar tube amplifiers with a small root mean square error and an high-end perceptual accuracy. This model is also able to take into account the behavior of some parameters of the amplifier. For example, the gain parameter of the amplifier, which influences directly the amount of distortion, has been modeled with success. The model has been transformed to reduce its computational cost in order to increase its real-time performance in [Schmitz and Embrechts, 2018b] and a perceptual audio evaluation of different structures of neural network models has been described in [Schmitz and Embrechts, 2019] proving the efficiency of the method even for high-gain (high-level of distortion) amplifiers. Another neural network has then been introduced in [Damskägg et al., 2019] to emulate the SPICE model of a guitar amplifier.

1.2 Contributions, objectives and applications

The main challenge in this research is to be able to accurately model the *nonlinear* (NL) behavior of the guitar signal chain. In particular, we have focused on: loudspeakers, overdrive effects and tube amplifiers but the method developed here is valid for many other nonlinear guitar effects. Compressors, overdrive effects, distortion effects, fuzz effects, tube amplifiers are always nonlinear, at least for some input signal ranges and are hereafter collectively referred as *guitar or audio effects*.

Since it is important for musicians to hear the final rendering of their guitar chain while they are playing (the sound they hear changes the way they play), it is crucial to design models running in *Real Time* (RT).

RT constraint:

A RT model must guarantee a response within specified time constraints. In the case of guitar playing, keeping the delay introduced by the model under 12ms results in a *good* experience (some artifacts may be perceptible but are not annoying and do not contribute badly to musicians' performance) [Lester and Boley, 2007].

Beyond these practical objectives is hidden a more general one, i.e., the identification and the emulation of nonlinear systems. To enable the emulation of the guitar chain, several methods have been tested: a block-oriented method using the parallel monomial Hammerstein model is introduced and developed in Chap. 3. Then some machine learning methods based on Neural Network (NN) models are developed in Chap. 4.

- Block-oriented models are, by their structure, easy to interpret and analyze. They contain few parameters which makes them suitable for emulation purposes. However, most of them have a lack of flexibility, which makes them difficult to use for real applications.
- On the contrary, neural network models are quite difficult to interpret, they can have an large number of parameters and are difficult to use for RT applications. However, their numerous parameters provide a good flexibility to the model. They are well suited for strongly nonlinear systems such as guitar distortion effects.

The contributions of this thesis are: on one hand, the development of the Hammerstein Kernels Identification by Sine Sweep (HKISS) method. Applying the original method [Farina et al., 2001] for the identification of the diagonal elements of the Volterra series by means of an Exponential Sine Sweep (ESS), we have found some limitations preventing to obtain a good emulation of the guitar chain. These limitations are discussed in the Chap. 3 of this thesis. In

particular, we present an exhaustive list of possible sources of errors/limitations when using this method and show how to avoid them.

On the other hand, the development and the improvement (based on objective and subjective tests) of NN structures enabling the emulation of guitar effects in RT is proposed. To our knowledge, the emulation of high-gain tube amplifiers by black-box techniques of this quality has never been reported earlier. We have proved that the Root Mean Square Error (RMSE) between the signal coming from the amplifier and the signal coming from the model is low (i.e., it depends on the tested amplifier as we will see in Chap. 4 but we have obtained normalized RMSE from 4% to 36% for best and worst case respectively). Some devices are so well emulated that listening tests have revealed that less than 1% of the tested listeners are able to hear the difference between the target and the prediction sounds.

However, comparing different methods is extremely difficult since the RMSE obtained depends on the chosen test set (the input signal selected to evaluate the method). In the same way, the performance indices obtained during a perceptual evaluation cannot be compared if they have been obtained in different studies using different test signals. Therefore, we also introduce a new dataset gathering guitar amplifier sounds aiming for the nonlinear emulation benchmarking.

Direct application of the theory and methods developed in this thesis is of course musical. From our point of view, the ideal application would be a processor integrated to the guitar: the effects would be activated by a foot-controller and the presets controlled by a smart phone. Then the guitar should send the processed signal to a wireless receptor plugged into a DAW station (or a mix table).

Also, the nonlinear identification methods presented in this research could serve to any black-box model in various domains as shown in these examples: physics (nonlinear thermal systems identification [Maachou et al., 2014]), psychoacoustic (peripheral auditory model [Pfeiffer, 1970]), communication (digital satellite communication systems operating over nonlinear channels [Benedetto et al., 1979]), psychology (observing patients' responses to stimuli), biology (human pupillary model [Krenz and Stark, 1991]), finance (neural network for forecasting financial and economic time series [Kaastra and Boyd, 1996]).

The child who tries to open a door has to manipulate the handle (the input) so as to produce the desired movement at the latch (the output); and he has to learn how to control the one by the other without being able to see the internal mechanism that links them. In our daily lives we are confronted at every turn with systems whose internal mechanisms are not fully open

to inspection, and which must be treated by the methods appropriate to the Black Box [Ashby, 1957].

1.3 Outline

Part I of this dissertation introduces the goal of this research as well as the background theory requested for the understanding of this manuscript. We begin by presenting the principal nonlinear models in Chapter 2 and proceed with comprehensive introduction of machine learning methods. The original contributions of this dissertation are gathered in Part II. Chapter 3 includes: an introduction to the identification of Hammerstein nonlinear systems through the exponential sine sweep method, an exhaustive listing of the possible sources of errors and limitations occurring when applying this method and finally a method to avoid or circumvent these errors/limitations. Chapter 4 proposes and evaluates several neural network structures enabling an accurate emulation of nonlinear guitar effects while keeping the real time constraint.

1.4 Publications

This dissertation summarizes several contributions to nonlinear systems identification methods. Publications resulting from this research include:

- [Schmitz and Embrechts, 2013] *Nonlinear Guitar Loudspeaker Simulation*. In 134th Convention of the Audio Engineering Society, May 2013.
- [Schmitz and Embrechts, 2014] *Improvement in non-linear guitar loudspeaker sound reproduction*. In the 5th International Conference on Systems, Signals and Image Processing, May 2014.
- [Schmitz and Embrechts, 2016] *A New Toolbox for the Identification of Diagonal Volterra Kernels Allowing the Emulation of Nonlinear Audio Devices*. In the 22th International Congress on Acoustics, September 2016.
- [Schmitz and Embrechts, 2017] *Hammerstein Kernels Identification by Means of a Sine Sweep Technique Applied to Nonlinear Audio Devices Emulation*. In the Journal of Audio Engineering Society, 65(9):696-710, September 2017
- [Schmitz and Embrechts, 2018c] *Real Time Emulation of Parametric Guitar Tubes Amplifier With Long Short Term Memory Neural Network*. In the 5th International Conference on Signal Processing, March 2018.

- [Schmitz and Embrechts, 2018b] *Nonlinear Real-Time Emulation of a Tube Amplifier with a Long Short Term Memory Neural-Network*. In the 144th Convention of the Audio Engineering Society, May 2018.
- [Schmitz and Embrechts, 2018a] *Introducing a Dataset of Guitar Amplifier Sounds for Nonlinear Emulation Benchmarking*. In the 145th Convention of the Audio Engineering Society, October 2018.
- [Schmitz and Embrechts, 2019] *Objective and Subjective Comparison of Several Machine Learning Techniques Applied for the Real-Time Emulation of the Guitar Amplifier Nonlinear Behavior*. In the 146th Convention of the Audio Engineering Society, March 2019.

1.5 Download materials

Different resources for this thesis are available online:

- The main website on the emulation of guitar signal chain by neural networks is: http://pc-dsp.montefiore.ulg.ac.be/download.
- The `Github` repository gathering the code for the *Hammerstein Kernels Identification by Sine Sweep* toolbox (see Sec. 3.6) and the codes for the 8 neural network models (see Sec. 4.2) is: https://github.com/TSchmitzULG/Thesis
- The dataset presented in App. A.3 gathering the different input/output signals used to train the neural networks can be downloaded from: http://www.montefiore.ulg.ac.be/services/acous/STSI/downloads.php

Chapter 2

Nonlinear models

Contents

Outline of chapter 2:

To be able to accurately describe the challenges encountered in this thesis, an introduction to the systems theory is presented in Sec 2.1. A brief recall of the properties of linear systems is given, then the concept of nonlinearity is introduced with some examples and a proposition of classification for its different forms. The general form of a nonlinear model is introduced in Sec. 2.2. The Volterra series, which is one of the fundamental theories of nonlinear systems, is defined in Sec. 2.3. We proceed, in Sec. 2.4, with the block oriented models and their relations to the Volterra model. The nonlinear state-space model is then introduced in Sec. 2.5 while Sec. 2.6 is devoted to the NARMAX model. The link between general formulation of a nonlinear system and neural network models is introduced in Sec. 2.7. Finally a comprehensive introduction to supervised machine learning is given in Sec. 2.8

2.1 Introduction to systems theory

If the theory of linear systems is well known and established, the theory of nonlinear systems is perceived as quite mysterious. This is due to the fact that if a system is nonlinear: *there is no possibility of generalizing from the responses of any class of inputs to the response for any other input* [Gelb and Vander Velde, 1968] which constitutes a fundamental barrier to the development of generalized methods. This chapter is devoted to the introduction of the most current nonlinear models.

2.1.1 Systems and signals

The description of a real system S by a set of equations $\hat{S}$ (the mathematical model of the system) is nowadays crucial in each engineering field. A mathematical model helps to simulate the system behavior. It can be used to make predictions of the system output corresponding to a specific input, to control it or to ease its design. System identification is a method to define the mathematical model of a process. There are three classes of models:

- The *analytical modeling*, called also *White Box*, aims to characterize the system using its physical laws. In electronics for example, a circuit can be modeled using the equations governing each component (Ohm law for a resistor,...). The result will be more or less satisfying depending on the knowledge, the number and the accuracy of these laws. Moreover the model has to be recomputed if a single component of the system changes.

- The *behavioral modeling*, called also *Black Box*, allows a more general approach. The idea is to find a method that could model all systems no matter their internal operation. Based on the relation between the input and the output of the system, the black-box model reproduces the behavior of the system but has no physical interpretation.

- The *Gray Box* approach uses the two techniques above. If the physical laws are unknown for some components, the introduction of mathematical laws, without physical meaning, completes the set of already known physical laws.

The black-box method offers the advantage of being applicable to all systems having measurable, distinct and causal inputs/outputs [Bunge, 1963]. Models can even be designed to track changes in the systems. The identification procedure of a black-box model should extract the required information in order to link the output of a system to its corresponding input. This is not a trivial task since the structure, the order of the nonlinearity and the memory length of the system are unknown.

2.1.2 Linear systems

This thesis is largely devoted to the modeling of nonlinear systems. Since, nonlinear systems belong to a class of systems that are not linear it is deemed relevant to start by a brief reminder of the *linear system* properties.

Linearity:

Linear systems must satisfy the superposition and homogeneity principles [Ogunfunmi, 2007]. A system obeys to the superposition principle if the outputs $y_i(t)$ from different inputs $x_i(t)$ are additive, and to the homogeneity principle if the output of a scaled version of an input is also scaled by the same factor. For a linear system with P sums (where $P \in \mathbb{N}_{\geq 2}$) of scaled inputs, we have:

$$\begin{cases} if \\ \qquad x(t) = \sum_{i=0}^{P} a_i x_i(t), \\ then \\ \qquad y(t) = \sum_{i=0}^{P} a_i y_i(t), \end{cases} \tag{2.1}$$

Another important property of systems is *time invariance*. Such linear systems are named Linear Time-Invariant (LTI) systems.

Time invariance:

A system S is *time-invariant* if its properties do not change with time.

$$\begin{cases} if \\ \qquad y(t) = S[x(t)], \\ then \\ \qquad y(t - t_0) = S[x(t - t_0)]. \end{cases} \tag{2.2}$$

To be *realizable* a system has to be *causal*.

Causality:

A system is *causal* if the output depends only on its present and/or past input. All physical realizable systems have to be causal since the future of its input is unknown.

Linear time-invariant causal systems can fully be described by their *impulse response h* such as:

Impulse response:

$$y(t) = \int_{\tau=0}^{+\infty} h(\tau)x(t-\tau)d\tau. \tag{2.3}$$

The study of linear models have led to important results and algorithms in the 1970s (non parametric models based on the concept of impulse responses and parametric methods based on least-squared algorithms). The researches on linear model identification are still very active, since the identification methods are generally straightforward, reliable and work well most of the time (at least in the neighborhood of a given operating point). In fact, linear models can be considered as an approximation to real systems. However, some systems cannot be modeled by this simple solution and need a more complex model. Nevertheless, being able to extract the linear behavior of such systems would be a good starting point before adding nonlinearity to a model [Schoukens, 2015].

2.1.3 Nonlinear systems

A nonlinear system is a system that does not satisfy the superposition and/or homogeneity principles, breaking the statements of Eq. (2.1). Such an obvious definition shows the difficulty to classify all nonlinear systems, due to their huge diversity. The study of nonlinear systems is challenging and is still an open topic whether in theory or in practice. While many theoretical methods for the identification of nonlinear systems exist, in practice there are no clear rules to choose the best method, depending on the application and on the system to model. This explains why the use of such methods is relatively marginal in industry although they are intensively studied since the 1940s.

Examples of nonlinear systems

If some systems can be considered as linear, some of them can only be considered as linear on a restricted bounded amplitude range of their input signal (this is the case of most electrical or hydraulic actuators). Others, like *gas jets* have no linear range at all. Nonlinear systems are present in various fields. For example, nonlinear systems are found in:

- Signal processing: the use of digital signal processors also implies nonlinearities in the inevitable quantization process.
- Biology: neurons, pupillary behavior, stiffness of the human body.
- Communications: nonlinear amplifiers.

- Acoustics: loudspeakers, sound wave propagation through materials.

In any way, the need for higher performance systems implies a better control. Therefore, more accurate methods are required to analyze, characterize and simulate the nonlinearities inherent to these systems.

Nonlinearity forms

The comprehension of the nonlinearity form present in a system can help to select an appropriate modeling method. According to [Ogunfunmi, 2007; Gelb and Vander Velde, 1968], the following classification has been selected:

- *Smooth* nonlinearities (can be modeled by polynomial equations).
- *Non-smooth* nonlinearities (nonlinearity with discontinuity).
- *Static* or *dynamic* nonlinearities: the nonlinearity is *static* if the output only depends on a nonlinear function of the input and *dynamic* if the output depends on a nonlinear function of the input signal and some of its derivatives. The nonlinearity has *memory*.
- *Static and multiple valued* nonlinearities lead to multiple output values for the same input (e.g., a relay with hysteresis).
- *Homomorphic* nonlinearity [Oppenheim and Schafer, 2004].

2.2 General form of a nonlinear system

According to [Ljung, 1998], a model is a mapping from past data (i.e., past input x and/or past output y) to the output. The predicted output value at time t of the model $\hat{y}(t)$ depending on the θ parameters can be written as :

$$\hat{y}(t|\theta) = G(Z^t, \theta), \tag{2.4}$$

where Z^t contains all the input/output pairs from the beginning to the instant t and $G(\cdot)$ is a nonlinear function that could be parametrized in terms of physical parameters or in black-box terms as the coefficients of a spline-function. It could be useful to write the Eq. 2.4 as a mapping of two functions. One function that transforms the past observations Z^t into a finite,

fixed dimension vector φ referred to the *regression vector* and whose components are named the *regressors*:

$$\varphi(t) = \varphi(Z^t), \tag{2.5}$$

and another function g mapping the regressors into the output space such as:

$$G(Z^t, \theta) = g(\varphi(t), \theta). \tag{2.6}$$

Defining a model $\hat{S}$ is choosing the regression vector $\varphi(t)$ for the observed input/output values and choosing a nonlinear mapping $g(\varphi(t), \theta)$ from the regressors space to the output space. The g function can also be decomposed in a series expansion of length K:

$$g(\varphi(t), \theta) = \sum_{k=1}^{K} \alpha_k g_k(\varphi(t)),\ \theta = [\alpha_1, \ldots, \alpha_K]^T, \tag{2.7}$$

where the g_k functions are referred to the *basis functions* [Ljung, 1998]. In fact the expansion from Eq. 2.7 with different basis functions plays the role of a unified theory for the different nonlinear models.

2.3 Volterra model

A special case of Eq.2.7 is the use of Taylor expansion choosing $g_k = \varphi^k$ where φ^k should be seen as all combinations of regressors φ_j having a summed exponent equal to k (e.g., if $\varphi = (\varphi_1, \varphi_2, \varphi_3)$ then $\varphi^{(2)} = \{\varphi_1\varphi_2;\ \varphi_1\varphi_3;\ \varphi_2\varphi_3;\ \varphi_1^2;\ \varphi_2^2;\ \varphi_3^2\}$. Choosing φ as the past inputs of the system from the beginning to the instant t leads to a special expansion called the *Volterra expansion*. An infinite sum of these terms is called *the polynomial Volterra series* which can be expressed in continuous time as:

$$y(t) = h_0 + \sum_{k=1}^{+\infty} \int_{\tau_1=-\infty}^{+\infty} \ldots \int_{\tau_k=-\infty}^{+\infty} h_k(\tau_1, \ldots, \tau_k) \cdot x(t-\tau_1) \ldots x(t-\tau_k) d\tau_1 \ldots d\tau_k, \tag{2.8}$$

where $h_k(\tau_1, \ldots, \tau_k), 1 \leq k \leq +\infty$ is a characteristic of the system and is called the kth *Volterra kernel* defined for $\tau_i \in]-\infty, +\infty[,\ 1 \leq i \leq k$. The kernels are continuous and bounded in each τ_i. For the representation to be unique, the kernels have to be symmetric in their k arguments $\tau_i,\ 1 \leq i \leq k$ (e.g., the product $h_2(\tau_1, \tau_2) \cdot x(t-\tau_1) \cdot x(t-\tau_2))$ has to be identical to the product $h_2(\tau_2, \tau_1) \cdot x(t-\tau_2) \cdot x(t-\tau_1)$. Moreover, for a causal system, $h_k = 0,\ \forall \tau_i < 0$.

A *truncated* polynomial Volterra series has a finite order O and a finite memory length T_M such that Eq. 2.8 can be written as:

$$y(t) = h_0 + \sum_{k=1}^{O} \int_{\tau_1=0}^{T_M} \dots \int_{\tau_k=0}^{T_M} h_k(\tau_1, \dots, \tau_k) \cdot x(t-\tau_1) \dots x(t-\tau_k) d\tau_1 \dots d\tau_k. \tag{2.9}$$

The Volterra series (1887) has been one of the most used methods in the last 40 years to model nonlinear systems. While Taylor series uses only a static polynomial expansion, the Volterra series introduces the notion of Taylor series with memory using the convolution method as presented in Eq. (2.3). According to [ULG, 2019], the number of peer reviewed articles including the terms "Volterra series" exceeds nineteen thousands which demonstrates its popularity and its activity in the research field. Since the 1990s the theory has known a renewed interest with the increasing computer capacities. However, the use of Volterra series is still challenging nowadays for multiple reasons:

- The number of parameters: in its digital form, a kernel h_k has M^k coefficients where M is the memory of the system (i.e., the number of past input values including the present one $\{x[n] \dots x[n-M+1]\}$). This number grows exponentially with the order of the system. It implies that the number of data required for the identification of a nonlinear system becomes excessively large. Indeed, applying the symmetry property of the kernels reduces the search of the coefficients of the kth Volterra kernel to the search of M_k independent coefficients [Reed and Hawksford, 1996] where M_k is given by the binomial [Abramowitz and Stegun, 1965, p.265]:

$$M_k = \binom{M+k-1}{k} = C^k_{M+k-1} = \frac{(M+k-1)!}{k!(M-1)!}. \tag{2.10}$$

- The number and the size of the kernels is unknown when starting the identification procedure. Although, a measure of the degree of nonlinearity of a system is introduced in [Rajbman, 1976].

- Special inputs, as Gaussian noise, are often requested for the identification of the kernels which is not always possible if the entry of the system is not accessible or for systems not allowing this kind of entry (e.g., chemical systems).

- As to whether or not the system can be approximated by a Volterra series is not a trivial question. Roughly speaking, if each component of the system can be modeled by an

analytic function [Krantz and Parks, 2002] (i.e., a power series which converges at least for small inputs), the system can mostly be represented by a Volterra series [Boyd, 1985]. However, as described in [Boyd, 1985] analyticity is not the only requirement, e.g., the following differential equation $\dot{y}(t) = -y^3(t) + x(t)$ does not have an exact Volterra series representation.

From an engineering point of view, this definition has a poor meaning, indeed, saying if a tension curve measured in a circuit is infinitely differentiable is only possible if the measurement instruments have an infinite precision, which is not the case. The essential matter is to know if an operator N representing the function of a system can be approximated by a Volterra operator $\hat{N}$ with a precision ε over a set of input signals X such that $||N(x) - \hat{N}(x)|| \leq \varepsilon$ for all $x \in X$. One major difficulty of this approach is to ensure that the Volterra operator is accurate for all inputs $x \in X$ (see examples of different operators to characterize the iris muscle dynamic for different inputs in [Krenz and Stark, 1991]). This is clearly of considerable importance since the choice of the input signal during the identification of the Volterra operator determines the input space for which the operator is valid.

- Identifying high-order kernels requires high-variance signals which can cause high identification error in the responses of the lower kernels, making the identification difficult [Orcioni, 2014].

- Volterra series can fail to approximate some real world systems. For example, dynamical systems with several stable equilibria do not have a *fading memory* (as explained hereafter) and cannot be represented by a Volterra series. Moreover, the Volterra series model is not appropriate for all systems. For example, discontinuities and saturation functions are difficult to approach with this method.

The physical interpretation of the kth Volterra kernel h_k is the measurement effect that the interaction of k past values of the input signal has on the output signal. The popular belief that any continuous, time-invariant, system can be modeled by a Volterra series has to be qualified by a stronger concept than continuity which is *fading memory*. It is introduced by [Boyd and Chua, 1985] in the Theorem 1. However, before enunciating this theorem, let us introduce some notations and definitions:

- **Definition of continuity** [Rudin et al., 1964, p.85]: Suppose X and Y are *metric spaces* [Rudin et al., 1964, p.30], $E \subset X$, $p \in E$, and f maps E into Y. Then f is said to be continuous at p if for every $\varepsilon > 0$ there exists a $\Delta > 0$ such that $d_Y(f(x), f(p)) < \varepsilon$ for all

points $x \in E$ for which $d_x(x,p) < \Delta$, where d is the distance function of the used metric space.

- **Definition of uniform continuity** [Rudin et al., 1964, p.90]: *uniform continuity* is a property of a function on a set whereas continuity can be defined in a single point. Let f be a mapping of *metric space* X into a metric space Y. We say that f is uniformly continuous on X if for every $\varepsilon > 0$ there exists $\Delta > 0$ such that $d_Y(f(p), f(q)) < \varepsilon \; \forall q, p \in X$ for wich $d_X(p,q) < \Delta$.

- **Definition of** $C(\mathbb{R})$: it denotes the space of bounded continuous function $\mathbb{R} \to \mathbb{R}$ with the *uniform norm* (also called *supremum norm* [Rudin et al., 1964]) such that $||f|| = \sup |f(x)| \; \forall x \in \mathbb{R}$.

- **Definition of equicontinuity:** in real and functional analysis, equicontinuity is a concept which extends the notion of uniform continuity [Stover, 2019] from a single function to collection of functions.

- **Definition of uniformly bounded:** a family of functions indexed by I such as $f_i : X \to Y, \forall i \in I$ is uniformly bounded if there exists a real number r such that: $|f_i(x)| \leq r \; \forall i \in I, \forall x \in X$.

- **Definition uniform approximation**: the quality measurement of the uniform approximation of a function f(x) by its approximated function P(x) on a given set is given by: $\sup |f(x) - P(x)| \; \forall a \leq x \leq b$.

We are now able to introduce the Theorem on the Volterra operator:

Volterra operator:

Theorem 1. Let $x(t) \in X$ where X is any uniformly bounded equicontinuous set in $C(\mathbb{R})$, and let N be any time-invariant, fading memory operator defined on X. Then for any $\varepsilon > 0$, there is a Volterra operator, $\hat{N}$, such that for all $x \in X$, we obtain

$$|N(x(t)) - \hat{N}(x(t))| \leq \varepsilon. \tag{2.11}$$

The only constraint on the system is the *fading memory* of its operator.

Definition of fading memory:

The operator N has a *Fading Memory* on a subset X of $C(\mathbb{R})$ if there exists a decreasing function $w : \mathbb{R}_+ \to]0,1]$, $\lim_{t \to \infty} w(t) = 0$, such that for each $x \in X$ and $\varepsilon > 0$ there is a $\delta > 0$ such that for

all $v \in X$:

$$\sup_{t \leq 0} |x(t) - v(t)|.w(-t) < \delta \rightarrow |N(x(0)) - N(v(0))| < \varepsilon. \tag{2.12}$$

[Boyd and Chua, 1985] explain that intuitively, an operator has fading memory if two input signals which are close in the recent past, but not necessarily close in the remote past yield present outputs which are close. In fact the *Fading Memory* condition on the operator is extremely weak and many systems have a fading memory operator. The concept of uniformly bounded equicontinuous set may seem obscure for those who are not familiar with functional analysis, it can be seen from an engineering point of view as the set X of signals $x(t)$ which is bounded by M_1 with a *slew-rate* bounded by M_2, which can be expressed formally by:

$$X = \left\{ x \in C(\mathbb{R}) \text{ such that } |x(t)| \leq M_1, \frac{|x(s) - x(t)|}{(s-t)} \leq M_2 \text{ for } t \leq s \right\}. \tag{2.13}$$

This formulation [Boyd, 1985, p.52] is very practical since it is easy to verify that the input signals are bounded and have a finite *slew-rate* (which is the case for many signals). However, all physical systems cannot be modeled by Volterra series. For example:

- Multiple valued nonlinearities such as hysteresis cannot be directly (work around is possible as it is presented in [Irving, 2008; Ran et al., 2014]) represented by a Volterra series since the characteristic subharmonics associated with memory nonlinearities are not generated by Volterra expansion [Billings, 1980].
- The *Peak-Hold Operator* given by:

  $$N(x(t)) = \sup(x(\tau)) \; \forall \; \tau \leq t, \tag{2.14}$$

  cannot be modeled by Volterra series since it has no fading memory.
- The Duffing oscillator which can exhibit three equilibrium points [Billings, 2013, p301] cannot be modeled by a Volterra series for the same reason.

For time varying systems, the Volterra kernels can be replaced by varying kernel functions where $h_k(t, \tau_1, \ldots, \tau_k)$ is the kth varying Volterra kernels [Yu et al., 1997, 1999; Guo et al., 2014].

A comprehensive review of Volterra series and associated methods is given in [Cheng et al., 2017]. The Volterra series are largely applied in the identification of nonlinear systems including: polynomial nonlinear systems [Jing et al., 2009; Peng et al., 2007b], piecewise linear systems [Zhu et al., 2008; Lin and Unbehauen, 1990], bilinear systems [Chatterjee, 2010; Peng

et al., 2007a], saturation systems [Hélie, 2010; Mastromauro et al., 2007], spatio-temporal nonlinear systems [Guo et al., 2010; Li and Qi, 2010; Hélie and Roze, 2008], hysteresis nonlinear systems [Irving, 2008; Ran et al., 2014], fractional-order nonlinear systems [Hélie and Hasler, 2004; Maachou et al., 2014].

Condition to belong to a Volterra series model:

A continuous-time, time-invariant, finite-memory, causal nonlinear system which is a continuous functional of its input, can be uniformly approximated over a uniformly bounded equicontinuous set of input signals, to an arbitrary degree of accuracy, by a Volterra series of sufficient but finite order [Korenberg, 1991]. This result can be extended to discrete-time nonlinear systems by the Stone-Weierstrass theorem [Dieudonné, 2013; Palm, 1979].

2.4 Block oriented models

Block oriented models represent one of the simplest classes of nonlinear models. These models consist of several parallel/series blocks where a block is either a static (i.e., memoryless) nonlinearity or a linear filter. If the dynamics of the system can be described by a linear function and if the system has a static nonlinearity at the input or at the output, the system can be represented by a *Hammerstein* model (as presented in Sec. 2.4.1) or a *Wiener* model (as presented in Sec. 2.4.2) respectively (e.g., a linear sensor having saturation nonlinearity). *Sandwich* models (as presented in Sec. 2.4.3) also called *Wiener-Hammerstein* models are composed of one static nonlinearity surrounded by two linear filters. These three models are, in fact, some simplified cases of the Volterra model. A fundamental property is that a Volterra operator can be exactly represented by a finite number of parallel cascades (as presented in Sec. 2.4.4) of Wiener-Hammerstein or Wiener models. Finally, the Wiener-Bose model (as presented in Sec. 2.4.5), composed of P linear filters followed by a P-inputs static nonlinearity, plays the role of unified framework for nonlinear models since it can be shown that parallel Wiener cascade and Volterra models are special cases of the Wiener-Bose model.

2.4.1 Hammerstein model

The Hammerstein model is composed of a static nonlinearity followed by a dynamic linear block as presented in Fig. 2.1 and Eq. (2.15).

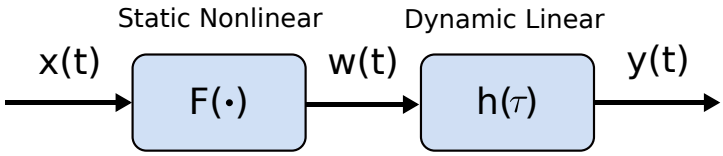


Figure 2.1 Hammerstein model.

$$\begin{aligned} y(t) &= \int_0^{+\infty} h(\tau) w(t-\tau) d\tau, \\ w(t) &= F(x(t)), \end{aligned} \tag{2.15}$$

where $F(\cdot)$ is a static nonlinear function usually assumed to be a polynomial function due to the Weierstrass' approximation theorem [Jeffreys and Jeffreys, 1999], stating that any function on a closed and bounded interval can be uniformly approximated on this interval by polynomials to any degree of accuracy. Considering such a function truncated to the order O where the DC component has been removed gives:

$$F(v) = \alpha_1 v + \ldots + \alpha_O v^O. \tag{2.16}$$

Eq. (2.15) can then be described as:

$$\begin{aligned} y(t) &= \int_0^{+\infty} h(\tau) \sum_{k=1}^{O} \alpha_k x(t-\tau)^k d\tau, \\ &= \sum_{k=1}^{O} \int_0^{+\infty} h(\tau) \cdot \alpha_k x(t-\tau)^k d\tau. \end{aligned} \tag{2.17}$$

The link with the Volterra series can be obtained by expressing Eq. (2.17) as:

$$y(t) = \sum_{k=1}^{O} \int_0^{+\infty} \cdots \int_0^{+\infty} \alpha_k h(\tau_1) \cdot \delta(\tau_1 - \tau_2) \cdots \delta(\tau_1 - \tau_k) \cdot x(t-\tau_1) \cdots x(t-\tau_k) d\tau_1 \cdots d\tau_k. \tag{2.18}$$

It can be derived from Eq. (2.18) and Eq. (2.8) that the Hammerstein model is a truncated Volterra series where the Volterra kernels are:

$$h_k(\tau_1, \cdots, \tau_k) = \alpha_k h(\tau_1) \cdot \delta(\tau_1 - \tau_2) \cdots \delta(\tau_1 - \tau_k) \tag{2.19}$$

and then,

$$h_k(\tau_1,\cdots,\tau_k) = \begin{cases} \alpha_k h(\tau_1) & \text{if } \tau_1 = \cdots = \tau_k, \\ 0 & \text{else.} \end{cases} \tag{2.20}$$

In the discrete time representation, the Hammerstein model can be represented by a Volterra series whose kernels coefficients are equal to zero unless on their diagonal elements, i.e., the elements of the tensor h_k having all its indices equal ($\tau_1 = \tau_2 = \cdots = \tau_k = \tau$). For example, the second order Volterra kernels of a Hammerstein system of memory M is represented by:

$$h_2 = \begin{pmatrix} \alpha_2 h[0] & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \alpha_2 h[M] \end{pmatrix}, \tag{2.21}$$

where $h[n]$ is the Hammerstein kernel of the system.

Belonging to the Hammerstein class:

A system S can be represented by a Hammerstein model if the non-diagonal elements of its Volterra kernels are equal to zero. Moreover, the diagonal of each Volterra kernel must be proportional to the impulse response of the linear subsystem. As a consequence, in an Hammerstein model, the diagonals of the Volterra kernels are proportional between them such that:

$$\begin{aligned} &h_k(\tau_1,\cdots,\tau_k) \propto h_j(\tau_1,\cdots,\tau_j), \\ &\textit{with } \tau_1 = \cdots = \tau_k = \tau_j \ \forall j,k \in \mathbb{N}^*. \end{aligned} \tag{2.22}$$

2.4.2 Wiener model

The Wiener model (not to be confused with Wiener series [Wiener, 1966] which is an orthogonal expansion of functional series) is composed by a static nonlinearity following a dynamic linear block as presented in Fig. 2.2 and Eq. (2.23).

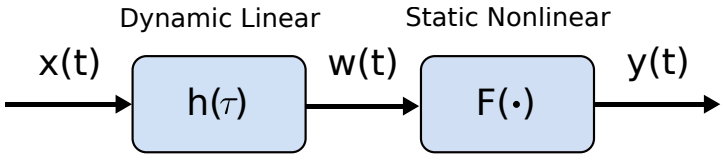


Figure 2.2 Wiener model

$$\begin{aligned} w(t) &= \int_0^{+\infty} h(\tau)x(t-\tau)d\tau, \\ y(t) &= F(w(t)). \end{aligned} \tag{2.23}$$

Using a polynomial function for the static nonlinearity, the Eq. (2.23) can be expressed as:

$$y(t) = \sum_{k=1}^{O} \alpha_k \left[\int_0^{+\infty} h(\tau) \cdot x(t-\tau)d\tau \right]^k, \tag{2.24}$$

which can be expressed as:

$$y(t) = \sum_{k=1}^{O} \int_0^{+\infty} \cdots \int_0^{+\infty} \alpha_k \cdot h(\tau_1) \cdots h(\tau_k) \cdot x(t-\tau_1) \cdots x(t-\tau_k) d\tau_1 \cdots d\tau_k. \tag{2.25}$$

The comparison of Eq. (2.8) and Eq. (2.25) shows that the Volterra kernels h_k of a Wiener system can be expressed as the product of k copies of the impulse response of its linear subsystem:

$$h_k(\tau_1, \cdots, \tau_k) = \alpha_k h(\tau_1). \cdots .h(\tau_k). \tag{2.26}$$

Necessary condition to belong to the Wiener class:

From Eq. (2.26) it appears that any *one-dimensional* slice of the discrete time representation of the Volterra kernel is proportional to the impulse response of the linear subsystem. For example, for a fixed value of $\tau_1 = k_1$ and $\tau_2 = k_2$, the third Volterra kernel of a Wiener system is :

$$h_3[k_1, k_2, \tau] = \alpha_3 h[k_1]h[k_2]h[\tau] = \beta h[\tau]. \tag{2.27}$$

This result is a necessary condition to belong to the Wiener class. Another formulation of this condition is that two one-dimensional slices of the Volterra kernels of a Wiener system have to be proportional [Westwick and Kearney, 2003].

2.4.3 Sandwich or Wiener-Hammerstein model

The Wiener-Hammerstein model (also named sandwich model) is composed by a static nonlinearity followed and preceded by linear blocks as presented in Fig. 2.3 and Eq. (2.28).

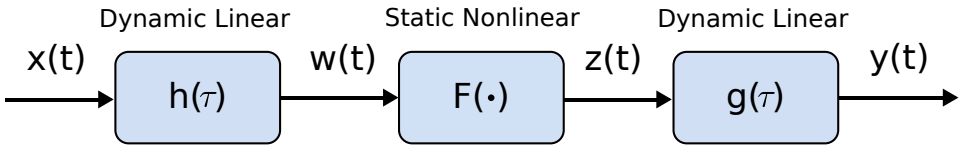


Figure 2.3 Wiener-Hammerstein model

$$\begin{aligned} w(t) &= \int_0^{+\infty} h(\tau)x(t-\tau)d\tau, \\ z(t) &= F(w(t)), \\ y(t) &= \int_0^{+\infty} g(\tau)z(t-\tau)d\tau. \end{aligned} \tag{2.28}$$

Using a polynomial static nonlinearity, Eq. (2.28) can be expressed as:

$$y(t) = \int_0^{+\infty} g(\tau) \sum_{k=1}^{O} \int_0^{+\infty} \cdots \int_0^{+\infty} \alpha_k h(\tau_1)\cdots h(\tau_k) \prod_{i=1}^{k} x(t-\tau-\tau_i)d\tau_1\cdots d\tau_k d\tau. \tag{2.29}$$

The Volterra form can be obtained by defining $\sigma_i = \tau + \tau_i$:

$$y(t) = \sum_{k=1}^{O} \int_0^{+\infty} \cdots \int_0^{+\infty} \left[\alpha_k \int_0^{+\infty} g(\tau)h(\sigma_1-\tau)\cdots h(\sigma_k-\tau)d\tau\right] \prod_{i=1}^{k} x(t-\sigma_i)d\sigma_1\cdots d\sigma_k. \tag{2.30}$$

The comparison of Eq. (2.8) and Eq. (2.30) shows that the Volterra kernels h_k of a Wiener-Hammerstein system can be expressed as:

$$h_k(\tau_1,\cdots,\tau_k) = \alpha_k \int_0^{+\infty} g(\tau)h(\tau_1-\tau)\cdots h(\tau_k-\tau)d\tau. \tag{2.31}$$

Necessary condition to belong to a Wiener-Hammerstein class:

In the discrete time domain, the first-order Volterra kernel must be proportional to the sum of the one-dimensional slices of the second-order Volterra kernel taken parallel to one axis, named the *marginal second-order kernel* $k(\tau)$ such that [Westwick and Kearney, 2003]:

$$h_1[\tau] \propto k[\tau] = \sum_{m=0}^{M-1} h_2[\tau, m], \tag{2.32}$$

where M is the memory length of the system.

This model is sometimes called LNL (Linear Nonlinear Linear). There are also models of the form NLN (Nonlinear Linear Nonlinear) which are not studied here.

2.4.4 Parallel cascade model

The Hammerstein model, the Wiener model and the Wiener-Hammerstein model are very convenient to model some nonlinear systems, they are more parsimonious (i.e., they have a smaller number of parameters) than the general Volterra model but their lack of flexibility reduces their application field. It has been shown that the output of any continuous, discrete-time, finite-dimensional system can be approximated by a parallel cascade of Wiener-Hammerstein models [Palm, 1979] as presented in Fig. 2.4. Later it has been shown that this Volterra model can be exactly represented by a finite number of P parallel Wiener cascade models [Korenberg, 1991]. The Volterra kernels associated to this structure are the sum of the Volterra kernels of each branch:

$$h_k(\tau_1, \cdots, \tau_k) = \sum_{p=1}^{P} \alpha_{kp} \int_0^{+\infty} g_p(\tau) h_p(\tau_1 - \tau) \cdots h_p(\tau_k - \tau) d\tau. \tag{2.33}$$

Moreover, Korenberg [1991] shows that the unidentified residue (i.e., the difference between the output of the system and the output of the parallel cascade model) is a Volterra series that can be treated by another parallel cascade model. The nonlinear system can then be approximated to an arbitrary level of accuracy (in the mean square error sense) by a sum of a sufficient number of individually identified cascades, where, the kth order of the Volterra kernels limited to a memory $(M+1)$ in the discrete time representation (the system output depends on input delays from 0 to M), is fully described by P parallel Wiener cascades [Korenberg, 1991]:

$$P = 2^{k-1} \cdot \binom{M+k-1}{k-1}. \tag{2.34}$$

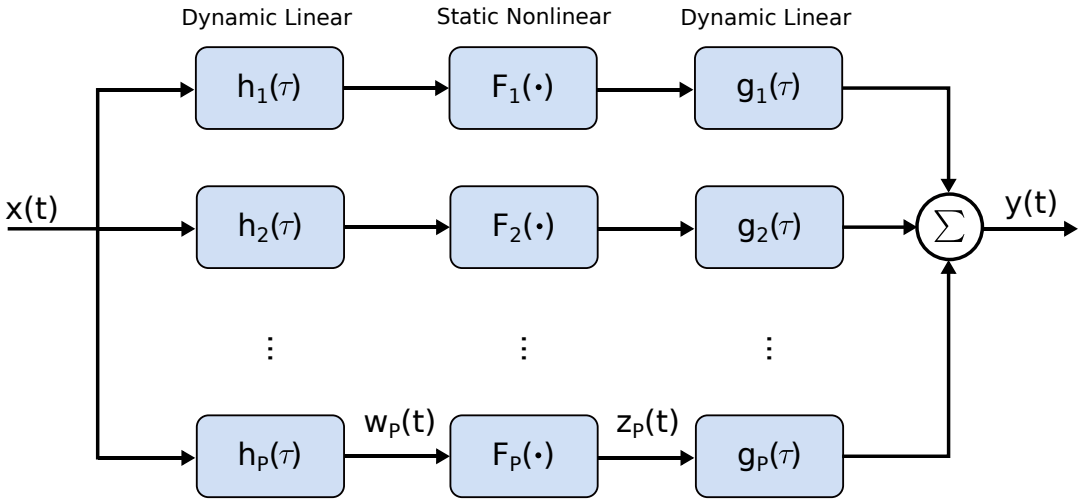


Figure 2.4 Parallel sum of Wiener-Hammerstein cascade models.

2.4.5 Wiener-Bose model

The Wiener-Bose model (named also the General-Wiener model) is constituted by a set of P-linear filters placed in parallel and followed by a multi-variable polynomial function F of degree Q as presented in Fig. 2.5. The Wiener-Bose model can be expressed by the following

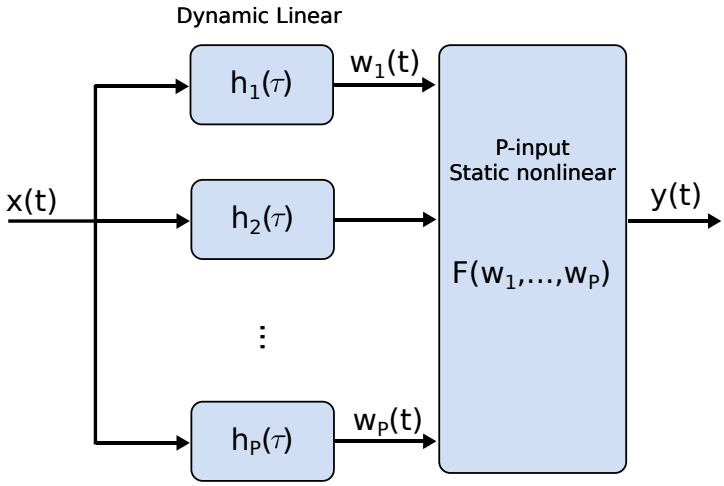


Figure 2.5 The Wiener-Bose model: P linear filters followed by a static multi-input polynomial nonlinearity.

equations:

$$w_p(t) = \int_0^{+\infty} h_p(\tau)x(t-\tau)d\tau,$$
$$y(t) = F^{(Q)}(w_1, \cdots, w_P),$$

where :

$$F^{(Q)}(w_1, \cdots, w_P) = \alpha_0^0 + \sum_{q=1}^{Q} \sum_{p_1=1}^{P} \sum_{p_2=p_1}^{P} \cdots \sum_{p_q=p_{q-1}}^{P} \alpha_{p_1,\cdots,p_q}^{(q)} \cdot w_{p_1}(t) \cdot \ldots \cdot w_{p_q}(t), \tag{2.35}$$

and $\alpha_{p1,\cdots,pq}^{(q)}$ is a tensor of order q containing parameters. $F^{(Q)}(w_1, \cdots, w_P)$ simply computes a polynomial function of order Q with all the combinations of inputs such that the sum of their exponents is $q \in [1, Q]$. For example:

$$\begin{aligned} F^{(2)}(x_1, x_2, x_3) = \alpha_0^{(0)} &+ \alpha_1^{(1)}x_1 + \alpha_2^{(1)}x_2 + \alpha_3^{(1)}x_3 \\ &+ \alpha_{1,1}^{(2)}x_1x_1 + \alpha_{1,2}^{(2)}x_1x_2 + \alpha_{1,3}^{(2)}x_1x_3 \\ &+ \alpha_{2,2}^{(2)}x_2x_2 + \alpha_{2,3}^{(2)}x_2x_3 + \alpha_{3,3}^{(2)}x_3x_3. \end{aligned} \tag{2.36}$$

It can be shown [Westwick and Kearney, 2003] that the Wiener, the Volterra expansion, the parallel Wiener and Wiener-Hammerstein cascade models are special cases of the Wiener-Bose model, making from this latter a good general framework for nonlinear models.

2.5 Nonlinear state-space model

The nonlinear state-space model provides a very general description of a finite-dimensional system, it can be described as:

$$\begin{aligned} \dot{w}(t) &= G(t, w(t), x(t), \omega(t); \theta), \\ y(t) &= H(t, w(t), x(t), \nu(t); \theta), \end{aligned} \tag{2.37}$$

where G and H are nonlinear functions of the input(s) $x(t)$, the state(s) $w(t)$ and independent random variable $\omega(t), \nu(t)$ (e.g., noise process), θ denotes the unknown parameters.

A special case of the nonlinear state-space is the Wiener-Bose model of Eq. (2.35). Indeed it is easy to show that the Wiener-Bose model can be described as:

$$\begin{aligned} \dot{w}(t) &= Aw(t) + bx(t), \\ y(t) &= F(w(t)), \end{aligned} \tag{2.38}$$

where $w(t)$ and b are P-dimensional vectors (P is the number of branches in the Wiener-Bose model). A is a P x P matrix and $F(\cdot)$: $\mathbb{R}^P \to \mathbb{R}$ is a polynomial as expressed in Eq. (2.35). The relation with the Wiener-Bose system is immediate if it is noticed that the state equation from Eq. (2.38) is a single input $x(t)$, P-output linear state-space system, which is equivalent to P separate linear filters [Westwick and Kearney, 2003, p.94]. This leads to the following theorem:

State-space equation:

Theorem 2. The approximation $\hat{N}$, for any time-invariant fading-memory operator as described by Theorem 1 in Sec. 2.3 can be realized using the state-space equation (2.38)

2.6 NARMAX model

System identification consists in the determination of the input/output law of a system, where the inputs/outputs are measured on a finite-time interval resulting in a dataset of input/output samples. Although, continuous-time models can be identified from this dataset, a discrete-time representation is usually more convenient. The *NARMAX* model [Billings and Leontaritis, 1982] is defined in the discrete-time domain by a set of nonlinear difference equations:

$$y[n] = F\left(y[n-1], \cdots, y[n-n_y], x[n-d] \cdots x[n-d-n_x], e[n-1], \cdots, e[n-n_e]\right) + e[n], \tag{2.39}$$

where $y[n]$, $x[n]$, $e[n]$ are the system output, input and noise sequence respectively. d is a time delay usually set to zero or one and n_y, n_x, n_e are the maximum lags for the system output, input and noise respectively. $F(\cdot)$ is a nonlinear function.

As the Infinite Impulse Response (IIR) filter can represent the Finite Impulse Response (FIR) filter in a more parsimonious way (i.e., much less parameters) by introducing the information of the past inputs in a few output lagged terms, the NARMAX model introduces the past of its outputs in its model. In comparison, the Volterra model only uses the past inputs of the system. The trade-off is the difficulty to choose an appropriate function F and to identify its arguments.

The Volterra model, the block-structured models and many neural network architectures can be considered as subsets of the NARMAX model [Chen and Billings, 1989]. The NARMAX

model can also represent some exotic models such as chaos, bifurcations and sub-harmonic systems [Billings, 2013].

Sufficient condition to belong to a NARMAX model:

[Leontaritis and Billings, 1985] proved that a nonlinear discrete-time, time-invariant system can be represented by Eq. (2.39) if the response of the system is finitely realizable, i.e., if and only if it has a state-space realization. This condition excludes infinite-dimensional state-space (also called *distributed parameter systems* [Banks and Kunisch, 2012]). A linearized model exists if the system operates close to an equilibrium point, i.e., a state w_e such that $w_e = G(w_e, x)$ in the meaning of the state-space equation (2.37) of the system.

Notice that the form of the nonlinear function $F(\cdot)$ may depend on the input region as for the Volterra model, therefore a NARMAX model is valid only in a region around some operating point (i.e., around a special state w). [Chen and Billings, 1989] present some examples of NARMAX models approximating mathematical functions valid around an operating point.

2.7 Neural networks models

Neural networks are generally presented in a matrix or tensor form. In this section, we choose to present neural networks with the same notation than used until now to highlight the strong relation existing between the general formulation of a nonlinear system and neural networks which is sometimes disregarded. We hope that readers will appreciate this unusual but interesting approach of the neural networks.

In the general formulation of a nonlinear system as presented in Eq. (2.7), the choice of the *basis functions* $g_k(\cdot)$ is free. We have seen in section 2.3 that Volterra series are based on the Taylor expansion, choosing $g_k = \varphi^k$. A common but more recent choice is to use a dilated (β_k parameters) and/or translated (γ_k parameters) version of the same function for all the g_k. This function is called the *mother basis function* κ:

$$g_k(\varphi(t)) = \kappa(\beta_k(\varphi(t) - \gamma_k)). \tag{2.40}$$

Using Eq. (2.40) in Eq. (2.7) gives the operator of the system such that:

$$\begin{aligned} g(\varphi(t)) &= \sum_{k=1}^{K} \alpha_k g_k(\varphi(t)), \\ &= \sum_{k=1}^{K} \alpha_k \kappa(\beta_k(\varphi(t) - \gamma_k)), \end{aligned} \tag{2.41}$$

where K is the length of the expansion.

Example: choosing κ as the *unit interval indicator* function:

$$\kappa(x) = \begin{cases} 1 \text{ for } 0 \leq x < 1, \\ 0 \text{ else}, \end{cases} \tag{2.42}$$

with $\beta_k = \frac{1}{\Delta}$, $\gamma_k = k\Delta$ and $\alpha_k = f(k\Delta)$ gives:

$$g(\varphi(t)) = \sum_{k=1}^{K} f(k\Delta)\kappa\left(\frac{1}{\Delta}(\varphi(t) - k\Delta)\right), \tag{2.43}$$

which is the piece-wise constant approximation of the function f over intervals of length Δ. In this case the basis functions are called local (variation takes place in local environment). On the contrary, the basis functions of the Volterra expansions are global basis functions (significant variation over the whole real axis).

2.7.1 Activation function

In neural networks jargon, the mother basis function is called the *activation function*, there are plenty of possible activation functions, their properties are discussed in [Juditsky et al., 1995] for nonlinear systems identification with a black-box approach, and in [Karlik and Olgac, 2011; Maas et al., 2013; Pascanu et al., 2013] for a more classical neural networks approach. One of the first used activation function was the step function in an Artificial Neural Network (ANN) named the *perceptron*:

$$\kappa(x) = \begin{cases} 1 \text{ for } x > 0, \\ 0 \text{ else}. \end{cases} \tag{2.44}$$

A smoothed version of this step function is the sigmoid function (presented in Fig. 2.6):

$$\kappa(x) = \frac{1}{1+e^{-x}}. \tag{2.45}$$

The choice of the sigmoid function in neural networks has been motivated by its similarity with the activation function of a human neuron. In modern neural networks, the recommendation is to use *rectified linear unit* [Jarrett et al., 2009] (as presented in Fig. 2.7):

$$\kappa(x) = \max(x, 0), \tag{2.46}$$

or *exponential linear unit* [Clevert et al., 2015] (as presented in Fig. 2.8):

$$\kappa(x) = \begin{cases} \alpha(e^x - 1) \ \forall x < 0, \\ x \text{ else}, \end{cases} \tag{2.47}$$

where $\lim_{x \to -\infty} \kappa(x) = -\alpha$.

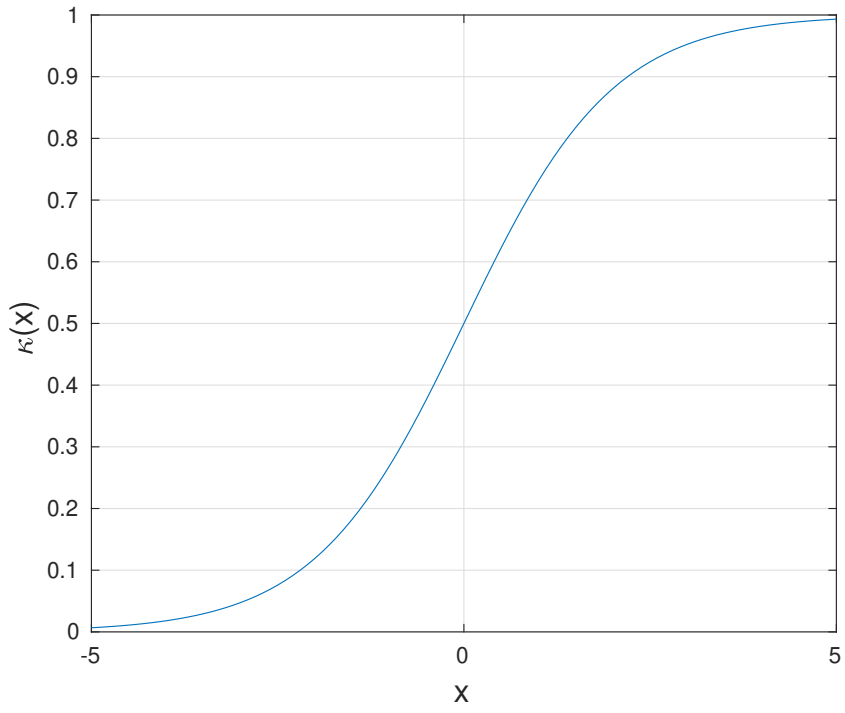


Figure 2.6 Sigmoid function.

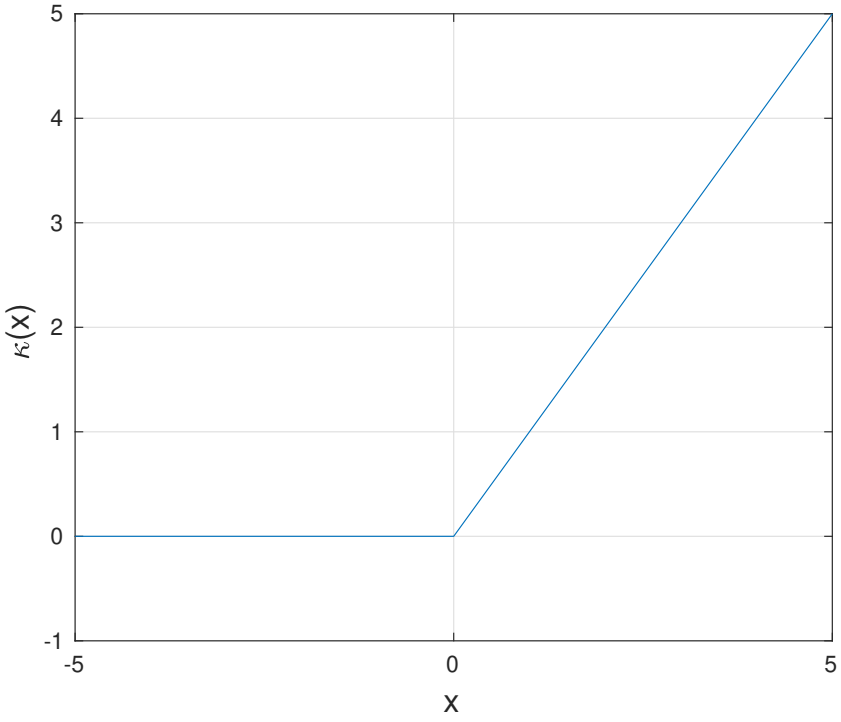


Figure 2.7 Rectified Linear Unit (ReLU) function.

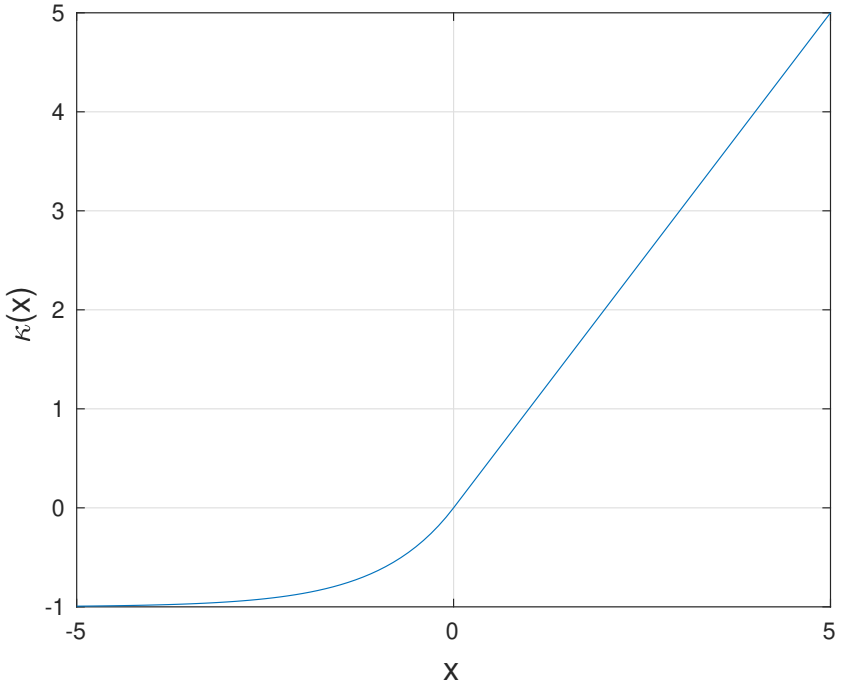


Figure 2.8 Exponential Linear Unit function ($\alpha = 1$).

2.7.2 Feedforward neural networks

A simplified neural network with only 2 inputs, one output and 2 neurons is presented in Fig. 2.9, where W represents a matrix of weights mapping the inputs $\mathbf{x}$ to $\mathbf{h}$ and where $\boldsymbol{\alpha}$ is a vector mapping $\mathbf{h}$ into the output y. Each h_k element is described as a transformation from an input vector to a scalar output by:

$$h_k = \kappa\left(\sum_{j=1}^{N} x_j W_{j,k} + b_k\right), \tag{2.48}$$

which can be written in a matrix form as:

$$h_k = \kappa\left(\boldsymbol{x}^T W_{:,k} + b_k\right), \tag{2.49}$$

where ":" stands for "select all the elements of this index". The output of the system can then be written as:

$$y = \sum_{k=1}^{K} \alpha_k \kappa\left(\boldsymbol{x}^T W_{:,k} + b_k\right), \tag{2.50}$$

which corresponds to the general form of neural networks introduced by Eq. (2.41). In this

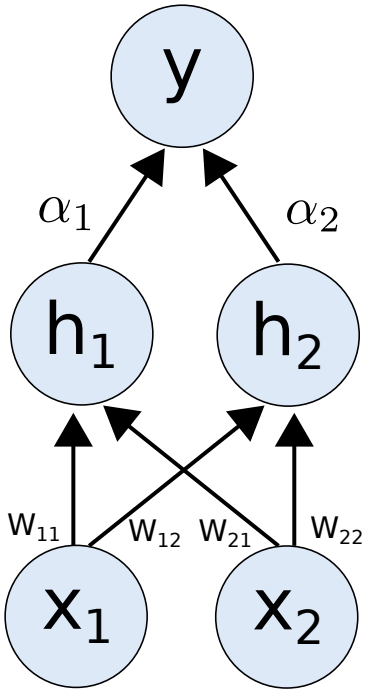


Figure 2.9 Simple feedforward neural network.

case, $\boldsymbol{W}$ (respectively $\boldsymbol{\beta}$ from Eq. (2.41)) are called the *weights* and $\boldsymbol{b}$ (respectively $-\boldsymbol{\gamma}$ from Eq. (2.41)) are the *bias* of the neural network.

2.7.3 Multi-layer neural networks

Multi-layer neural networks, also called Deep Neural Networks (DNN), are the cascade of several feed-forward networks as presented in Fig. 2.10. Instead of linearly combining the

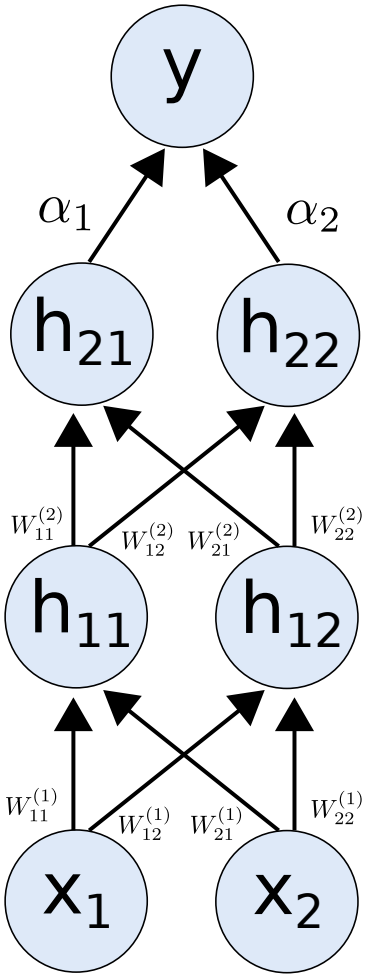


Figure 2.10 Two layers feed-forward neural network.

output of the mother basis functions by the vector $\boldsymbol{\alpha}$, they are collected in a new regression vector:

$$\varphi^{(2)}(t) = [\kappa(\varphi(t), \beta_1, \gamma_1), \ \ldots \ , \kappa(\varphi(t), \beta_K, \gamma_K)]\,. \tag{2.51}$$

Then, this vector is introduced in a new *layer* of basis function forming a new expansion:

$$g(\varphi^{(2)}(t)) = \sum_{l=1}^{L} \alpha_l \kappa(\beta_l(\varphi^{(2)}(t) - \gamma_l)), \tag{2.52}$$

where L is the size of the new expansion (i.e., the number of *neurons* of this new layer) and $\alpha_l, \beta_l, \gamma_l$ are the parameters associated with this layer.

In the matrix form, the multi-layer neural network can be described by a succession of layers $\boldsymbol{h}^{(l)}$ given by:

$$\boldsymbol{h}^{(l)} = \kappa^l \left(\boldsymbol{h}^{(l-1)T} \boldsymbol{W}^{(l)} + \boldsymbol{b}^{(l)} \right), \tag{2.53}$$

where $\boldsymbol{h}^{(0)} = \boldsymbol{x}$ is the input of the neural network. Although one hidden layer is sufficient (in principle) to model almost all reasonable systems, [Sontag, 1993] presents many useful insights to understand the importance of a second hidden layers in a neural networks (i.e., the second layer can improve the convergence rate).

2.7.4 Recurrent neural networks

In a generic approach, a Recurrent Neural Network (RNN) refers to a model whose some regressors $\varphi_k(t)$ are the outputs of the model at a previous moment:

$$\varphi_k(t) = g(\varphi(t - \tau_k), \theta). \tag{2.54}$$

In a more classical neural networks approach, this kind of network is specialized to process discrete-time sequences $\mathbf{x}_1, \cdots, \mathbf{x}_N$, where each $\mathbf{x}_n$ is a scalar, or a vector if the system has several inputs. The RNN (as presented in Fig. 2.11) can be built with a collection of hidden units h_t such as:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t, \theta), \tag{2.55}$$

where the function f is often the result of a simple feed-forward neural network such that the hidden state h_t for a particular time t (called a *time step*) can be computed in a matrix form:

$$\boldsymbol{h}_t = \kappa \left(\boldsymbol{x}_t^T . \boldsymbol{W}_x + \boldsymbol{h}_{t-1}^T . \boldsymbol{W}_h + \mathbf{b} \right). \tag{2.56}$$

The particularity of RNN in machine learning is the sharing of its parameters $\boldsymbol{W}_x, \boldsymbol{W}_h$ (the weights of the network) over the different times steps (see Fig. 2.12). The idea is to have the same processing of the inputs whatever the input lengths or whatever the position of a particular input in the sequence. For example, the sentences "Yesterday, I have eaten an apple" or "I

have eaten an apple yesterday" should be treated equally if the purpose of the network is to determine the moment where I have eaten the apple. The computational graph of a recurrent neural network is said *unfolded* (as it can be seen in Fig. 2.12) if it is presented as a succession of time steps representing the states at each point of the time. The size of matrices and vectors given in Eq. (2.56) are:

- $\boldsymbol{h}_t$ is a vector of size $N_{neurons}$,
- $\boldsymbol{x}_t$ is a vector of size $N_{inputs} \neq 1$ if the system is multi-inputs,
- $\boldsymbol{W}_x$ is a matrix of size $N_{inputs} \times N_{neurons}$,
- $\boldsymbol{W}_h$ is a matrix of size $N_{neurons} \times N_{neurons}$,
- $\mathbf{b}$ is a vector of size $N_{neurons}$.

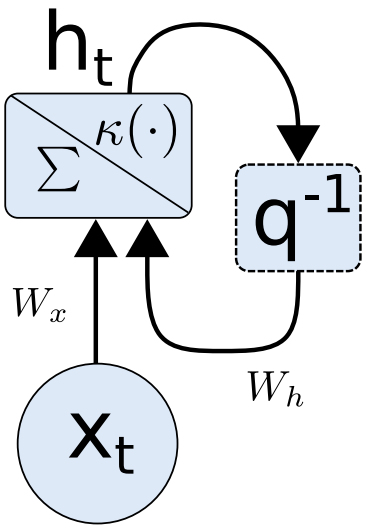


Figure 2.11 Recurrent neural network.

2.8 Introduction to machine learning

In the previous sections, different nonlinear models have been presented but nothing is said about the learning method of the model parameters. Nowadays, the tendency goes to the machine learning. As this thesis is not purely devoted to machine learning, the terms usually related to the field of the neural networks are presented in Sec. 2.8.2.

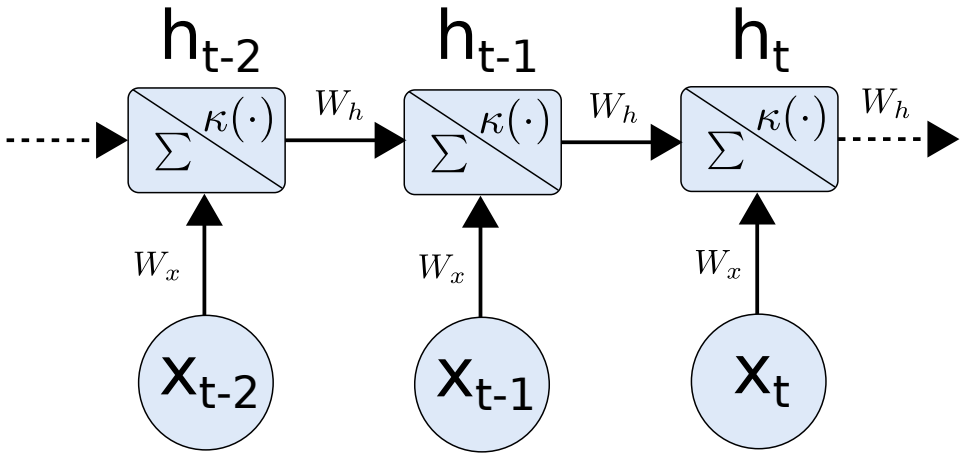


Figure 2.12 Recurrent neural network unfolded over three time steps.

2.8.1 What is machine learning ?

A simple definition could be: *machine learning* is the science of programming computers so they can learn from data [Géron, 2017]. A more rigorous definition is given in [Mitchell, 1997]: **Learning**: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

An experience E

The experience E can be mainly represented by three types of learning process:

- *Supervised learning*: where the data fed in the learning algorithm includes the desired solutions called the *targets* or the *labels*, e.g., regression tasks, classification tasks.
- *Unsupervised learning*: where the data fed in the learning algorithm is unlabeled. The algorithm learns useful properties on the dataset based on its features, e.g., clustering, dimensionality reduction.
- *Reinforcement learning*: interact with its environment. Based on some observations, the system chooses an action and gets rewards or penalties in return. The system has to learn a strategy to get the best reward, e.g., training a robot to return a table tennis ball over the net [Mülling et al., 2013], *Alphago* program [Wang et al., 2016] which has beaten the world champion of Go in March 2016.

For the emulation of the guitar signal chain, the supervised learning for a regression task will be used. From now on, "machine learning" refers to supervised learning process only.

A task T

The task (T) can be divided in several types [Goodfellow et al., 2016, p.99] but most of the time it belongs to one of these two main classes:

- The *Classification tasks*: are tasks where we try to predict (among a finite set of classes) to which class each input belongs to, e.g.,
  - Input mails have to be classified in email or spam classes.
  - Handwriting recognition, each symbol has to belong to one class where each class represents a specific alphanumeric symbol.
- The *Regression tasks*: are tasks where the operator governing a system is approximated to make a quantitative prediction of the output, e.g.,
  - Estimate the probability of cancer based on clinical and demographic variables.
  - Predict the behavior of the stock market based on the politico-economical variables.
  - Emulate the behavior of a physical system.

Our interest here is about the regression class, given the past entries of a guitar signal, we would like to predict the output signal of the system (e.g., a distortion effect, the output of a tube amplifier).

A performance measure P

Mean square error:

In a regression task, the *performance measure* (P) is often a measure of the Mean Square Error (MSE) such that for K targets $y[k] \in \mathbb{R}$ and their associated predictions $\hat{y}[k] \in \mathbb{R}$, $k \in [1, K]$ the performance is measured by:

$$P = MSE = \frac{1}{K} \sum_{k=0}^{K} (y[k] - \hat{y}[k])^2 . \tag{2.57}$$

In the case where y is a discrete-time signal, the percentage of the error relatively to the energy of this output signal can be computed as the Normalized Root Mean Square Error (NRMSE) defined such that:

$$P = NMRSE = \sqrt{\frac{\sum_{k=0}^{K} (y[k] - \hat{y}[k])^2}{\sum_{k=0}^{K} y[k]^2}}. \tag{2.58}$$

This performance measure seems reasonable since it minimizes the distance between the target and the prediction in the mean square sense. Although the choice of the performance measure should be dependent on the desired behavior of the system, it is not always straightforward. For example, the relation between the NRMSE of an emulated audio device and its perceived accuracy could be meaningless.

Coefficient of determination R^2:

The coefficient of determination is a measure of the quality of a prediction for a linear regression. R^2 is equal to 1 if the model perfectly explains the variance of the output from the input variables and is equal to 0 if the model is not better than a constant.

$$P = R^2 = 1 - \frac{MSE}{Var(y)} = 1 - \frac{\sum_{k=0}^{K} (y[k] - \hat{y}[k])^2}{\sum_{k=0}^{K} (y[k] - \bar{y})^2}, \tag{2.59}$$

where $\bar{y}$ is the mean value of the output target $\bar{y} = \frac{1}{K} \sum_{k=0}^{K} y[k]$.

Speed and inference in real time:

Of course, having a model enabling the emulation of an amplifier with a good accuracy is important. However, a faster but less accurate model enabling the emulation in real time could also be useful. For example, the faster model can be used during recording when musicians want to hear what they play and the most accurate model can be used in post production for the final rendering of the sound. Therefore, the performance measure of the model could also depend on the Computational Time (CT) to process a buffer of audio samples. A simple performance measure that takes the CT and the accuracy into account could be:

$$P = CT \cdot MSE. \tag{2.60}$$

This performance measure is useful to compare different model structures, or different set of hyperparameters for the same model. However, it is important to be aware of the trivial cases

where the accuracy is bad but compensated by very small computational time.

Other performance measures have been used to analyze the accuracy of the different models, even if these measures are not used during the learning phase of a model:

- The **Total Harmonic Distortion (THD)**: is the ratio of the sum of the powers of all harmonic components excluding the fundamental frequency to the sum of the powers of all harmonic components including the fundamental frequency [Zölzer, 2011, p.119]:

$$THD_{dB} = 10.\log_{10}\left(\frac{\sum_{i=2}^{N} P_{harmo}(i)}{\sum_{i=1}^{N} P_{harmo}(i)}\right), \tag{2.61}$$

  where $P_{harmo}(i)$ is the power of the ith harmonic and N the number of harmonics considered (including the fundamental).

- The **Signal to Noise Ratio (SNR)**: is the ratio of the power of the desired signal to the power of background noise (unwanted signal):

$$SNR_{dB} = 10.\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right). \tag{2.62}$$

  From now on, the SNR is computed using $P_{signal} = P_{harmo}(1)$: the power of the fundamental.

- The **harmonic content accuracy** (Δ_{harmo}): is the root mean square error between the harmonic Power $P_{harmo,dB}(i)$ of the prediction and target signals:

$$\Delta_{harmo} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(P^{target}_{harmonic,dB}(i) - P^{prediction}_{harmonic,dB}(i)\right)^2}, \tag{2.63}$$

  where N is the number of harmonics taken into accounts. Δ_{harmo} gives information about the accuracy of the predicted harmonic spectrum for a chosen model. From now on, Δ_{harmo} is computed with $N = 10$.

The learning of the neural networks

Historically, Artificial Neural Network (ANN) were first introduced by [McCulloch and Pitts, 1943] in a simplified model on how biological neurons might work together. The parameters are learned using the gradient descent algorithm. They are chosen to minimize (or maximize) a

performance function. The gradient of the function is computed (or estimated) by using the current parameters. Then, those parameters are updated to move against the gradient toward a better minimum of the function. However, neural networks are often composed of a tremendous number of parameters. Therefore, ANN have to wait until the 1980s to know a true success. Indeed until then the computation of the gradient was too expensive, especially for multi-layer neural networks. ANN have known a revival interest when a better algorithm to compute the gradient (called the back propagation algorithm) has been developed. More information about back propagation can be found in Appendix A.1. Actually, most of the work made in the neural networks field is about improving the learnability of the parameters by introducing some novel methods, an overview of these methods is given in Appendix A.2.

Universal approximation theorem

One fundamental contribution in the field of neural networks is the *universal approximation theorem* [Csáji, 2001] which was first proved in [Cybenko, 1989] under a mild assumption on the choice of the activation function (i.e., κ is an activation function if and only if κ is bounded and $\lim_{x\to+\infty}\kappa(x)=a$ and $\lim_{x\to-\infty}\kappa(x)=b$ with $a\neq b$). Actually, [Hornik, 1991] proved that it is the multilayer feedforward architecture and not the choice of the activation function which gives to the neural network the property of being a universal approximator. For notation convenience, the following universal approximation theorem is expressed for a single output unit. The general case, for multiple output units, can be deduced from this:

Universal approximation theorem [Csáji, 2001]:

Theorem 3. Let $\kappa(\cdot)$ be an arbitrary activation function. Let $X\subseteq\mathbb{R}^m$ and X is compact. The space of continuous functions on X is denoted by $\mathbb{C}(X)$. Then $\forall f\in\mathbb{C}(X),\forall\varepsilon>0$: $\exists\, n\in\mathbb{N}_0,\ a_{ij},b_i,w_i\in\mathbb{R},\ i\in\{1...n\},j\in\{1...m\}$:

$$\hat{y}(x_1,\cdots,x_m)=\sum_{i=1}^{n}w_i\kappa\left(\sum_{j=1}^{m}a_{i,j}x_j+b_i\right) \tag{2.64}$$

is an approximation of the function $y=f(x)$ such that:

$$\|\, y-\hat{y}\,\|<\varepsilon \tag{2.65}$$

2.8.2 Terminology and notation used in machine learning

As this thesis is not purely devoted to machine learning, the machine learning terms usually used in this field are presented in this section. Readers who are already familiar with the machine learning field can skip this reminder and directly go to the next section.

Hyperparameter: in machine learning, a hyperparameter is a parameter which is set before the training phase. Consequently, this parameter will not vary during the learning of the training set.

Training set, Test set and Validation set: the dataset used in machine learning is usually split in three parts:

- the *training set* gathers examples that the system uses to learn,
- the *test set* gathers data that are reserved to verify if the trained model generalizes well to unseen data. The error computed on these new cases is called the *generalized error*,
- the *validation set* is used to avoid the search of hyperparameters that fits only with the test set. A common approach is to have a second holdout set used to test different hyperparameters. Once those ones are chosen, the generalized error is computed on the test set to be sure that the model and its hyperparameters generalize well to unseen data.

An instance: is an element of the training set, it is one input of the system at a given time.

Target: corresponds to the true output of the system for a given instance. It is also called the *ground truth* and is noted $target[n]$ or $y[n]$.

Prediction: corresponds to the output of the model noted hereafter $pred[n]$ or $\hat{y}[n]$.

Feature and Attribute: in machine learning, an *attribute* or a *feature* has several meanings depending on the context, but it generally refers to an individual measurable property or characteristic of an object or an observed phenomenon (e.g., *Mileage=15000*, *Temperature=30*). Many people use the words *attribute* and *feature* interchangeably [Géron, 2017].

Batch: optimization algorithms using all the data present in the training set before computing the gradient are called *batch* gradient algorithms or even *deterministic* gradient algorithms.

Minibatch: is a subset of a batch. The reason to use mini-batch is that computing the mean gradient function for all the instances of the training set can be very time consuming. In practice, an estimation of the true gradient can be obtained by randomly sampling some instances of the batch, then taking average over these picked instances. The gradient method is then called *minibatch stochastic*. If the size of the minibatch is equal to one, the method is called *stochastic* or *online*. These definitions are often subject to abuses, e.g., minibatch stochastic are often simply called stochastic methods. Moreover batch and minibatch are often exchanged, e.g., "the size of the batch" is often used to refer to the size of the minibatch.

Epoch: the gradient descent is an iterative method, meaning that the weights are updated at each *iteration* (i.e., it computes the gradient then it updates the weights). The moment where the algorithm has seen all the data of the training set is called an *epoch*. Many epochs may be necessary to reach the minimum of the cost function.

Learning rate: The learning rate or step size in machine learning is a hyperparameter which determines to what extent newly acquired information overrides old information [Zulkifli, 2018]. This parameter is often represented by α or η. If the learning rate is too low, the algorithm takes too long to converge to the minimum, moreover it can stay stuck in a local minimum. On the contrary, a too high learning rate can make the learning to jump over the minimum.

Fully connected: A fully connected (FC) layer also called a *Dense* layer, is a layer where each neuron is connected at each neuron in the previous layer by weights. A fully connected layer is usually followed by a nonlinear activation function.

Part II

Emulation of the Guitar Signal Chain

Table of Contents

Chapter 3

Hammerstein kernels identification by exponential sine sweeps

Contents

Outline of this chapter:

This chapter is devoted to the emulation of guitar effects by the parallel cascade of Hammerstein models and is organized as follows : the basic identification technique is recalled in Sec. 3.2. This theory is illustrated with a simple third-order model and then extended to any finite order. In Sec. 3.3, five problems that may occur during the implementation of the method are described. An algorithm for testing the reconstruction of the Hammerstein kernels is proposed to highlight and solve these problems. This leads to the definition of an *Hammerstein Kernels Identification by Sine Sweep* (*HKISS*) method. Its efficiency is proven in Sec. 3.4 where the method is applied on three different nonlinear audio devices: two tube amplifiers and a distortion guitar effect. The Sec. 3.5 offers an overview on the limitations of the *HKISS* method. Finally, the Sec. 3.6 presents a *Matlab* toolbox helping to identify the Hammerstein kernels of a nonlinear device through the *HKISS* method.

3.1 Introduction

3.1.1 Related works

The polynomial Volterra series as described in Sec. 2.3 is a nice nonlinear operator which enables the approximation (with a chosen bounded error) of any time-invariant nonlinear system under the mild assumption that the system has fading memory (see Theorem 1 in Sec. 2.3). However, the number of computations required to describe a highly nonlinear system grows exponentially with its order. Therefore, the Volterra series could be unappropriated since:

- these numerous kernel coefficients are difficult to identify,
- there are too much computational operations to emulate a strong nonlinear system in real time.

In the last few years, a nonlinear convolution method [Farina et al., 2001; Farina, 2000] has been elaborated with the aim of emulating nonlinear audio systems. The authors proposed to use the properties of the Exponential Sine Sweep (ESS) signal in order to identify a sub-model of the Volterra model by considering only the static (memoryless) nonlinearities. This model is called the *Hammerstein* model (as presented in Sec. 2.4.1). More precisely, the studied model is the *polynomial parallel cascade of Hammerstein models* (as presented in Fig. 3.1).

It is composed of M (the system order) parallel branches where each branch m gathers the cascade of a static monomial nonlinearity (putting the input signal to the power m) followed by a linear filter $h_m[n]$ which corresponds to the m^{th} Hammerstein kernel (also called the diagonal Volterra kernel) [Bendat, 1998; Ogunfunmi, 2007; Kadlec et al., 2008; Rébillat et al., 2011]. With the ESS signal sent through a nonlinear system, they show that it is possible to separate the different harmonic impulse responses created by the harmonic distortion. This method has been quite successfully applied to model nonlinearities in loudspeakers [Farina et al., 2001; Schmitz and Embrechts, 2013, 2014], guitar distortion generators [Tronchin, 2013; Novak et al., 2010a] (the famous *Tube-Screamer* [Ibanez, 1978] is often taken as an example) and compressors [Novak et al., 2009].

However, this method has some limitations (as it will be explained in Sec. 3.3) that may lead to some inaccuracies during the emulation phase. One of these limitations (the nonsynchronization of the phase) has been addressed in [Novak et al., 2009] and then improved in [Novak et al., 2015] by using a special ESS signal called *the exponential synchronized sine sweep* or simply the *Synchronized Sine Sweep* (SSS). We have proposed another method to circumvent the problem of nonsynchronized phase whithout the need of using a SSS [Schmitz and Embrechts, 2013, 2014]. Moreover, we have proposed in [Schmitz and Embrechts, 2017] to list and solve the possible error sources when dealing with this method.

An early solution to the problem of input level dependency (see Sec. 3.2.4) of the Hammerstein kernels is discussed in [Tronchin and Coli, 2015; Schmitz and Embrechts, 2017]. Finally, the effect of using a Hammerstein model on a non-Hammerstein nonlinear system (creating some artifacts in the measured impulse responses) has been discussed in [Torras-Rosell and Jacobsen, 2011; Schmitz, 2012; Ćirić et al., 2013].

In this chapter, we first highlight some difficulties that must be considered during the implementation of the *ESS* method, in order to perform an accurate identification of nonlinear (*NL*) systems. The nature of these problems is first described, then a solution is proposed for each of them. Finally, these solutions are validated by the emulation of real nonlinear audio devices.

3.2 Hammerstein kernels identification method

This section briefly reminds the principles of the used Hammerstein kernels identification method. In order to ensure a good understanding, we illustrate it with a third-order model, before extending it to a general order M.

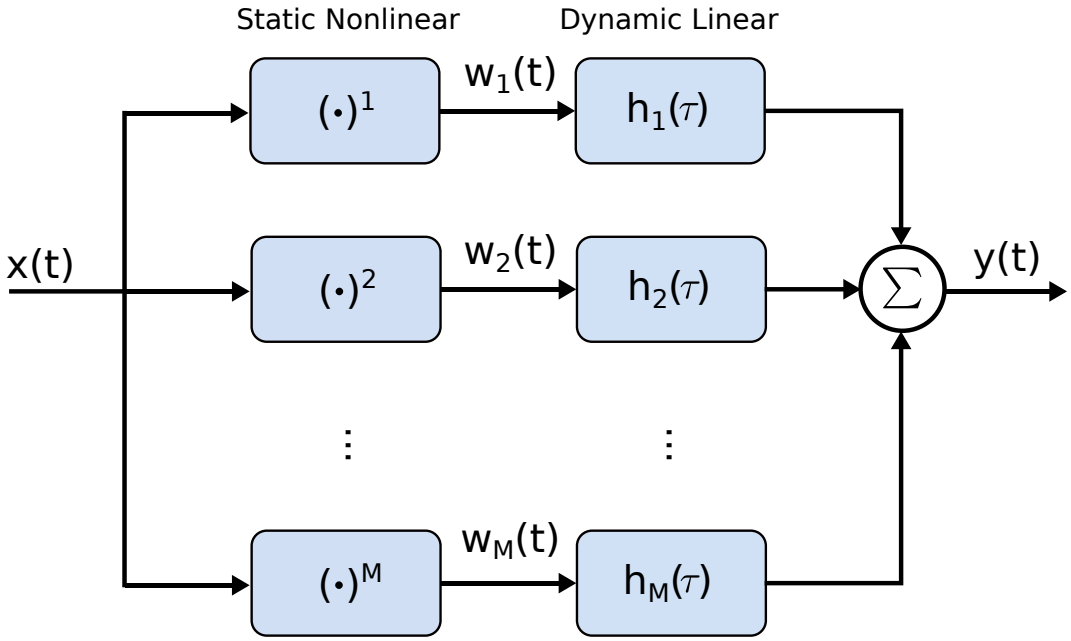


Figure 3.1 The polynomial parallel cascade of Hammerstein models

3.2.1 Volterra series

Considering time-invariant nonlinear systems, the Volterra series expresses the relationship between an input signal $x(t)$ and its corresponding output signal $y(t)$ as an infinite sum of multiple convolutions [Ogunfunmi, 2007]:

$$y(t) = \sum_{m=1}^{\infty} \int_{-\infty}^{+\infty} \dots \int_{-\infty}^{+\infty} v_m(\tau_1, \dots, \tau_m) . \prod_{i=1}^{m} x(t-\tau_i)\, d\tau_1 \dots \tau_m, \tag{3.1}$$

where $\{v_m(\tau_1, \dots, \tau_m)\}_{\forall m \in \mathbb{N}^+}$ are the Volterra kernels characterizing the system.

3.2.2 Cascade of Hammerstein models

It has been shown in Sec. 2.4.3 that any continuous nonlinear system can be represented by a finite set of M parallel branches composed by a static nonlinearity $P_m(.)$ surrounded by two linear filters. The parallel cascade of Hammerstein models is a subclass of this general model where only static nonlinear monomial functions are followed by a linear filter $h_m(t)$ (the Hammerstein kernels), as presented in Fig. 3.1. A nonlinear, continuous time, causal Hammerstein model with a finite order M can express its input/output relationship as:

$$y[n] = \sum_{m=1}^{M} \int_{-\infty}^{+\infty} h_m(\tau) . x^m(t-\tau). \tag{3.2}$$

The Volterra kernels from Eq. (3.1) and the Hammerstein kernels from Eq. (3.2) are equivalent if $v_m(\tau_1,\ldots,\tau_m) = 0$ when $\tau_i \neq \tau_j \ \forall i,j \in [1,m]$ and $i \neq j$. In this case, $v_m(\tau_1,\ldots,\tau_m) = v_m(\tau_1,\ldots,\tau_m = \tau_1) = h_m(\tau)$ and the Hammerstein kernels correspond to the Volterra kernels taken on their principal diagonals (as presented in Eq. (2.21)). Therefore, the polynomial parallel cascade of Hammerstein models is also called a diagonal Volterra model. Its equivalent in discrete time is given by:

$$y[n] = \sum_{m=1}^{M} h_m[n] * x^m[n], \tag{3.3}$$

where the symbol "$*$" refers to the convolution operator defined in Eq. (2.3).

3.2.3 Hammerstein kernels identification using the *ESS* method

In order to enable the extraction of the Hammerstein kernels, the nonlinear system has to be excited with a special input signal (i.e., the Exponential Sine Sweep (ESS)). Such a signal should enable the separation of the different harmonic impulse responses after the deconvolution of the nonlinear system output by the inverse filter of the ESS. The next two subsections explain the properties of the ESS input signal and how it can be used to compute the Hammerstein kernels.

The following is greatly inspired by our paper published in the Journal of the Audio Engineering Society [Schmitz and Embrechts, 2017].

Exponential sine sweep phase properties

In the following, our development starts from the original formulation of Farina et al. [Farina et al., 2001; Farina, 2000]. The equations are expressed in discrete time for a better correspondence with their practical implementations. Let $ss[n]$ be the *ESS* signal such that:

$$ss[n] = A\sin(\phi[n]), \tag{3.4}$$

with a phase $\phi[n]$ that grows exponentially such that :

$$\phi[n] = \omega_1 \frac{R}{f_s}.(e^{\frac{n}{R}} - 1) \qquad \forall n \in [0, N-1], \tag{3.5}$$

where:

- $R = (N-1).(\log \frac{\omega_2}{\omega_1})^{-1}$ is the inverse of the frequency changing rate. Note: in this dissertation, the log symbol refers to the *natural logarithm* (logarithm in base e) also often noted as ln.
- N is the length of the sweep in samples.
- ω_1, ω_2 are respectively the initial and final angular frequencies ($\mathrm{rad\,s^{-1}}$).
- f_s is the sampling rate (Hz).

An interesting property of the *ESS* (as explained in Appendix. B.1) is that the m^{th} harmonic of the *ESS* can be derived from the *ESS* itself delayed by Δ_m, as:

$$m.\phi[n] = \phi[n+\Delta_m] - B(m-1) \quad \forall n \leq N-1-\Delta_m, \tag{3.6}$$

where:

$$\begin{cases} B = \dfrac{\omega_1.R}{f_s}, \\ \Delta_m = R.\log(m). \end{cases} \tag{3.7}$$

The B term of Eq. (3.6) could be neglected in the following operations if $B = 2k\pi$, *with* $k \in \mathbb{N}$, since it just adds an integer multiple of 2π to the sine phase. However, this forces the ESS signal to have specific parameters which could be not ideal.

The property of the *ESS* signal in Eq. (3.6) is interesting since after its deconvolution by the *ESS* inverse filter $\overline{ss}[n]$, the contribution of a harmonic m in the impulse response will be delayed by $-\Delta_m$ samples from the fundamental impulse response. In particular, if $B \in 2k\pi$, the deconvolution of the ESS by its inverse filter can be written as:

$$\begin{cases} ss[n] * \overline{ss}[n] = \sum_{k=0}^{N-1} ss[k].\overline{ss}[n-k] = C.\delta[n-n_0], \\ \sin(m.\phi[n]) * \overline{ss}[n] = C.\delta[n-(n_0-\Delta_m)], \end{cases} \tag{3.8}$$

where C is a constant, δ is the Dirac delta function [Oppenheim et al., 1996, p.30] and n_0 is the position along the time axis of the first order (linear) impulse response which depends on the *ESS* length (i.e., $n_0 = N-1$ [Holters et al., 2009]). One can notice that the values Δ_m are real and most often non-integer (except for m=1). Consequently, a phase correction will be required during the extraction of the harmonic impulse responses in order to avoid phase reconstruction problems when computing the Hammerstein kernels as it is explained in Sec. 3.3.4. We must note at this point that if $m \neq 1$, the second convolution in Eq. (3.8) is not the same delayed

version of the Dirac pulse than in the case where $m = 1$. This problem will be addressed in Sec. 3.3.6.

The inverse *ESS* can be computed in several ways [Rébillat et al., 2011; Novak et al., 2015; Holters et al., 2009; Meng et al., 2008; Norcross et al., 2002; Kirkeby and Nelson, 1999]. However, the advantage of using an analytical computation of the inverse sine sweep as described in [Novak et al., 2015] is that the bandwidth of the inverse filter can be extended to any frequency if needed. As explained in Appendix. B.2, the analytical version of $\overline{ss}[n] \xrightarrow{\mathscr{F}} \overline{SS}(\omega)$ can be computed as:

$$\begin{cases} \overline{SS}(\omega) = 2\sqrt{\dfrac{\omega}{2\pi L}}e^{-j\left[L\left(\omega\left(1-\ln\left(\frac{\omega}{\omega_1}\right)\right)-\omega_1\right)-\frac{\pi}{4}\right]}, \\ \text{with } L = \dfrac{R}{f_s}. \end{cases} \tag{3.9}$$

A third order model example

In order to illustrate the origin of the difficulties mentioned in the introduction, we propose to establish the Hammerstein kernels equations for a model developed up to the third order:

$$y[n] = x[n] * h_1[n] + x^2[n] * h_2[n] + x^3[n] * h_3[n]. \tag{3.10}$$

An extension of the method is then developed up to any order M at the end of this section. The goal is to obtain M equations giving the M Hammerstein kernels $h_m[n]$.

Overview of the Hammerstein Kernel Identification by Sine Sweep (HKISS) method:

1. The system to model is excited by an ESS and its output y is recorded.
2. The signal y is deconvolved by the inverse filter of the ESS (i.e., $\overline{ss}$). The resulting signal is called the global impulse response g.
3. g is cut into M separated impulse responses g_m corresponding to the M harmonic impulse responses of the system.
4. A transformation T is applied to the g_m impulse responses in order to compute the h_m Hammerstein kernels.
5. Eq. (3.3) can then be applied to emulate the system for other input signals.

The first step is to use the *ESS* signal (3.4) as an input of the model. Developing the Eq. (3.3) up to the third order gives:

$$\begin{aligned} y[n] = \; & A\sin(\phi[n]) * h_1[n] \\ & + A^2\sin^2(\phi[n]) * h_2[n] \\ & + A^3\sin^3(\phi[n]) * h_3[n]. \end{aligned} \tag{3.11}$$

Using trigonometric formulas [Zwillinger, 2002], the sine powers are decomposed into an order-one formulation:

$$\begin{aligned} y[n] = \; & A\sin(\phi[n]) * h_1[n] \\ & - \frac{A^2}{2}\left[\cos(2\phi[n]) - 1\right] * h_2[n] \\ & + \frac{A^3}{4}\left[3\sin(\phi[n]) - \sin(3\phi[n])\right] * h_3[n]. \end{aligned} \tag{3.12}$$

In order to deconvolve the signal $y[n]$ properly, the cosine term has to be changed in a sine expression. This can be done by using the *Hilbert Transform* [Zwillinger, 2002; Qin et al., 2008] :

$$\begin{cases} \mathscr{H}\{\sin(u(t))\} \triangleq \sin(u(t)) * \hbar(t) = -\cos(u(t)), \\ \mathscr{H}\{\cos(u(t))\} \triangleq \cos(u(t)) * \hbar(t) = \sin(u(t)), \end{cases} \tag{3.13}$$

where $\mathscr{H}\{.\}$ is the *Hilbert Transform* and $\hbar(t) = \frac{1}{\pi t}$ its kernel. Introducing (3.13) in (3.12) and using the phase property (3.6) of the ESS, the output of the system can be written as:

$$\begin{aligned} y[n] &= A\sin(\phi[n]) * h_1[n] \\ &\quad + \frac{A^2}{2}\left[\sin(\phi[n+\Delta_2] - B) * \hbar[n] + 1\right] * h_2[n] \\ &\quad + \frac{A^3}{4}\left[3\sin(\phi[n]) - \sin(\phi[n+\Delta_3] - 2B)\right] * h_3[n] \\ &= ss[n] * h_1[n] \\ &\quad + \frac{A}{2}\left(ss[n+\Delta_2]\cos(B) * \hbar[n] - ss[n+\Delta_2]\sin(B)\right) * h_2[n] \\ &\quad + \frac{A^2}{4}\left(3ss[n] - ss[n+\Delta_3]\cos(2B) - ss[n+\Delta_3]\sin(2B) * \hbar[n]\right) * h_3[n], \end{aligned} \tag{3.14}$$

where the DC offset of the even power has been neglected as it is only related to the mean value of the *ESS*. Moreover, in practice, most audio devices stop the DC component. In the

frequency domain, using the trigonometric exponential property, the *Euler* formula and the *Hilbert Transform* property [Zwillinger, 2002], the Eq. (3.14) becomes (see Appendix. B.3 for the development) for $\omega > 0$:

$$\begin{aligned} Y(\omega) &= SS(\omega).H_1(\omega) \\ &- \frac{A}{2}\left(jSS(\omega).e^{-jB}.e^{\frac{j\omega\Delta_2}{f_s}}\right).H_2(\omega) \\ &+ \frac{A^2}{4}\left(3.SS(\omega) - SS(\omega).e^{-j2B}e^{\frac{jw\Delta_3}{f_s}}\right).H_3(\omega). \end{aligned} \tag{3.15}$$

One can notice that since the Hammerstein kernels are real in the time domain, the Hermitian property [Oppenheim et al., 1996] can be used in the frequency domain to find the Hammerstein kernels for negative frequencies such that:

$$H_m(-\omega) = H_m^*(\omega). \tag{3.16}$$

The second step is to deconvolve the signal $Y(\omega)$ with the inverse filter $\overline{SS}(\omega)$:

$$\begin{aligned} G(\omega) = \frac{Y(\omega).\overline{SS}(\omega)}{C} &= e^{-j\omega n_0/fs}H_1(\omega) \\ &- \frac{A}{2}\left(j.e^{-jB}.e^{\frac{-j\omega(n_0-\Delta_2)}{f_s}}\right).H_2(\omega) \\ &+ \frac{A^2}{4}\left(3e^{-j\omega n_0/fs} - e^{-j2B}e^{\frac{-jw(n_0-\Delta_3)}{f_s}}\right).H_3(\omega). \end{aligned} \tag{3.17}$$

Using the inverse Fourier transform, the Eq. 3.17 can be expressed in the time domain such that:

$$\begin{aligned} g[n] &= h_1[n-n_0] \\ &- \frac{A}{2}\left(j.e^{-jB}.h_2[n-n_0+\Delta_2]\right) \\ &+ \frac{A^2}{4}\left(3h_3[n-n_0] - e^{-j2B}h_3[n-n_0+\Delta_3]\right) \end{aligned} \tag{3.18}$$

As it can be seen, the deconvolved output of the system is an impulse response $g[n]$ formed by the sum of the Hammerstein kernels $h_m[n]$ positioned around some specific locations given by n_0 or $n_0 - \Delta_m$ (as presented in Fig.3.2). The difficulty is now to be able to separate the different kernels occupying the same position on the time axis (e.g., the Hammerstein kernels

number 1 and 3 both bring contributions centered on n_0). By grouping in g_m the kernels having the same position in the time domain or by grouping the $G_m(\omega)$ terms having the same phase delay $e^{-j\frac{\omega}{f_s}(n_0-\Delta_m)}$ in the frequency domain, the Eq. (3.17) can be written in a matrix form such that:

$$\begin{pmatrix} G_1(\omega) \\ G_2(\omega) \\ G_3(\omega) \end{pmatrix} = \begin{pmatrix} 1 & 0 & \frac{3A^2}{4} \\ 0 & \frac{-jAe^{-jB}}{2} & 0 \\ 0 & 0 & \frac{-A^2e^{-j2B}}{4} \end{pmatrix} \cdot \begin{pmatrix} H_1(\omega) \\ H_2(\omega) \\ H_3(\omega) \end{pmatrix}. \tag{3.19}$$

By inverting this relation, the Hammerstein kernels ($\forall \omega > 0$) are given by:

$$\begin{pmatrix} H_1(\omega) \\ H_2(\omega) \\ H_3(\omega) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 3e^{j2B} \\ 0 & \frac{2je^{jB}}{A} & 0 \\ 0 & 0 & \frac{-4e^{j2B}}{A^2} \end{pmatrix} \cdot \begin{pmatrix} G_1(\omega) \\ G_2(\omega) \\ G_3(\omega) \end{pmatrix}. \tag{3.20}$$

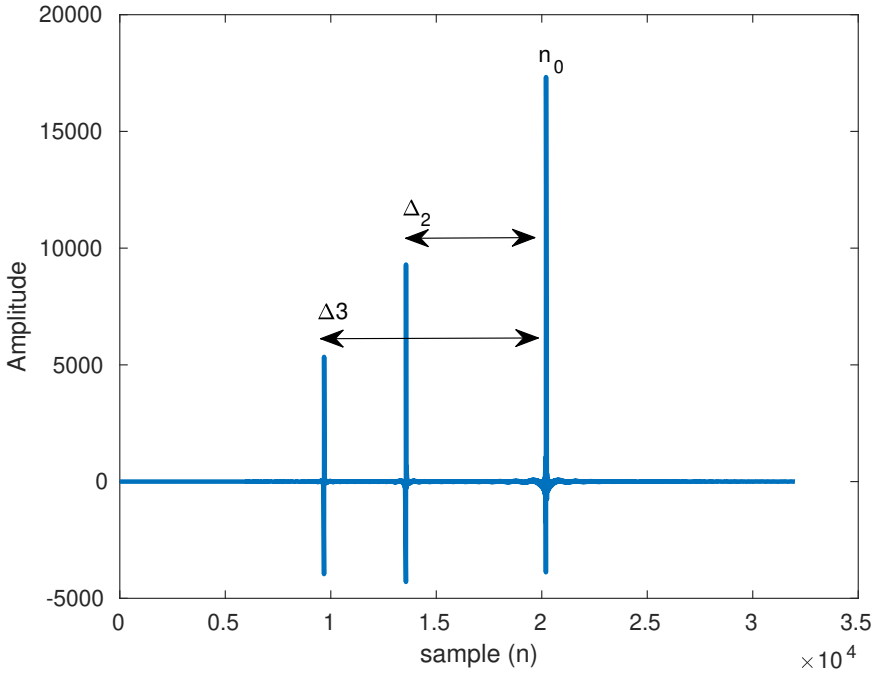


Figure 3.2 Deconvolution of the power series (3.10) up to the third order when the input is an ESS in the frequency range [50, 5000] Hz.

Extension of the method up to the order M

The generalization up to the order M can be done by means of the trigonometric power formulas of sine [Spiegel, 1968]:

$$\begin{cases} \sin^{2n+1}(t) = \dfrac{-1^n}{4^n}\displaystyle\sum_{k=0}^{n}(-1)^k C_{2n+1}^k \ \sin([2n+1-2k]t) \\ \\ \forall n \in \mathbb{N}, \\ \\ \sin^{2n}(t) = \dfrac{-1^n}{2^{2n-1}}\displaystyle\sum_{k=0}^{n-1}(-1)^k C_{2n}^k \cos([2n-2k]t) + \dfrac{C_{2n}^n}{2^{2n}} \\ \\ \forall n \in \mathbb{N}_0, \end{cases} \tag{3.21}$$

where C is the binomial [Abramowitz and Stegun, 1965, p.265]:

$$C_n^k = \frac{n!}{k!(n-k)!}. \tag{3.22}$$

Let first assume that $B \in 2k\pi$. Using Eq. (3.6) it can then be written that:

$$\sin(m.\phi[n]) = \sin(\phi[n+\Delta_m]) \quad \forall n \leq N-1-\Delta_m. \tag{3.23}$$

Using this equation in Eq. (3.21), enables to derive power formulas of the *ESS* signal for the positive frequency domain:

$$\begin{cases} SS^{2n+1} = SS\dfrac{-1^n}{4^n}\displaystyle\sum_{k=0}^{n}(-1)^k C_{2n+1}^k e^{j\omega\Delta_{2n+1-2k}/f_s} \\ \\ \qquad\quad = a_n.SS \ \forall n \in \mathbb{N}, \\ \\ SS^{2n} = j.SS.\dfrac{-1^n}{2^{2n-1}}\displaystyle\sum_{k=0}^{n-1}(-1)^k C_{2n}^k e^{j\omega\Delta_{2n-2k}/f_s} \\ \\ \qquad\quad = b_n.SS \ \forall n \in \mathbb{N}_0, \end{cases} \tag{3.24}$$

where SS^m is the Fourier transform of mth power of the ESS signal and considering that the term $\frac{C_{2n}^n}{2^{2n}}$ corresponding to the DC component can be removed as explained earlier.

Deconvolution process

Taking the development back to the Eq. (3.17) but applying (3.24) for sine powers, the deconvolution in the frequency domain of the Hammerstein model given in Eq. (3.3) leads to:

$$\begin{cases} Z(\omega) = Y(\omega)\overline{SS}(\omega)e^{j\omega n_0/f_s} \\ \qquad = H_1(\omega) + \sum_{m=1}^{MM} \left[a_m H_{2m+1}(\omega) + b_m H_{2m}(\omega)\right], \end{cases} \tag{3.25}$$

where, $MM = (M-1)/2$. The transformation of Eqs. (3.24), (3.25) into the time domain shows that $z[n]$ is a superposition of several versions of each Hammerstein kernel $h_m[n]$ delayed by Δ_k ($e^{j\omega\Delta_k/f_s}$ in the Fourier domain). We name $g_m[n]$ the superposition of the $h_i[n]$ having the same delay Δ_m. Furthermore, we consider that all these harmonic impulse responses "start" at n=0, which requires to apply a delay of Δ_m to all of them:

$$\begin{pmatrix} G_1(\omega) \\ \vdots \\ G_M(\omega) \end{pmatrix} = \mathbf{T}. \begin{pmatrix} H_1(\omega) \\ \vdots \\ H_M(\omega) \end{pmatrix}, \tag{3.26}$$

where the elements $\mathbf{T}_{u,v}$ with $u, v \leq M$ can be developed using Eq. (3.25) in the matrix form and grouping in $G_m(\omega)$ the $H_m(\omega)$ having the same phase delay:

$$\mathbf{T}_{u,v} = \begin{cases} \frac{(-1)^{\frac{1-u}{2}}}{2^{v-1}} C_v^{\frac{v-u}{2}} & \forall v \geq u \;\&\quad \mod\left(\frac{u+v}{2}\right) = 0 \\ 0 & \text{else.} \end{cases} \tag{3.27}$$

Finally, the Hammerstein kernels in the positive frequency domain are given by the following relationship:

$$\begin{pmatrix} H_1(\omega) \\ \vdots \\ H_M(\omega) \end{pmatrix} = (\mathbf{T})^{-1} \begin{pmatrix} G_1(\omega) \\ \vdots \\ G_M(\omega) \end{pmatrix}. \tag{3.28}$$

The case where $B \neq 2k\pi$ with $k \in \mathbb{N}$ is discussed in Sec. 3.3.3.

3.2.4 Note on the Hammerstein model and its input level dependency

If the nonlinear system to model belongs to the Hammerstein class, the cascade of Hammerstein models takes the amplitude of the input signal into account. For example, if the nonlinear system to model is a power series up to the 6^{th} order, Fig. 3.3 compares the output signal

of this power series when the input signal is $x = 0.5\sin(2\pi 1000t)$ with its emulation using the Hammerstein kernels which have been obtained by using an *ESS* of amplitude 1. The signal is reconstructed correctly. However, a real nonlinear system, could no longer match perfectly the Hammerstein model (see the condition to belong to a Hammerstein model in Sec.2.4.1). Therefore, the Hammerstein kernels could be different according to the amplitude of the input signal [Novak et al., 2010a; Tronchin and Coli, 2015]. From now on, $H_m^{\alpha}(\omega)$ are the Hammerstein kernels measured using an *ESS* whose amplitude is α :

$$ss[n] = \alpha\sin(\phi[n]). \tag{3.29}$$

To illustrate the Hammerstein kernels amplitude dependency, when the system to model does not perfectly match the Hammerstein model, Fig. 3.4 presents the comparison between an *ESS* (focus on frequencies close to f=1000 Hz) with $A = 0.5$ passing through the distortion part of the amplifier `Ibanez TSA15h` and its emulation using the same input signal. The emulation is processed using the Hammerstein kernels measured using an *ESS* of amplitude $\alpha = 1$ (i.e., $H_m^1(\omega)$). As it can be seen, the signal coming from the distortion effect (y_1) is different from the emulated signal (y_2), while the emulation of this amplifier matches perfectly if using $A = \alpha$ (see Sec. 3.4.2).

For the remaining part of this chapter, the symbol α over the Hammerstein kernels will be omitted if $\alpha = 1$.

3.3 Optimization of the Hammerstein kernels identification method

Eq. (3.28) can be used to compute the Hammerstein kernels. However, several difficulties must be taken into account to enable an accurate emulation. In order to illustrate them, a top down approach is chosen, starting with the optimized case (in Sec. 3.3.2) where all the precautions are taken. Then, they are removed one by one to prove their necessity (see Sec. 3.3.3, 3.3.4, 3.3.5, 3.3.6).

3.3.1 Test algorithm to highlight the potential difficulties of the HKISS method

We propose a test algorithm (see Fig. 3.5) based on the nonlinear system described by Eq. (3.3) which belongs to the Hammerstein class. The kernels $h_m[n]$ are chosen to be simple all-pass

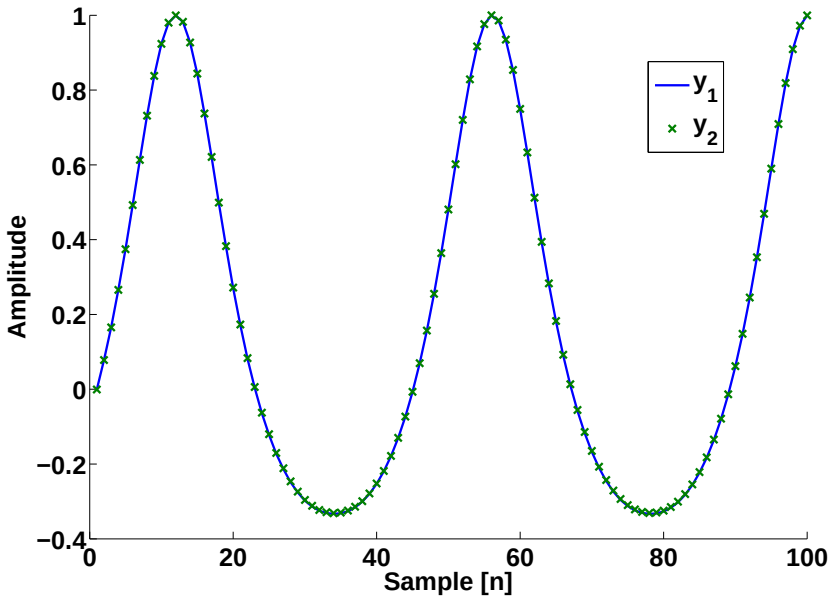


Figure 3.3 Comparison between a sine of amplitude 0.5 with a frequency f=1000 Hz passing through a power series up to the order 6 (y_1) and its emulation (y_2) using the Hammerstein kernels measured for an *ESS* of amplitude 1 (y_1 and y_2 have been normalized to be compared).

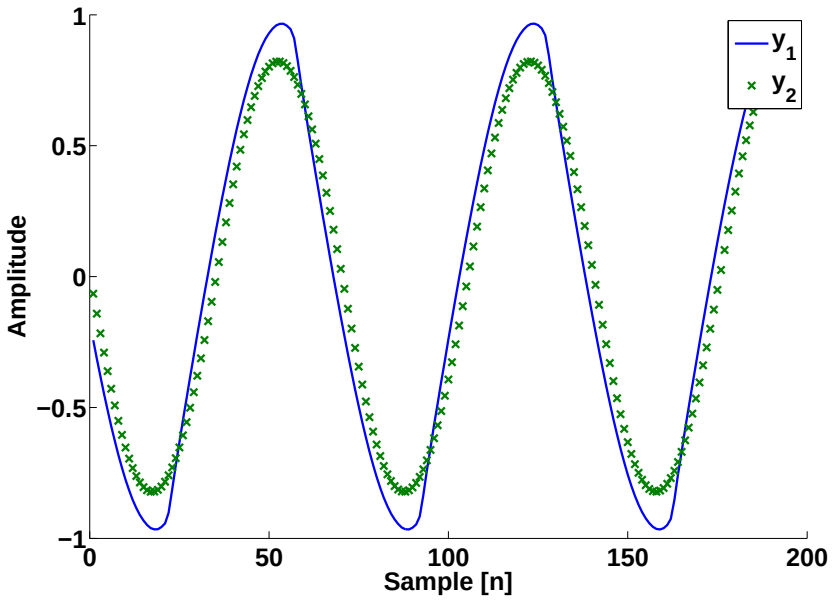


Figure 3.4 Comparison between an *ESS* of amplitude 0.5 (with frequencies close to f=1000 Hz) passing through the distortion effect of the amplifier `Ibanez TSA15h` (y_1) and its simulation (y_2) by means of Hammerstein kernels measured for an *ESS* of amplitude 1.

filters restricted to the Nyquist band $[0, f_s/2]$ in such a way that the created system is equivalent to a power series.

Using the *ESS* as input signal, the output y_1 is the result of the power series (see Sec. 3.3.7 for aliasing consideration). Deconvolving y_1 with the *ESS* inverse filter $\overline{ss}[n]$ enables to compute the Hammerstein kernels of this nonlinear system. Then, using Eq. (3.3) the emulated signal y_2 can be computed based on the Hammerstein kernels and the *ESS* powers. If the Hammerstein kernels are correctly reconstructed, y_1 and y_2 have to be identical.

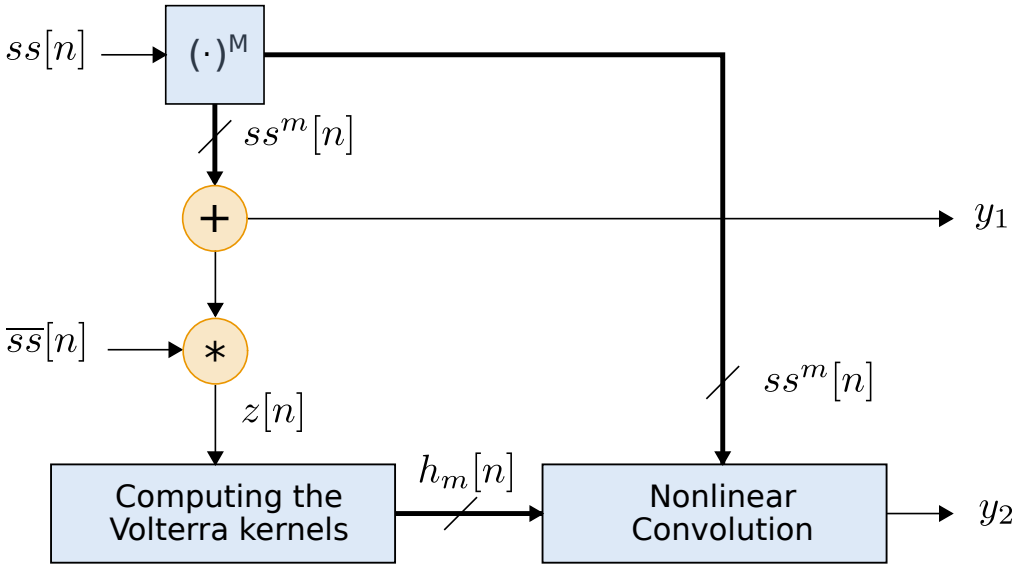


Figure 3.5 Test algorithm: the output of the power series (y_1) and its emulation by means of Hammerstein kernels (y_2) have to be identical.

3.3.2 The best case: a correct reconstruction through the HKISS method

Following the top down approach, Fig. 3.6 presents the *best match* between the output of the power series which is the target y_1 (up to the order M=6) and its emulation by means of the Hammerstein kernels (i.e., the prediction y_2). For clarity, a focus on a part of the sweep signals (arbitrarily chosen at frequencies close to $f = 1000\,\text{Hz}$) has been applied. As it can be seen in Fig. 3.6, the power series is correctly reconstructed. This *best match* has been obtained by following all the precautions and corrections described in the next subsections.

3.3.3 Hammerstein kernels phase errors

The model (3.20) shows that the Hammerstein kernels depend on a constant factor B introduced in the phase of the power of the *ESS*. During the general formulation given at Eq. (3.28), the

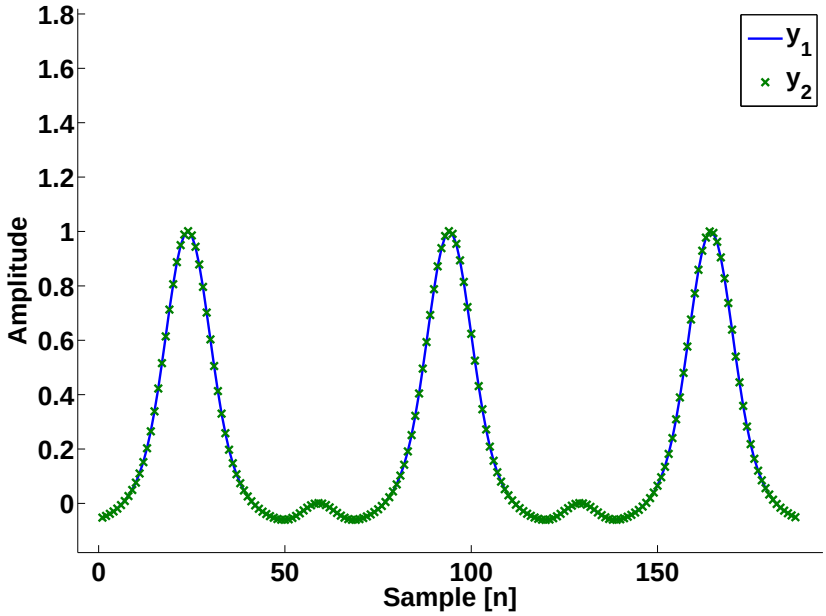


Figure 3.6 Comparison between the target $y_1[n]$ and the prediction $y_2[n]$ at $f \simeq 1000\,\text{Hz}$ (best case).

simplifying assumption that $B \in 2k\pi,\ with\ k \in \mathbb{N}$ (see Sec. 3.2.3) has been made. Forcing B close enough to $2k\pi$ would imply to tweak the *ESS* parameters as it has been done by [Novak et al., 2015]. In this research, we rather propose to directly take B into account using Eqs. (3.6) and (3.21). This can be done by multiplying term by term the $(\mathbf{T})^{-1}$ matrix with $Corr_B$: a phase corrective matrix of size $M \times M$ as explained in Appendix. B.4:

$$Corr_B(u,v) = \begin{pmatrix} e^{j0} & \dots & e^{j(v-1)B} \\ \vdots & \ddots & \vdots \\ e^{j0} & \dots & e^{j(v-1)B} \end{pmatrix}. \tag{3.30}$$

Indeed, after application of the *CorrB* matrix in Eq. 3.28, all the phases of the Hammerstein kernels are equal to zero (see Fig.3.7), which is a sufficient condition to avoid phase reconstruction problems [Vanderkooy and Thomson, 2016].

Fig. 3.8 shows the comparison between y_1 and y_2 when the matrix $Corr_B$ is not applied (with $\mod(B, 2\pi) = 0.7$). The emulated signal (y_2) is very sensitive to the B factor. As it can be seen, it no longer corresponds to the original power series signal (y_1).

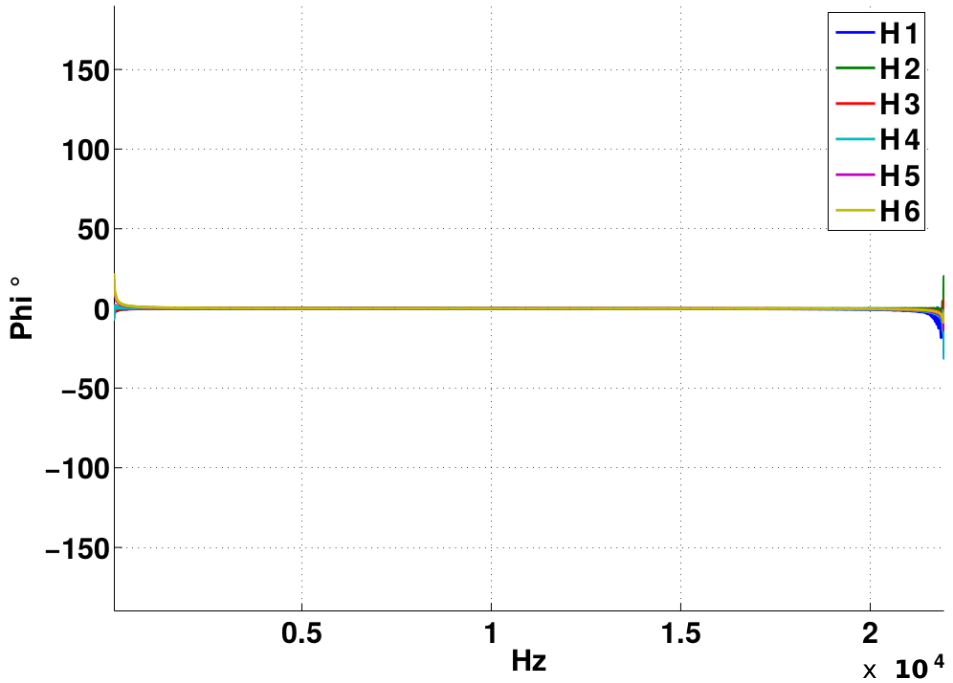


Figure 3.7 Phase of the Hammerstein kernels $H_m(\omega)$ computed for a power series after application of the matrix $Corr_B$ ($f_s = 44\,100$ Hz).

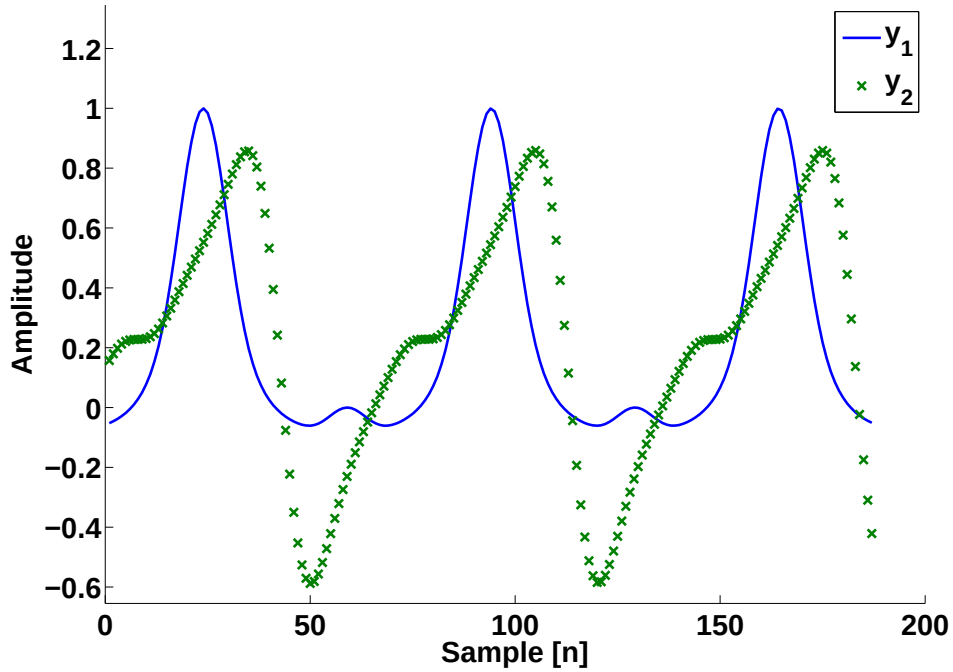


Figure 3.8 Comparison between $y_1[n]$ and $y_2[n]$ when the B correction is not applied. Focus on a part of the ESS signal where the frequency is close to f=1000 Hz.

3.3.4 Phase mismatch when extracting the impulses $g_m[n]$

The harmonic impulse responses ($g_m[n]$) appear in the deconvolved signal $z[n]$ at specific positions on the time axis. They can be extracted, around the position $n_0 - \Delta_m$. However, no matter how the *ESS* parameters are adjusted, it is impossible to satisfy the condition $\Delta_m \in \mathbb{N},\ \forall m \in [1,M]$ (except for the trivial case $M = 1$). Actually, the $g_m[n]$ impulses are extracted around the position $n_0 - Ceil(\Delta_m)$. This leads to a position error $\varepsilon_m = Ceil(\Delta_m) - \Delta_m$ specific to each order m, as shown in Fig. 3.9. In the frequency domain, this leads to a phase mismatch between the kernels $G_m(\omega)$. Consequently, the computation of the Hammerstein kernels are impacted as they depend on a weighted sum of the kernels $G_m(\omega)$ (see Eq. (3.28)).

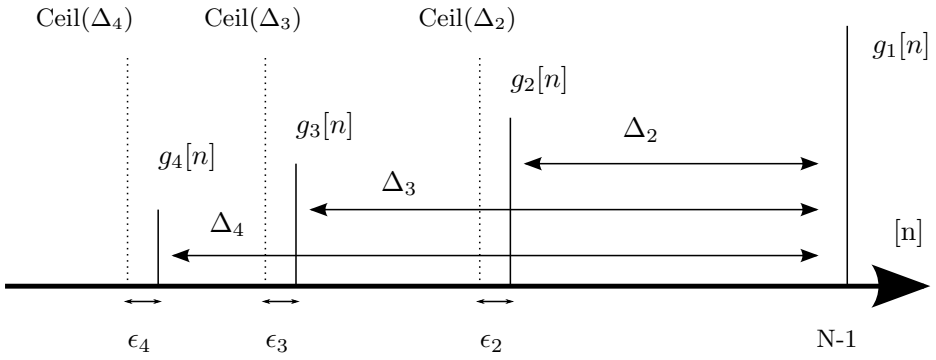


Figure 3.9 Deconvolution of an *ESS* passing through a power series, position error ε_m.

Choosing the sweep parameters in order to minimize the ε_m leads to a nonlinear optimization problem having no exact solutions. Moreover, it could be very restrictive for the ESS parameters when the order M increases. [Novak et al., 2015] propose to shift the signal $z[n]$ by $(\Delta_m - Ceil(\Delta_m))$ before extracting each $g_m[n]$ to solve this problem:

$$Z_m(\omega) = Z(\omega)e^{j\frac{\omega}{f_s}(Ceil(\Delta_m)-\Delta_m)}. \tag{3.31}$$

This phase mismatch mostly introduces some reconstruction errors at high frequencies. Fig. 3.10 shows the comparison between the target y_1 and the prediction y_2 if the *phase mismatch correction* is not applied. There is a significant difference between the two signals, mostly starting at 3000 Hz and increasing with frequency.

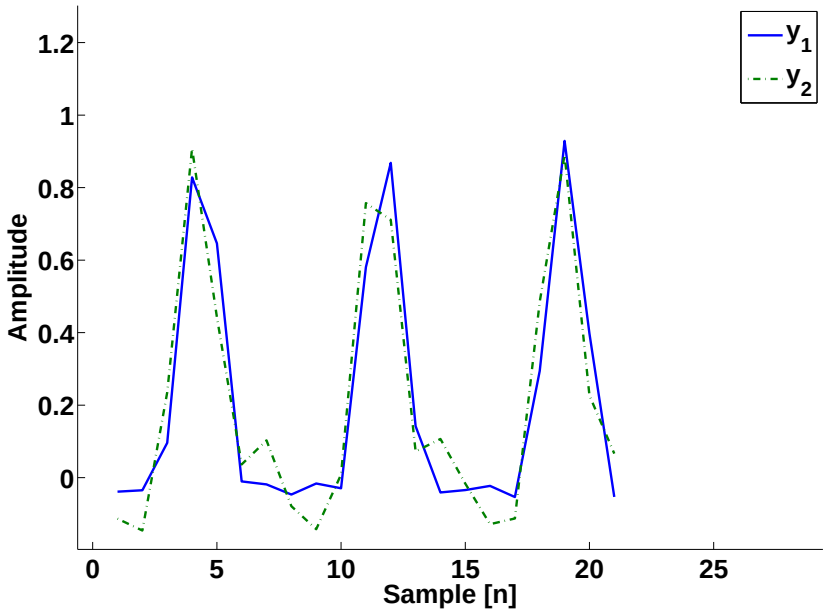


Figure 3.10 Comparison between the target $y_1[n]$ and the prediction $y_2[n]$ for input frequencies close to 6000 Hz when the phase correction from Eq. (3.31) is not applied ($y_2[n]$ perfectly matches $y_1[n]$ if the correction is applied).

3.3.5 *ESS* fade in/out

In order to avoid spectral leakage due to abrupt termination of the *ESS*, an appropriate fade-in and fade-out procedure has been applied. Similarly to [Novak et al., 2015] a raise cosine window has been chosen. If the loss of bandwith is expressed as a fraction of f_1 and f_2, for the fadeIn and fadeOut respectively, then the length of the window to apply is given by:

$$\begin{aligned} \text{length}_{\text{fadeIn}} &= R \cdot log(1+loss), \\ \text{length}_{\text{fadeOut}} &= R \cdot log(f_2 \cdot \frac{1-loss}{f_1}) + 1 - N. \end{aligned} \tag{3.32}$$

3.3.6 Spreading of the Dirac impulse

We have assumed in Eq. (3.8) that the deconvolution of $ss(m.\phi[n])$ is a delayed Dirac impulse $\delta[n-(n_0-\Delta_m)]$. Indeed, as the signal is band-limited, the Dirac impulse is a Sync function δ'_m. It follows that each measured kernel $g_m[n]$ is in fact:

$$g'_m[n] = g_m[n] * \delta'_m. \tag{3.33}$$

Moreover, as the signal $ss(m.\phi[n])$ starts at frequency $m.f_1$ instead of f_1, δ'_m is more and more band limited as the order m increases. These limitations spread the kernels $g'_m[n]$ around the positions $n_0 - \Delta_m$.

Extracting the kernels $g_m[n]$ from $z[n]$, by starting at the exact positions Δ_m can therefore be problematic. We prefer cutting the kernels $g_m[n]$ a few samples before that position to have a better reconstruction of the Hammerstein kernels. In practice, 1000 samples have been chosen and there is no need to take a larger number of samples (at $f_s = 44\,100$ Hz). For example, Fig. 3.12 shows the comparison between the emulated signal using kernels $g_m[n]$ cut at $n_0 - \Delta_m - 1000$ (signal y'_1) and the emulated signal using kernels $g_m[n]$ cut at $n_0 - \Delta_m$ (signal y_2). The signal y'_1 is better aligned to the signal y_1 of the Fig. 3.6 than y_2. Table 3.1 shows the mean absolute error between y'_1 and y_2 for frequencies close to f=1000 Hz in the ESS for different cutting offsets.

Table 3.1 Mean absolute error for frequencies close to 1000 Hz between y'_1 and y_2 for different offsets.

cutting offset	-1000	-500	-100	-10	0
Mean error	0.005	0.007	0.014	0.062	0.155

Note: bandwidth = [5,22000] Hz with $f_s = 44\,100$ Hz

For real-time emulations, introducing such a delay in the impulse responses could affect the output signal in the same way since these impulse responses will be convolved with the signal to emulate. This signal will therefore be delayed by the same offset. To avoid this delay, we propose to cut the kernels $g_m[n]$ at $n_0 - \Delta_m - 1000$, to compute Hammerstein kernels and then delete the first thousand samples of the Hammerstein kernels. Fig. 3.13 shows that this method gives excellent results.

In the case where the bandwidth of the ESS is short (measurement of a sub-woofer for example), the Dirac impulses δ'_m become larger and larger (as presented in Fig.3.11). Only cutting the first thousand samples after the computation of the Hammerstein kernels could lead to a wrong representation. In this case, a compromise has to be found between the accuracy of the Hammerstein kernels and their possible use for real-time emulations. In the Toolbox, described in Sec. 3.6, the choice of the offset before the beginning of the Hammerstein kernels is therefore left open to the user, depending on the bandwidth and the real-time constraint of the user's application.

One can notice that the sampling rate used here is always 44 100 Hz as we try to ensure compatibility with basic sound cards. Nevertheless, the use of higher sampling rates allows a

better compactness of the impulse responses as the harmonic m extends its bandwidth to the frequency $\min(m.f_2, f_s/2)$.

The offset of 1000 samples as been chosen according to the results of Table 3.1. However, since the number of samples between two harmonic impulses evolves logarithmically, they are increasingly closer when the order M increases (see Eq. 3.7). In order to avoid interferences between the impulse $g_{m+1}[n]$ and the impulse $g_m[n]$, the following condition is imposed:

$$\Delta_M - \Delta_{M-1} > 2 \cdot \text{offset}. \tag{3.34}$$

In practice, this case happens only when the following conditions are combined:

- The length of the sweep is short.
- The bandwidth is short.
- Numerous Hammerstein kernels are requested.

For example, when the parameters of the ESS are $N = 20 \cdot 44100$, $f_1 = 20\,\text{Hz}$, $f_2 = 20\,000\,\text{Hz}$, the condition (3.34) is satisfied until the order reaches M=64 (with an offset=1000). However for an ESS whose parameters are $N = 1 \cdot 44100$, $f_1 = 20\,\text{Hz}$, $f_2 = 100\,\text{Hz}$ the order should be limited to M = 14 (or the offset has to be decreased).

3.3.7 Computing the powers of the input signal

Eq. (3.3) shows that implementing the Hammerstein model requires the powers of the input signal. Some precautions must be taken to avoid aliasing during the emulation of the nonlinear Device Under Test (DUT). In order to raise the input signal $x[n]$ to the power m, the following procedure is used:

- Resampling the input signal at $m.f_s$ by zeros padding (where f_s is the original sampling rate).
- Filtering the signal in the initial Nyquist band $[0, \frac{f_s}{2}]$.
- Raising the obtained signal to the power m (at $m.f_s$ sampling frequency).
- Filtering the result in the Nyquist band $[0, \frac{f_s}{2}]$.
- Downsampling the filtered signal at the f_s sampling frequency.

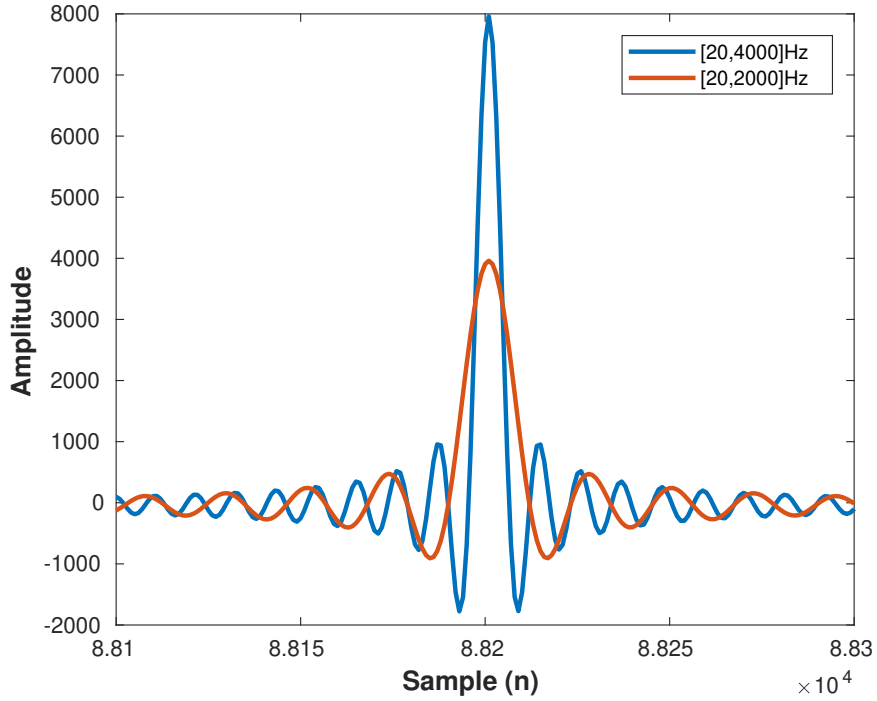


Figure 3.11 Comparison in time domain of the impulse response obtained when deconvolving an ESS signal having a bandwidth comprised between [20,2000]Hz or between [20,4000]Hz.

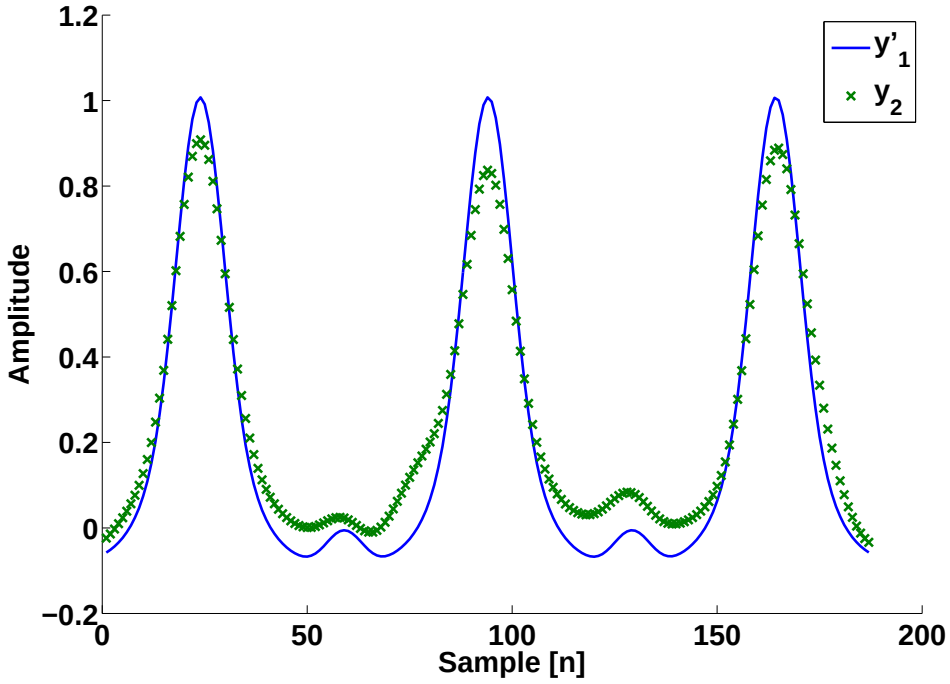


Figure 3.12 Comparison between emulated signals cutting $g_m[n]$ at $n_0 - \Delta_m - 1000$ (y_1') and at $n_0 - \Delta_m$ (y_2).

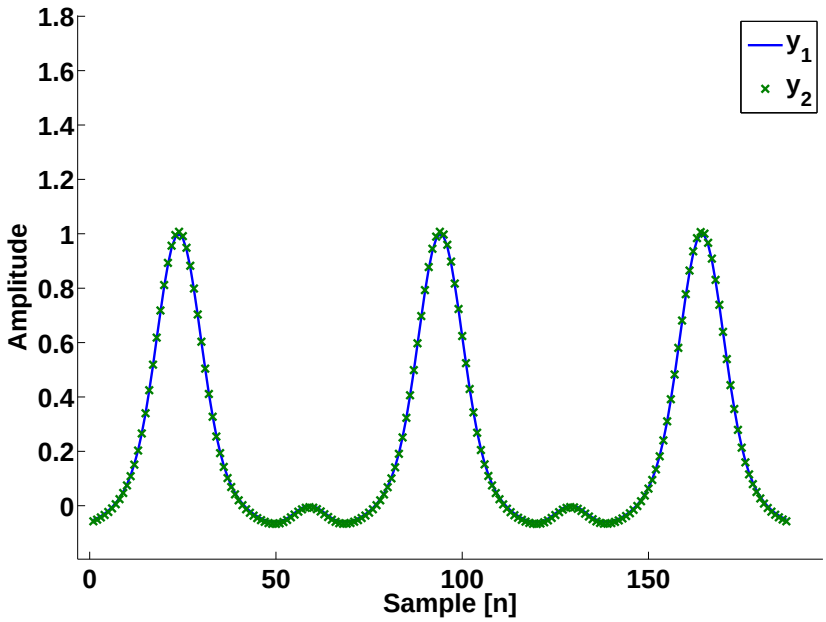


Figure 3.13 Comparison between the output power series y_1 and the emulated signal y_2 computed by cutting $g_m[n]$ at $n_0 - \Delta_m - 1000$ and deleting the first thousand samples of $h_m[n]$.

3.4 Emulation of nonlinear audio devices using HKISS method

In the precedent section, a set of precautions, which are necessary for a correct computation of the Hammerstein kernels has been presented. Taking these precautions leads to the *Hammerstein Kernels Identification by Sine Sweep (HKISS)* method. This section presents several tests made on real nonlinear audio devices in order to prove the efficiency of this method.

3.4.1 Evaluation algorithm

The following method (illustrated in Fig. 3.14) is proposed to evaluate the HKISS method over different nonlinear audio devices:

- An ESS signal is sent to the audio Device Under Test (DUT).
- The output of the system is recorded and called the *target* y_1.
- From y_1, the Hammerstein kernels can be computed using Eq. (3.28).
- The DUT is emulated through Eq. (3.3) using the ESS as input signal and the Hammerstein kernels previously computed. The resulting signal is called the prediction y_2.

The comparison between the target and the prediction can be made:

1. In the time domain.
2. In the frequency domain to compare the harmonic content of the target and the prediction. The root mean square error between the ten first harmonics of the target and prediction signals is used. This measure, called the harmonic content accuracy, is defined in Eq. (2.63).

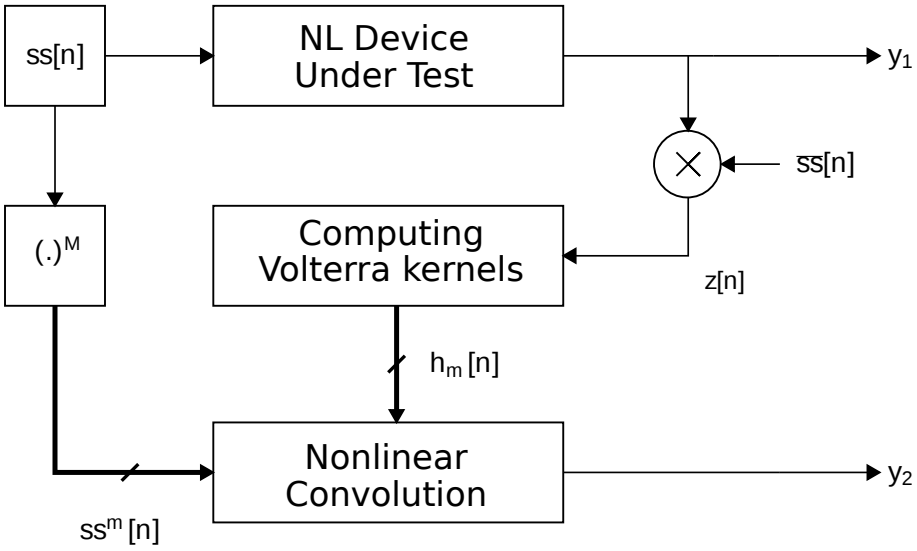


Figure 3.14 Comparison between the signal coming from the nonlinear device under test y_1 and its emulation y_2.

3.4.2 Emulation of a guitar distortion effect (tube screamer) through the HKISS method

The first nonlinear audio device under test is the distortion effect of the Amplifier `TSA15h` [Ibanez, 2015]. This effect is the copy of the famous Tubescreamer distortion effect [Ibanez, 1978]. The parameter *Gain* of this effect is set at its maximum. Figs 3.15, 3.16 and 3.17 compare the target y_1 (i.e., the signal coming from the distortion device) with the prediction y_2 (i.e., the signal computed through the *HKISS* method by using 10 Hammerstein kernels) for the frequencies close to 500, 1000 and 5000 Hz respectively. As it can be seen, the curves are very close.

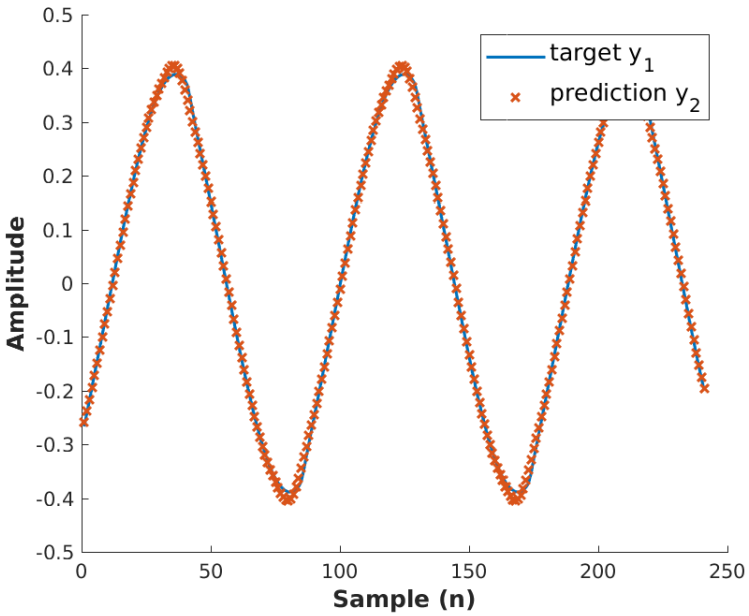


Figure 3.15 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 500Hz. Emulation of the distortion effect of the `TSA15h` amplifier.

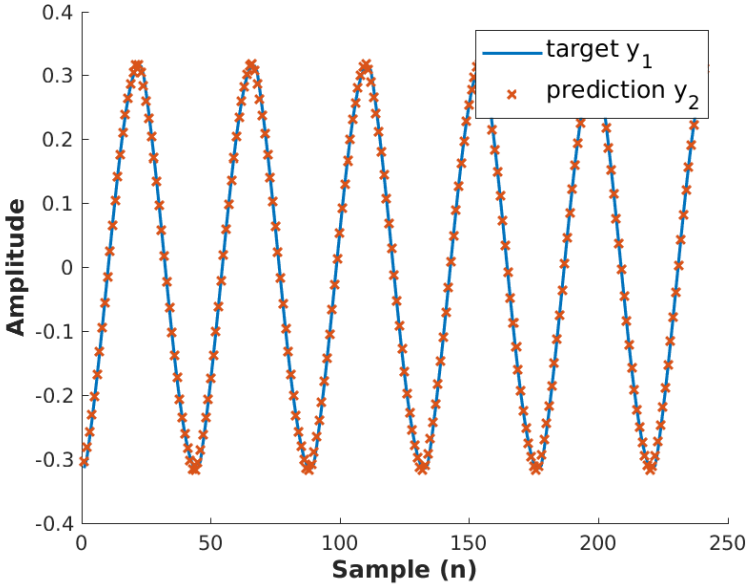


Figure 3.16 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 1000 Hz. Emulation of the distortion effect of the `TSA15h` amplifier.

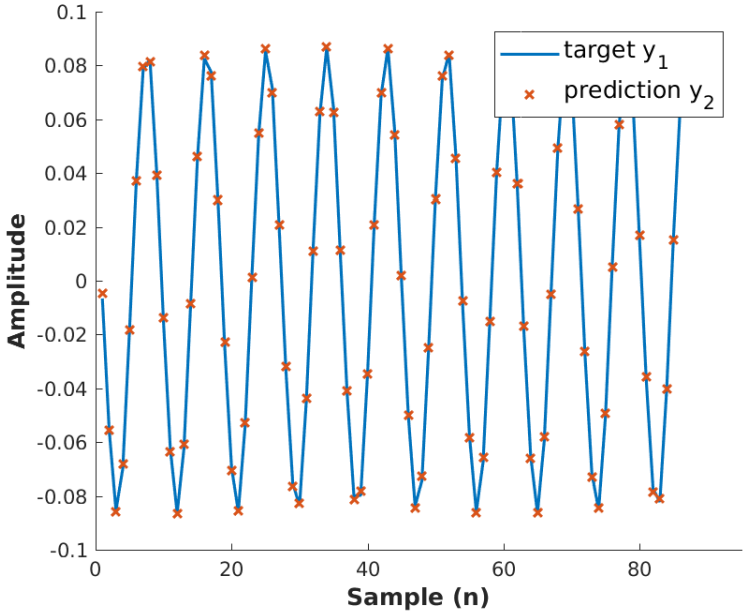


Figure 3.17 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 5000Hz. Emulation of the distortion effect of the `TSA15h` amplifier.

Fig. 3.18 presents the power spectrum of the target and the prediction are compared when the frequency of the input ESS signal is close to $f = 1000\,\text{Hz}$. The harmonic content accuracy is equal to 0.09dB. This proves that the harmonic content is accurately represented. However when a guitar signal is used, its amplitude is not constant and does not correspond to the amplitude of the ESS anymore. Consequently, the prediction is less accurate. This is shown in Fig. 3.19 where the power spectrum of the target and the prediction when the input is a guitar note whose fundamental is close to the frequency f= 400Hz.

3.4.3 Emulation of the tube amplifier `TSA15h` through the HKISS method

One of the most popular application in the field of nonlinear audio modeling is the emulation of tube amplifiers, so dear to guitarists. The emulation of the tube amplifier `TSA15h` with the *Gain* parameter at its maximum position is presented in Figs. 3.20 , 3.21 & 3.22, using an ESS as input signal, with a focus on the frequencies close to 200, 1000 and 5000Hz respectively. The prediction y_2 is very close to the measured signal y_1, in each case.

The power spectrum comparison of the target and the prediction when the frequency of the input ESS signal is close to $f = 1000\,\text{Hz}$ is given in Fig. 3.23. The harmonic content accuracy

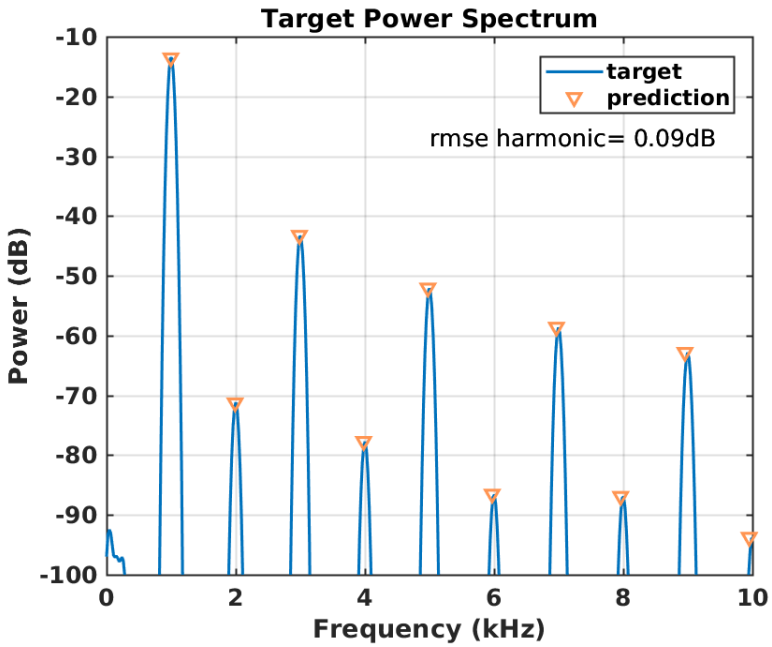


Figure 3.18 Comparison in the frequency domain, between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 1000 Hz. Emulation of the distortion effect of the `TSA15h` amplifier (i.e., the *Tube-screamer* circuit).

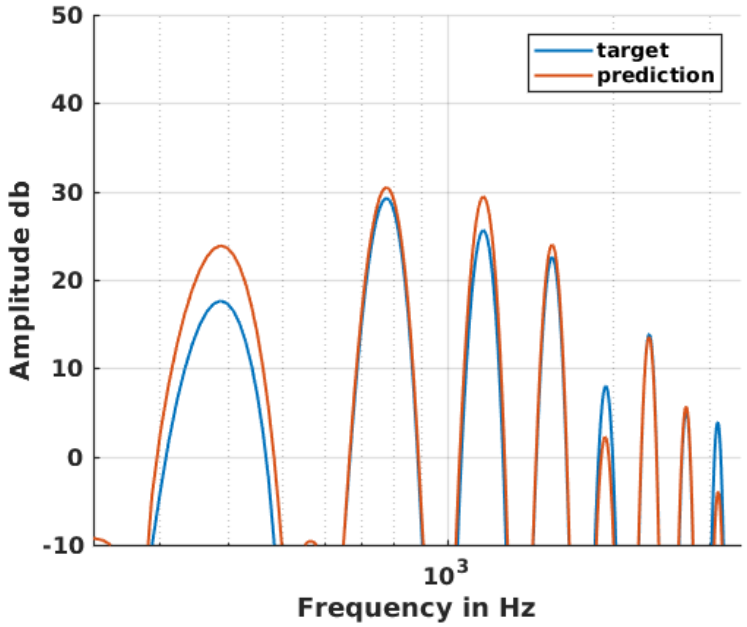


Figure 3.19 Comparison in the frequency domain, between the target y_1 and the prediction y_2 when the input is a guitar note whose fundamental is close to the frequency f=400Hz. Emulation of the distortion effect of the `TSA15h` amplifier.

is equal to 0.21dB. This proves that the nonlinearities specific to the vacuum tube are also well emulated by the HKISS method.

For this amplifier, the emulation of a guitar note is worse than for the emulation of the distortion part only (previously presented in Fig. 3.19). This is shown in Fig. 3.24 where the power spectrum of the target and the prediction when the input is a guitar note close to the frequency f= 400Hz. One can notice that the THD of this amplifier is higher than the THD of the Tube-screamer only. This could explain why the emulation is worse than the one for the Tube-screamer distortion circuit.

Finally, a time domain comparison between the target and the prediction is given for different orders (i.e., using different numbers of Hammerstein kernels) in Fig. 3.25. A good agreement can already be obtained with only few Hammerstein kernels (about five or six) which is interesting for real-time implementations.

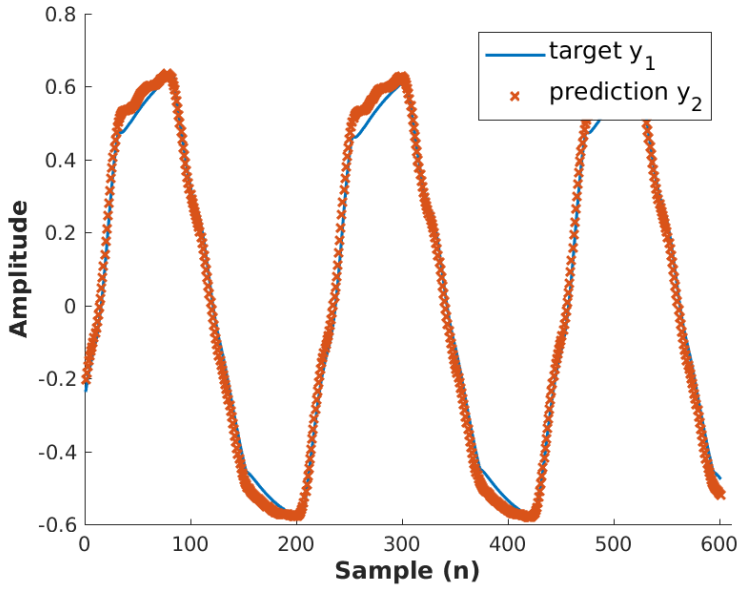


Figure 3.20 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 200Hz. Emulation of the `TSA15h` tube amplifier.

3.4.4 Emulation of the tube amplifier: *Engl retro tube 50* through the HKISS method

Another tube amplifier has been tested, the `Engl retro tube 50` [Engl, 2015]. This amplifier is more nonlinear than the `TSA15h`. The emulation of the `Engl retro tube 50` amplifier with

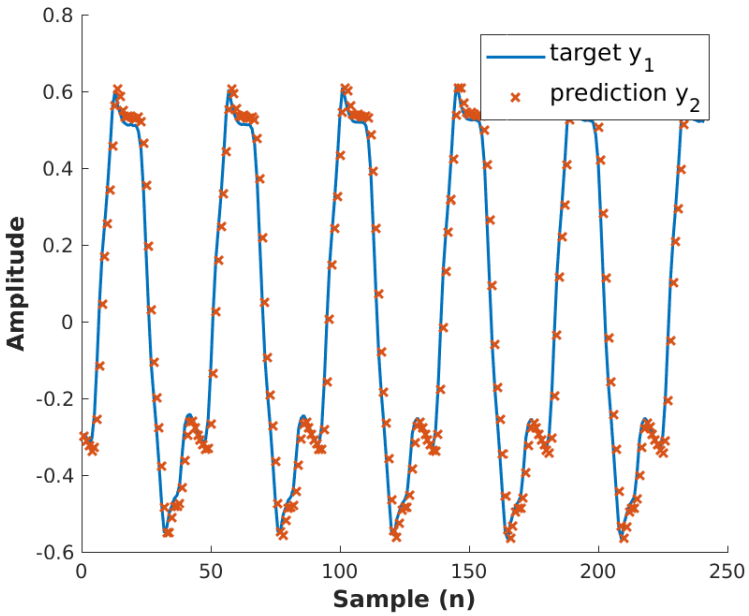


Figure 3.21 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 1000 Hz. Emulation of the `TSA15h` tube amplifier.

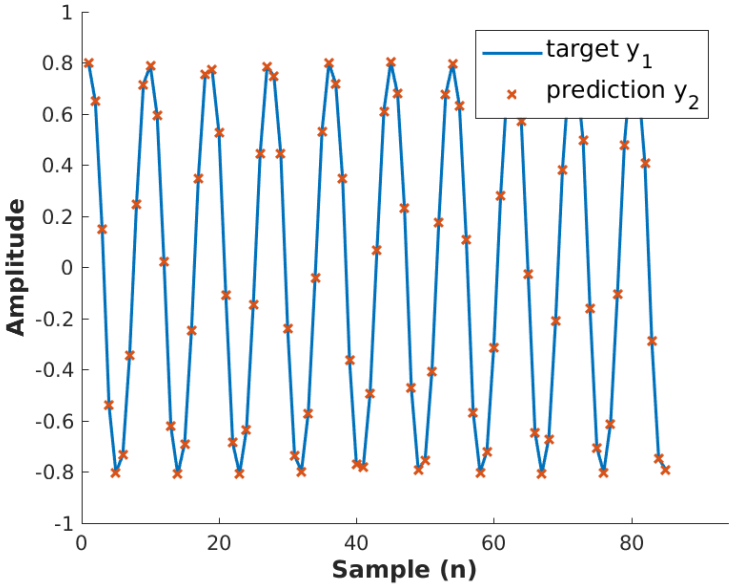


Figure 3.22 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 5000Hz. Emulation of the `TSA15h` tube amplifier.

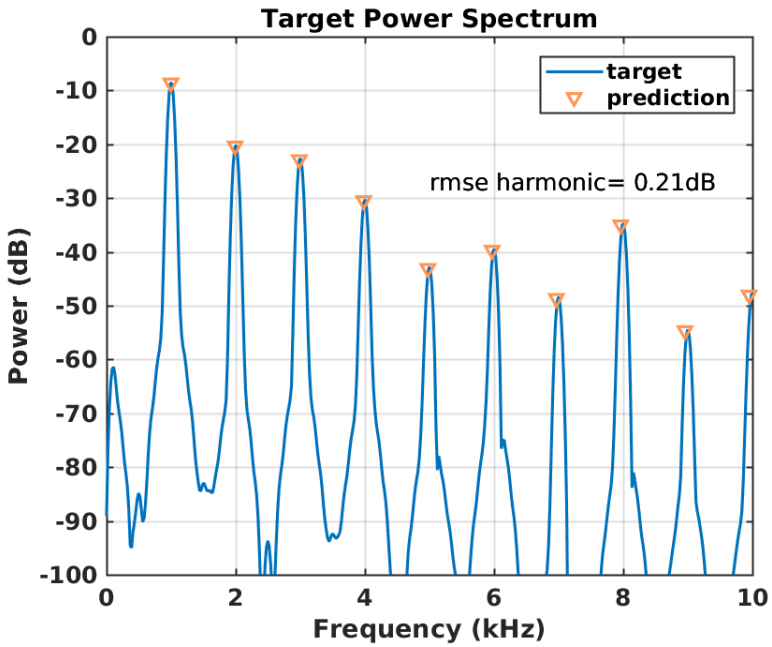


Figure 3.23 Comparison in the frequency domain, between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 1000 Hz. Emulation of the `TSA15h` tube amplifier.

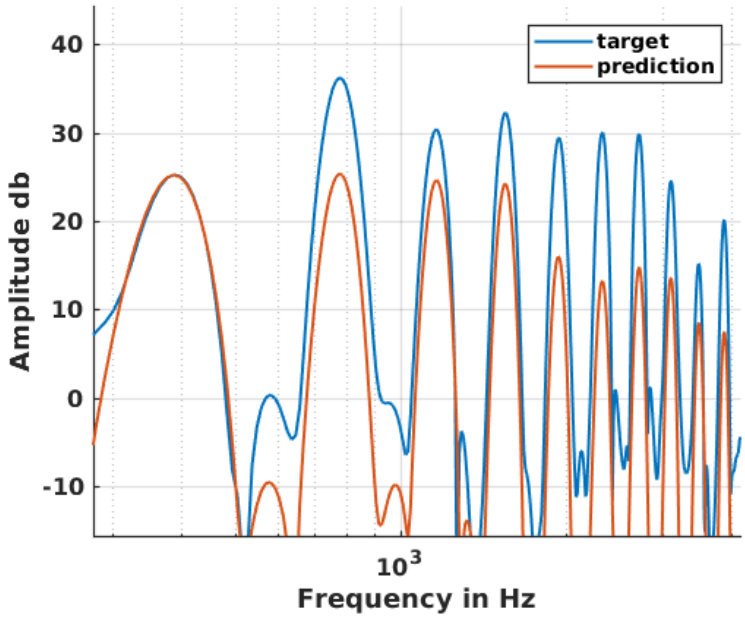


Figure 3.24 Comparison in the frequency domain, between the target y_1 and the prediction y_2 when the input is a guitar note close to the frequency f=400Hz. Emulation of the `TSA15h` amplifier.

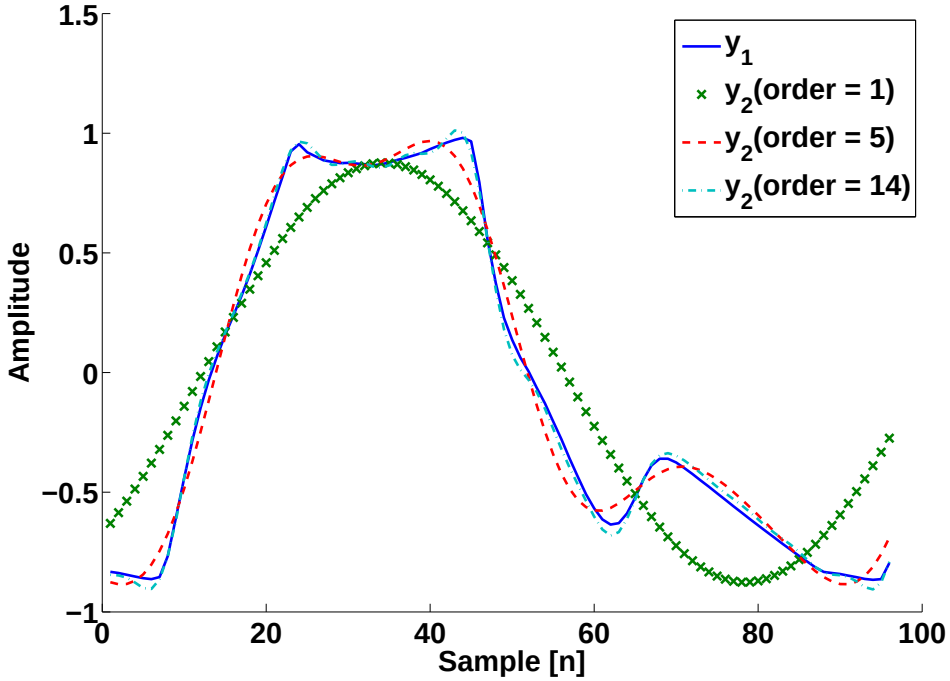


Figure 3.25 Emulation of the tube amplifier `TSA15h` when the input signal is a sine wave a the frequency f=500Hz using different orders of simulation. Comparison in the time domain between the target y_1 and the prediction y_2.

the *Gain* parameter at its maximum position is presented in Figs. 3.26, 3.27 & 3.28 using an ESS as input signal with a focus on frequencies close to 200, 1000 and 5000Hz respectively. The prediction y_2 is very close to the measured signal y_1, in each case.

The power spectrum comparison of the target and the prediction when the frequency of the input ESS signal is close to $f = 1000\,\text{Hz}$ is given in Fig. 3.29. The harmonic content accuracy is equal to 0.04dB.

3.5 Discussion on the limitations of the method

The *HKISS* method is pretty robust. For example, applying again the test algorithm of Sec. 3.3 (power series model) up to the order 20 causes no problem, as presented in Fig. 3.30. The model is also accurate for high frequencies (see Fig. 3.31). The remaining limitations are discussed in this section.

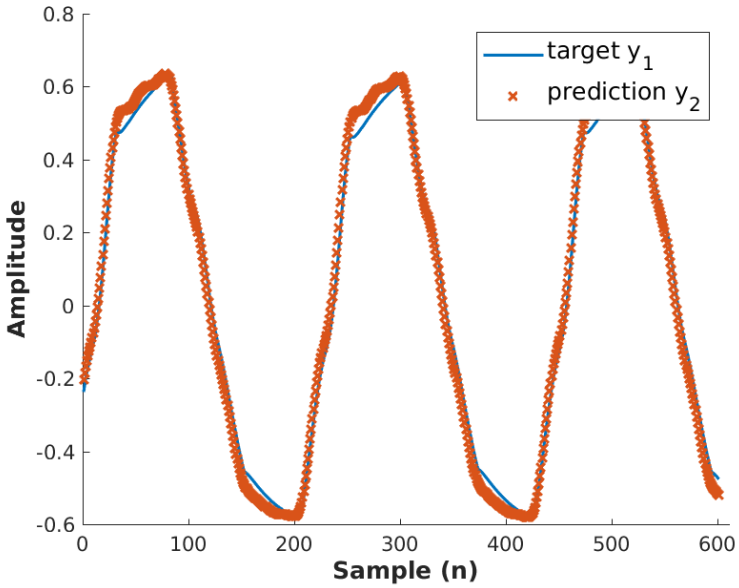


Figure 3.26 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 200Hz. Emulation of the `Engl` tube amplifier.

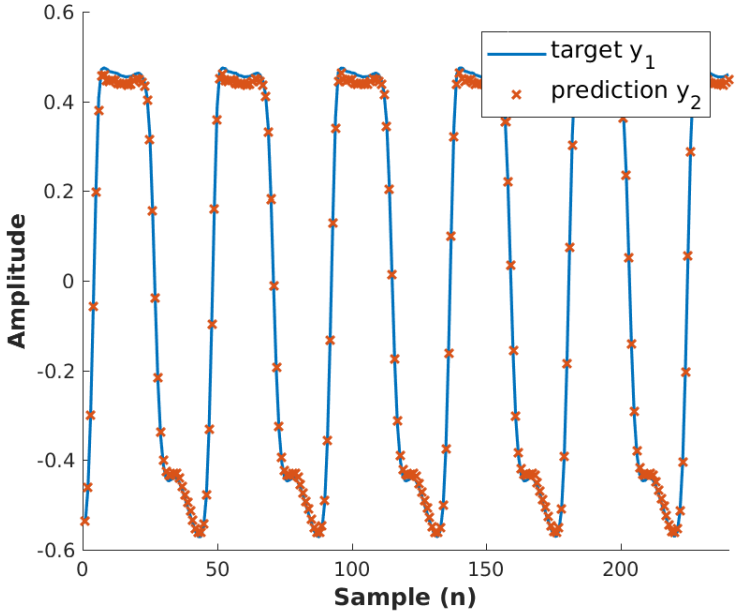


Figure 3.27 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 1000 Hz. Emulation of the `Engl` tube amplifier.

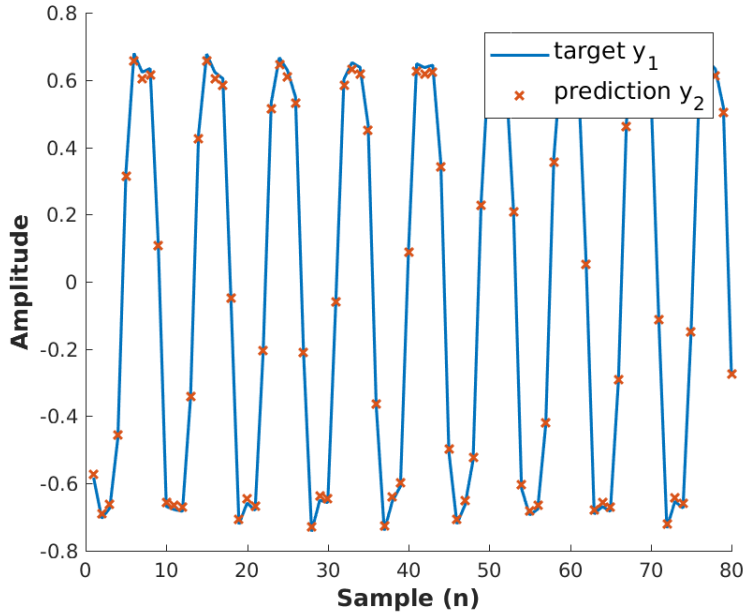


Figure 3.28 Comparison in the time domain between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 5000Hz. Emulation of the `Engl` tube amplifier.

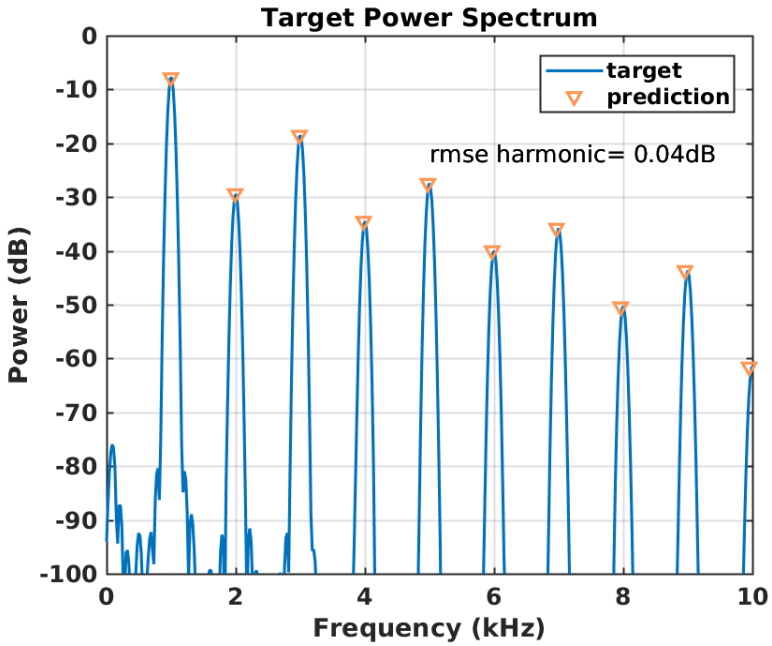


Figure 3.29 Comparison in the frequency domain, between the target y_1 and the prediction y_2 when the input is an ESS. Focus on frequencies close to 1000 Hz. Emulation of the `Engl` tube amplifier.

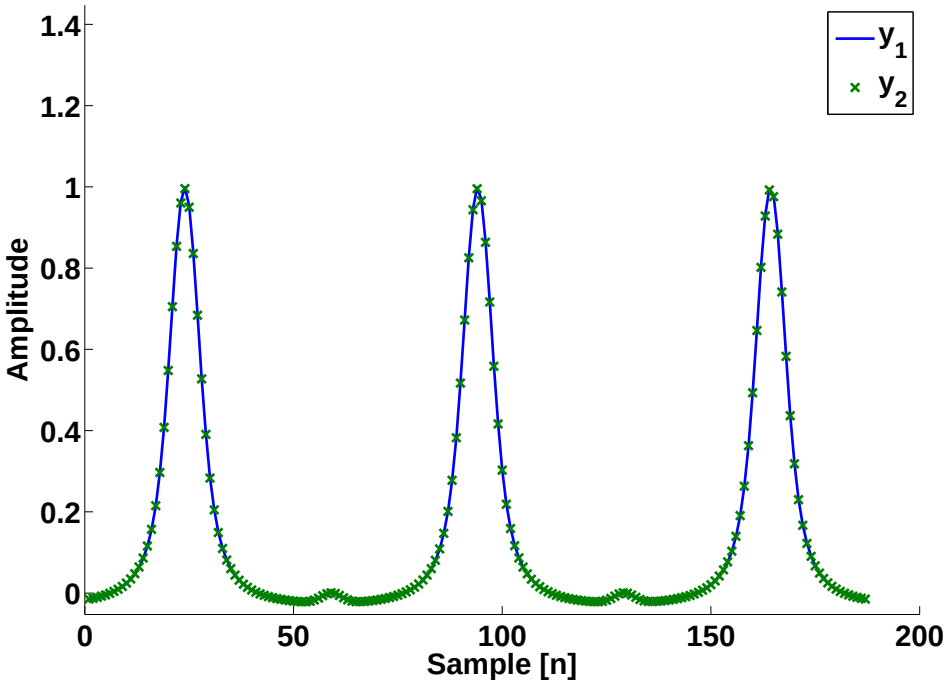


Figure 3.30 Comparison between the target y_1 and the prediction y_2 for a power series up to the 20^{th} order when the input is an ESS. Focus on frequencies close to 1000 Hz.

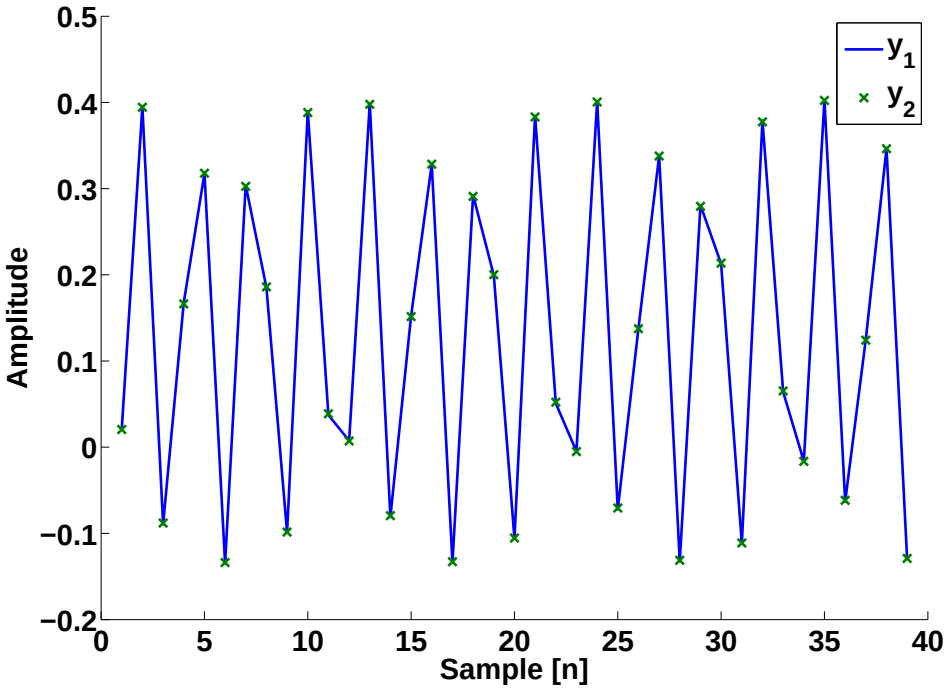


Figure 3.31 Comparison between the target y_1 and the prediction y_2 for a power series up to the 20^{th} order when the input is an ESS. Focus on frequencies close to 16000Hz.

3.5.1 Order limitation

The amplitude of the kernels $g_m[n]$ decreases when the order m increases. It could be a problem for noisy nonlinear systems when the kernels reach the background noise creating a maximum order limit. Improving the signal to noise ratio (*SNR*) is thus important if high order emulation is required. It could be convenient to take several measurements of a device and averaging the results in order to improve the *SNR*. However, we must be sure that the system is perfectly time invariant. For example, the measurement of a loudspeaker at a high power could be difficult as the heating of the coil changes its characteristics.

3.5.2 Low-frequency limitation

As the *ESS* is band limited in the frequency interval $[\omega_1, \omega_2]$, its m^{th} harmonic is band limited in $[m.\omega_1, m.\omega_2]$ (the upper bound could be the Nyquist frequency). It means that the deconvolution by the inverse filter is also band limited in the frequency interval $[m.\omega_1, m.\omega_2]$ which gives a band limited Dirac impulse δ'_m.
Eq. (3.8) can then be rewritten as:

$$\sin(m.\phi[n]) * \overline{ss}[n] = D.\delta'_m[n-(n_0-\Delta_m)], \tag{3.35}$$

where D is an amplitude constant. In the frequency domain, the impulse δ'_m delayed by Δ_m is a step function $\mathbb{1}(\omega)$ on the domain $[m.\omega_1, m\omega_2]$ (see Fig. 3.32). Therefore (considering that the impulse is normalized such that D=1) :

$$SS_m(\omega).\overline{SS}(\omega) = \mathbb{1}_{[m.w_1, m.w_2]}(\omega). \tag{3.36}$$

In order to understand the impact on the Hammerstein kernels computations, let's go back over Eq. (3.15) again and convolve its output signal $y(t)$ with the *ESS* inverse filter using Eq. (3.36):

$$\begin{cases} G_1(\omega) = \mathbb{1}_{[w_1,w_2]}.H_1(\omega) + \dfrac{3A^2}{4}H_3(\omega).\mathbb{1}_{[w_1,w_2]}, \\ G_2(\omega) = \dfrac{-A}{2}j.e^{-jB}.H_2(\omega).\mathbb{1}_{[2.w_1,2.w_2]}, \\ G_3(\omega) = \dfrac{-A^2}{4}e^{-j2B}.H_3(\omega).\mathbb{1}_{[3.w_1,3.w_2]}. \end{cases} \tag{3.37}$$

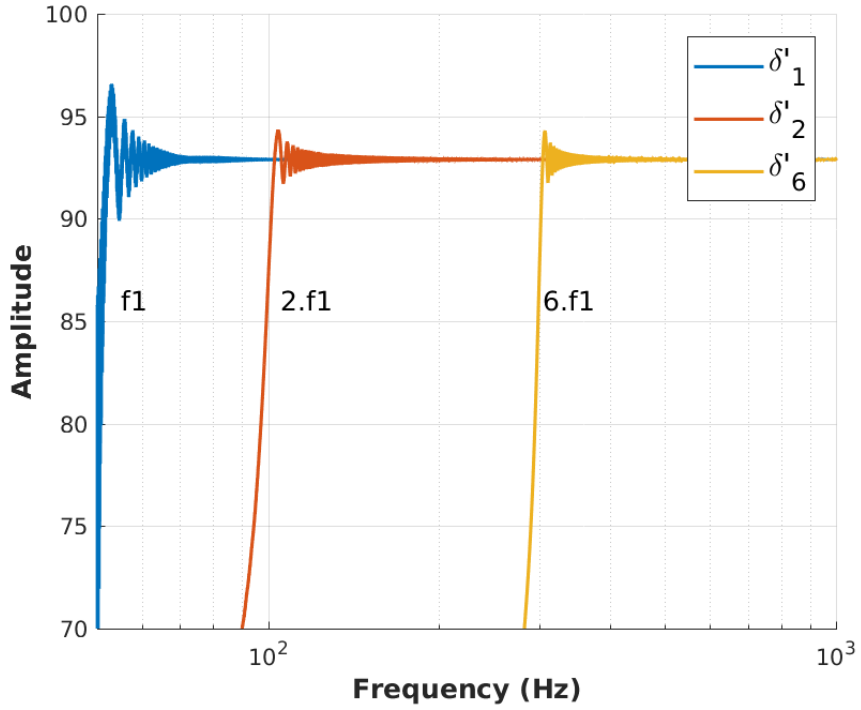


Figure 3.32 FFT of δ'_m with $m \in \{1,2,6\}$.

By inverting the relation, the Hammerstein kernels can be obtained such that:

$$\begin{cases} H_3(\omega).\mathbb{1}_{[3.w_1,3.w_2]} = \dfrac{-4}{A^2} e^{j2B}.G_3(\omega), \\ H_2(\omega).\mathbb{1}_{[2.w_1,2.w_2]} = \dfrac{2}{A} j.e^{jB}.G_2(\omega), \\ H_1(\omega).\mathbb{1}_{[w_1,w_2]} = G_1(\omega) - \dfrac{3A^2}{4} H_3(\omega).\mathbb{1}_{[w_1,w_2]}. \end{cases} \tag{3.38}$$

This last expression indicates that $H_3(\omega).\mathbb{1}_{[w_1,w_2]}$ must be known in order to compute $H_1(\omega).\mathbb{1}_{[w_1,w_2]}$, but only $H_3(\omega).\mathbb{1}_{[3.w_1,3.w_2]}$ is known from the first expression in Eq. (3.38). Consequently, for a power series up to the third order, the Hammerstein kernel $H_1(\omega)$ is well reconstructed only in the frequency range $[3\omega_1, 3\omega_2]$.

In the general case, the Hammerstein kernels are correctly reconstructed for the frequencies ω_{valid} which correspond (for a nonlinear system of order M) to:

$$\omega_{valid} \geq M \cdot \omega_1. \tag{3.39}$$

The Fig. 3.33 illustrates this problem by presenting the comparison in the frequency domain between the first Hammerstein kernel of a power series up to the 6^{th} order (where $H_m(\omega) = \mathbb{1}_{[w_1,w_2]}$) and the first Hammerstein kernel reconstructed with the *HKISS* method. The reconstructed one does not correspond to the initial one until it reaches the frequency $f = 6 \cdot f_1$. A way to circumvent this problem is to choose the starting frequency of the ESS (ω_1) according to the desired valid frequency range. For example, if a 10 order nonlinear system has to be emulated on the frequency range [100, 20000]Hz, the starting frequency has to be chosen such that $f_1 = 10Hz$.

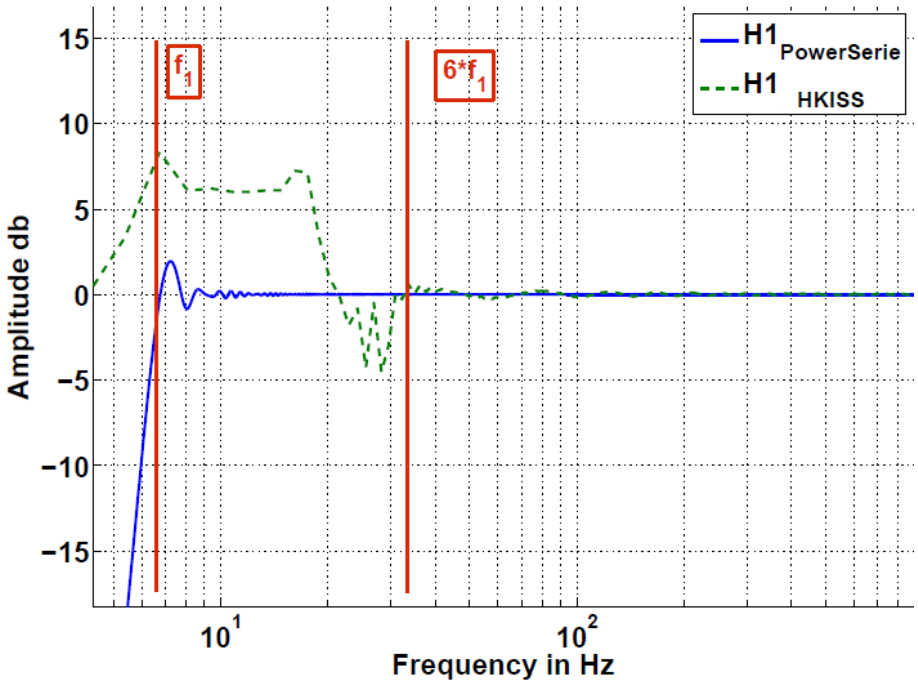


Figure 3.33 Comparison between the first Hammerstein kernel coming from a power series up to the order 6 and its reconstruction by the *HKISS* method.

3.5.3 Hammerstein kernels input level dependency

As discussed in Sec. 3.2.4, the accuracy of the emulated signal depends on the amplitude matching between the signal to emulate and the amplitude of the ESS used to measure and compute the Hammerstein kernels. During the emulation test of the amplifier `TSA15h`, we have noticed that the Hammerstein kernels $H_m^{\alpha}(\omega)$ have the same shape for a given order m but have different relative amplitudes (especially for higher-order kernels). For example, the comparison

of $H_1^1(\omega)$, $H_1^{0.5}(\omega)$, $H_5^1(\omega)$, $H_5^{0.5}(\omega)$ is presented in Fig. 3.34 for the Tube-screamer distortion circuit.

This suggests that the Hammerstein kernels $H_m^\alpha(\omega)$ could be computed from $H_m^1(\omega)$ by changing the amplitude factor A appearing in Eq. (3.20). The Fig. 3.35 shows an example of such adaptation: the kernels $H_m^{0.5}(\omega)$ are obtained from the kernels $H_m^1(\omega)$ by changing the A factor. With these adapted Hammerstein kernels $H_m^\alpha(\omega)$ based on the measured kernel at full amplitude $H_m^1(\omega)$, a signal of amplitude α can be emulated.

For example, a sine of amplitude 0.5 is emulated in Fig. 3.36 using the $H_m^1(\omega)$ corrected to give $H_m^{0.5}(\omega)$. In comparison, the same signal emulated using $H_m^1(\omega)$ was presented in Fig. 3.4. It can be seen that the corrected version is much better. In these examples, the parameter A has been manually adjusted in order to have the best correspondence between the kernels. This allows to avoid having a large set of impulse responses $H_m^\alpha(\omega)$ corresponding to each level α in the simulator (space memory consumption). To avoid some difficulties with variable kernels convolution , the Hammerstein model equation can be weighted as follows:

$$y[n] = \sum_{m=1}^{M} \frac{1}{[A(\alpha)]^{m-1}} h_m^1[n] * x^m[n]. \tag{3.40}$$

A way to determine the function $A(\alpha)$ is to find for each level α, the A factor minimizing the areas between all the $H_m^\alpha(\omega)$ and the $Corrected(A, H_m^1(\omega))$ ones. It is important to notice that some devices could have their Hammerstein kernels H_m^α too different from each other for different α values. Indeed, as it has been seen in Sec. 2.4.1, an ideal Hammerstein system has all its kernels proportional together. For these systems, the Hammerstein kernels can then be approximated for a continuous range of levels by means of an interpolation procedure as proposed by Tronchin and Coli in [Tronchin and Coli, 2015], but a truncated Volterra model should be also considered as in [Orcioni et al., 2018].

3.6 Implementation with the Matlab toolbox

The practical implementation of the Hammerstein identification technique is complex and can be subject to several sources of errors. Hence, a Matlab Toolbox (see Fig. 3.37) for the Hammerstein kernels identification is proposed. The toolbox is composed of three parts:

1. The generation of the *ESS* signal.

2. The computation of the Hammerstein kernels through the measured signal at the output of the DUT.

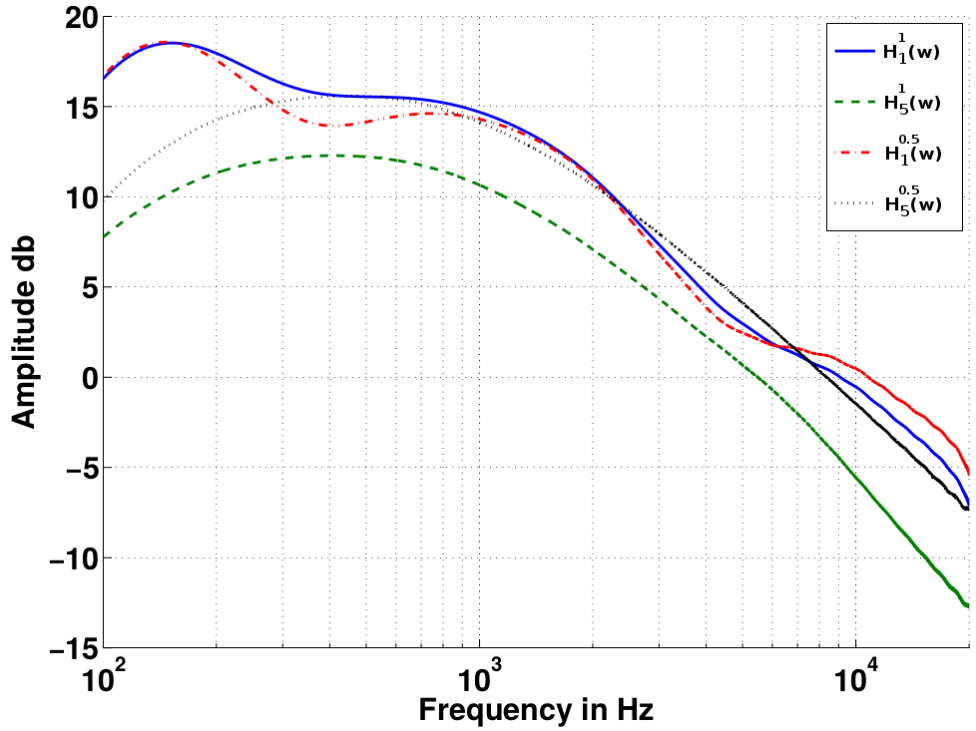


Figure 3.34 FFT of the first and fifth Hammerstein kernels of the Tube-screamer distortion circuit measured with an *ESS* with full and half amplitudes.

3. The emulation of the nonlinear DUT by convolving the powers of the input signal with the Hammerstein kernels.

The toolbox can be downloaded in [Schmitz, 2019a].

3.7 Example: emulation of a power series through the HKISS method

A short example using the toolbox to compute the Hammerstein kernels of a power series up to the order six is provided here.

3.7.1 Sine sweep generation

The tab *ESS generation* is the first step to emulate a nonlinear device. It allows the generation of the *ESS* with the following parameters (to be given by the user):

- f_1 and f_2 are the initial and final frequencies of the *ESS* respectively,

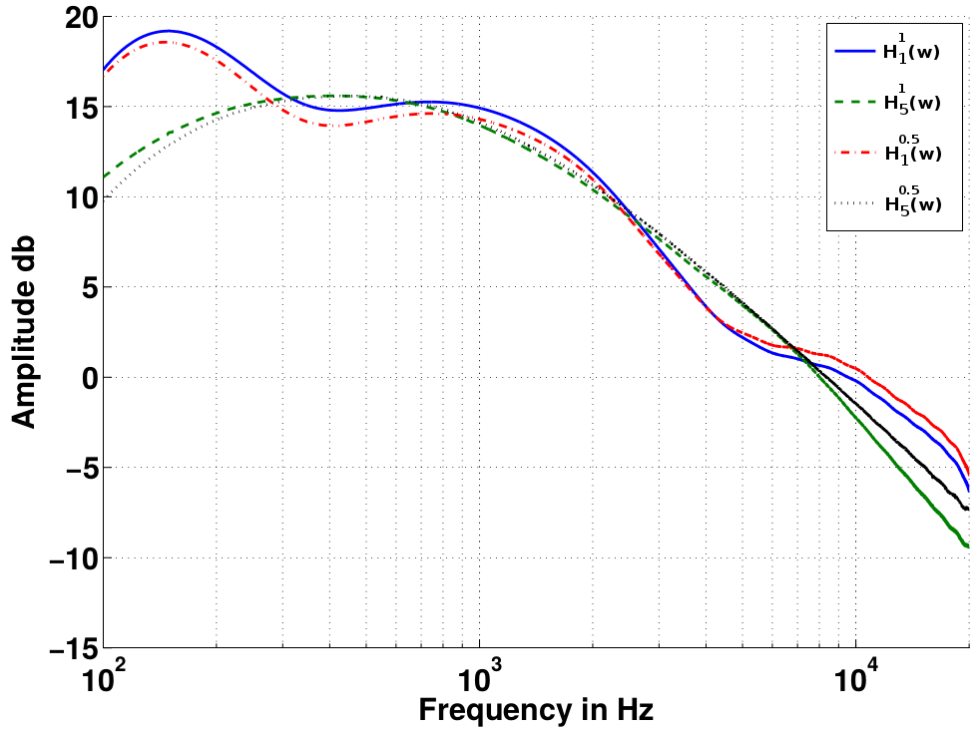


Figure 3.35 Corrected version of $H_m^1(\omega)$ in order to fit the $H_m^{0.5}(\omega)$ Hammerstein kernels for $m = [1,5]$. Tube-screamer distortion circuit of the amplifier `TSA15h`.

- N is the length of the *ESS* signal in samples,
- *fade in (or out)* is the percentage of the value of f_1 (or f_2) until a fade in (or out) is allowed. For example if $f_1 = 10\,\text{Hz}$ and *fade in = 0.1*, a *fade in* will be applied between the frequency $f = 10\,\text{Hz}$ and $f = 11\,\text{Hz}$. The fade out will be applied between the frequency f_2 and $f_2 - 0.1 \cdot f_2$. These values are displayed in the frame *Bandwidth* (see Fig. 3.37).

The *generate* button exports the *ESS* in a *.wav* format.

3.7.2 Hammerstein kernels calculation

Once the *ESS* signal has been sent to the *DUT*, its output signal y has to be loaded in the toolbox to be convolved with the inverse filter of the *ESS*. In this example, the output signal of a power series until the 6^{th} order named *yPowerSeries* is provided. It can be loaded in the toolbox by pushing the *Load y response* button (see Fig. 3.38). After having pressed the *Deconvolve* button, the z impulse is obtained as depicted at Fig.3.38. Before cutting z into the harmonic impulses g_m, the following parameters have to be specified :

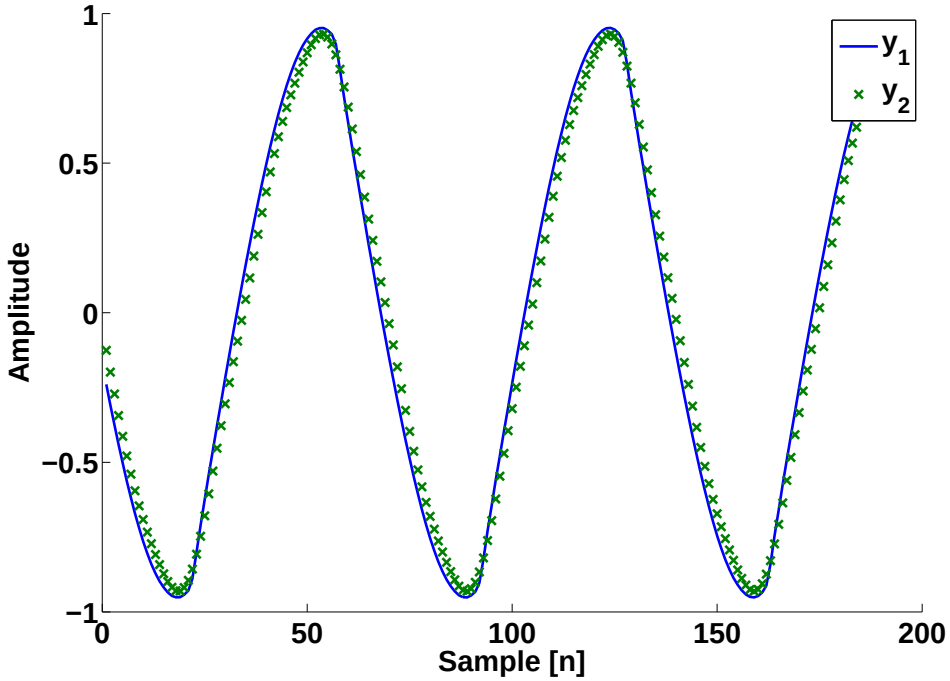


Figure 3.36 Comparison between an *ESS* of amplitude 0.5 passing through the distortion effect `TSA15h` and it simulation (y_2) by means of Hammerstein kernels measured for an *ESS* of amplitude 1 but corrected with the *A* factor.

- *Nb Kernels* is the number of Hammerstein kernels requested for the emulation of the DUT.
- *Length Kernels* is the desired length for the Hammerstein kernels.
- The *Delay cut* is the number of samples kept before the theoretical position of the Hammerstein kernels. This number has to be comprised in the interval [-1000 0]. Note: the user should verify that the condition from Eq. (3.34) is respected.

3.7.3 Emulation of the DUT

The third tab (see Fig. 3.39) enables the emulation of the *DUT* by convolving the Hammerstein kernels with any chosen signal in a *.wav* format. For this example, we can load the *sweep.wav* signal and convolve it with the Hammerstein kernels. This prediction signal can then be compared to the target *yPowerSeries*.

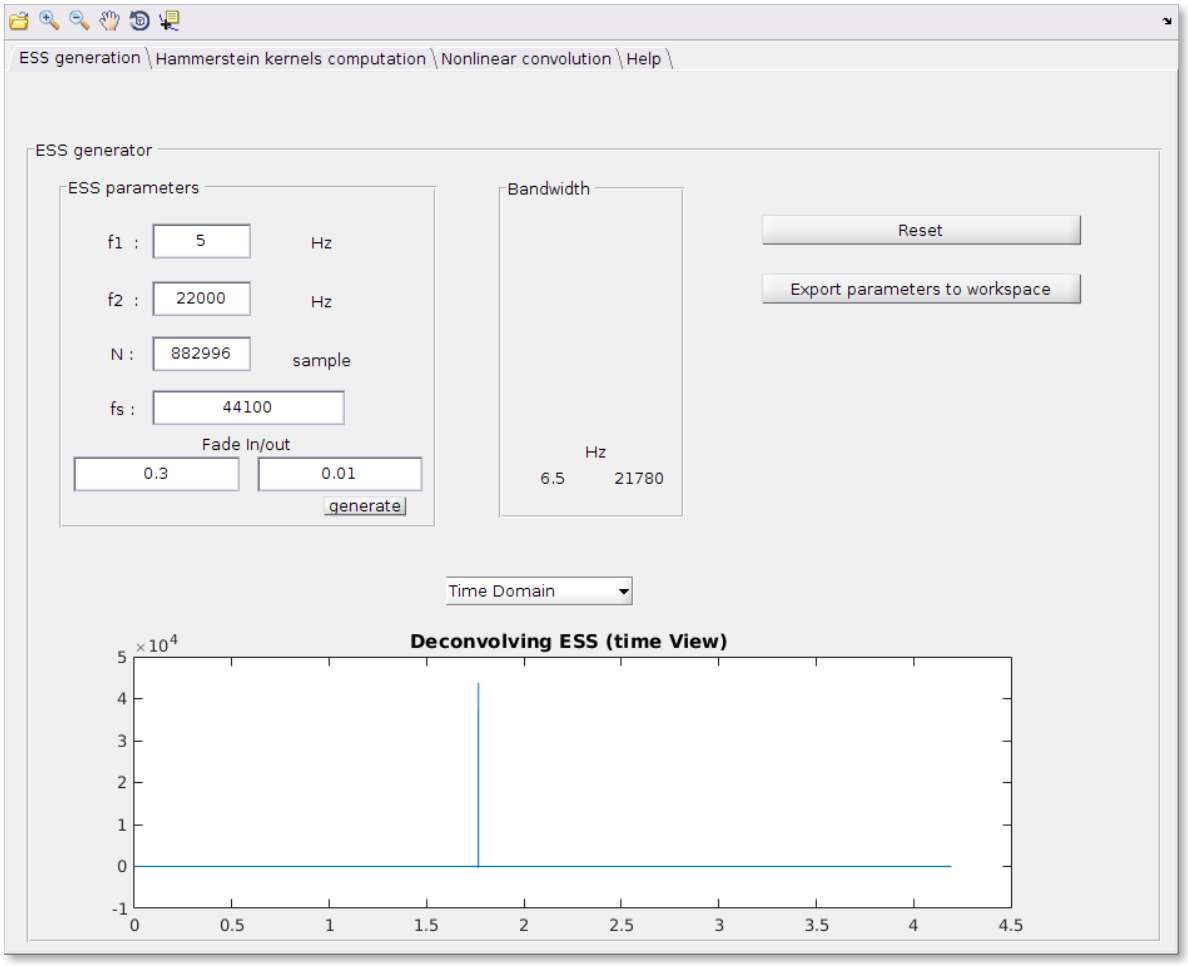


Figure 3.37 The Matlab Hammerstein identification toolbox.

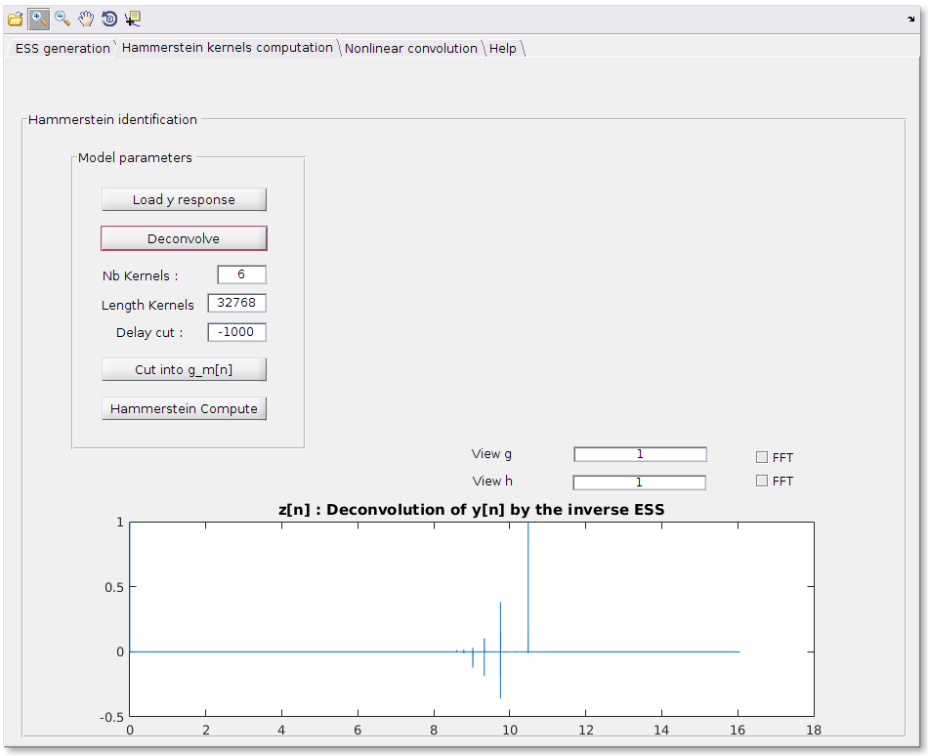


Figure 3.38 The Hammerstein kernels computation tab.

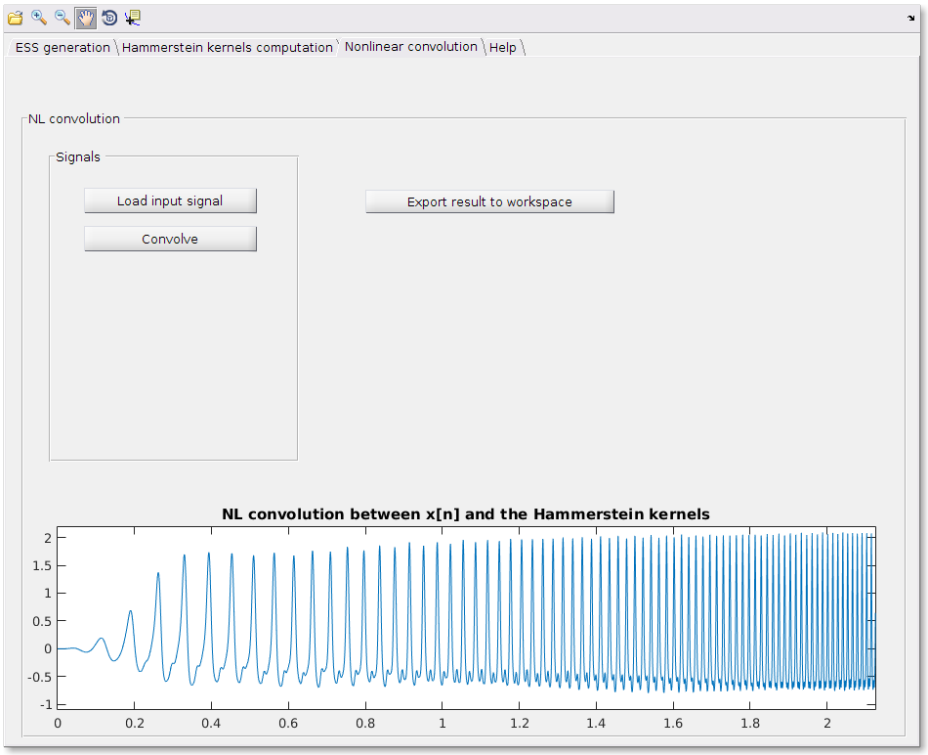


Figure 3.39 The nonlinear convolution tab.

3.8 Conclusion

Many researches have been done on Volterra kernels identification in the last few years. As the impulse response characterizes a linear system very precisely, we expect to extend the method to nonlinear systems with an equivalent success. As we have shown, the cascade of Hammerstein models can lead to a great sensitivity to inaccuracies during its implementation. In this research, we have highlighted different kinds of errors and we have shown how they can affect the emulation of a nonlinear system. Moreover we have proposed a way to get around each problem leading us to an accurate emulation of three nonlinear systems: the *tube-Screamer* like overdrive effect, the `TSA15h` tube amplifier and the `Engl retro tube 50` amplifier. We have also described the limitations of the model (i.e., maximum order, low-frequency limitation, input level dependency). Finally, we have proposed a solution in Eq. (3.40) for the input amplitude level dependency of the Hammerstein kernels which seems to be valid for some nonlinear devices. This problem is still a major limitation of this method for guitar signal emulation purpose. Indeed, the guitar signal amplitude is continuously changed according to the strength of the playing but also during the ringing of a note. Indeed, the first part of a note (the attack) can reach an amplitude of 1 V if the string is hardly picked while the amplitude can be of 10 mV if the string is softly picked. Further investigation are required to continue to improve the robustness of this method.

Chapter 4

Emulation of guitar amplifier by neural networks

Contents

Outline of chapter 4:

This chapter is devoted to the emulation of tube amplifiers by neural network models. The introduction in Sec. 4.1 explains why and how neural networks could be used as a reliable method to emulate tube amplifiers. In Sec. 4.2, different models of neural networks are presented and analyzed. The comparison between them is discussed in Sec. 4.3 according to objective metrics. Two models, i.e., the most accurate and the fastest, are discussed and evaluated in terms of perceived accuracy (through some listening tests) in Sec. 4.4.

4.1 Introduction to the emulation of guitar amplifiers by neural networks

4.1.1 Motivation to use neural networks for the guitar emulation task

Several reasons have led us to the machine learning field:

- The parallel cascade of Hammerstein models was not flexible enough to represent the distortion effect of a tube amplifier. Its static nonlinearity does not represent the nonlinear memory effects present in audio devices. The first idea was to switch to a Wiener-Hammerstein model, but no analytic identification of the filter coefficients was possible using the Exponential Sine Sweep (ESS) method. The parameters of the Wiener-Hammerstein model had to be learned by a machine learning technique.
- The HKISS method requests the use of an ESS as input signal, the estimated operator $\hat{N}$ was then more appropriate for these kinds of sinusoidal signals as introduced in Theorem. 1 Sec. 2.3. Machine learning techniques allow to get rid of the exponential sine sweep as input signal and use guitar signals during the identification phase. The estimated operator $\hat{N}$ is then well suited to emulate guitar signals.
- Neural network models are very flexible, and can represent strong nonlinearities.

An adequate neural network structure has to be chosen. As Recursive Neural Networks (RNN) are designed to treat time series, they appear as an appropriate choice for our application.

4.1.2 Related works

Only few researches have used neural networks for the emulation of tube amplifiers [Covert and Livingston, 2013; Schmitz and Embrechts, 2018c,b, 2019; Damskägg et al., 2019], even if similar models have been widely applied in the machine learning field for other types of tasks. To understand their capabilities and their limitations, it is important to trace some fundamental contributions in the machine learning field (see Appendix A.2). Artificial Neural Networks (ANN) (in opposition with biological neural networks) often simply called neural networks are the core of deep learning. They are powerful, versatile and they are used in a very large set of machine learning tasks such as: speech recognition services (SIRI), classification of images (Google), recommendation of video to watch (Youtube), finding relations between data, e.g., how S&P 500 stocks are correlated with twitter informations.

Other uses of neural networks in audio

Typically, in the audio field, neural networks have also been used for different tasks, such as (non-exhaustive list):

- Correction of loudspeaker nonlinearities [Low and Hawksford, 1993].
- Augmented reality (AR) applications, to reproduce the acoustic reverberation field [Kon and Koike, 2018], to emulate head related transfer functions [Chun et al., 2017].
- The classification of spatial audio localization and content [Hirvonen, 2015].
- Enhancement of speech intelligibility in noisy environments [Thoidis et al., 2019].
- Audio sources separation and identification [Nugraha et al., 2016].
- Audio processing technique to increase the sampling rate [Kuleshov et al., 2017].
- Music information retrieval [Shuvaev et al., 2017].

4.1.3 Dataset for guitar amplifiers

The age of *Big data* (i.e., the record and the centralization of all our activities) has made the development of efficient machine learning tasks easier. Indeed, making a general model is easier with a lot of data than with few data. In the 1980s the datasets were constituted by approximately thousands of instances. Nowadays, some datasets are created with a billion of instances [Goodfellow et al., 2016, p.21]. The same algorithms which had some difficulties to achieve some simple tasks in 1980s are now able to take care of complex tasks due to the increase of the collected data.

In [Schmitz and Embrechts, 2018a], we have introduced a new dataset gathering several guitar amplifiers output sounds. This dataset has two purposes:

- The first one is to collect enough data to train our neural networks.
- The second one is to ease the comparison of different models. Indeed, it is hard to prove that a model generalizes so well that any test sets will give exactly the same measured performance. Therefore, the comparison of two models has to be made on the same test set.

To create this dataset, different guitar signals have been recorded and then sent through several amplifiers. For each amplifier, the output signal has been recorded (as presented in Fig. 4.1) for ten positions of the *gain* parameter (a parameter of the amplifier enabling the control of the amount of distortion, where 1 is the lowest level and 10 the strongest one).

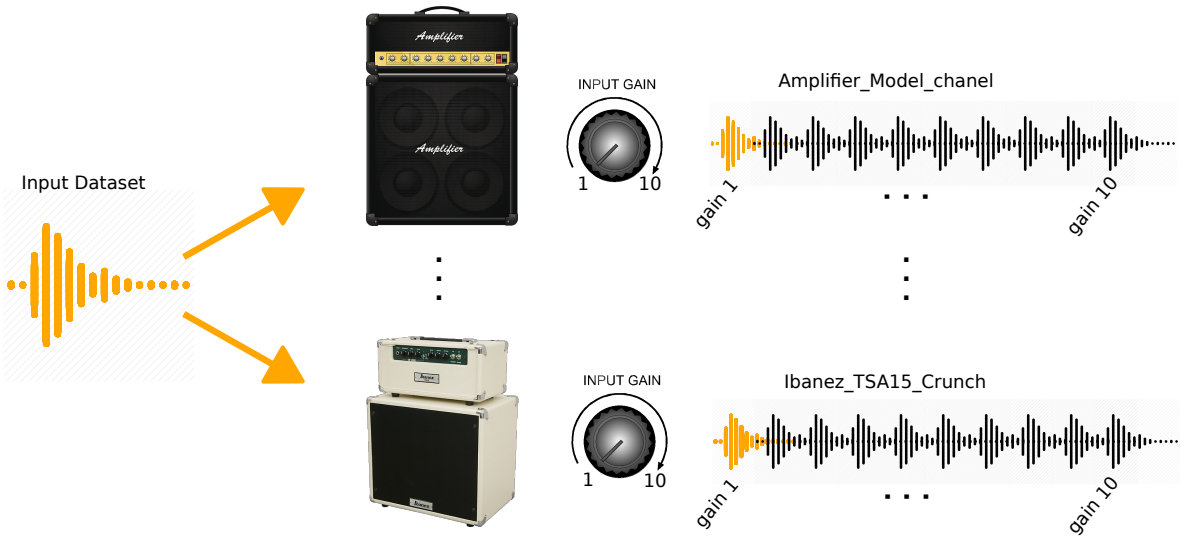


Figure 4.1 A guitar signal (*Input Dataset*) is sent through different amplifiers and is recorded for different gain levels for each amplifier.

Input set

One important point in machine learning is to obtain a model that generalizes well, i.e., from a number of training instances, the model has to be able to generalize to instances it has never seen before. The model can then be used for *inference* (i.e., emulating the output of an amplifier for any guitar signals in the input). To do so, it is important that the class of input signals is sufficiently representative of the guitar signals used during the training phase. The input dataset has to be:

- Long enough: for each amplifier and each gain, our input set contains 8,408,005 instances (i.e., guitar signal sampled at 44 100 Hz).
- Diversified: the dataset gathers 5 different musical *styles*:
  1. A chromatic scale.
  2. A set of Major and minor chords.
  3. A song with chords only "La Bamba".

4. A song with chords and single musical notes.
5. A song with single musical notes in a "Am" blues scale.

Output set

The recorded sounds are taken at the output of each amplifier (i.e., the sound card is connected to the output of the power amplifier by means of the `Torpedo` loadbox [TwoNoteEngineering, 2011] as presented in Fig. 4.2). The role of the loadbox is to attenuate the amplified output signal to a line level.

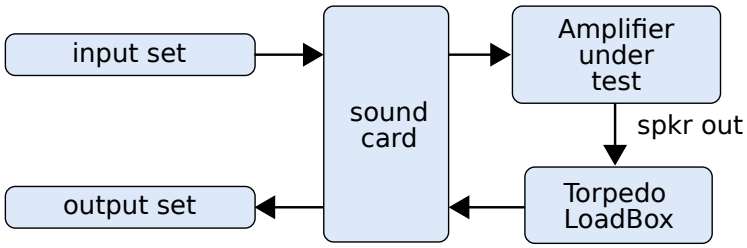


Figure 4.2 Recording of the dataset, *the output set* is a scaled (in amplitude level) version of the output signal of the amplifier (speaker out).

The amplifiers present in the dataset are:

- The `Ibanez Tsa15H` channel crunch.
- The `Mesa Boogie 550` channel clean.
- The `Mesa Boogie 550` channel crunch.
- The `Mesa Boogie 550` channel disto.
- The `Mesa Boogie MarkV` channel clean.
- The `Mesa Boogie MarkV` channel crunch.
- The `Mesa Boogie MarkV` channel disto.
- The `Engl Retro Tube 50` channel disto.
- The `Blackstar HT5M` channel disto.

More informations on this dataset are presented in Appendix. A.3. The dataset can be downloaded from [Schmitz, 2018b].

Training, testing, validating, listening

Each model presented in this chapter, is trained and evaluated as following:

- The model is trained during 15 hours using 80% of the dataset A.3 (i.e., 6 726 404 instances).
- The generalized error is computed using the remaining 20% of this dataset (i.e., 1 681 601 instances). These 20% are randomly picked over each different musical style of the dataset.
- Hyperparameters search are carried out on another dataset of 374 224 instances.
- Figures and listening tests use a third dataset of 910 337 instances to compare the differences between the target and the prediction signals.

4.2 Different neural network models for guitar amplifier emulation

Eight neural network models are presented in the following sections. Their main characteristics are summarized in Table 4.1. The accuracy in time domain given by the NMRSE (see Eq. (2.58)), the speed of the model (given in computational time CT) and the harmonic content accuracy (Δ_{harmo} from Eq. (2.63)) of each model are given on a relative scale where:

- $\star$ means that the model performs less than the others,
- $\star\star$ means that the model is in the average,
- $\star\star\star$ means that the model performs better than the average.

Motivation summary for the different models:

- Model 1: recurrent neural network, based on LSTM cells limiting the problem of vanishing or exploding gradient. This first model gives accurate results but is not optimized in terms of computational time.
- Model 2: a second layer of LSTM cells is added to gives more flexibility to the model and to determine if the resulting accuracy is improved.
- Model 3: a sequence of output samples is predicted instead of one single output sample for each input instance. The goal is to make a faster model than the model 1.
- Model 4: the gain parameter of the amplifier is added as input feature of the LSTM layer in order to see if the parameters of the amplifier can be taken into account.
- Model 5: a convolutional layer is added at the input of the neural network to reduce the length of the input sequence allowing the use of a smaller number of LSTM cells in the LSTM layer in order to reduce the computational time.
- Model 6: small feedforward neural network, this network is very fast and is used to determine if such a small neural network can reach an appropriate level of accuracy.
- Model 7: the idea is the same than for model 6 but using a small convolutional neural network.
- Model 8: this model is the same than the model 5 where the LSTM cells are replaced by simpler cells (less computation in each cell). The goal is to determine if there is a reduction of the computational time while keeping the same accuracy.

4.2.1 Model 1: Guitar amplifier emulations with a LSTM neural network

As a first approach in the neural network field, it seemed natural to work with Recurrent neural networks (RNN), as presented in Sec. 2.7.4, since they are designed to take care of time series prediction. One major problem encountered when dealing with RNN is the difficulty to learn long-term dependencies. It is known as the *vanishing* or *exploding* gradient problem as described in Appendix A.4. A special RNN cell called the Long Short Term Memory

Table 4.1 Overview of the different proposed neural network models.

	Short description	**Code**	**NMRSE**	**CT**	Δ_{harmo}
Model 1 (p.105)	One layer of LSTM cells	Appendix. A.5.2	⋆⋆	⋆	⋆⋆
Model 2 (p.121)	Two layers of LSTM cells	Appendix. A.5.3	⋆	⋆	⋆⋆⋆
Model 3 (p.126)	Sequence to sequence prediction	Appendix. A.5.4	⋆⋆	⋆	⋆⋆⋆
Model 4 (p.131)	Taking the amplifier parameters into account	Appendix. A.5.5	⋆	⋆	⋆
Model 5 (p.136)	LSTM layer with convolutional input reduction layer	Appendix. A.5.6	⋆⋆⋆	⋆⋆	⋆⋆
Model 6 (p.144)	Stacked feedforward layers	Appendix. A.5.7	⋆	⋆⋆⋆	⋆
Model 7 (p.148)	Two convolutional layers with two pooling layers	Appendix. A.5.8	⋆	⋆⋆⋆	⋆⋆⋆
Model 8 (p.153)	One layer with simple recurrent cells and convolutional input reduction layer	Appendix. A.5.9	⋆	⋆⋆	⋆⋆⋆

(LSTM) cell has been introduced [Hochreiter and Schmidhuber, 1997] in order to overcome this problem.

Introduction to the LSTM cell

The Long Short Term Memory (LSTM) cell is indeed a special RNN cell with gates to control when information enters or leaves the memory. This architecture lets the network learn long-term dependencies more easily than with simple RNN cells since the network can let the information flow through the entire network with only few modifications. Indeed, in a classic RNN, the information from the first cell is transformed at each time step. Therefore, some information is lost at each time step. In a LSTM NN, the information can be passed without alteration from the first cell to the last cell using *gates*. The Fig. 4.3 presents a LSTM cell. It is composed of:

- A short state memory vector $\mathbf{h}[n]$: this vector represents the state of the cell, it also corresponds to the output of the nth cell i.e., $\mathbf{h}[n] = \mathbf{y}[n]$. The size of this vector is called the number of *hidden units* of the cell (N_h).
- A long terms state vector $c[n]$: this vector represents a long term state since information can easily flow from a long remote cell to the current cell.
- An input signal $\mathbf{z}[n]$: The use of a vector in the notation can be confusing. In general, $\mathbf{z}[n]$ is an input vector gathering N_f *features* for the time step n: for example, the input of the LSTM cell can be the current input signal value and a set of amplifier parameters (i.e., $\mathbf{z}[n] = [x[n], Gain, Bass, Medium, Treble]$, where Gain, Bass, Medium, Treble are usual parameters of guitar amplifiers). $\mathbf{z}[n]$ can also simply be the scalar value corresponding to the value of the guitar signal sampled at a time $t = nT_e$ where T_e is the sampling period, i.e., $\mathbf{z}[n] = x[n]$.

Each LSTM cell has two tasks:

- Computing an output state $\mathbf{h}[n]$ depending on the input vector $\mathbf{z}[n]$, the previous state vector $\mathbf{h}[n-1]$ and the previous long terms dependency vector $\mathbf{c}[n-1]$.
- Transform the long terms dependency vector $\mathbf{c}[n]$ to let information leaves or enters the memory.

These two operations are made through the concept of gates. A gate is an element wise multiplication ($\otimes$) between an input vector $a[k]$ and a *gate* vector $b[k]$ containing values between zero and one. The two extreme cases are:

- if $b[k] = 0$ then $b[k] \otimes a[k] = 0 \ \rightarrow$ the element $a[k]$ is forgotten and replaced by 0.
- if $b[k] = 1$ then $b[k] \otimes a[k] = a[k] \ \rightarrow$ the element $a[k]$ is fully kept.

The LSTM cell has three gate vectors, $\mathbf{f}[n]$, $\mathbf{i}[n]$ and $\mathbf{o}[n] \in [0,1]$: $\mathbf{f}[n]$: selects which part of the long terms dependency vector $\mathbf{c}[n]$ will be kept and which part will be forgotten. The content of this gate vector is chosen according to the previous short term state $\mathbf{h}[n]$ and to the input vector $\mathbf{z}[n]$. In fact the input vector and the previous short state memory vector constitute the inputs of a fully connected layer of N_h hidden units. A sigmoid activation function is used to obtain the gate vector whose elements are comprised between 0 and 1 such that:

$$\mathbf{f}[n] = \sigma\left(\mathbf{W}_{zf}\mathbf{z}[n] + \mathbf{W}_{hf}\mathbf{h}[n-1] + \mathbf{b}_f\right). \tag{4.1}$$

Similarly, the *input gate* vector $\mathbf{i}[n]$ and the *output gate* vector $\mathbf{o}[n]$ are computed such that:

$$\mathbf{i}[n] = \sigma\left(\mathbf{W}_{zi}\mathbf{z}[n] + \mathbf{W}_{hi}\mathbf{h}[n-1] + \mathbf{b}_i\right). \tag{4.2}$$

$$\mathbf{o}[n] = \sigma\left(\mathbf{W}_{zo}\mathbf{z}[n] + \mathbf{W}_{ho}\mathbf{h}[n-1] + \mathbf{b}_o\right). \tag{4.3}$$

Their purposes are respectively: to choose which part of the *input candidate* $\mathbf{g}[n]$ will be added to the long term dependency vector and to choose which part of the long term dependency vector will be used to compute the output state of the cell $\mathbf{h}[n]$. The *input candidate* $\mathbf{g}[n]$ to add to the long term dependency vector is computed as:

$$\mathbf{g}[n] = \tanh\left(\mathbf{W}_{zg}\mathbf{z}[n] + \mathbf{W}_{hg}\mathbf{h}[n-1] + \mathbf{b}_g\right). \tag{4.4}$$

One can notice that the only difference with the computation of the gate vectors is the use of the hyperbolic tangent instead of the sigmoid as activation function since the result does not need to be comprised between zero and one anymore. The long term dependency vector $\mathbf{c}[n]$ can then be computed as:

$$\mathbf{c}[n] = \mathbf{c}[n-1] \otimes \mathbf{f}[n] + \mathbf{g}[n] \otimes \mathbf{i}[n]. \tag{4.5}$$

Finally, the short memory state $\mathbf{h}[n]$ can be computed as:

$$\mathbf{h}[n] = \tanh\left(\mathbf{c}[n]\right) \otimes \mathbf{o}[n], \tag{4.6}$$

where:

- $\mathbf{W}_{hf}$, $\mathbf{W}_{hi}$, $\mathbf{W}_{hg}$, $\mathbf{W}_{ho}$ are matrix weights of size $[N_\mathrm{h} \times N_\mathrm{h}]$.

- $\mathbf{W}_{zf}$, $\mathbf{W}_{zi}$, $\mathbf{W}_{zg}$, $\mathbf{W}_{zo}$ are matrix weights of size $[N_\mathrm{h} \times N_\mathrm{f}]$.
- $\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_o$, $\mathbf{b}_g$ are bias vector of size $[N_\mathrm{h} \times 1]$.
- $\mathbf{c}$ and $\mathbf{h}$ are state vectors of size $[N_\mathrm{h} \times 1]$.

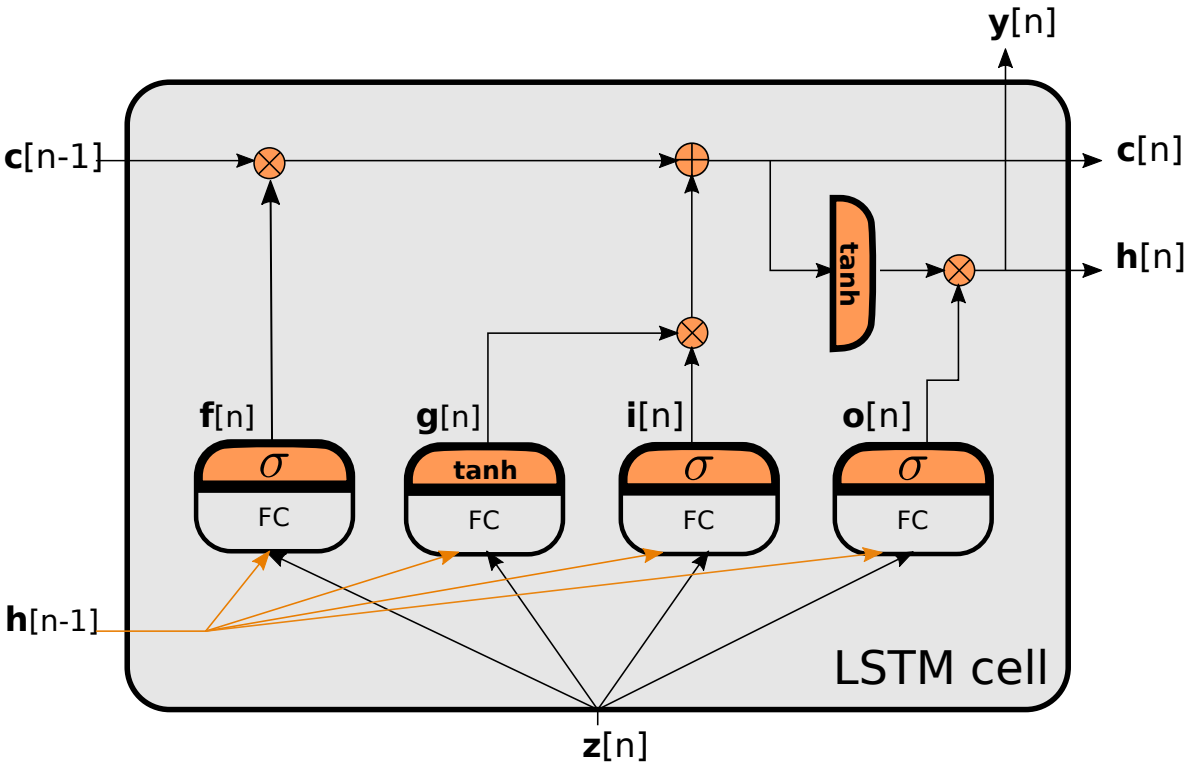
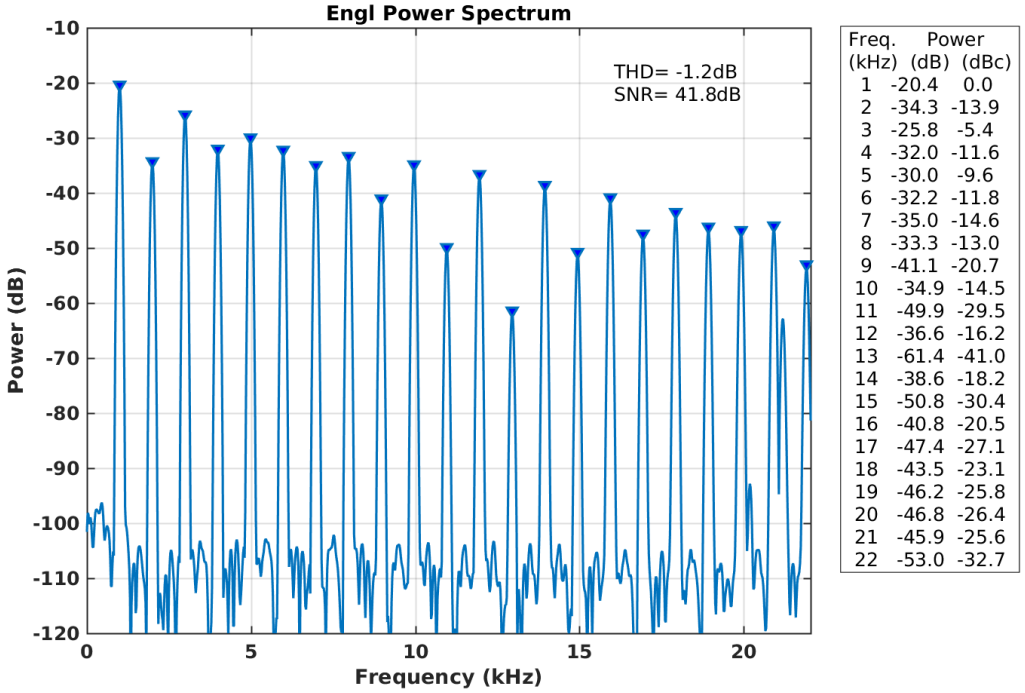


Figure 4.3 A Long Short Term Memory cell.

Presentation of the tested amplifier

The amplifier *Engl retrotube 50* (Distortion channel) has been used to evaluate the eight models presented in this chapter. The power spectrum of the output of the amplifier, when the input signal is an ESS with a focus on frequencies close to f=1000 Hz, is presented in Fig. 4.4. In

this power spectrum, there are 22 harmonics from which ten have less than 20 dB difference with the amplitude of the fundamental frequency. This illustrates the strong nonlinear behavior of this tube amplifier. The amplitude level of the harmonics are given in dB. The column " dBc" presents the normalized harmonics such that the fundamental has an amplitude level of "0 dBc". This will ease the comparison between the amplitude levels of the target harmonics and the prediction harmonics. The Total Harmonic Distortion (THD) and the Signal to Noise Ratio (SNR) are presented, for the same input frequency, in Figs. 4.5 and 4.6 respectively.



Freq. (kHz)	Power (dB)	(dBc)
1	-20.4	0.0
2	-34.3	-13.9
3	-25.8	-5.4
4	-32.0	-11.6
5	-30.0	-9.6
6	-32.2	-11.8
7	-35.0	-14.6
8	-33.3	-13.0
9	-41.1	-20.7
10	-34.9	-14.5
11	-49.9	-29.5
12	-36.6	-16.2
13	-61.4	-41.0
14	-38.6	-18.2
15	-50.8	-30.4
16	-40.8	-20.5
17	-47.4	-27.1
18	-43.5	-23.1
19	-46.2	-25.8
20	-46.8	-26.4
21	-45.9	-25.6
22	-53.0	-32.7

Figure 4.4 Power spectrum of the amplifier *Engl retro tube 50 Disto* for a frequency $f \simeq 1000\,\text{Hz}$.

Presentation of model 1

Our first model is presented in Fig. 4.7: the N last values (samples) of the input guitar signal x are sent to a LSTM neural network unrolled over N time steps. One particularity of a recurrent neural network is the use of only one set of parameters for all the unrolled time steps. This means that the parameters $\mathbf{W}_{hf}$, $\mathbf{W}_{hi}$, $\mathbf{W}_{hg}$, $\mathbf{W}_{ho}$, $\mathbf{W}_{zf}$, $\mathbf{W}_{zi}$, $\mathbf{W}_{zg}$, $\mathbf{W}_{zo}$,$\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_o$, $\mathbf{b}_g$ are shared between all the LSTM cells. The output state $\mathbf{h}[n]$ is transformed into a single value by a Fully Connected (FC) layer. This single value is the prediction $\hat{y}[n]$ of the neural network for the time sequence $\mathbf{x} = [x[n-N+1], \cdots, x[n]]$. The prediction of the neural network $\hat{y}[n]$ is

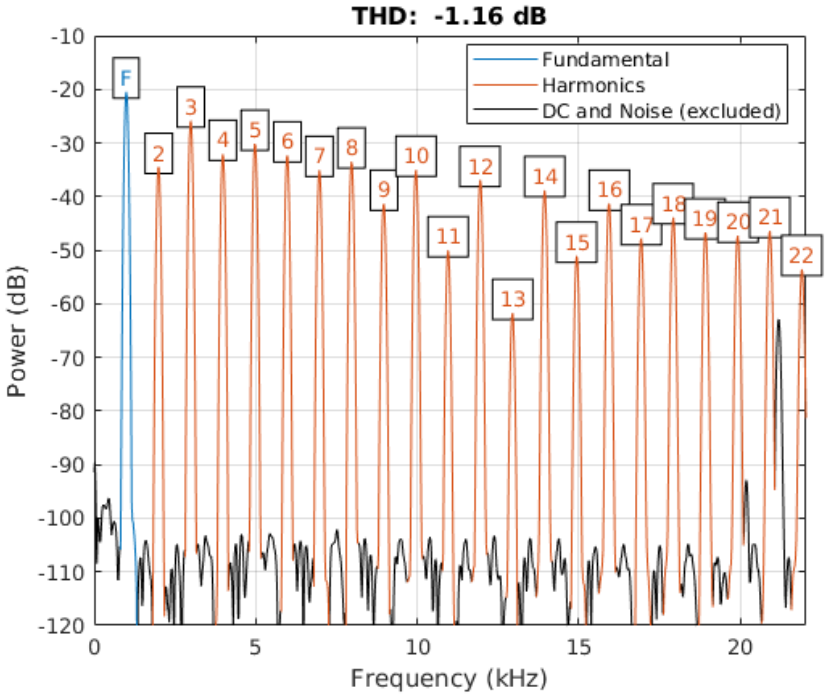


Figure 4.5 Total Harmonic Distortion of the amplifier *Engl retro tube 50 Disto* for a frequency $f \simeq 1000\,\text{Hz}$.

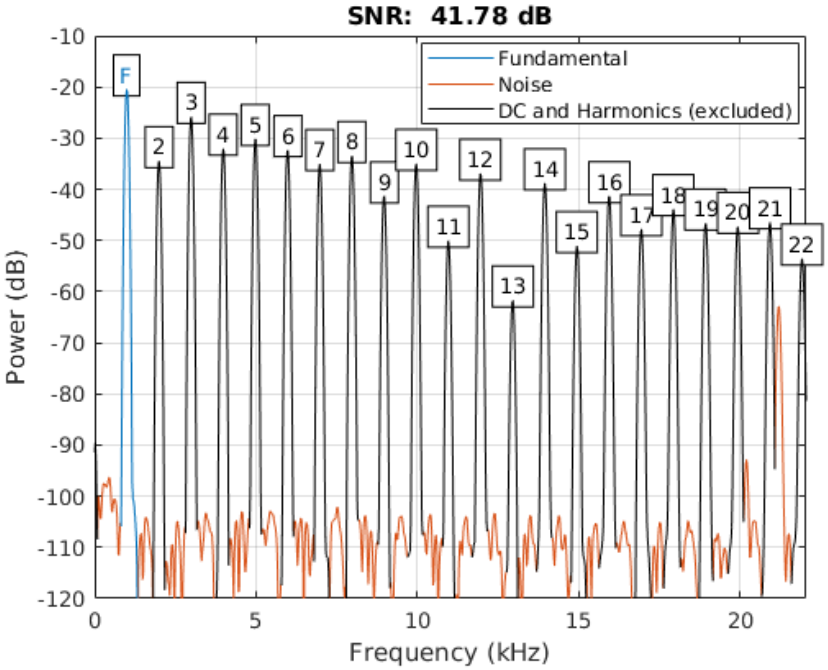


Figure 4.6 Signal to Noise Ratio of the amplifier *Engl retro tube 50 Disto* for a frequency $f \simeq 1000\,\text{Hz}$.

then compared to the real output of the amplifier i.e., the *target* $y[n]$ by mean of a Normalized Root Mean Square Error (NRMSE) cost function (as presented in Eq. 2.58). The Tensorflow framework [Abadi et al., 2015] is used in this research, the code to build the graph of this neural network model can be found in Appendix A.5.2 and the entire code can be downloaded in [Schmitz, 2019b].

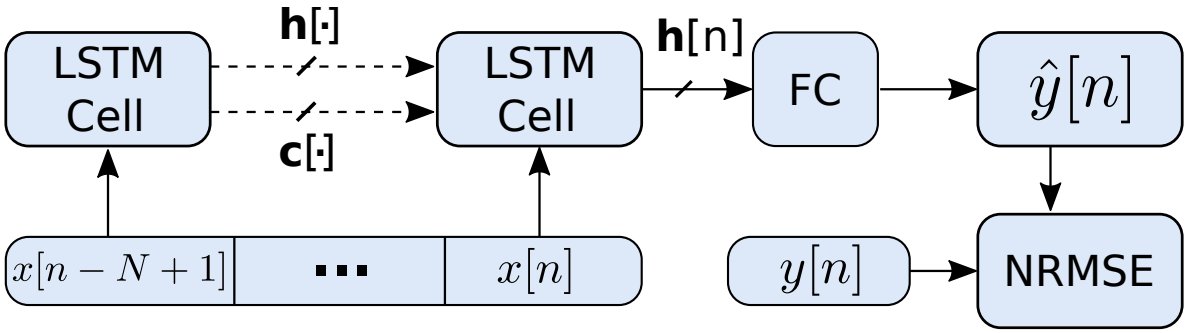


Figure 4.7 Model 1: emulation of tube amplifier with LSTM neural network.

Parameters of the model

The hyperparameters of the model (i.e., the minibatch size, the number of hidden units N_h, the number of unrolled time steps N) have been chosen by random exploration. Actually, some bounds have been fixed for each hyperparameter. These bounds have been chosen by eliminating the hyperparameter values which give not accurate results (too small number of hidden units for example) or by eliminating values which increase too much the computational time (too many time steps for example). To estimate the hyperparameters, the model has been trained during three hours on a validation set using hyperparameters selected randomly between these bounds (using an Intel 7-4930K @3.4 GHz*12 equipped with a Nvidia Titan xp graphic card). Then, an accuracy performance (the RMSE) has been measured on a subset of this validation set. This validation set is composed of 30 seconds of guitar signal sampled at 44 100 Hz. 10 % of this validation set are reserved to measure the performances of the model according to the choice of the hyperparameters. This gives approximately 1190 000 instances for the training phase and 132000 instances for the evaluation. The results are presented in Fig. 4.8. The upper subfigure represents the RMSE of the network in regard of the following hyperparameters: the minibatch size, the number of time steps (N) and the number of hidden units (N_h), while the lower subfigure is its projection into the $(N,\ N_h)$ plane. One can notice that the learning rate is not in the exploration space since the used optimization algorithm Adam [Kingma and Ba, 2014]) is an adaptative learning rate algorithm. The learning

rate hyperparameter η can usually be set to its default value (i.e., $\eta = 0.001$). The others hyperparameters are discussed below:

- The minibatch size: rather than computing the gradient on all the instances of the training set, an estimation can be obtained using a subset called a minibatch. A compromise, between the accuracy of the computed gradient and the speed to compute it, has to be made. In fact, accelerating the gradient computation has a stronger impact in the algorithm convergence than increasing the accuracy of the estimated gradient. However, the use of a GPU enables the parallel computation of the gradient for several instances. Depending on the used GPU, the size of the minibatch can be chosen to take advantage of its numerous processing cores, in order to increase the accuracy of the computed gradient without any expense in computational time. As it can be seen in Fig. 4.8 for a too small minibatch size, the GPU is under used and the convergence is slow leading to a too high RMSE after 3 hours of training. Similarly, a too large minibatch size led to a slow computation of the gradient and to a too high RMSE after 3 hours of training. A good compromise seems to choose a minibatch size comprised between [500, 1500] instances.

- The number of hidden units (N_h) represents the length of the vector state in the LSTM cell. The Fig. 4.8 shows that best results are obtained by using a number of hidden units comprised between [150, 200].

- The number of unrolled time steps (N) represents the memory length of the system. As it can be seen in Fig. 4.8 there is no need to increase this hyperparameter too much since the models having a number of time steps comprised between [100, 200] give already good results. Results for $N < 100$ are not shown but previous explorations have shown that models with less than a hundred of time steps perform poorly.

Discussion and results

The Fig. 4.9 shows how a vector of N_{data} samples will be reshaped into $minibatch\_size$ instances of size N to be treated in parallel by the GPU.

Computational time of a model

The purpose of the models is to enable an accurate emulation of a tube amplifier in real time. Usually, the speed of a model is measured in Floating Point Operation Per Second (FLOPS).

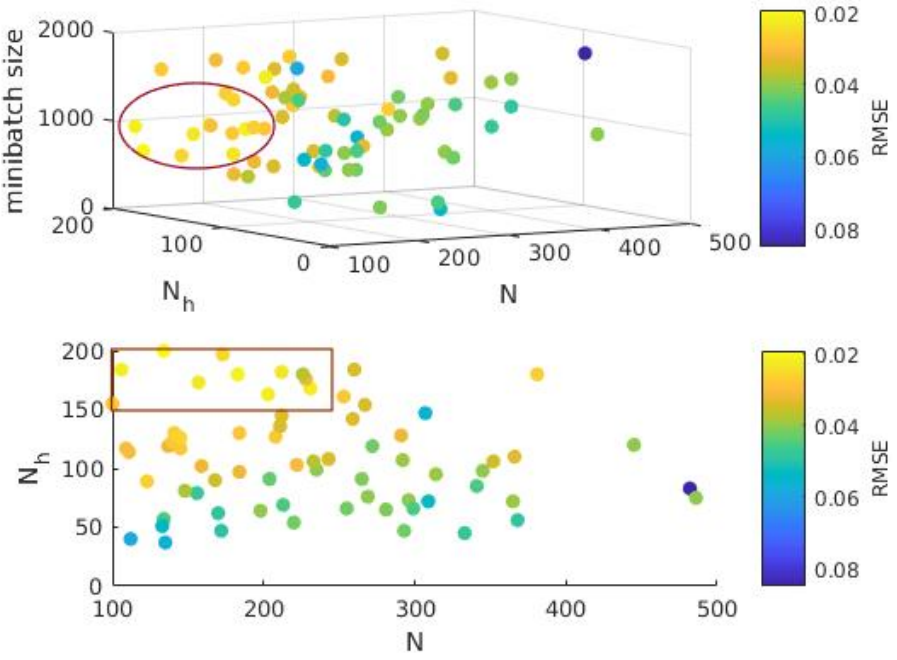


Figure 4.8 Model 1: hyperparameters search (i.e., minibatch size, number of hidden unit N_h, the number of unrolled time steps N) random exploration for one layer of LSTM neural network.

However, the use of GPU and their thousands of cores gives the opportunity to highly parallelize the computations. The number of FLOPS is thus inconsistent since it does not tell how much the model is parallelizable. Therefore, it is not a good indicator to estimate if a model respects the real-time constraint. Rather than using FLOPS, the Computational Time (CT) to process a buffer (i.e., a vector of L input samples) is used. In this research, the size of the buffer has been set to $L = 400$ samples in order to keep a small latency (i.e., at a sample rate $f_s = 44\,100\,\text{Hz}$, a new input buffer of 400 samples is sent every 9.1 ms, such that, the maximal allowed CT of the model should be of 9.1 ms). The CT is measured between the moment when the incoming buffer is sent to the neural network model and the moment when the model returns the corresponding output values.

Remarks:

- The CT depends on the computer choice (i.e., an Intel core i7-4930K CPU at $3.4\,\text{GHz} \cdot 12$, 32 Gb of ram and a GPU Nvidia Titan Xp). During this research, the same computer has been used for all the model evaluations which makes the CT a consistent comparative/relative measure.

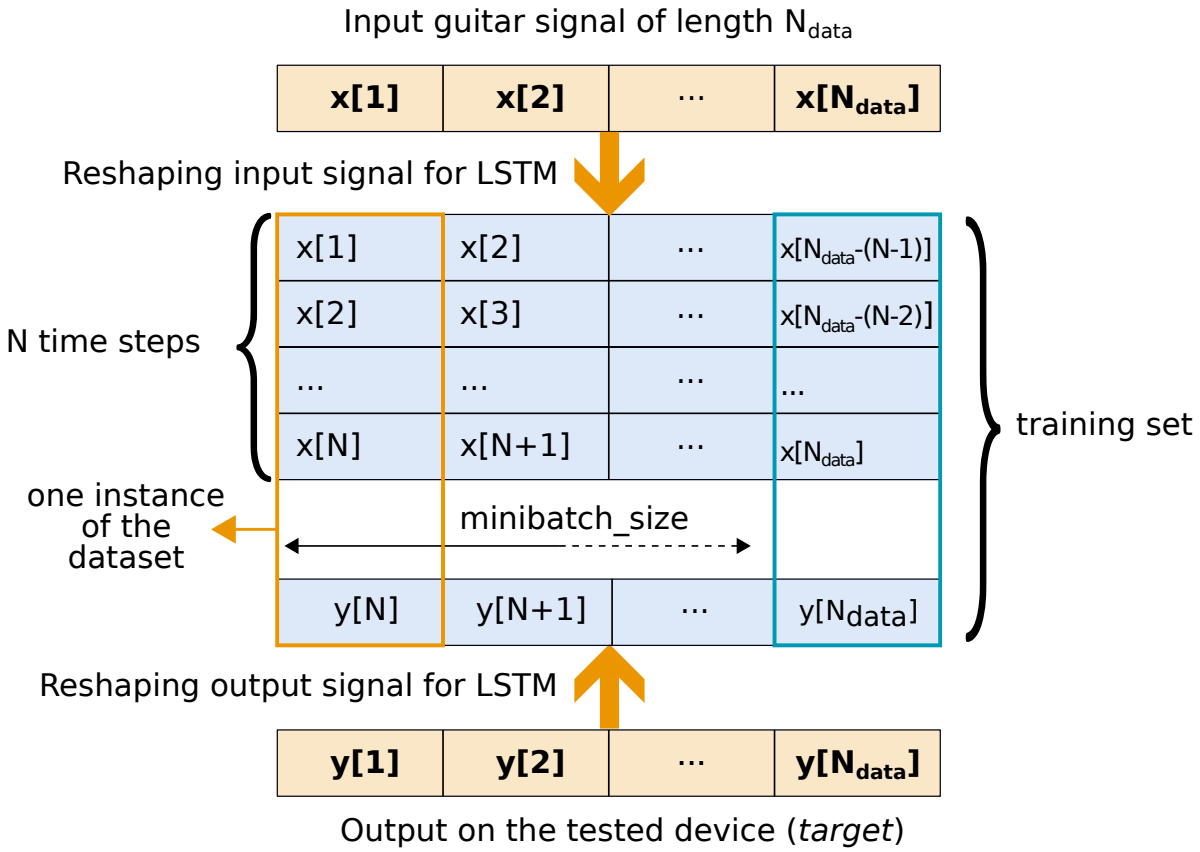


Figure 4.9 Reshaping a guitar signal of length N_{data} to have $N_{data} - (N-1)$ instances where each instance is composed of a sequence of input signal of length N (the number of time step) and the output sample of the device under test corresponding to the last sample of the input sequence. The *minibatch_size* corresponds to the number of instances treated in parallel by the GPU.

- One advantage of the method is to measure the time to process a buffer of L samples including the time to transfer data to the GPU.

Table 4.2 presents the normalized root mean square error and the computational time to process a buffer of 400 samples using the model 1 to emulate the *Engl Retro Tube 50* amplifier on its *Disto* channel with a gain of 5. The hyperparameters used to train the model are : minibatch size = 1000, number of hidden units $N_\text{h} = 150$, number of unrolled time steps N=100.

Table 4.2 Model 1: normalized root mean square error, computational time and harmonic content accuracy for a single LSTM layer model (parameters: N=100, $N_\text{h} = 150$).

Amplifier	channel	Gain	CT(ms)	NRMSE(%)	Δ_harmo(dB)
Engl	Disto	5	7.1	39%	5.3

To illustrate the accuracy of the model, we compare in Fig. 4.10 the signal at the output of the *Engl Retro Tubes 50* (the tube amplifier) and its prediction by the neural network model in the time domain. The *validation set* used for this test is a guitar signal never seen by the model before (see Sec. 4.1.3). Moreover, Fig. 4.11 presents the comparison between the spectrogram of the target (Fig. 4.11a) and the spectrogram of the prediction (Fig. 4.11b) when the input signal is an ESS with a bandwidth comprised between 50 and 5000 Hz. Fig. 4.12a presents the spectrogram of this ESS input signal. For clarity, Fig. 4.12b shows the evolution of its instantaneous frequency on a logarithmic vertical axis. All the spectrograms presented in this chapter are computed by using the following parameters:

- NFFT = 256, the Fast Fourier Transform (FFT) length,
- noverlap = 128, the number of overlapping samples between two consecutive FFT,
- the window applied is a Kaiser window [Harris, 1978] with a beta parameter equal to 8.

Several observations on the spectrograms:

- As it can be seen in Fig. 4.11a, there are many curves in the spectrogram, showing that the *Engl Retro Tube 50* is a high-order nonlinear system.
- Fig. 4.13 presents the comparison between the power spectrum of the target and the prediction when the input frequency is $f \simeq 1000$ Hz in the ESS. In this figure, $\Delta \triangleq \text{dB}c_\text{prediction} - \text{dB}c_\text{target}$ represents the difference between the amplitude level of the harmonics such that:

  - if $\Delta < 0$, the harmonic amplitude is underestimated in the model,
  - if $\Delta > 0$, the harmonic amplitude is overestimated in the model.

  From now on, a predicted harmonic is said:

  - *underestimated* if $\Delta < -3$ dB,
  - *overestimated* if $\Delta > 3$ dB,
  - *well estimated* if -3 dB $< \Delta < 3$ dB.

  As it can be seen in Fig. 4.13, the harmonics number 4, 6, 8, 10 are underestimated while the harmonic number 9 is over estimated. The model also creates some noisy signals which are not present in the amplifier system. As shown by the spectrogram in Fig. 4.11b, this noisy behavior increases with frequency, after $t = 13$ s (i.e., after 1500 Hz in the ESS input signal): the signal seems to be mixed with a white noise as it can also be seen in Fig. 4.14 where the comparison between the power spectrum of the target and the prediction when the input frequency is $f \simeq 2000$ Hz in the ESS is presented. The Signal to Noise Ratio (SNR) of the model for frequencies close to f=1000 Hz is equal to 15.5 dB while it is equal to 1.88 dB for frequencies close to 2000 Hz.

- The THD measured at 1000 Hz shows that the model overestimates the harmonic content ($THD_{\text{target}} = -1.2$ dB $< THD_{\text{prediction}} = -0.65$ dB).

- The noise becomes very important from $t = 13$ s which corresponds to a frequency of 1500 Hz. But this is also the higher fundamental frequency of our guitar. So after $t = 13$ s, the model does not know how to respond since it has never been trained for these frequencies. The same happens for low frequencies, below $t = 2$ s, since the input ESS frequency is below 80 Hz which is also the lowest fundamental frequency of our guitar.

- Finally, the model is more accurate between 5s and 13s, which corresponds (in the ESS input signal) to the frequency range [160-1500] Hz.

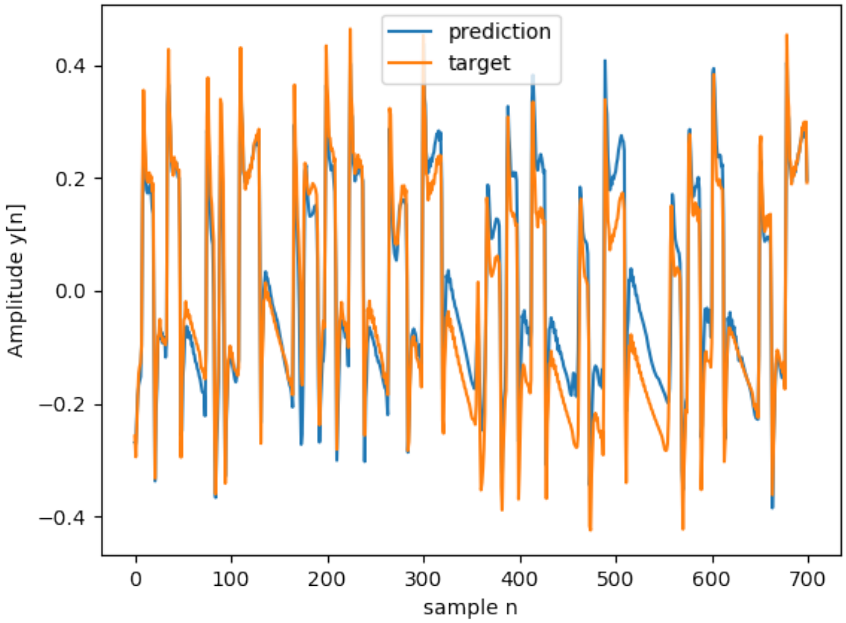
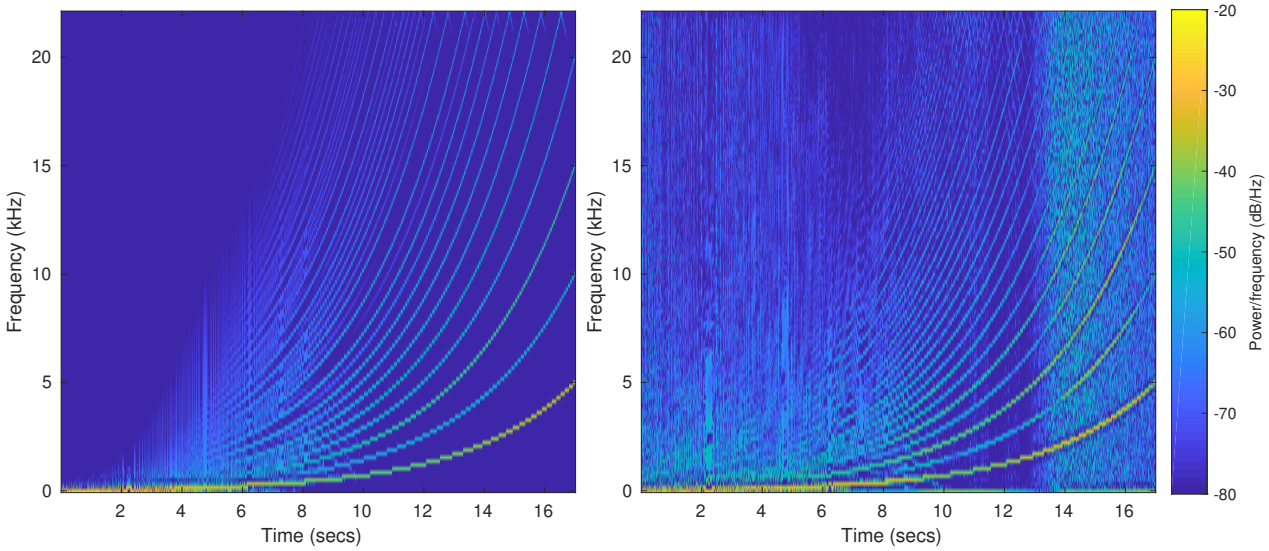


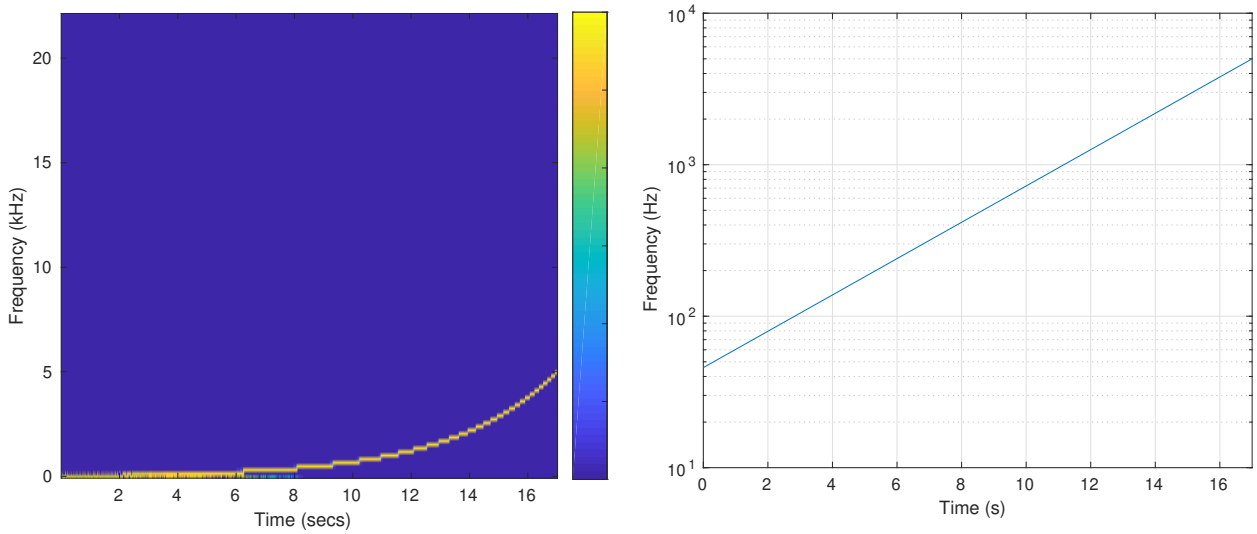
Figure 4.10 Comparison in the time domain of the output of the amplifier (*target*) and the output of the neural network model 1 (*prediction*) when a guitar signal is provided at the input.



(a) Target Engl retro tube 50 gain 5.

(b) Prediction model 1.

Figure 4.11 Comparison of the spectrograms of the target(a) and the prediction(b) for model 1 when the input is an ESS in the frequency range [50-5000] Hz.



(a) Spectrogram of the ESS in the frequency range [50-5000] Hz.

(b) ESS frequency as a function of time.

Figure 4.12 ESS input signal in the frequency range [50-5000] Hz.

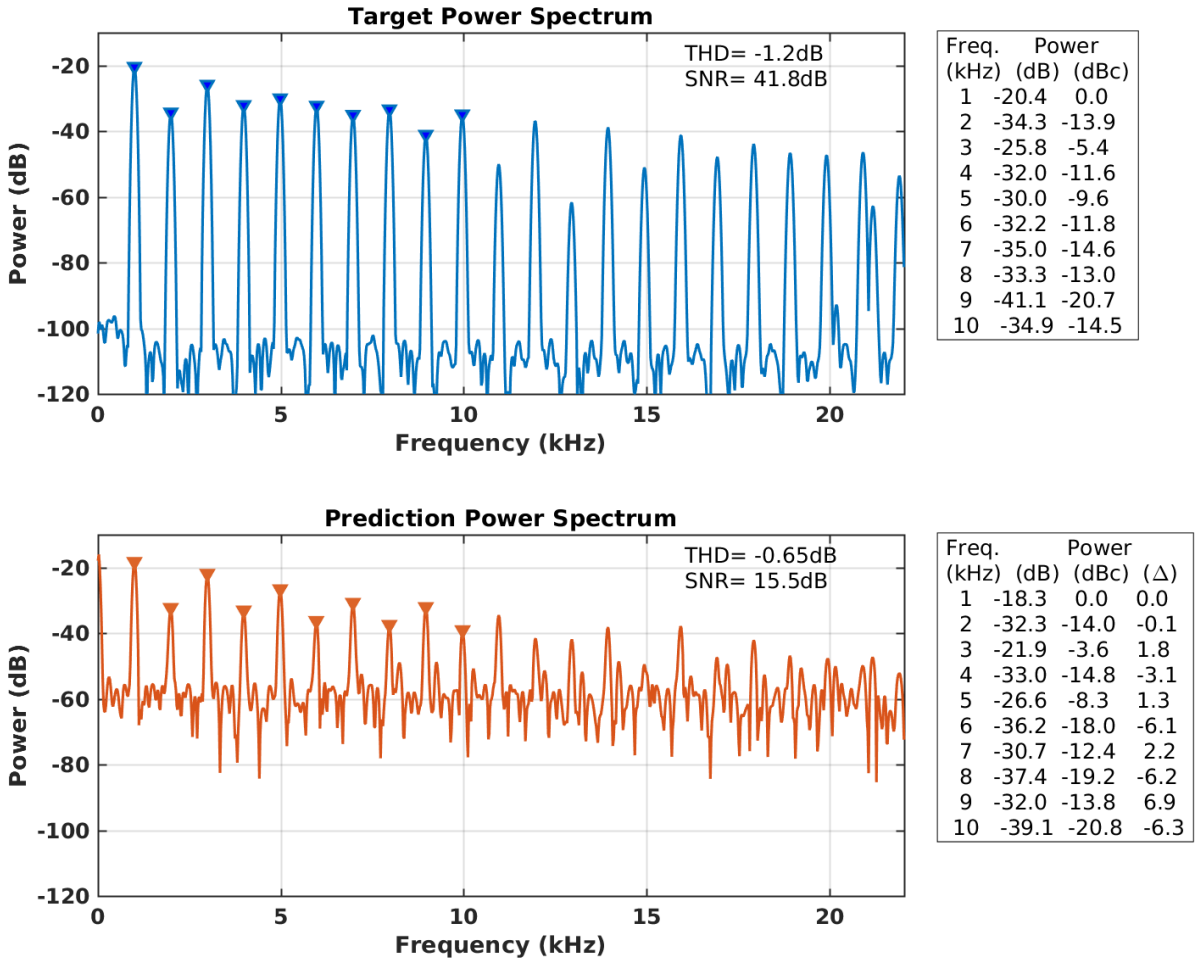
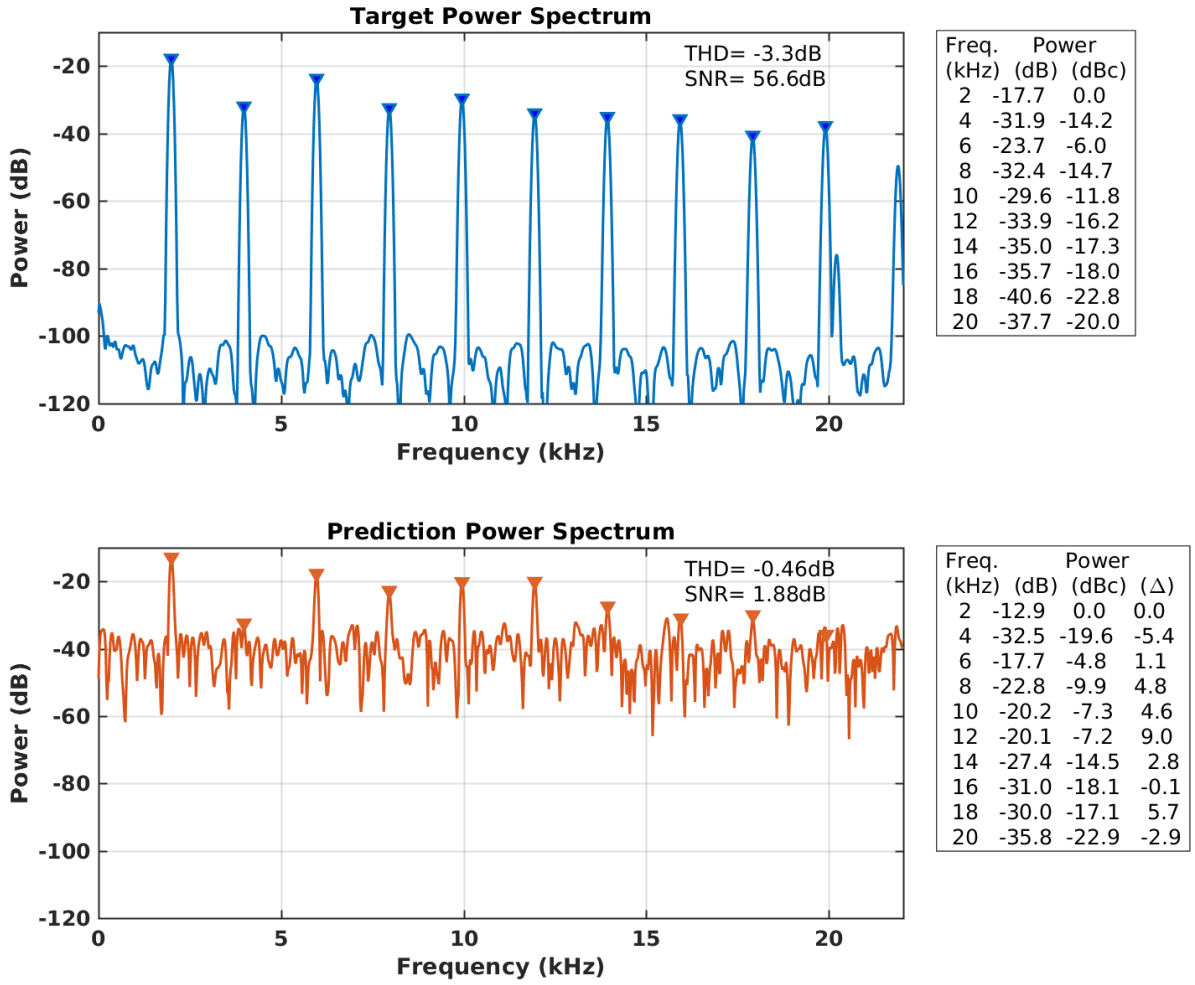


Figure 4.13 Power spectrum of the target compared with the power spectrum of the prediction for frequencies close to $f \simeq 1000\,\mathrm{Hz}$ in the ESS (model 1). $\Delta = \mathrm{dB}c_{\mathrm{prediction}} - \mathrm{dB}c_{\mathrm{target}}$.



Freq. (kHz)	Power (dB)	(dBc)
2	-17.7	0.0
4	-31.9	-14.2
6	-23.7	-6.0
8	-32.4	-14.7
10	-29.6	-11.8
12	-33.9	-16.2
14	-35.0	-17.3
16	-35.7	-18.0
18	-40.6	-22.8
20	-37.7	-20.0

Freq. (kHz)	Power (dB)	(dBc)	(Δ)
2	-12.9	0.0	0.0
4	-32.5	-19.6	-5.4
6	-17.7	-4.8	1.1
8	-22.8	-9.9	4.8
10	-20.2	-7.3	4.6
12	-20.1	-7.2	9.0
14	-27.4	-14.5	2.8
16	-31.0	-18.1	-0.1
18	-30.0	-17.1	5.7
20	-35.8	-22.9	-2.9

Figure 4.14 Power spectrum of the target compared with the power spectrum of the prediction for a frequency $f \simeq 2000\,\text{Hz}$ in the ESS (model 1). $\Delta = \text{dB}c_{\text{prediction}} - \text{dB}c_{\text{target}}$.

4.2.2 Model 2: two layers of LSTM cells

Presentation of model 2

A single layer of LSTM cells already gives promising results. This second model is a neural network with two LSTM layers as presented in Fig. 4.15. The Tensorflow code used to build the graph of this neural network model can be found in Appendix A.5.3 and the entire code can be downloaded in [Schmitz, 2019b].

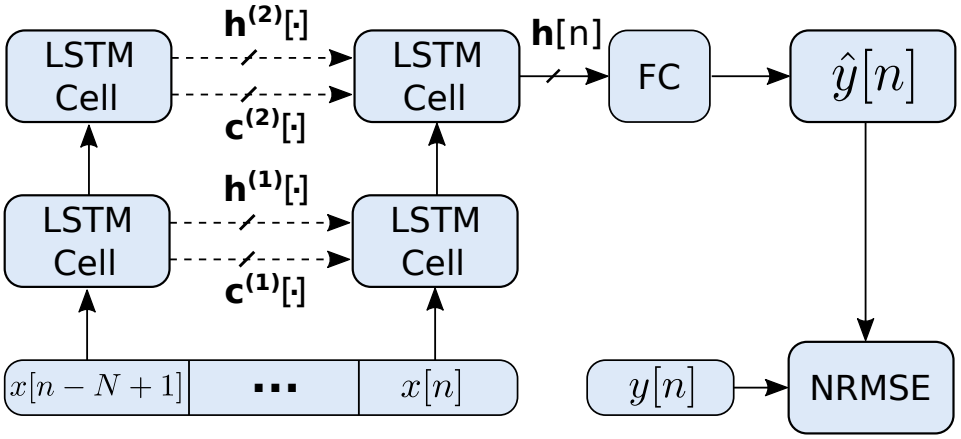


Figure 4.15 Model 2: emulation of tube amplifiers with two staked LSTM layers.

Parameters of the model

The results of the hyperparameters exploration are presented in Fig. 4.16. The results are similar to those obtained for the model 1, i.e., the best scores are given by a number of hidden units N_h comprised between [150, 200] and a number of unrolled time steps N comprised between [100, 200].

Discussion and results

As presented in Table. 4.3, the NRMSE is worse than the one reached by a single LSTM layer and the computational time is approximately two times bigger. The fact that this model does not have a better NRMSE than the previous model is surprising, since the model is more flexible and has much more parameters than the model 1. Some optimization of the learning rate can be made but since all models have used the same learning rate, it has not been realized. However, this model has the best harmonic spectrum accuracy as presented in Fig. 4.19 and Table 4.3.

The hyperparameters used to train the model are: the minibatch size = 1000, the number of hidden units $N_\text{h} = 150$ and the number of unrolled time steps N=100. The model is too slow to run in real time since it computes a buffer of 400 samples in 12.6 ms which is superior to the computational time available at a sample rate of 44 100 Hz (i.e., CT available = $\frac{400}{44100} = 9.1$ ms). Two options are still possible to run this model in RT (at the expense of the latency):

- Decrease the sample rate to increase the CT available.
- Increase the number of samples in the buffer in order to increase the CT available. However, increasing the number of samples in the buffer also increases the number of computations but if the model is well parallelized and the resources of the GPU are not

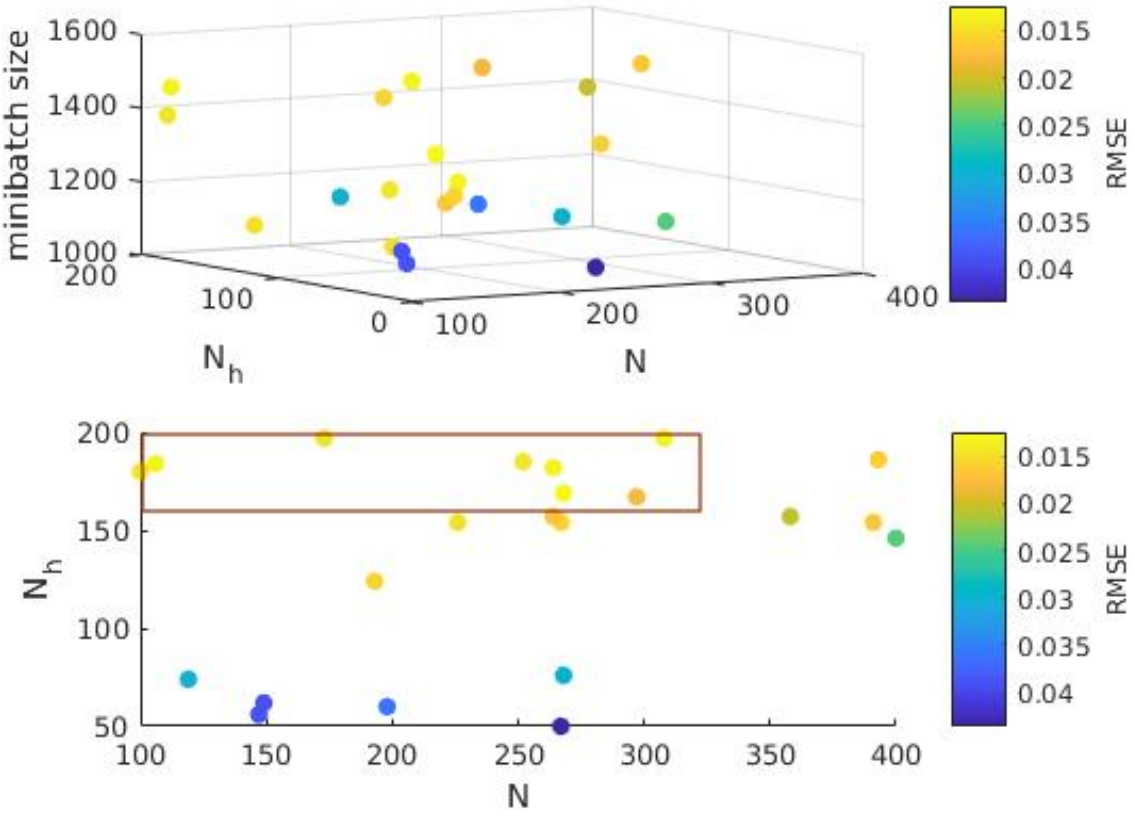


Figure 4.16 Model 2: hyperparameters exploration (i.e., minibatch size, number of hidden unit N_h, the number of unrolled time steps N), random exploration for two stacked layers of LSTM neural network.

overwhelmed, the CT will increase less than proportionately. For example, the CT to emulate a buffer 3 times bigger (buffer = 1200 instead of 400) is $CT_{1200} = 11.2$ ms which is less than 2 times the CT for a buffer of 400 samples $CT_{400} = 7.1$ ms. This is due to the possible parallel computation of several instances of the minibatch by the different cores of the GPU.

Table 4.3 Model 2: normalized root mean square error, computational time and harmonic content accuracy (parameters: N=100, $N_\text{h} = 150$).

Amplifier	channel	Gain	CT(ms)	NRMSE(%)	Δ_harmo(dB)
Engl	Disto	5	12.6	42%	2.6

To illustrate the accuracy of the model, we compare in Fig. 4.17 the signal at the output of the *Engl Retro Tubes 50* (the tube amplifier) with its prediction by the neural network model. Moreover, Fig. 4.18 presents the comparison between the spectrogram of the target (Fig. 4.18a) and the spectrogram of the prediction (Fig. 4.18b) when the input signal is an ESS with a bandwidth comprised between 50 and 5000 Hz. Even if the NRMSE is worse than for the model 1, the spectrogram of the model 2 is cleaner (i.e., less noisy). The Fig. 4.19 presents the comparison between the power spectrum of the target and the prediction when the input frequency is close to $f \simeq 1000$ Hz in the ESS. The SNR measured at 1000 Hz is better than for model 1 ($SNR_{model2} = 23.2$ dB $> SNR_{model1} = 15.5$ dB). The harmonic number 4 is overestimated while the harmonic number 9 is underestimated. The THD measured at 1000 Hz shows that the model still overestimates the harmonic content ($THD_\text{target} = -1.2$ dB $< THD_\text{prediction} = -0.78$ dB).

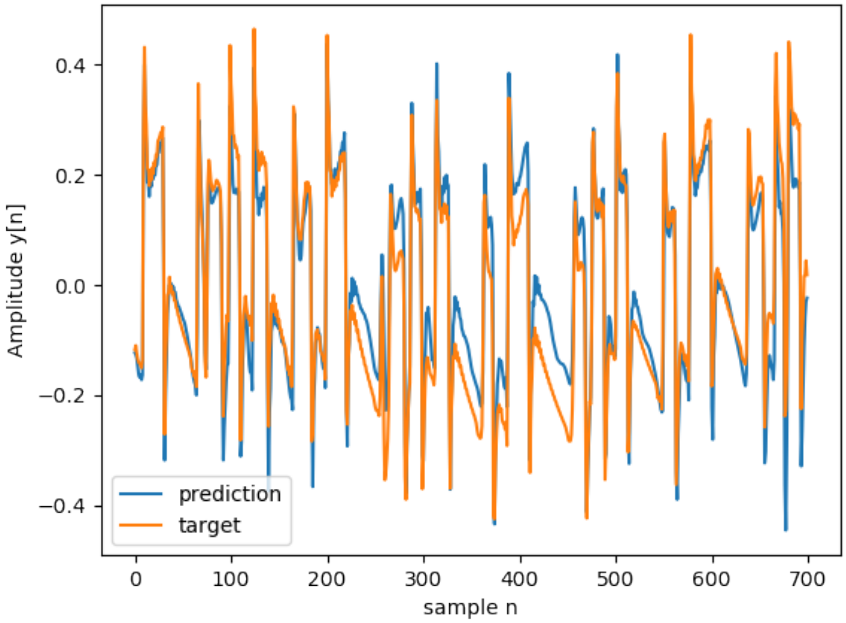
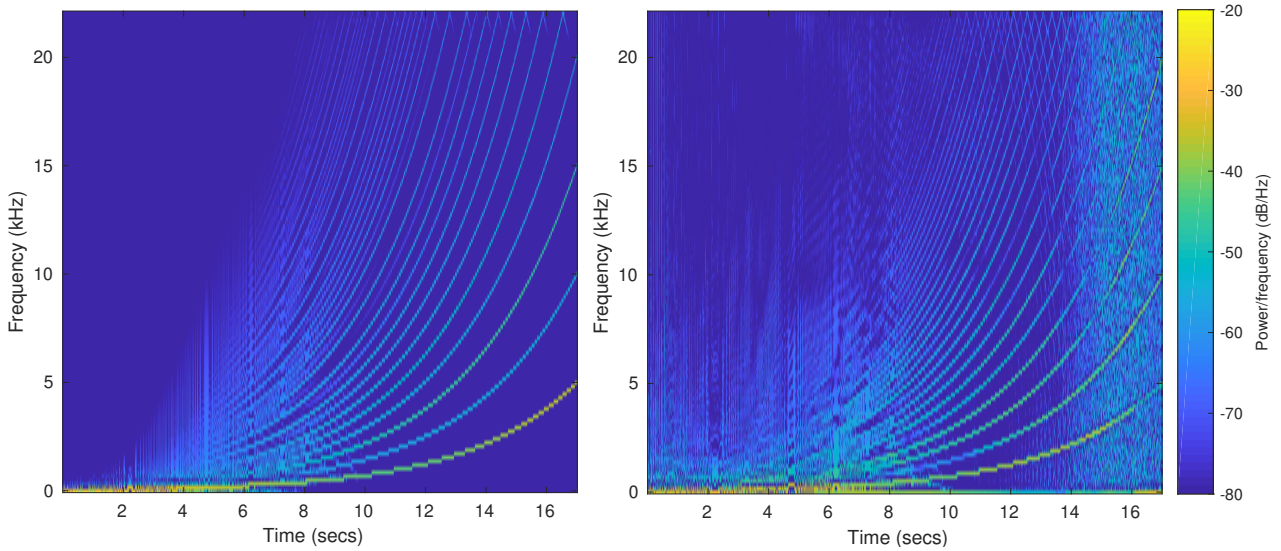


Figure 4.17 Comparison in the time domain of the output of the amplifier (*target*) and the output of the neural network model (*prediction*) when a guitar signal is provided at the input.



(a) Target Engl retro tube 50 gain 5.

(b) Prediction model 2.

Figure 4.18 Comparison of the spectrograms of the target(a) and the prediction(b) for model 2 when the input is an ESS in the frequency range [50-5000] Hz.

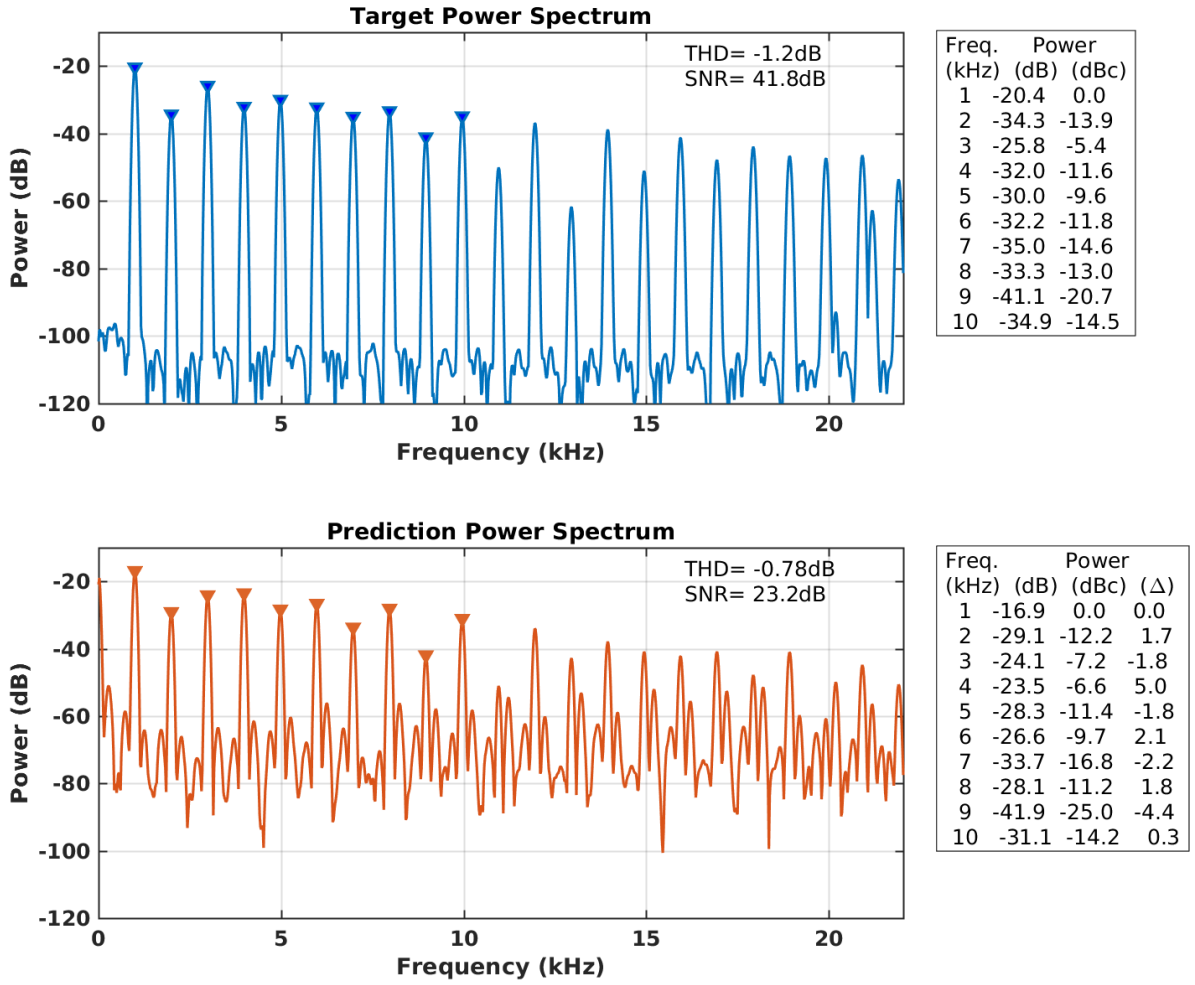


Figure 4.19 Power spectrum of the target compared with the power spectrum of the prediction for a frequency $f \simeq 1000\,\text{Hz}$ in the ESS (model 2). $\Delta = \text{dB}c_{\text{prediction}} - \text{dB}c_{\text{target}}$.

4.2.3 Model 3: sequence-to-sequence prediction with one LSTM layer

Presentation of model 3

In model 1, a sequence of N input samples $\mathbf{x} = \{x[n-N+1], \cdots, x[n]\}$ is used to compute the output value $\hat{y}[n]$. Since the LSTM cells share the same weights and biases (as presented in Sec.4.2.1), it would make sense to use each state vector $\mathbf{h}[\cdot]$ (and not just the last one) to compute a sequence of output values $\hat{y} = \{\hat{y}[n-N+1], \ldots, y[n]\}$ as presented in Fig. 4.20.

The Tensorflow code used to build the graph of this neural network model can be found in Appendix A.5.4 and the entire code can be downloaded in [Schmitz, 2019b].

At first glance, it seems that the sequence-to-sequence prediction should be N times faster than the sequence-to-sample model (i.e., model 1). Indeed, the time to compute a minibatch of size $N_{miniBatch}$ is given by:

- **For a sequence-to-sample model** : the time t_1 to compute one instance multiplied by the number of instances in the minibatch:

$$CT = t_1 * N_{miniBatch}. \tag{4.7}$$

- **For a sequence-to-sequence model** : if the number of outputs are noted N_{out} then there is N_{out} output samples computed for each instance. Therefore, the number of instances required to compute the whole minibatch is equal to $N_{miniBatch}/N_{\text{out}}$ (if the minibatch size is an integer multiple of the output size). Finally, the time to compute all the instances of the minibatch is :

$$CT = t_1.N_{miniBatch}/N_{\text{out}} \quad \text{with } N_{\text{out}} \in [1,N]. \tag{4.8}$$

However, this result is not straightforward since during inference (i.e., when the model is used for emulation after the training phase) all samples in the buffer are treated in parallel such that if the GPU has enough parallel cores, processing one sample or 400 may take approximately the same amount of computational time, even though, the sequence-to-sequence model has N times less FLOPS than the sequence-to-sample model. One can notice that if just one CPU is used, the sequence-to-sequence model is more interesting.

There is another reason that makes the sequence-to-sequence model less interesting when using a GPU. Indeed, in an unrolled LSTM neural network, the state vector entering in the first cell (i.e., $\mathbf{h}[n-N]$) is set to 0. Consequently, the estimation of $\hat{y}[n-N+1]$ will be wrong since the incoming vector $\mathbf{h}[\cdot]$ is 0 only for this cell (the other cells receive a non null entering state $\mathbf{h}[\cdot]$). One possible way to circumvent this problem is to store the state $\mathbf{h}[n-N+1]$ when computing the instance $\mathbf{x} = \{x[n-N+1],\cdots,x[n]\}$ and to use this state vector as the incoming state vector of the first LSTM cell when computing the next instance $\mathbf{x} = \{x[n-N+2],\cdots,x[n+1]\}$. This method will break the parallelization on the minibatch since the input state vector to compute an instance depends on a state vector computed in a previous instance, making this process a serial process.

However, this method is still interesting for CPU use. Moreover, computing the gradient for a sequence can be more robust than computing the gradient for a unique output sample. This method can be seen as a regularization technique: each cell $k \in [n-N+1, n]$ can be represented by a function $f(\mathbf{W}, \mathbf{b}, x[k])$ where all cells share the same weights $\mathbf{W}$ and biases $\mathbf{b}$. Consequently, there are more constraints on the weights and biases if each cell has to fit its own target (sequence-to-sequence model) than if only the last cell has to fit its target (sequence-to-sample model). Therefore, it can be interesting to predict a sequence of a few output samples during the training phase to have a better constraint on the weights and biases of the LSTM layer, even if only the last output sample $\hat{y}[n]$ will be used during the inference. In this case, the next output sequence $\{\hat{y}[n-N_{\text{out}}+2], \cdots, \hat{y}[n+1]\}$ is computed using the input sequence $\{x[n-N], \cdots, x[n+1]\}$ to give the next output value $\hat{y}[n+1]$ and so on and so fourth. In comparison, if the N_{out} values $\{\hat{y}[n-N_{\text{out}}+1], \cdots, \hat{y}[n]\}$ are computed for the input instance $\{x[n-N+1], \cdots, x[n]\}$, the next instance is composed by the output values $\{\hat{y}[n+1], \cdots, \hat{y}[n+N_{\text{out}}]\}$ computed using the input sequence $\{x[n-N+N_{\text{out}}+1], \cdots, x[n+N_{\text{out}}]\}$.

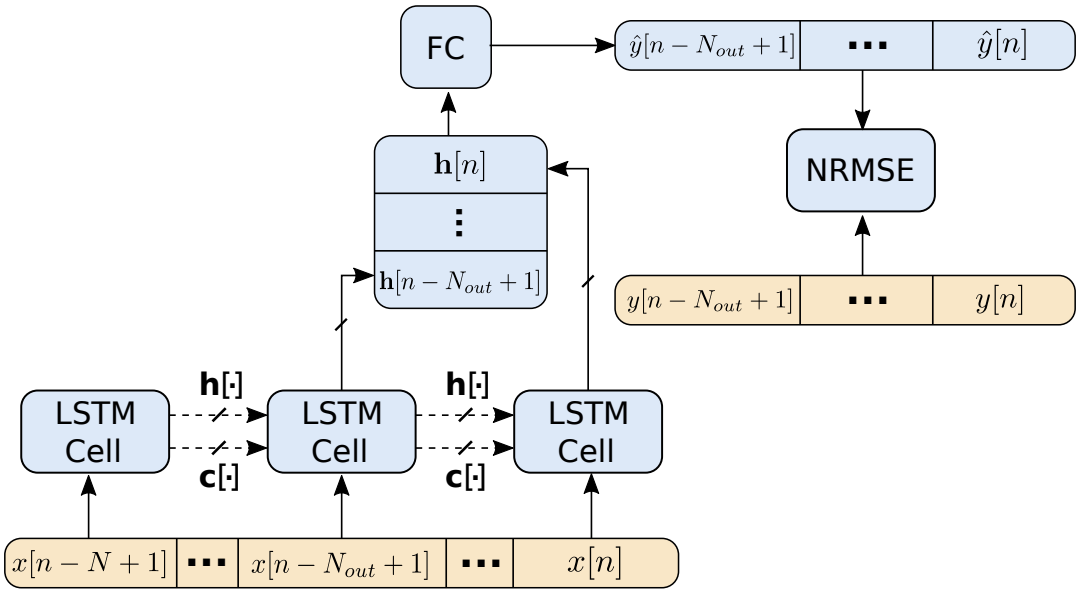


Figure 4.20 Model 3: using the N_{out} outputs of the LSTM layer to predict a sequence.

Discussion and results

As it can be seen in Table 4.4, the NRMSE, the CT and the harmonic content accuracy are slightly better when using a sequence-to-sequence model with an output sequence length = 5 rather than when using a sequence-to-sample model (output sequence length = 1). But, by continuing to increase the output sequence length (N_{out}), the NRMSE can increase again (the

model is not flexible enough to predict a long sequence). The CT can also become worse if the output sequence is too long. This can be explained since the gain in number of operations is counterbalanced by the increase of output state data ($N_{\text{out}} \times N_{\text{h}}$ elements). Consequently, the terms t_1 from Eqs. (4.7),(4.8) are not equal anymore. This increases the load of post processing (i.e., processing after the LSTM layer). More informations on this post processing can be found in the code of Appendix A.5.4 .

Table 4.4 Model 3: normalized root mean square error, computational time and harmonic content accuracy for model 3 on the *Engl Disto* (parameters: N=100, $N_{\text{h}} = 150$, gain of the amplifier = 5).

Model	N_{out} (samples)	CT(ms)	NRMSE(%)	Δ_{harmo}(dB)
1	1	7.1	39%	5.3
3	5	6.4	37%	3.2
3	10	7.1	39%	(not computed)
3	100	12	60%	(not computed)

To illustrate the accuracy of the model, we compare in Fig. 4.21 the signal at the output of the *Engl Retro Tubes 50* (the tube amplifier) and its prediction by the neural network model. Moreover, Fig. 4.22 presents the comparison between the spectrogram of the target (Fig. 4.22a) and the spectrogram of the prediction (Fig. 4.22b) when the input signal is an ESS with a bandwidth comprised between 50 and 5000 Hz. Clearly, these figures show that this model generates a lot of noise. Fig. 4.23 presents the comparison between the power spectrum of the target and the prediction when the input frequency is close to $f \simeq 1000$ Hz in the ESS. The SNR measured at 1000 Hz is worse than the one obtained for the model 1 ($SNR_{model3} = 11.5$ dB $< SNR_{model1} = 15.5$ dB). There is no overestimated harmonics while the harmonics number 4, 8, 10 are underestimated. The THD measured at 1000 Hz shows also that the model tends to underestimate the harmonic content ($THD_{\text{target}} = -1.2$ dB $< THD_{\text{prediction}} = -3$ dB).

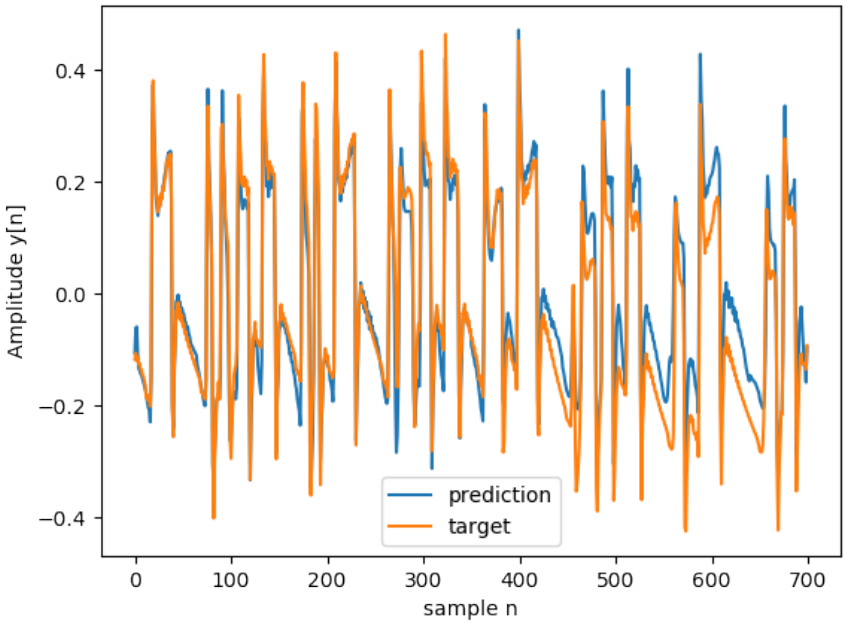
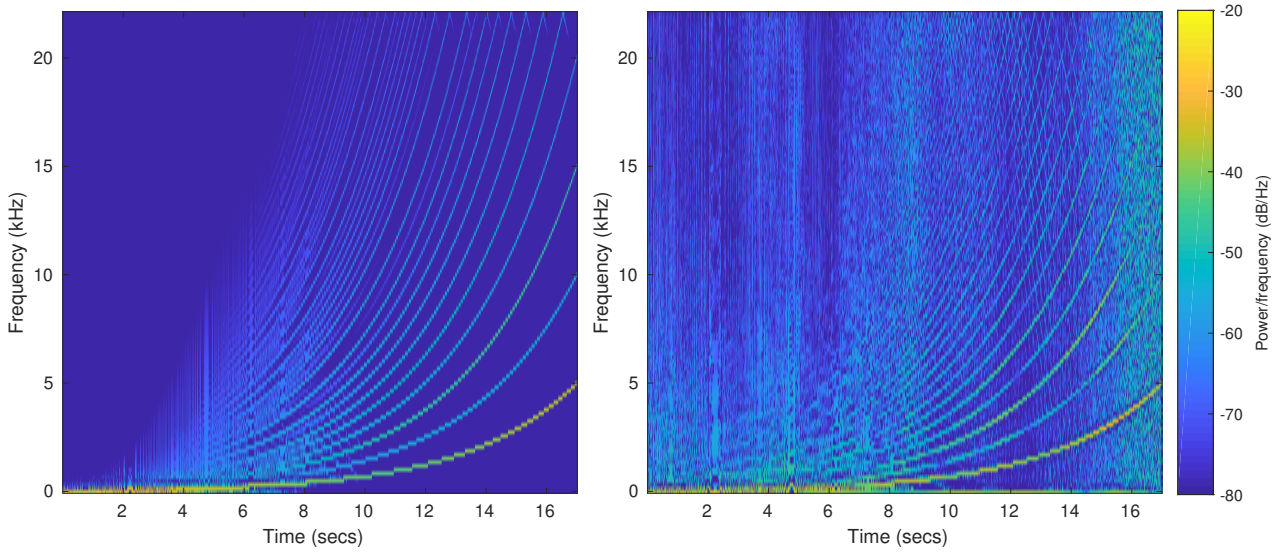


Figure 4.21 Comparison in the time domain of the output of the amplifier (*target*) and the output of the neural network (model 3) with an output sequence length = 5 (*prediction*), when a guitar signal is provided at the input.



(a) Target Engl retro tube 50 gain 5.

(b) Prediction model 3.

Figure 4.22 Comparison of the spectrograms of the target(a) and the prediction(b) for model 3 when the input is an ESS in the frequency range [50-5000] Hz.

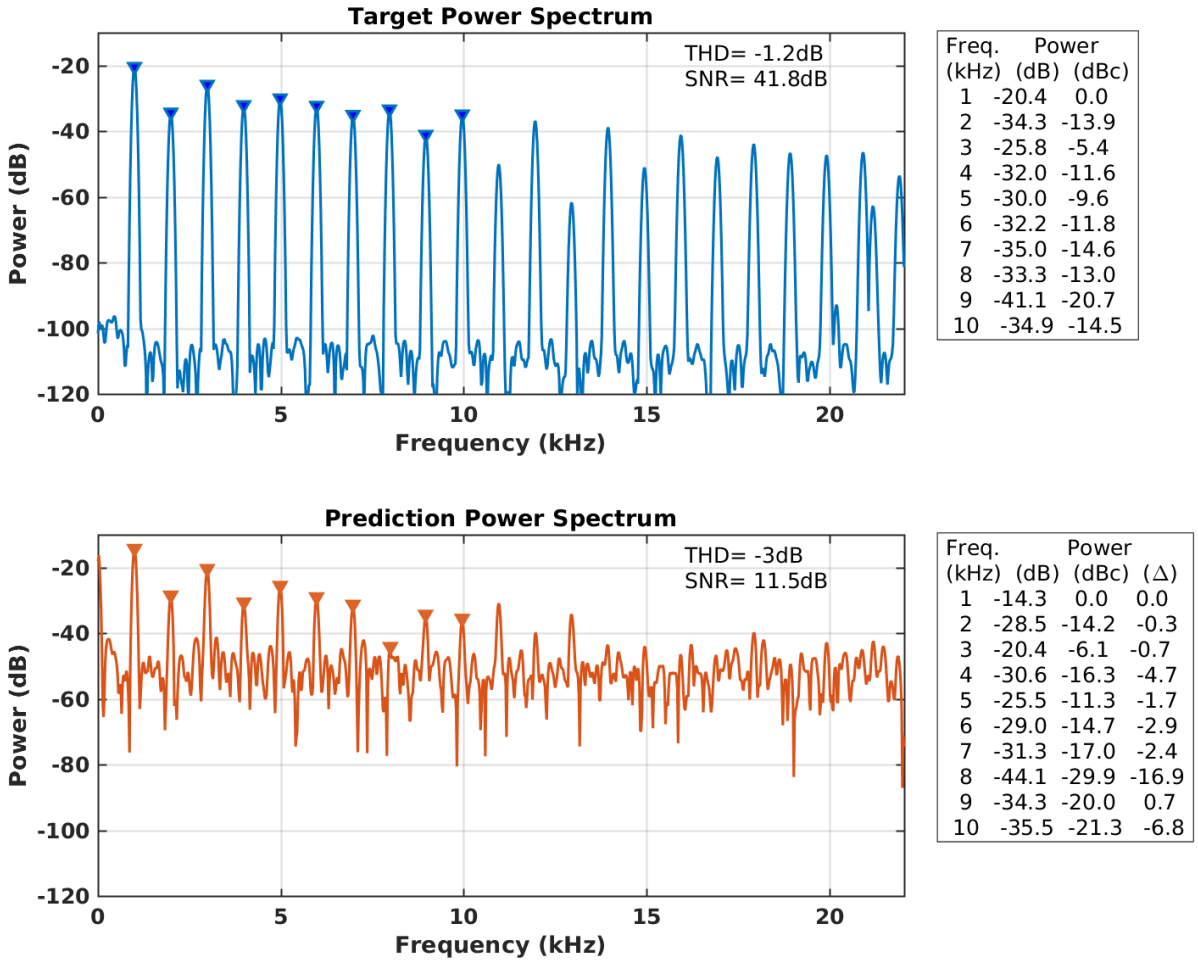


Figure 4.23 Power spectrum of the target compared with the power spectrum of the prediction for a frequency $f \simeq 1000\,\text{Hz}$ in the ESS (model 3). $\Delta = \text{dB}c_{\text{prediction}} - \text{dB}c_{\text{target}}$.

4.2.4 Model 4: taking the parameters of the amplifier into account in LSTM cells

In the previous models, the parameters of the amplifier are fixed (i.e., all in their middle position). Indeed, the target is measured for a specific set of the amplifier parameters. Then the model is trained for this dataset. If the user wants to emulate the amplifier with another set of parameters, he has to load another model. Usually, tube amplifiers have at least four parameters:

- The *Gain* which controls the amount of distortion.

- The *Bass* which controls the low-frequency response.
- The *Mid* which controls the medium-frequency response.
- The *Treble* which controls the high-frequency response.

These parameters are controlled by potentiometers, usually graduated in ten steps. This gives 10^4 possible combinations for a simple amplifier. Having numerous models is not convenient for two reasons:

- Training many models is time consuming.
- Storing many models with their weights, biases and graphs consumes data storage.

Moreover, there is no easy interpolation between stored models. For example, it could be convenient to compute the model of an amplifier whose gain is 1.5 when available measurements are for gain 1 and 2 only. For these reasons, a model that directly takes the parameters of the amplifier into account is developed in this section.

Presentation of model 4

As specified in Sec. 4.2.1 the input $\mathbf{z}[n]$ of each LSTM cell can be a vector of N_f *features*. Until now, only one feature was used (i.e., the current input guitar sample $x[n]$). To take into account the parameters of the amplifier, the values of the parameters can be added as input features. For example, to take the amplifier gain parameter into account, the entry of each LSTM cell becomes $\mathbf{z}[n] = [x[n], gain[n]]$ as presented in Fig. 4.24. The Tensorflow code used to build the graph of this neural network model can be found in Appendix A.5.5 and the entire code can be downloaded in [Schmitz, 2019b].

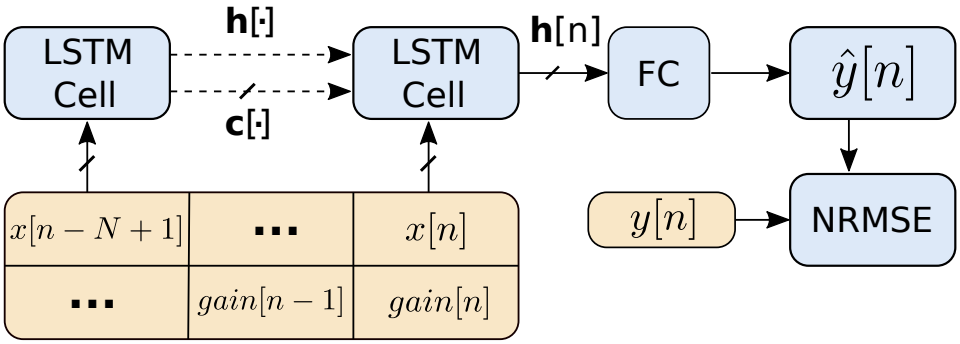


Figure 4.24 Model 4: Using two input features to take the parameter *gain* of the amplifier into account.

Parameters of the model

The hyperparameters are the same than those used in model 1, i.e., a number of hidden units $N_\text{h} = 150$ and a number of unrolled time steps $N = 100$.

Discussion and results

For this model, the targets of the neural network are the output samples coming from the amplifier when its gain is set to 1 or 5. The other gain levels are not present for practical reasons, i.e., to be able to fit the entire dataset in the Random Access Memory (RAM) without having to change the code to take into account the split dataset. Since the gain five was chosen as default value for the previous models, it is also chosen for this model in order to ease the comparison. The gain 1 was chosen since the sound produced by the amplifier for this gain level is very different than the sound produced when the gain is five. Moreover, since the amplifier produces less distortions at low gain level, its emulation should be eased (by contrast, the choice of gain 5 and 10 should be more difficult to emulate).

It is important to mix the output samples of gain 5 with those of gain 1 so that a minibatch gathers approximately 50% of instances of gain 1 and 50% of instances of gain 5 at each iteration (i.e., for each computation of the gradient). If the datasets were simply concatenated $[data_{gain1}, data_{gain5}]$, the training algorithm might optimize the weights and biases for instances of gain 1 on the first half of the epoch then for instances of gain 5 on the second half but may not converge to both of them. Mixing them ensures that at each iteration the gradient is computed using both instances of gain 1 and gain 5. To do so, the dataset is reshaped as usual (see Fig. 4.9) except that the input is a tuple ($x[n]$,$gain[n]$) and not a single value as before (see Fig.4.24). Then the different instances of the dataset are randomly shuffled.

Table. 4.5 shows that the NRMSE for this model is similar when it is used for gain 1 and gain 5. The model performs a little bit less than its equivalent for a fixed gain. However, the technique is promising in order to take the parameters of the amplifier into account. Some might think that the model would perform better for gain 1 since the model is more linear. However, the energy of the target for gain 1 is lower than the energy of the target for gain 5 (as explained below). Consequently, the predicted signal obtained with gain 5 has a stronger impact on the minibatch MSE than the signal with gain 1. This difference of energy can be explained by the increasing saturation effect of the target signal when the gain is increased. Indeed, the saturation effect makes the output signal look like to a square signal. As presented in Fig. 4.25, the signal of gain 5 has more energy than the signal of gain 1 even if their amplitudes are normalized to the same level Vcc.

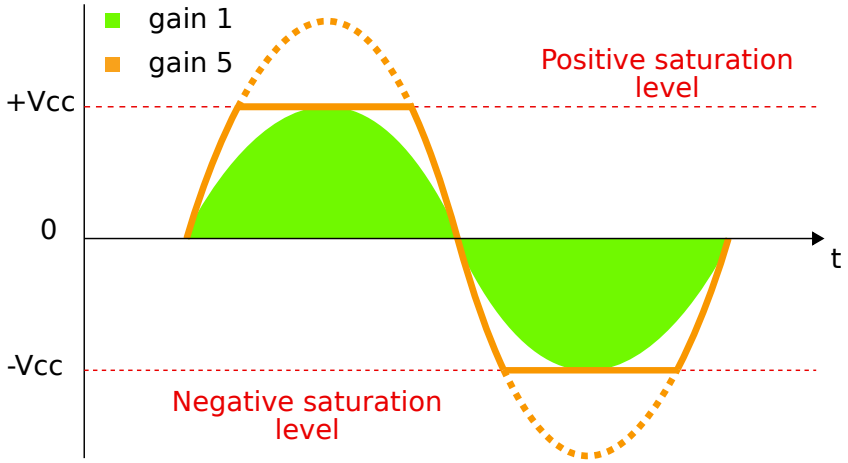


Figure 4.25 Saturation effect when increasing the gain of the amplifier.

Table 4.5 Model 4: normalized root mean square error, computational time and harmonic content accuracy for a model that takes the gain of the amplifier into account (parameters: $N = 100$, $N_h = 150$, gain=1&5).

Amplifier	channel	Gain	CT(ms)	NRMSE(%)	Δ_{harmo}(dB)
Engl	Disto	1	6.5	43%	/
Engl	Disto	5	6.5	42%	7.2

To illustrate the accuracy of the model, we compare in Fig. 4.26 the signal at the output of the *Engl Retro Tubes 50* (the tube amplifier) and its prediction by the neural network model for the gain 5. Moreover, Fig. 4.27 presents the comparison between the spectrogram of the target (Fig. 4.27a) and the spectrogram of the prediction (Fig. 4.27b) when the input signal is an ESS with a bandwidth comprised between 50 and 5000 Hz. The Fig. 4.28 presents the comparison between the power spectrum of the target and the prediction when the input frequency is close to $f \simeq 1000$ Hz in the ESS. The SNR measured at 1000 Hz is better than the one obtained for the model 1 ($SNR_{model4} = 16.4$ dB $> SNR_{model1} = 15.5$ dB). The odd harmonics are overestimated while the harmonics number 4, 6, 8 are underestimated. The THD measured at 1000 Hz shows that the model tends to overestimate the harmonic content ($THD_{target} = -1.2$ dB $< THD_{prediction} = +2$ dB).

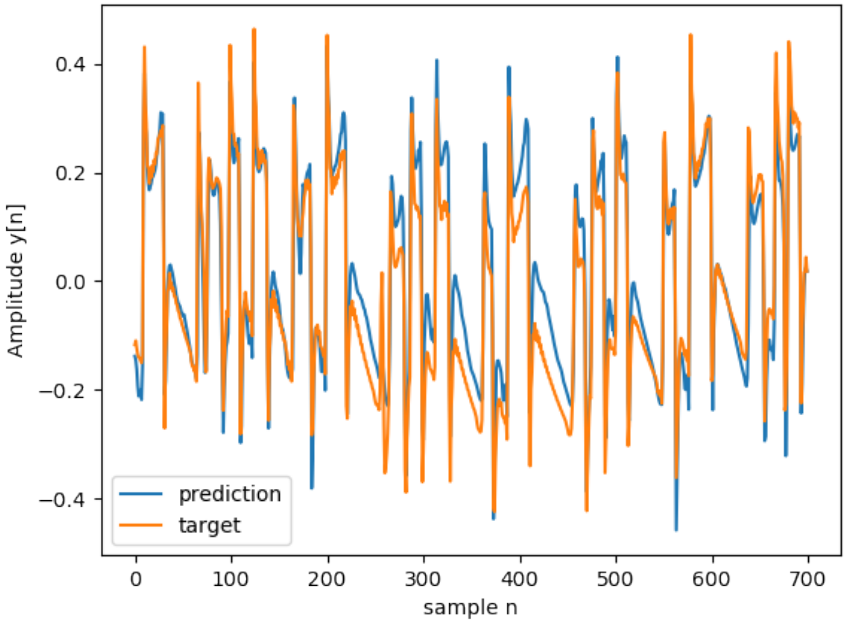
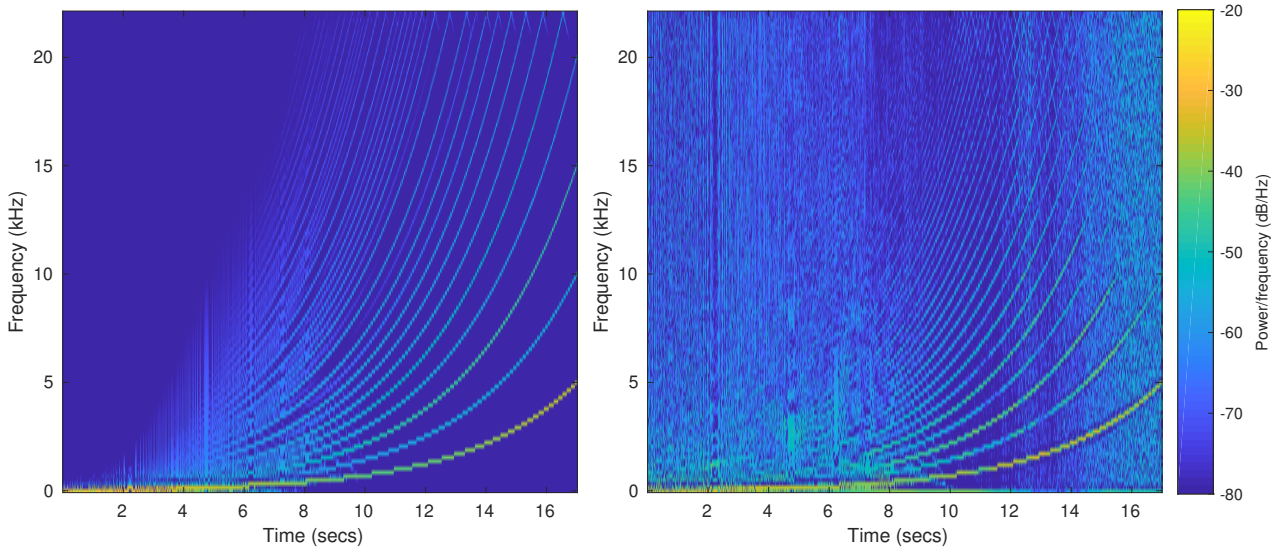


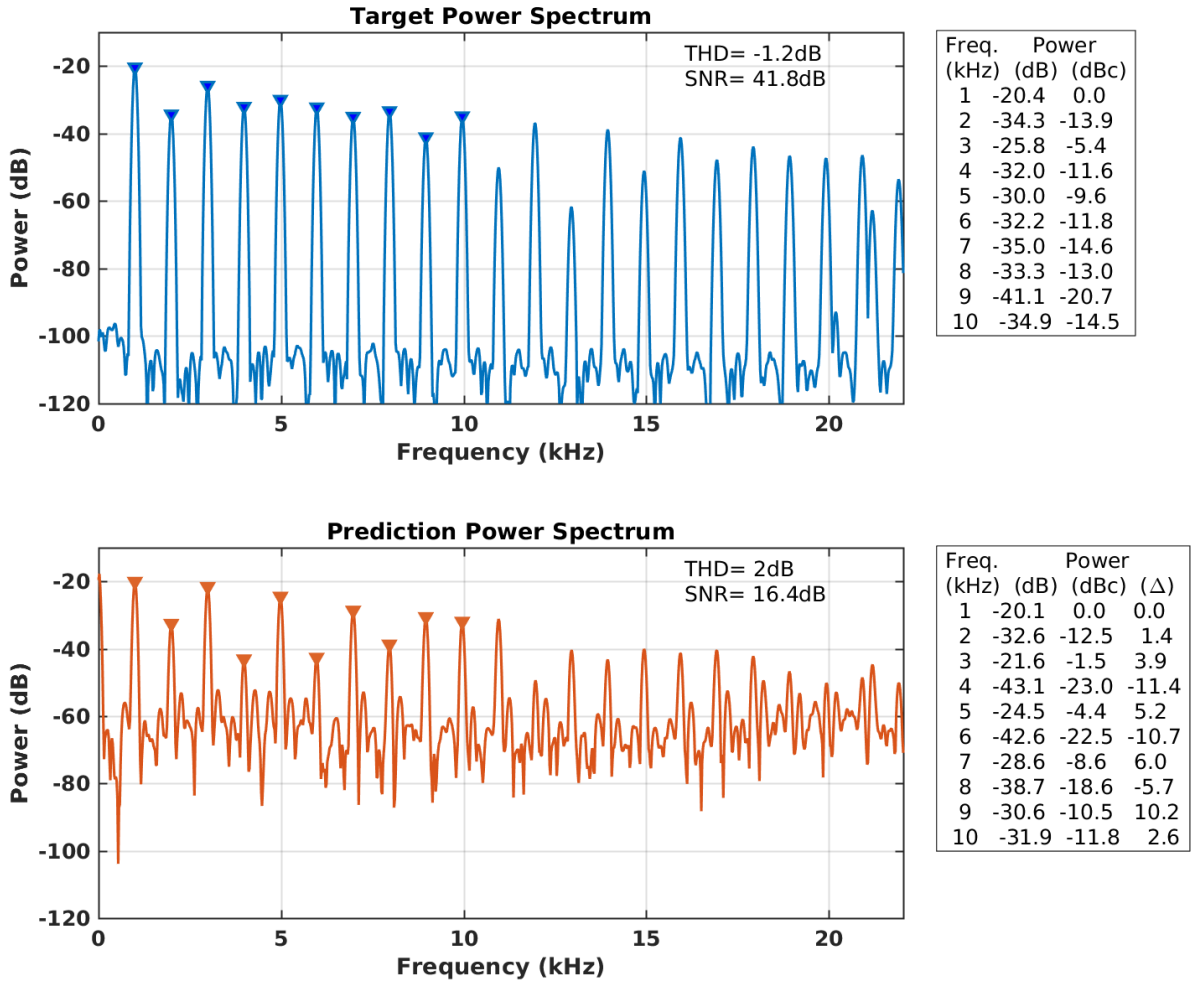
Figure 4.26 Comparison in the time domain of the output of the amplifier (*target*) and the output of the neural network model 4 (*prediction*) when a guitar signal is provided at the input.



(a) Target Engl retro tube 50 gain 5.

(b) Prediction model 4.

Figure 4.27 Comparison of the spectrograms of the target(a) and the prediction(b) for model 4 when the input is an ESS in the frequency range [50-5000] Hz.



Freq. (kHz)	Power (dB)	(dBc)
1	-20.4	0.0
2	-34.3	-13.9
3	-25.8	-5.4
4	-32.0	-11.6
5	-30.0	-9.6
6	-32.2	-11.8
7	-35.0	-14.6
8	-33.3	-13.0
9	-41.1	-20.7
10	-34.9	-14.5

Freq. (kHz)	Power (dB)	(dBc)	(Δ)
1	-20.1	0.0	0.0
2	-32.6	-12.5	1.4
3	-21.6	-1.5	3.9
4	-43.1	-23.0	-11.4
5	-24.5	-4.4	5.2
6	-42.6	-22.5	-10.7
7	-28.6	-8.6	6.0
8	-38.7	-18.6	-5.7
9	-30.6	-10.5	10.2
10	-31.9	-11.8	2.6

Figure 4.28 Power spectrum of the target compared with the power spectrum of the prediction for a frequency $f \simeq 1000\,\text{Hz}$ in the ESS (model 4). $\Delta = \text{dB}c_{\text{prediction}} - \text{dB}c_{\text{target}}$.

4.2.5 Model 5: a faster LSTM model using convolutional input

Presentation of model 5

The main problem when dealing with recurrent neural networks is the limited use it makes of the GPU capabilities. Indeed, in a recurrent neural network, each cell has to wait for the state vectors of the previous cell as explained in Eq. 2.56. Consequently, the states cannot be computed in parallel, loosing the advantage of using a GPU.

The speed of the model can be increased by reducing the number of time steps N as it can be seen in Table 4.6. The gain in CT when reducing the number of time steps is less than linear. Indeed, the computations in the LSTM layer is not the only source of CT consumption since the model also has to reshape the input/output data and transfers them from the sound card to the GPU. If the CT of the LSTM layer is decreased, it progressively becomes similar to the CT of the other operations.

Our goal is to reduce the number of LSTM cells from $N = 100$ to $M \in [10, 20]$ cells. However, reducing the number of time steps also reduces the accuracy of the model since the model has less information from the past. One idea to circumvent this problem is to use the same number of time steps for the input data but to reduce this number before the treatment by the LSTM layer. A Convolutional Neural Network (CNN) is chosen to reduce the number of time steps. This model, called the Long Short Term Memory with Convolutional input (*LSTMConv*) model is presented in Fig. 4.29. The Tensorflow code used to build the graph of this neural network model can be found in Appendix A.5.6 and the entire code can be downloaded in [Schmitz, 2019b].

Table 4.6 Model 5: Computational time when reducing the number of time steps N on the *Engl Disto* amplifier (parameters: N, $N_\mathrm{h} = 150$).

	N=100	N=50	N=20	N=10	N=5
CT(ms)	7.1	5.1	3.8	3.3	3

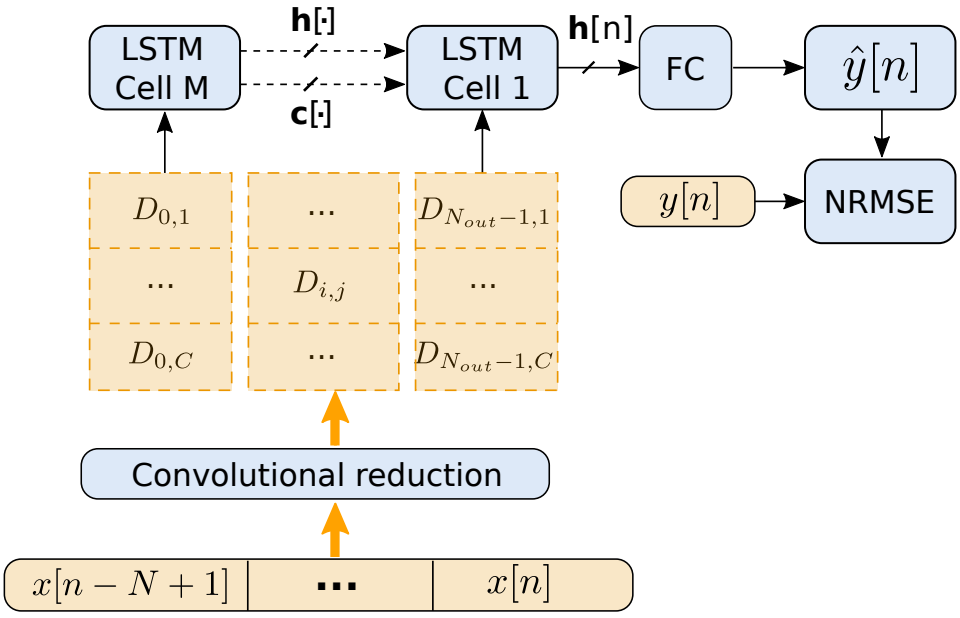


Figure 4.29 Model 5: LSTMConv.

Convolutional reduction layer

The objective of this layer is to reduce the number of LSTM *Cells* without reducing the number of input time steps N. Until now, the number of LSTM cells M was equal to N. A layer projecting the N input values into the M LSTM cells is required (with $M < N$). This layer will change the representation of the input signal, reducing its length while attempting to preserve the global information it contains (see Fig. 4.29).

A CNN is used to reduce the length of the input signal: a kernel $\mathbf{k}$ of length L is used to compute a representative version of the input $\mathbf{x}$ (size of $\mathbf{x} = N_{\text{in}}$) such as the output $\mathbf{d}$ of a Convolutional Reduction (CR) layer (size of $\mathbf{d} = N_{\text{out}}$) is given by:

$$\begin{cases} \mathbf{d}[i] = \kappa\left(\sum_{u=0}^{L-1} x[u+i*S].k[u] + b\right), \\ \forall i \in [0, N_{\text{out}} - 1], \end{cases} \tag{4.9}$$

where:

- S is called the *stride size* and allows to skip input samples. The output size of the CR layer is given by $N_{\text{out}} = \frac{N_{\text{in}}}{S}$. In fact, if N_{in} is not an integer multiple of S, the output size is given by its upper bound $N_{\text{out}} = ceil\left(\frac{N_{\text{in}}}{S}\right)$ and the signal x is (left and right) zero padded in order to obtain this output size.
- κ and b are respectively the activation function and the bias term as presented in Sec. 2.7.2

The kernel $\mathbf{k}$ and the bias b are the parameters of the neural network to optimize. Instead of using a unique kernel, it is possible to use C different kernels where C is also called the number of *maps*. The output of such convolutional layer can be written as a matrix $\mathbf{D}$:

$$\begin{cases} \mathbf{D}_{i,j} = \kappa\left(\sum_{u=0}^{L-1} x[u+i*S].k_{u,j} + b_j\right), \\ \forall i \in [0, N_{\text{out}} - 1] \text{ and } \forall j \in [1, C]. \end{cases} \tag{4.10}$$

$\mathbf{D}$ is a matrix constituted of C maps of length N_{out}. This matrix can be sent to N_{out} LSTM cells where each cell has C input features. In order to decrease the size further, $\mathbf{D}$ can be sent in another CR layer. The output of the l^{th} CR layer can be written as:

$$\begin{cases} \mathbf{D}^{(l)}_{i,j} = \kappa\left(\sum_{m=1}^{C_{l-1}} \sum_{u=0}^{L-1} \mathbf{D}^{(l-1)}_{u+i*S_l,m}.k^{(l)}_{u,j} + b^{(l)}_j\right), \\ \forall i \in [0, N^{(l)}_{\text{out}} - 1] \text{ and } \forall j \in [1, C_l], \end{cases} \tag{4.11}$$

where C_l is the number of maps of the l^{th} CR layer. The number of maps and the stride sizes S do not have to be the same in each CR layer. The output length of the l^{th} layer $N_{\text{out}}^{(l)}$ is given by :

$$N_{\text{out}}^{(l)} = \begin{cases} ceil\left(\dfrac{N_{\text{in}}}{S_l}\right) & \text{if } l = 1, \\ ceil\left(\dfrac{N_{\text{out}}^{(l-1)}}{S_l}\right) & \text{else.} \end{cases} \tag{4.12}$$

We now have a way to transform an input vector $\mathbf{x}$ of size $[N,1]$ into a matrix $\mathbf{D}$ of size $[M,C]$ with $M < N$ by adjusting the number of CR layers and the stride size S of each layer. Due to the parallelization of the operations in a *GPU*, computing the *cell_states* of the LSTM layer based on C channels instead of one does not raise the CT in a significant way.

Parameters of the model

The *LSTMConv* model requires the setting of a few hyperparameters:

- the number of input time steps N,
- the number of LSTM cells M,
- the stride size S in each CR layer,
- the number of maps C in each CR layer,
- the length L of the kernels in each CR layer.

In order to choose the hyperparameters, we have defined a performance score for the model depending on the inverse of the RMSE and the time per epoch (i.e., the time required to process all the training data once). The higher the score, the better:

$$perf\_score = \frac{1}{RMSE.time\_epoch}. \tag{4.13}$$

The objective of this score is to show which set of hyperparameters gives a good accuracy while maintaining a low CT. Some bounds on the hyperparameters have to be set to avoid trivial cases where the time is so small that it can counterbalance a very bad accuracy. Each point in Fig. 4.30 represents the score of the model 5 obtained on the *test set* after 3 hours of training. Each point has been obtained using a random combination of the following three hyperparameters : the length of the kernels L, the number of maps C (identical in the two *CR*

layers for this test) and the number of input time steps N. One can notice that the number of LSTM *Cells* is not represented. Indeed, as the CT strongly depends on the number of LSTM *Cells*, it has been fixed to the maximum time allowing a real-time emulation. Consequently, the stride size parameter S is chosen such that 15 LSTM *Cells* are used in each experiment (i.e., for each point of Fig. 4.30).

In Fig. 4.30, the red ellipse represents a selection of the best performance scores, i.e., small RMSE and small CT. The bottom figure of Fig. 4.30, is a projection of these scores on the (C,N) plane. It shows that most of the best performance scores are over the red line of 150 time steps. The horizontal line defines a limit on the number of channels, it helps to prevent the overfitting of the database by reducing the flexibility of the model. Finally, the *best* parameters' combination is situated around $[N, L, C] = [160, 12, 35]$.

Discussion and results

For this model, we have chosen to use two CR layers with the stride sizes $S_1 = 4$ and $S_2 = 3$. The choice of the stride sizes determines the number M of LSTM cells present in the model. On one hand, it impacts the resulting accuracy of the model and on the other hand, it impacts the CT since less LSTM cells reduce the number of sequential computations. The number of LSTM cells is then set by:

$$M = ceil\left(\frac{ceil\left(\frac{N}{S_1}\right)}{S_2}\right). \tag{4.14}$$

Depending on the ceiling operation, M will be equal to $\frac{N}{12}$ or to $\frac{N}{12}+1$. The model has been implemented according to the general structure shown in Fig. 4.29. Two *CR* layers have been used to reduce the number of input time steps from N=150 to M=13 LSTM *Cells*. Table 4.7 shows that the resulting model is 4% more accurate and 58% faster than the model 1.

Table 4.7 Model 5: normalized root mean square error, computational time and harmonic content accuracy for a LSTMConv model (parameters: N=150, $N_h = 150$).

Amplifier	channel	Gain	CT(ms)	NRMSE(%)	Δ_{harmo}(dB)
Engl	Disto	5	4.1	35%	5.6

To illustrate the accuracy of the model, we compare in Fig. 4.31 the signal at the output of the *Engl Retro Tubes 50* (the tube amplifier) and its prediction by the neural network model. Moreover, Fig. 4.32 presents the comparison between the spectrogram of the target (Fig. 4.32a) and the spectrogram of the prediction (Fig. 4.32b) when the input signal is an ESS with a

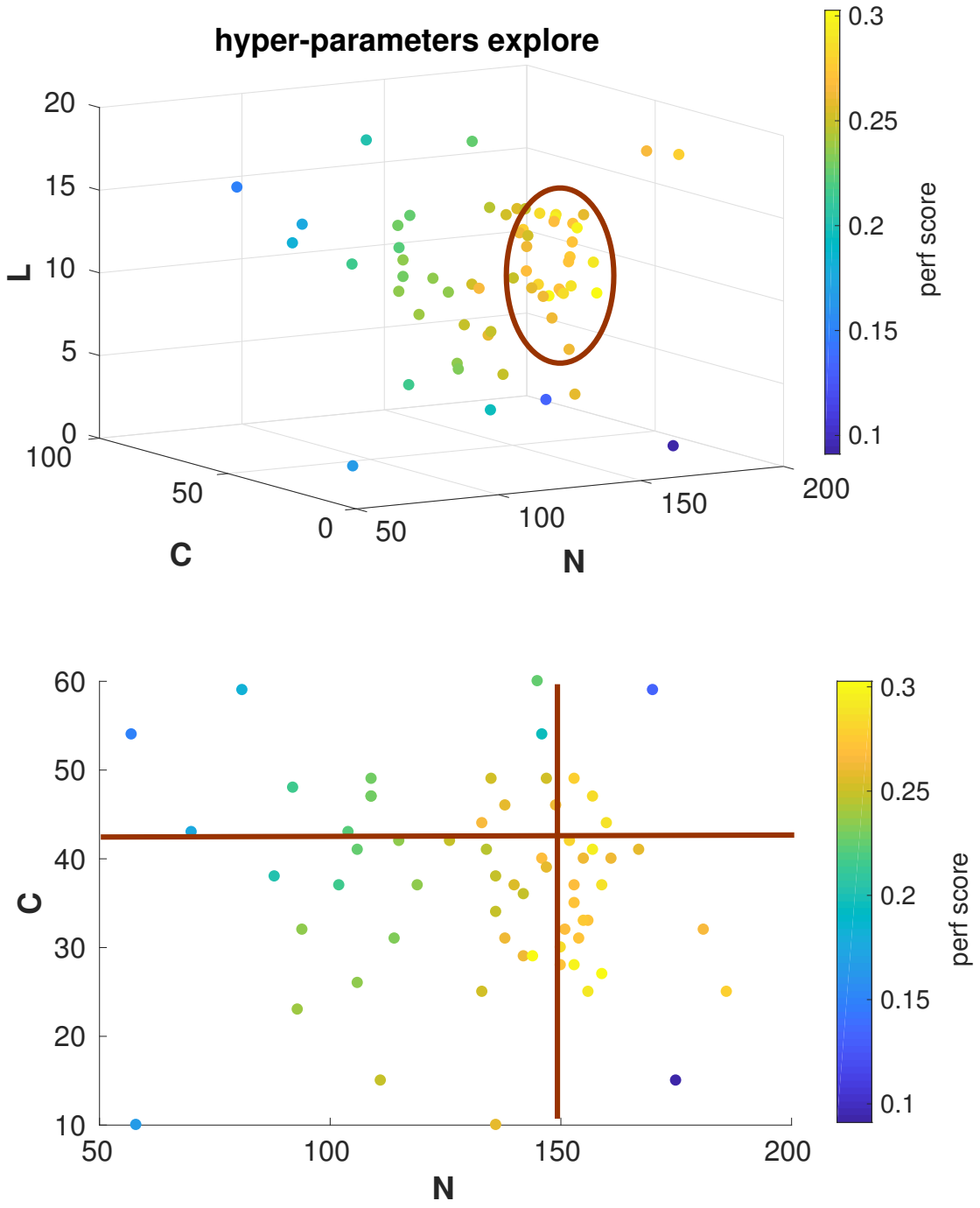


Figure 4.30 Performance scores of the model 5 according to the values of the hyperparameters : (*N*,*C*,*L*).

bandwidth comprised between 50 and 5000 Hz. The Fig. 4.33 presents the comparison between

the power spectrum of the target and the prediction when the input frequency is close to $f \simeq 1000\,\text{Hz}$ in the ESS:

- The even harmonics are underestimated (except for the harmonic number 4) while the odd harmonics are overestimated (except for the harmonic 3).
- The THD measured at 1000 Hz shows that the model tends to overestimate the harmonic content ($THD_{\text{target}} = -1.2\ \text{dB} < THD_{\text{prediction}} = 0.2\ \text{dB}$).
- The harmonic number 2 is very badly represented: this is a serious default since it is one of the major contribution to the output sound. 10 dB are missing for this harmonic while the harmonic number 9 has 8.1 dB too much. The balance between the even and odd harmonics is compromised. However, the listening tests still give good results as presented in Sec. 4.4.
- One can notice that, this spectrogram is one of the cleanest, its noise floor is lower than those of previous models even for high frequencies. Since high frequencies have never been trained for this model (guitar signal frequency range is [80-1500] Hz) it also means that this model generalizes particularly well to unseen data. The SNR measured at 1000 Hz measured at 1000 Hz is better than the one obtained for the other models ($SNR_{model5} = 21.4\ \text{dB}$).

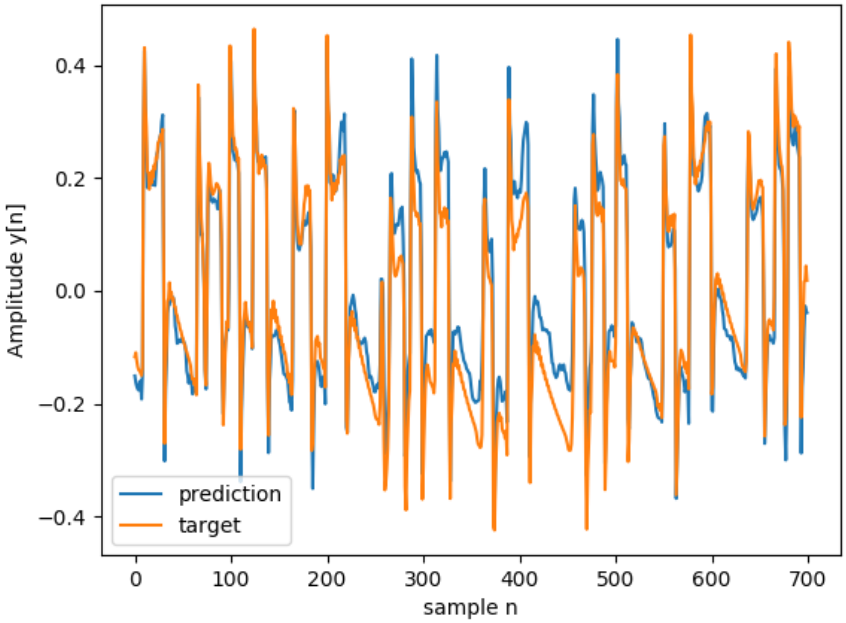
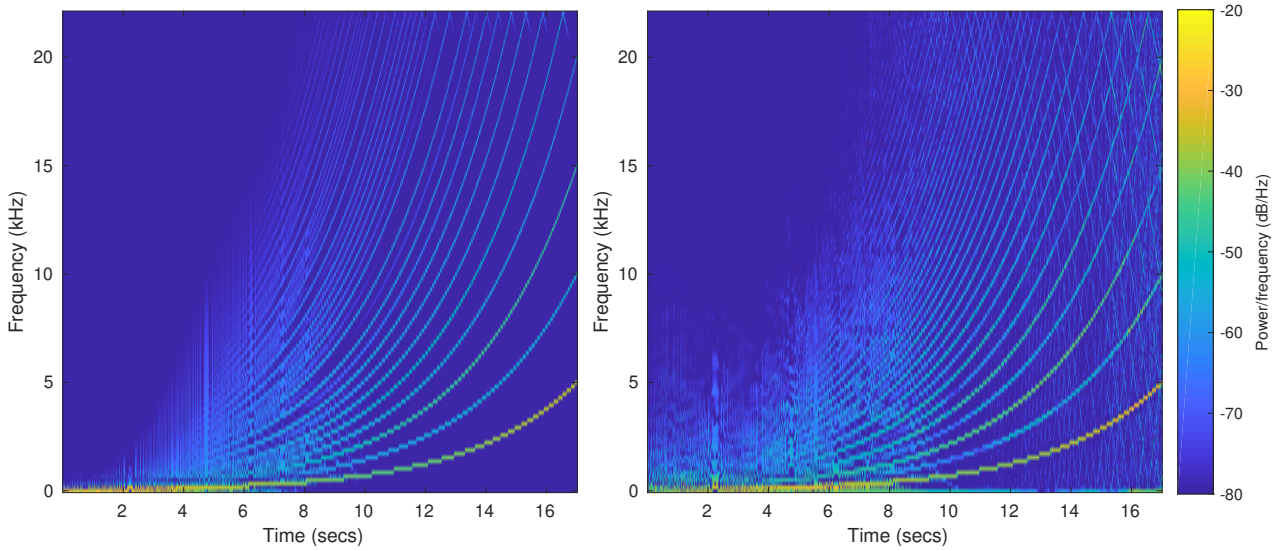


Figure 4.31 Comparison in the time domain of the output of the amplifier (*target*) and the output of the neural network model 5 (*prediction*) when a guitar signal is provided at the input.



(a) Target Engl retro tube 50 gain 5.

(b) Prediction model 5.

Figure 4.32 Comparison of the spectrograms of the target(a) and the prediction(b) for model 5 when the input is an ESS in the frequency range [50-5000] Hz.

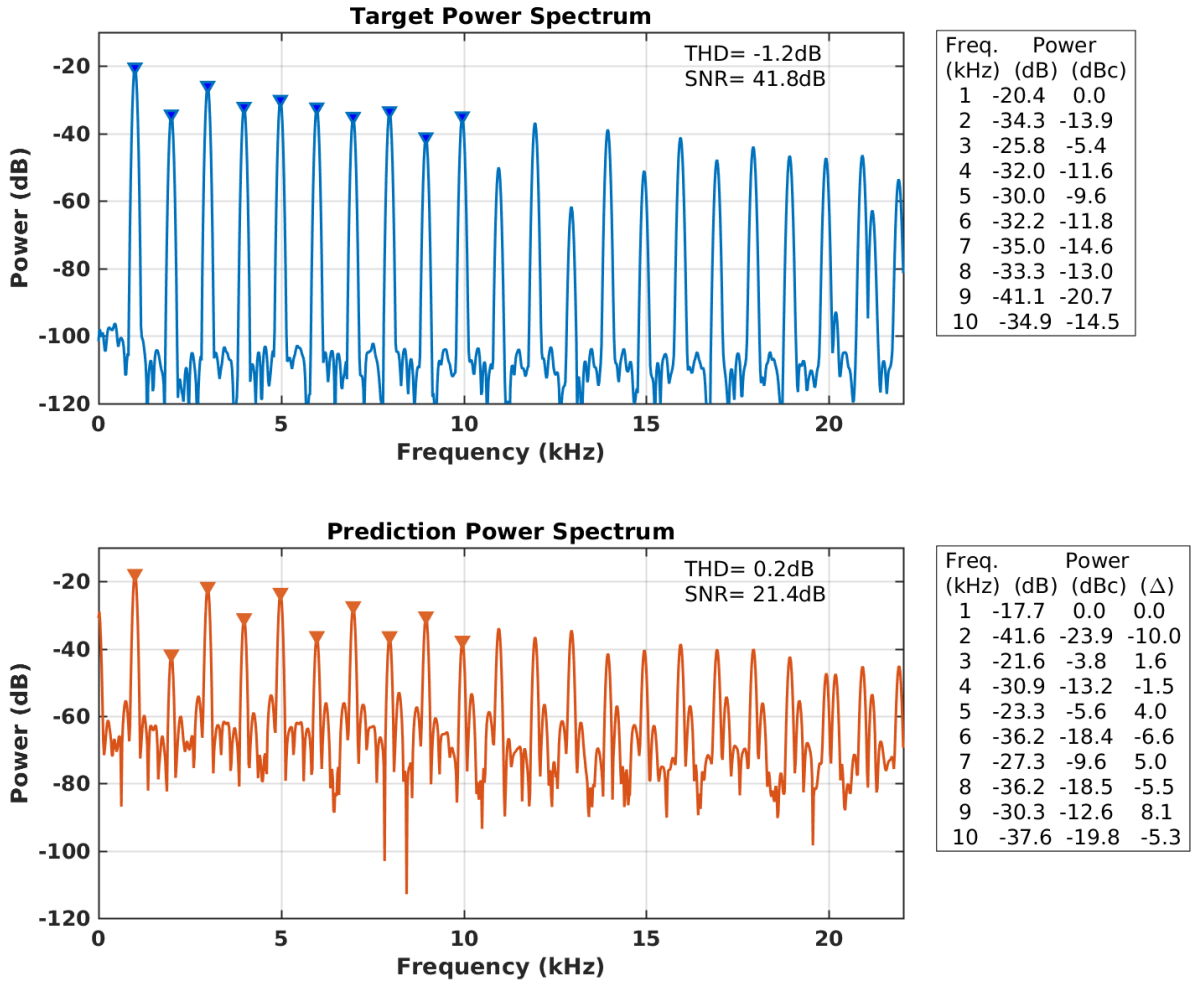


Figure 4.33 Power spectrum of the target compared with the power spectrum of the prediction for a frequency $f \simeq 1000\,\text{Hz}$ in the ESS (model 5). $\Delta = \text{dB}c_{\text{prediction}} - \text{dB}c_{\text{target}}$.

4.2.6 Model 6: feedforward neural network

Presentation of model 6

The model 6 (as presented in Fig. 4.34) is a multi-layer feedforward (i.e., no recurrent connexions) neural network composed of three fully connected layers as defined in Sec. 2.7.2. From now on, this model is called, the Deep Neural Network (DNN) model. The Tensorflow code used to build the graph of this neural network model can be found in Appendix A.5.7 and the entire code can be downloaded in [Schmitz, 2019b].

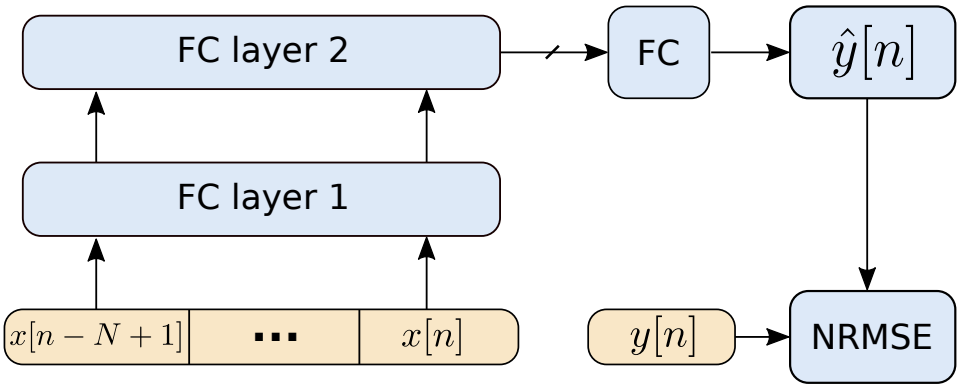


Figure 4.34 Model 6: DNN.

Parameters of the model

This model has few hyperparameters:

- the number of layers which has been fixed to 3,
- the layers 1 and 2 have respectively 1000 and 500 hidden units and use a *relu* activation function (see Sec. 2.7.1),
- the last fully connected layer maps the 500 hidden units from layer 2 to the output prediction sample $\hat{y}[n]$ using a hyperbolic tangent activation function,
- the input layer is composed of the $N = 150$ last input samples as described in model 5.

Discussion and results

The purpose of this model is to limit the number of sequential computations to obtain the shortest CT as possible. This is possible by removing the recurrent operations (i.e., the unrolled time steps) and by limiting the number of stacked layers. Indeed, since the output of the layer $l-1$ is the input of the layer l, the latter has to wait for the results of the previous layer before computing its own output. Table 4.8 presents the computational time to process a buffer of 400 samples and the NRMSE of the model 6 trained on the *Engl retro tube 50* channel *Disto*. As it can be seen, the accuracy of this model is lower than for the other models but in terms of computational time, it is the fastest.

To illustrate the accuracy of this model, we compare in Fig. 4.35 the signal at the output of the *Engl Retro Tubes 50* (the tube amplifier) and its prediction by the neural network model. Moreover, Fig. 4.36 presents the comparison between the spectrogram of the target (Fig. 4.36a) and the spectrogram of the prediction (Fig. 4.36b) when the input signal is an ESS with a

Table 4.8 Model 6: normalized root mean square error, computational time and harmonic content accuracy for a DNN model emulating the *Engl Disto* amplifier (parameters: N=150, layer 1 $N_\mathrm{h} = 1000$, layer 2 $N_\mathrm{h} = 500$, gain of the amplifier = 5).

Amplifier	channel	Gain	CT(ms)	NRMSE(%)	Δ_harmo(dB)
Engl	Disto	5	2.2	41%	6

bandwidth comprised between 50 and 5000 Hz. Noise is present in low frequencies (below t=5s which corresponds to frequencies f<200 Hz in the ESS input signal) and in high frequencies (above t=14s which corresponds to frequencies f>2000 Hz). Fig. 4.37 presents the comparison between the power spectrum of the target and the prediction when the input frequency is close to $f \simeq 1000$ Hz in the ESS:

- The SNR measured at 1000 Hz is slightly better than the one obtained for the model 1 ($SNR_{model6} = 16$ dB $> SNR_{model1} = 15.5$ dB).
- The harmonics number 2, 3, 7 are overestimated while the harmonics number 4, 5, 8 are underestimated. The THD measured at 1000 Hz shows that the model tends to overestimate the harmonic content ($THD_\mathrm{target} = -1.2$ dB $< THD_\mathrm{prediction} = +4.2$ dB).
- The harmonics number 2 is even louder than the fundamental frequency.

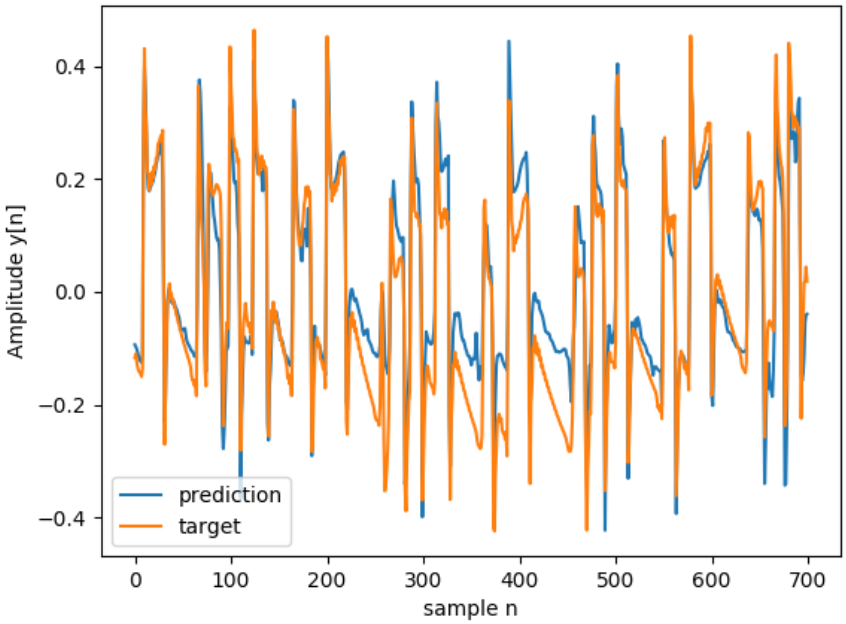
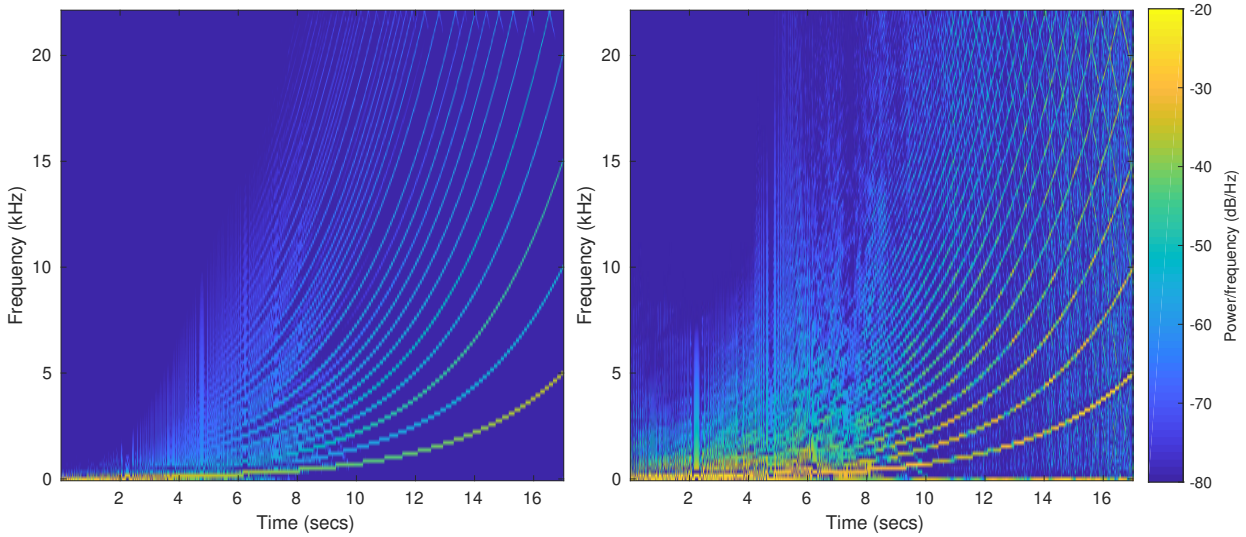


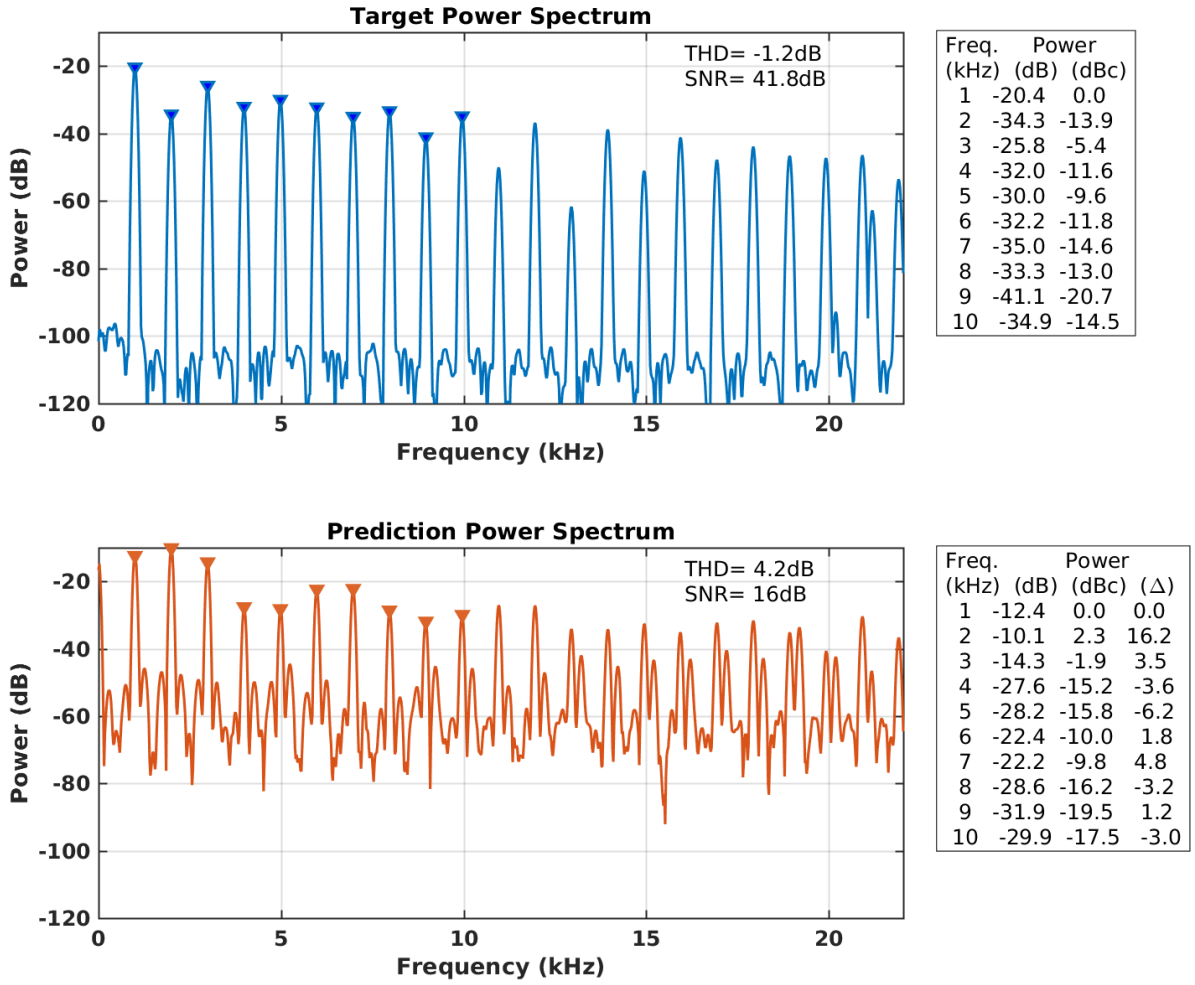
Figure 4.35 Comparison in the time domain of the output of the amplifier (*target*) and the output of the neural network model 6 (*prediction*) when a guitar signal is provided at the input.



(a) Target Engl retro tube 50 gain 5.

(b) Prediction model 6.

Figure 4.36 Comparison of the spectrograms of the target(a) and the prediction(b) for model 6 when the input is an ESS in the frequency range [50-5000] Hz.



Freq. (kHz)	Power (dB)	(dBc)
1	-20.4	0.0
2	-34.3	-13.9
3	-25.8	-5.4
4	-32.0	-11.6
5	-30.0	-9.6
6	-32.2	-11.8
7	-35.0	-14.6
8	-33.3	-13.0
9	-41.1	-20.7
10	-34.9	-14.5

Freq. (kHz)	Power (dB)	(dBc)	(Δ)
1	-12.4	0.0	0.0
2	-10.1	2.3	16.2
3	-14.3	-1.9	3.5
4	-27.6	-15.2	-3.6
5	-28.2	-15.8	-6.2
6	-22.4	-10.0	1.8
7	-22.2	-9.8	4.8
8	-28.6	-16.2	-3.2
9	-31.9	-19.5	1.2
10	-29.9	-17.5	-3.0

Figure 4.37 Power spectrum of the target compared with the power spectrum of the prediction for a frequency $f \simeq 1000\,\text{Hz}$ in the ESS (model 6). $\Delta = \text{dB}c_{\text{prediction}} - \text{dB}c_{\text{target}}$.

4.2.7 Model 7: convolutional neural network

Presentation of model 7

The Convolutional Neural Networks (CNN) have already been used in model 5 to transform the N input samples into a matrix of size $M \times C$ where M is the new number of time steps and C is the number of *maps*. A classic CNN (as presented in Fig. 4.38) is composed by convolutional layers (as introduced in Eq. (4.10)) combined with *pooling layers*.

The goal of a *pooling layer* is to subsample the output of the convolutional layer in order to reduce its size before the next convolutional layer. It helps to reduce the computational load, the memory usage and the risk of overfitting. The sampling function is free of choice but is often chosen as the *max* function over a small area called the *receptive field* or the *kernel length*. As in the convolutional layer, a stride size and a length for the receptive field have to be chosen. The output $\mathbf{D}_{i,j}^{(l)}$ of a pooling layer l is given in function of the output of the previous output layer $\mathbf{D}_{i,j}^{(l-1)}$ such that:

$$\begin{cases} \mathbf{D}_{i,j}^{(l)} = \max\left(\mathbf{D}_{i*S_l+[1:L_l],j}^{(l-1)}\right), \\ \forall i \in [0, N_{\text{out}}^{(l)} - 1] \text{ and } \forall j \in [1, C_{l-1}], \end{cases} \tag{4.15}$$

where:

- $C_{(l-1)}$ is the number of maps of the previous layer.
- L_l is the length of the receptive field of the layer l.
- S_l is the stride size of the layer l as defined in Sec. 4.2.5.
- $N_{\text{out}}^{(l)}$ is the output size of the layer l, it can be computed by using Eq. (4.12).

The Tensorflow code used to build the graph of this neural network model can be found in Appendix A.5.8 and the entire code can be downloaded in [Schmitz, 2019b].

Parameters of the model

The hyperparameters of the CNN model are given in Table 4.9 .

Table 4.9 Model 7: hyperparameters of each layer of the CNN model (model 7).

layer l	C_l	S_l	L_l	κ_l
conv layer 1	35	3	12	relu
pool layer 2	35	2	3	max
conv layer 3	70	1	6	relu
pool layer 4	70	2	3	max
FC	1	/	/	tanh

Where: κ_l is the activation function of the layer l

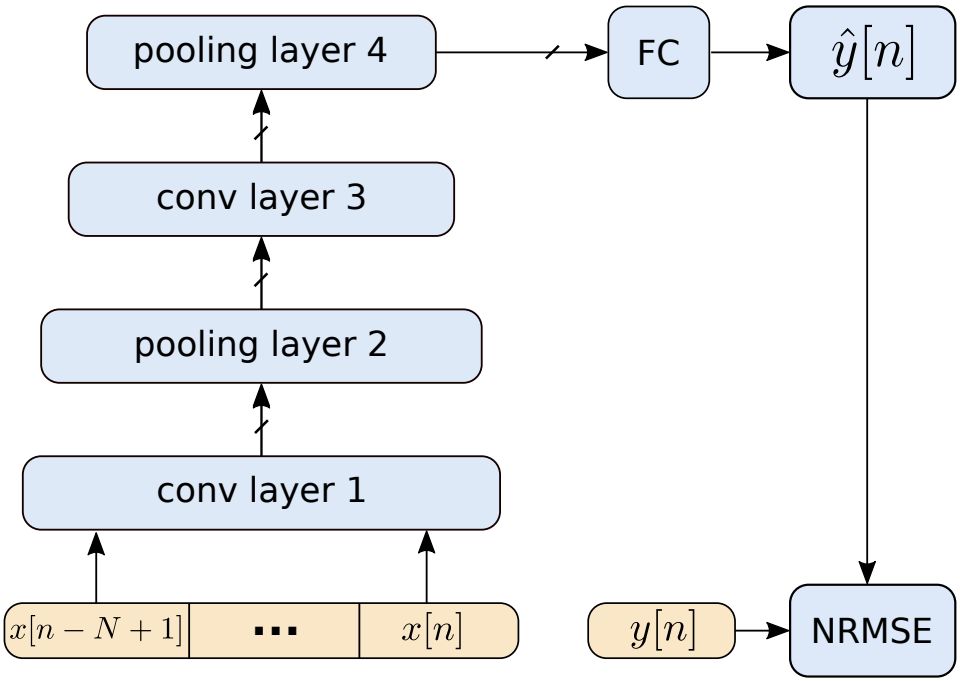


Figure 4.38 Model 7: CNN.

Discussion and results

The results of this model in terms of NRMSE and CT are presented in Table. 4.10. This model is faster than the other models (except for the model 6 based on a feedforward neural network) at the expense of the NRMSE. Indeed, this model has the worst NRMSE of all the models presented in this chapter (for the *Engl Disto* amplifier). However, the harmonic content accuracy is 3.3 dB which is still accurate for such a fast model.

Table 4.10 Model 7: normalized root mean square error, computational time and harmonic content accuracy for a CNN model emulating the *Engl Disto* amplifier with a gain of 5 (the parameters of the model are given in Table 4.9).

Amplifier	channel	Gain	CT(ms)	NRMSE(%)	Δ_{harmo}(dB)
Engl	Disto	5	2.9	43%	3.3

To illustrate the accuracy of this model, we compare in Fig. 4.39 the signal at the output of the *Engl Retro Tubes 50* (the tube amplifier) and its prediction by the neural network model. Moreover, Fig. 4.40 presents the comparison between the spectrogram of the target (Fig. 4.40a) and the spectrogram of the prediction (Fig. 4.40b) when the input signal is an ESS with a bandwidth comprised between 50 and 5000 Hz. The Fig. 4.41 presents the comparison between

the power spectrum of the target and the prediction when the input frequency is close to $f \simeq 1000\,\text{Hz}$ in the ESS:

- The SNR measured at 1000 Hz is significantly better than the one obtained for the model 1 ($SNR_{model7} = 19.2\,\text{dB} > SNR_{model1} = 15.5\,\text{dB}$).
- The harmonic number 9 is overestimated while the harmonics number 4, 10 are underestimated.
- The THD measured at 1000 Hz shows that the model tends to underestimate the harmonic content ($THD_{\text{target}} = -1.2\,\text{dB} < THD_{\text{prediction}} = -2\,\text{dB}$).

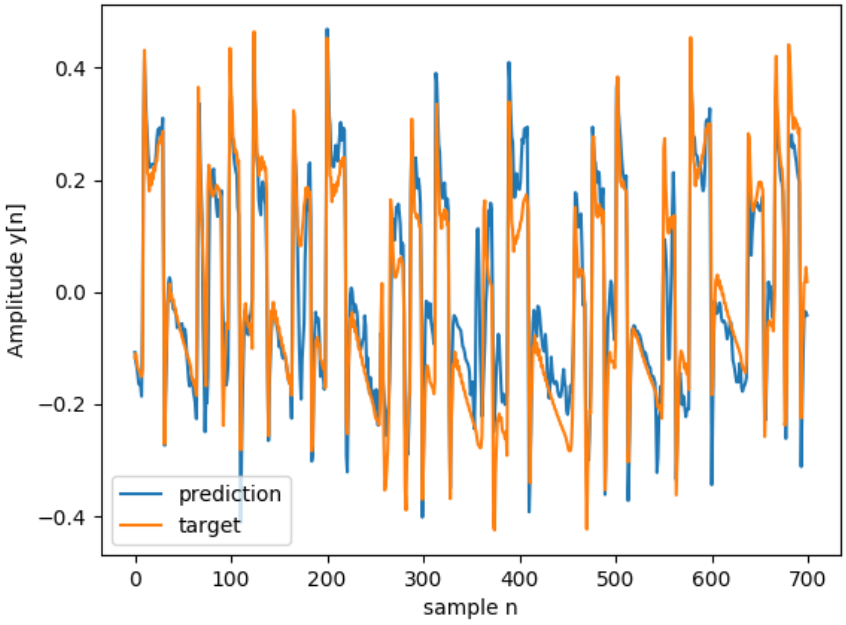
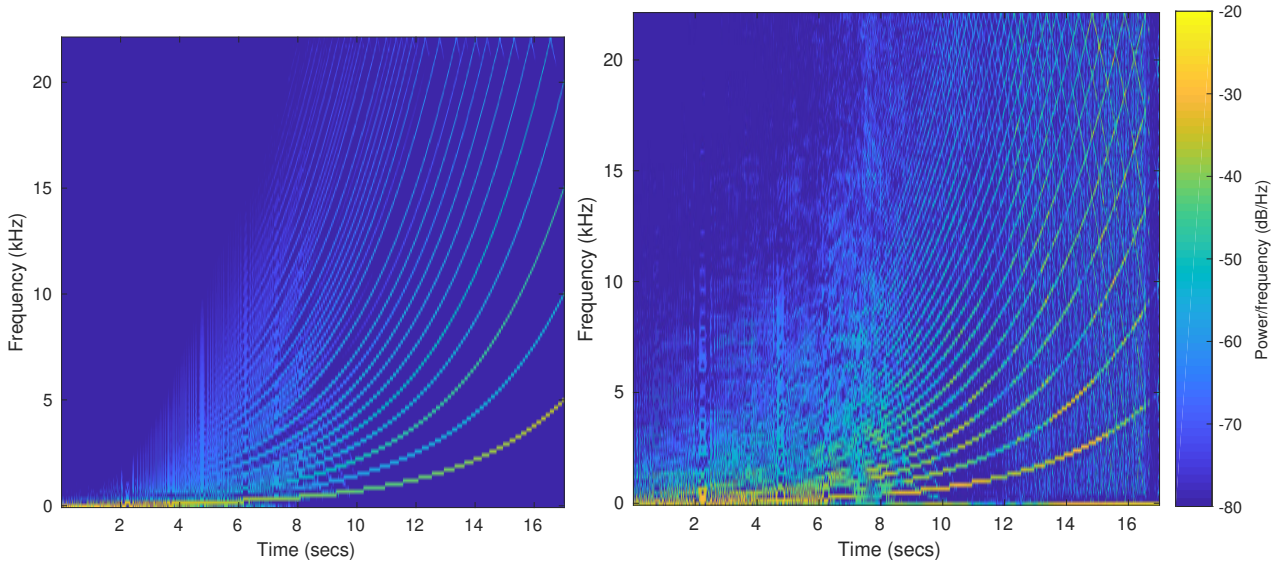


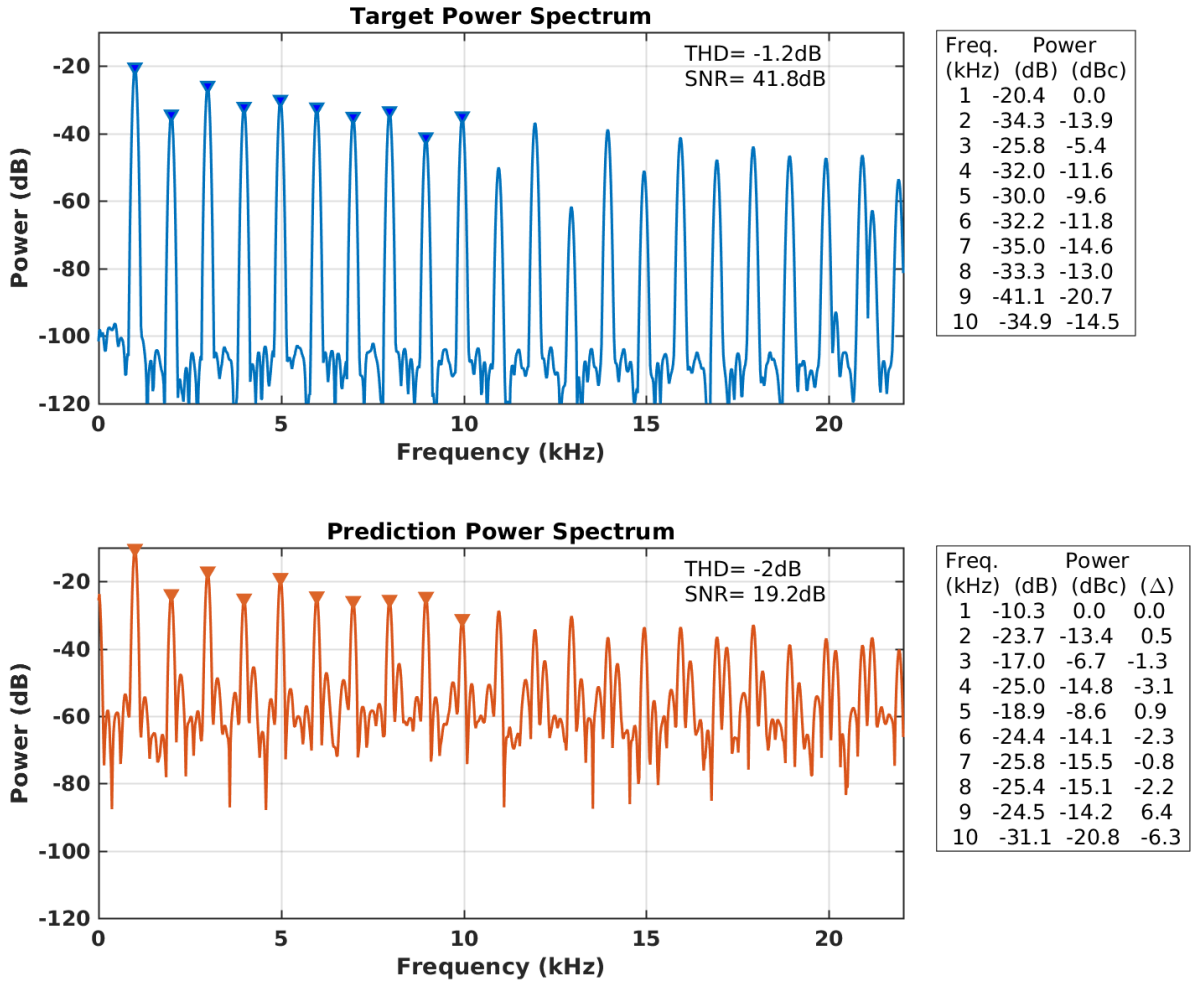
Figure 4.39 Comparison in the time domain of the output of the amplifier (*target*) and the output of the neural network model 7 (*prediction*) when a guitar signal is provided at the input.



(a) Target Engl retro tube 50 gain 5.

(b) Prediction model 7.

Figure 4.40 Comparison of the spectrograms of the target(a) and the prediction(b) for model 7 when the input is an ESS in the frequency range [50-5000] Hz.



Freq. (kHz)	Power (dB)	(dBc)
1	-20.4	0.0
2	-34.3	-13.9
3	-25.8	-5.4
4	-32.0	-11.6
5	-30.0	-9.6
6	-32.2	-11.8
7	-35.0	-14.6
8	-33.3	-13.0
9	-41.1	-20.7
10	-34.9	-14.5

Freq. (kHz)	Power (dB)	(dBc)	(Δ)
1	-10.3	0.0	0.0
2	-23.7	-13.4	0.5
3	-17.0	-6.7	-1.3
4	-25.0	-14.8	-3.1
5	-18.9	-8.6	0.9
6	-24.4	-14.1	-2.3
7	-25.8	-15.5	-0.8
8	-25.4	-15.1	-2.2
9	-24.5	-14.2	6.4
10	-31.1	-20.8	-6.3

Figure 4.41 Power spectrum of the target compared with the power spectrum of the prediction for a frequency $f \simeq 1000\,\text{Hz}$ in the ESS (model 7). $\Delta = \text{dB}c_{\text{prediction}} - \text{dB}c_{\text{target}}$.

4.2.8 Model 8: simple recurrent neural network

Presentation of model 8

In Sec. 4.2.1, the LSTM cell is introduced to avoid vanishing or exploding gradient problems. However, the use of CR layers introduced in model 5 makes the recurrent layer smaller ($N \Longrightarrow M$). This reduces the risk of vanishing/exploding gradients as explained in Appendix A.4. The model 8 is identical to the model 5 except that the LSTM cells are replaced by RNN cells as introduced in Sec. 2.7.4 and by Fig. 4.42. The output of each RNN cell can be computed as

shown in Eq. (2.56). The Tensorflow code used to build the graph of this neural network model can be found in Appendix A.5.9 and the entire code can be downloaded in [Schmitz, 2019b].

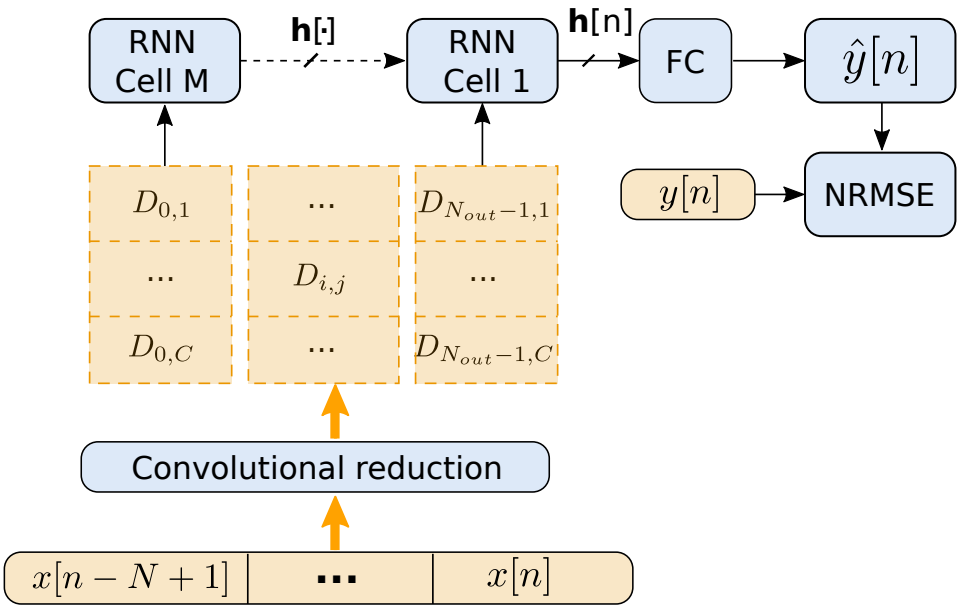


Figure 4.42 Model 8: RNN.

Parameters of the model

The model uses exactly the same hyperparameters than those presented in Sec. 4.2.5 for the model 5.

Discussion and results

The results for this model in terms of NRSME and CT are presented in Table. 4.11. As it can be seen, this model performs significantly less (in terms of NRMSE) than the model 5 (i.e., the model using LSTM cells) but has the same CT.

Table 4.11 Model 8: normalized root mean square error, computational time and harmonic content accuracy for a RNN model (parameters: are the same than those used for the model 5).

Amplifier	channel	Gain	CT(ms)	NRMSE(%)	Δ_{harmo}(dB)
Engl	Disto	5	4.1	42%	3.7

To illustrate the accuracy of the model, we compare in Fig. 4.43 the signal at the output of the *Engl Retro Tubes 50* (the tube amplifier) and its prediction by the neural network model.

Moreover, Fig. 4.44 presents the comparison between the spectrogram of the target (Fig. 4.44a) and the spectrogram of the prediction (Fig. 4.44b) when the input signal is an ESS with a bandwidth comprised between 50 and 5000 Hz. As, this model is very similar to the model 5, it is not surprising that its spectrogram looks similar. Although its NRMSE is one of the worst, this model seems less noisy than the other models. The Fig. 4.45 presents the comparison between the power spectrum of the target and the prediction when the input frequency is close to $f \simeq 1000\,\text{Hz}$ in the ESS:

- On the contrary of model 5, the harmonic number 2 is very well emulated.
- The accuracy of the predicted harmonic amplitudes is one of the best in comparison with the other models. In particular, the THD measured at 1000 Hz shows that the model has practically the same THD than the target ($THD_{\text{target}} = -1.2\,\text{dB} \simeq THD_{\text{prediction}} = -1.1\,\text{dB}$).
- The SNR measured at 1000 Hz is better than the one obtained for the model 1 ($SNR_{model7} = 16.2\,\text{dB} > SNR_{model1} = 15.5\,\text{dB}$).
- The harmonic number 9 is overestimated while the harmonic number 4 is underestimated. The others are well estimated.
- The THD measured at 1000 Hz shows that the model tends to underestimate the harmonic content ($THD_{\text{target}} = -1.2\,\text{dB} < THD_{\text{prediction}} = -1.1\,\text{dB}$).

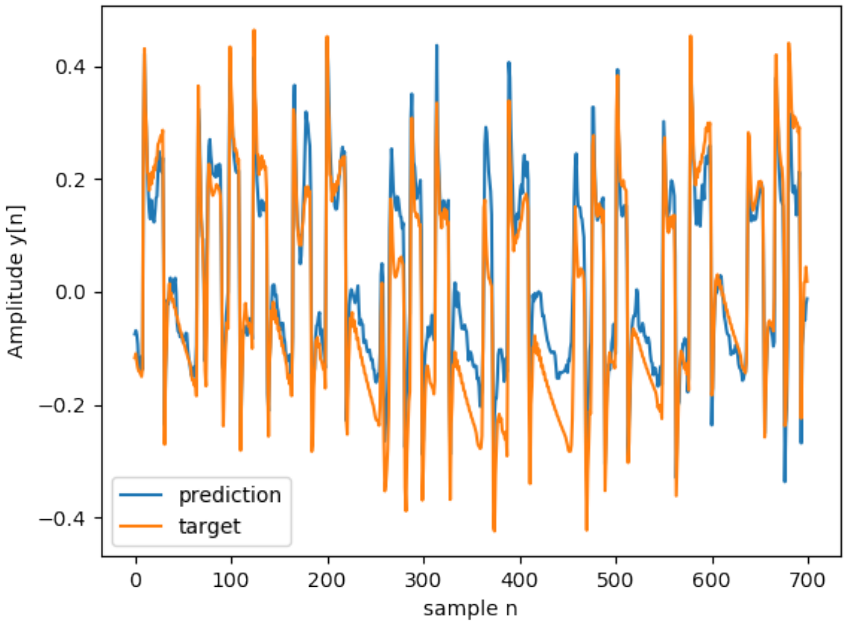
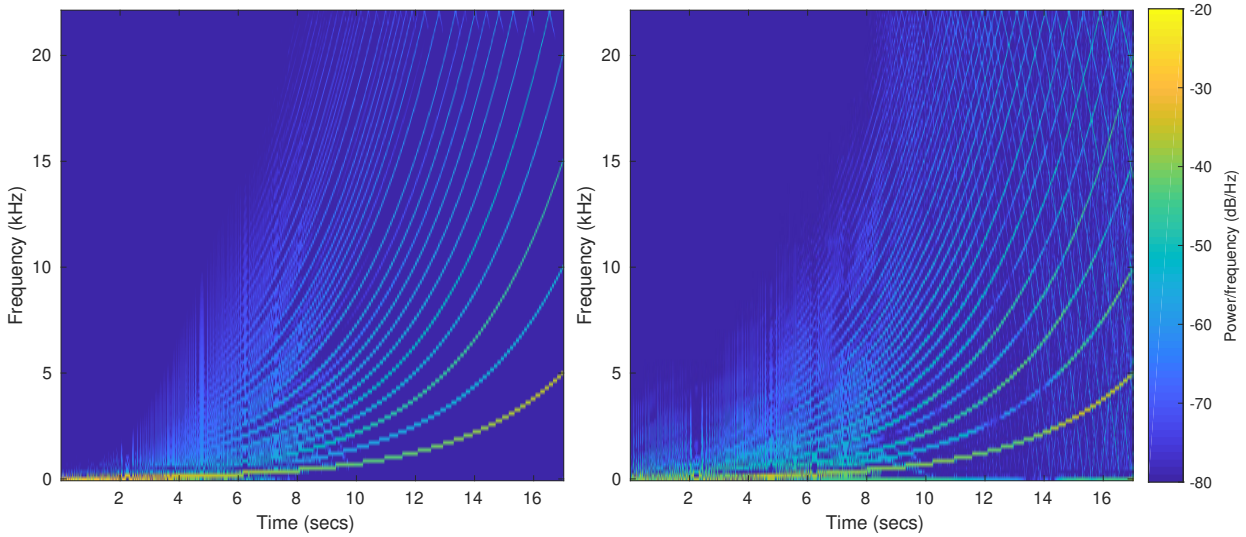


Figure 4.43 Comparison in the time domain of the output of the amplifier (*target*) and the output of the neural network model 8 (*prediction*) when a guitar signal is provided at the input.



(a) Target Engl retro tube 50 gain 5.

(b) Prediction model 8.

Figure 4.44 Comparison of the spectrograms of the target(a) and the prediction(b) for model 8 when the input is an ESS in the frequency range [50-5000] Hz.

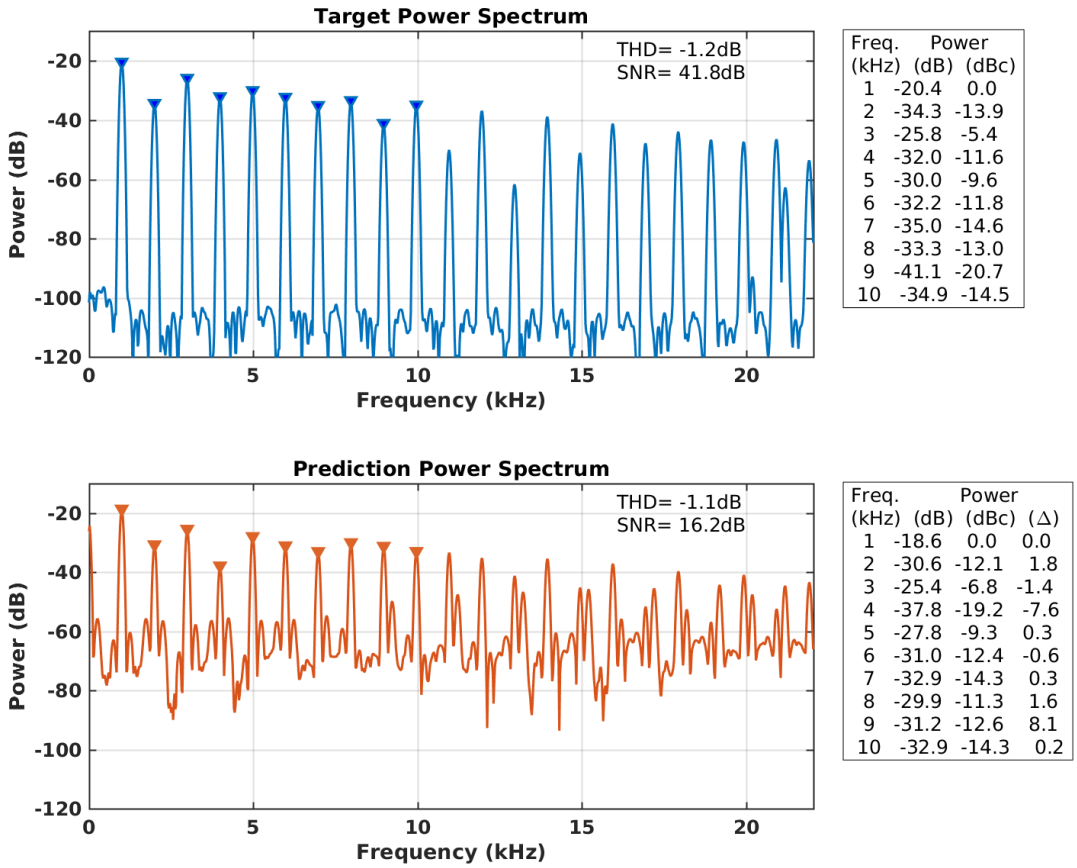


Figure 4.45 Power spectrum of the target compared with the power spectrum of the prediction for a frequency $f \simeq 1000\,\text{Hz}$ in the ESS (model 8). $\Delta = \text{dB}c_{\text{prediction}} - \text{dB}c_{\text{target}}$.

4.3 Evaluation of the different neural network models through performance metrics

4.3.1 Comparison in the time domain of normalized root mean square error, computational time and signal to noise ratio

The normalized root mean square error is one of the main objective measures of model *accuracy*. From now on, it is called the Performance Index (PI). Another performance measure is the Computational Time (CT) to process a buffer of audio sample. Indeed, a fast model is required to enable the emulation of the guitar signal chain in real time with the smallest latency. The

size of this audio buffer is fixed to 400 samples. Finally, the harmonic content accuracy Δ_{harmo} and the SNR are also used to compare the different models. These measures are presented for all models in Table 4.12 (Δ_{harmo} is defined in Eq. (2.63)). All these models are trained and evaluated using the same dataset and the same amplifier, the *Engl Retro Tube 50 channel Disto* with the same gain (i.e., gain=5) except for model 4 which is trained for gains 1 and 5. It appears that the CNN and the DNN models outclass the LSTM models in terms of computational time. However, the LSTM model 5 has the best PI. The RNN, CNN and DNN models have approximately the same accuracy but the DNN model is significantly faster than the others. As discussed before, the chosen PI may not be perfectly representative of the human perceptual accuracy. Consequently, if the differences in terms of PI were inaudible, the DNN model would be the most interesting one. One can notice that, all these models are close in terms of PI. However, as it will be seen in Sec. 4.4.1, a difference of a few percents in terms of PI may involve a huge difference of perceptual accuracy. The analysis of The SNR measured at 1000 Hz shows that all the models have a lower SNR than the one measured for the target ($SNR_{target} = 41.8$ dB). The models number 2, 5 and 7 have the best SNR, at least for frequencies close to $f = 1000$ Hz. The model 3 is particularly noisy.

Table 4.12 Comparison of all the models in terms of Normalized Root Mean Square Error, Computational Time, Signal to Noise Ratio and harmonic content accuracy for the *Engl retro tube 50* amplifier at gain=5:

Model	Type	NRMSE	$CT_{400samples}$	$SNR_{1000\,\text{Hz}}$	Δ_{harmo}
1	LSTM	39%	7.1ms	15.5 dB	5.3 dB
2	LSTM	42%	12.6ms	**23.2** dB	**2.6** dB
3	LSTM	37%	6.4ms	11.5 dB	3.2 dB
4	LSTM	42%	6.5ms	16.4 dB	7.2 dB
5	LSTM	**35%**	4.1ms	21.4 dB	5.6 dB
6	DNN	41%	**2.2ms**	16 dB	6 dB
7	CNN	43%	2.9ms	19.2 dB	3.3 dB
8	RNN	42%	4.1ms	16.2 dB	3.7 dB

4.3.2 Evaluation in the frequency domain based on spectrograms and power spectrums

The spectrograms obtained for models number 1 to 8, when the input signal is an ESS with a bandwidth comprised between 50 and 5000 Hz are presented in Fig. 4.46. These spectrograms have already been presented and discussed earlier but they are placed side by side to ease their comparison. Fig. 4.47 presents the power spectrum (focus on the tenth first harmonics) of the target and prediction signals for the different models when the frequency of the input signal is close to $f = 1000$ Hz in the ESS. The prediction level is adjusted such that the amplitude of the fundamental matches the amplitude of the fundamental of the target. Based on Figs. 4.46 and 4.47, several observations can be made:

- The spectrograms of models 1,3,4 are the noisiest, especially for input signals in the frequency range 2000-5000 Hz (i.e., t=[14,17]s in the ESS) and in the range 50-150 Hz (i.e., t=[0,4]s in the ESS).
- The spectrogram of model 2 (2 LSTM layers) is "cleaner" (i.e., less noisy) than the one obtained from the model 1 (1 LSTM layer), even if the PI of the model 2 (42%) is a bit worse than the PI of the model 1 (39%). This proves that the noise of a model is not directly correlated to the PI. Consequently, the PI alone is not able to characterize the global quality of a model. In Fig. 4.47, the comparison of the power spectrums, for an input frequency close to 1000 Hz, shows that the model 2 tends to overestimate the even harmonics and to underestimate the odd ones. The model 2 has more than 270 000 trainable parameters (in comparison, the model 1 has only 91000 trainable parameters) enabling a more flexible model. This improves the harmonic content accuracy Δ_{harmo} as shown in Table 4.12.
- Model 3 (sequence-to-sequence prediction) has the noisiest spectrogram, even if its PI (37%) is slightly better than the one obtained for the model 1 (PI of 39%). This shows, once again, that the PI alone cannot explain the accuracy of a model. As it can be see in Table 4.12, the harmonic content accuracy of this model is one of the best.
- Model 4 (taking into account the gain of the amplifier in the LSTM cells) is a little bit noisier than the model 1 in low-frequency but is a little bit quieter in high-frequency. The power spectrum for $f \simeq 1000$ Hz shows that the harmonic content accuracy is lower than the one obtained for model 1 (see Fig. 4.47 and Table. 4.12). Actually, this model has the worst harmonic content accuracy of the eight models. However this model shows that

it is possible to take the parameters of the amplifier into account in the neural network. Moreover, the gain parameter is the most difficult parameter to emulate since it directly controls the amount of distortion (i.e., the nonlinearity strength of the system). Some other parameters such as: the bass, the middle and the treble (called the tonestack in the musician world) should be easier to model since the tonestack of an amplifier is often placed at the last stage of the pre-amplifier. Consequently, it will act as a set of simple filters and does not contribute to the nonlinear behavior of the amplifier.

- Model 5 (LSTMConv) is a very accurate model (in terms of PI). Its spectrogram is one of the cleanest, but the power spectrum shows that the harmonics are not so well estimated (see also Table 4.12).

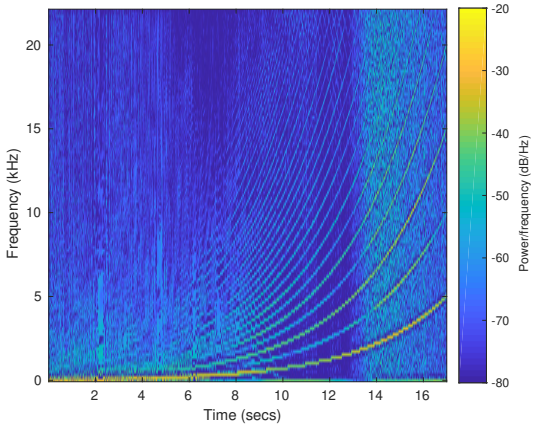
- Model 6 (DNN) is globally not too noisy (even if at 1000 Hz the SNR is low). As it can be seen in the power spectrum (for a frequency close to $f = 1000$ Hz) and in Table 4.12, the harmonic content accuracy is one of the worst of the eight models. However, this is also the fastest model. Depending on the resources and on the computational time available, it could be interesting to consider the use of this model.

- Model 7 (CNN) is quiet, gives a very good harmonic content accuracy and is one of the fastest models. Surprisingly, this model has a poor accuracy in terms of PI=NRMSE.

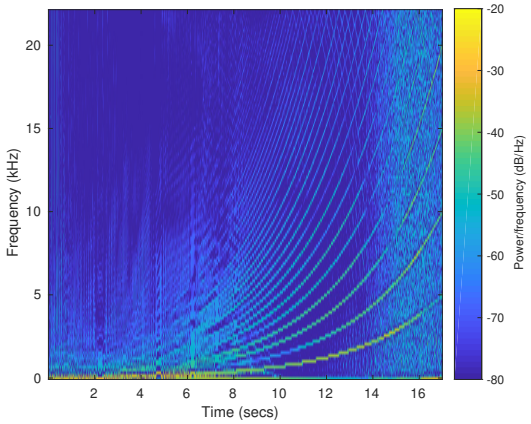
- Model 8 (RNNConv) has a PI of 42% which is not very accurate. However, globally, it is one of the quietest models. The power spectrum for an input frequency close to $f = 1000$ Hz shows that the harmonics are very well estimated.

- One can notice that the noise in the spectrograms, the PI and the harmonic content accuracy are not fully correlated. These quality measures are therefore complementary. Also note that the SNR and the Δ_{harmo} are given for an input frequency close to $f = 1000$ Hz which is not representative of the entire frequency range used.

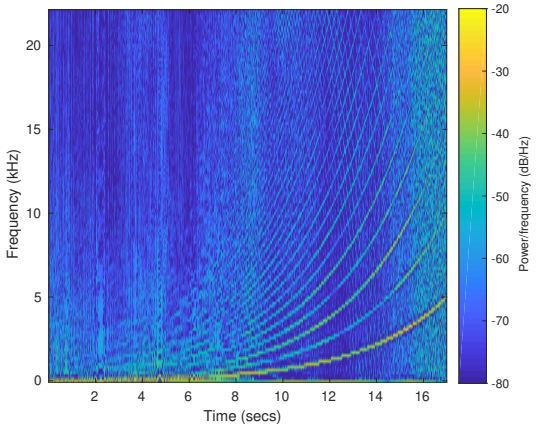
From these observations, we have decided to focus on two models: the first one is our most accurate model in terms of PI (i.e., the *LSTMConv* model 5) and the second one is our fastest model (i.e., the DNN model 6) called from now on, the LSTM and DNN models respectively. To ensure that the characteristics of both models are not specific to the Engl amplifier, these two models are going to be compared for the emulation of different amplifiers (introduced in Sec.4.1.3). Their PI are given in Table. 4.13, it can be noticed that:



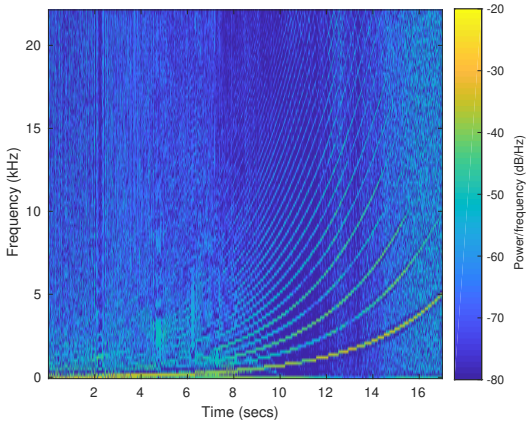
(a) model 1.



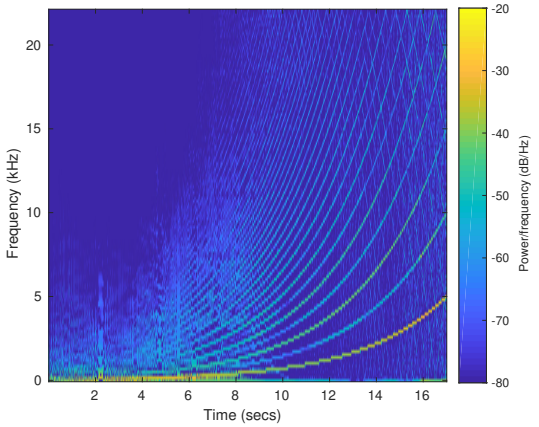
(e) model 2.



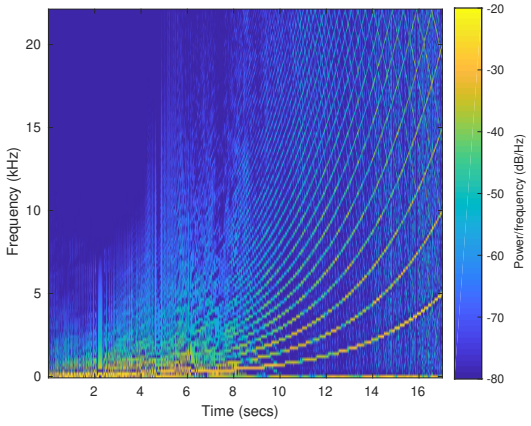
(b) model 3.



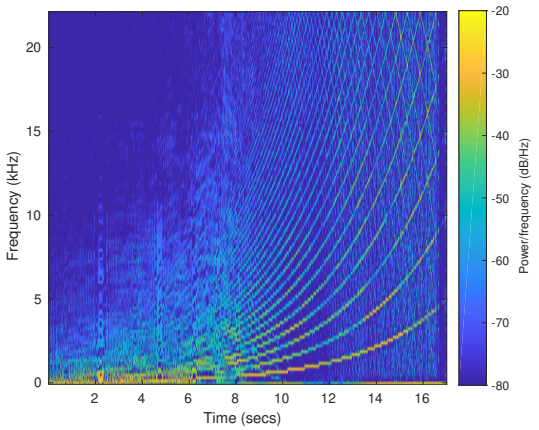
(f) model 4.



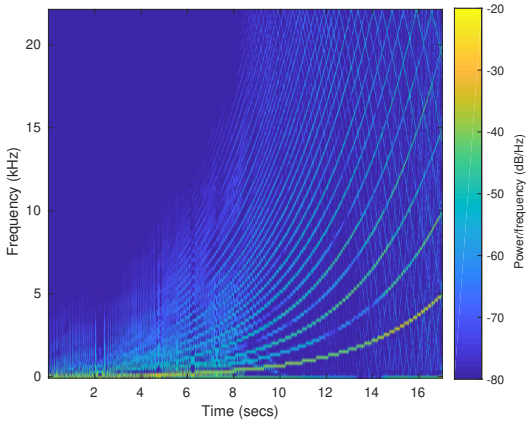
(c) model 5.



(g) model 6.

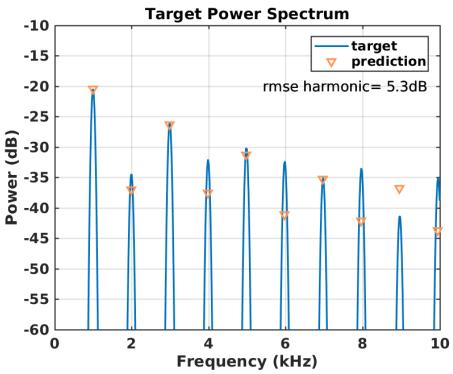


(d) model 7.

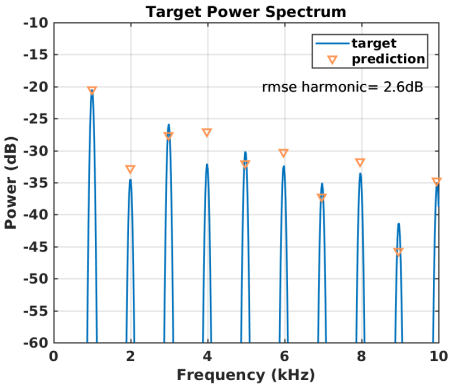


(h) model 8.

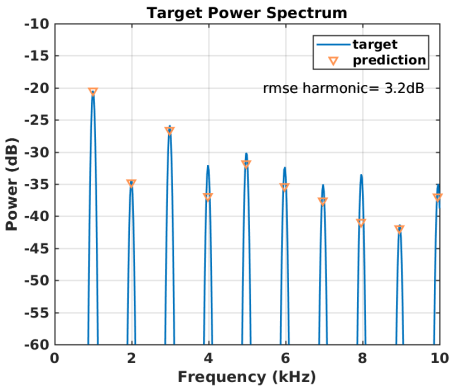
Figure 4.46 Comparison of the spectrograms of the models 1-8 for an ESS as input signal.



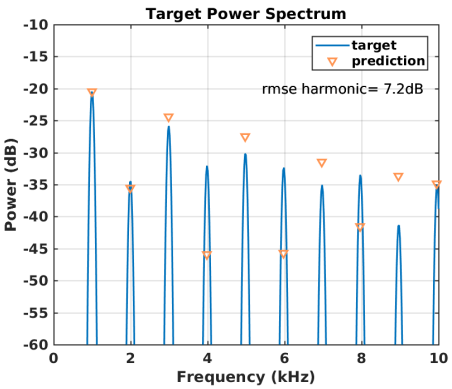
(a) model 1.



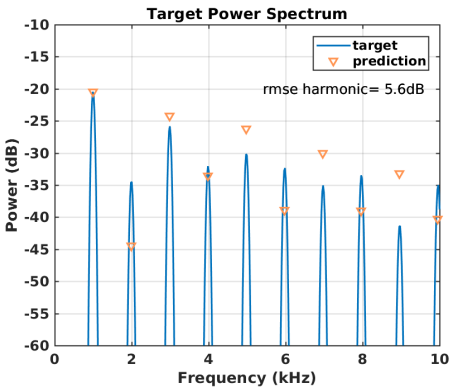
(e) model 2.



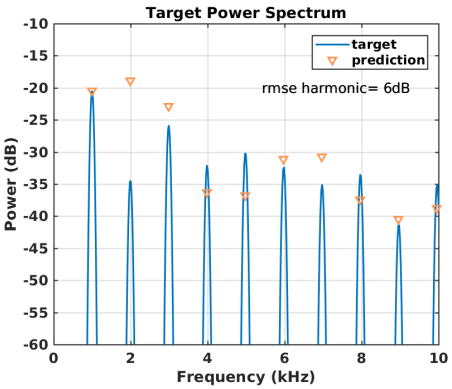
(b) model 3.



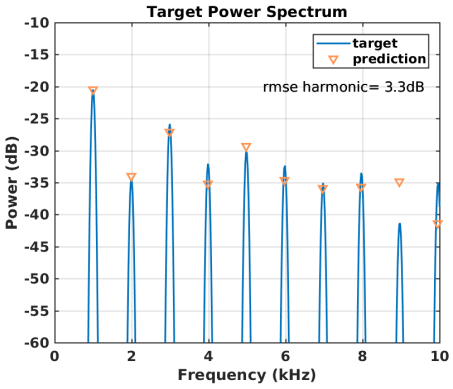
(f) model 4.



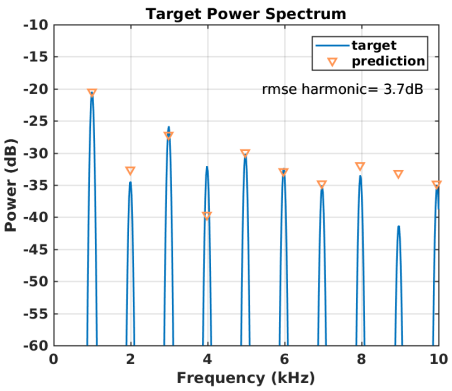
(c) model 5.



(g) model 6.



(d) model 7.



(h) model 8.

Figure 4.47 Power spectrums of the models 1-8 for $f \simeq 1000\,\mathrm{Hz}$ in the ESS.

- The LSTM model always makes a better prediction than the DNN model. However, for the amplifiers: *MesaBoogie Mark V disto*, *MesaBoogie Mark V clean* and *MesaBoogie 550 crunch*, the difference in terms of PI is less than 5%.
- The amplifiers have an increasing amount of distortion when passing from a *clean* channel to a *crunch* channel and from a *crunch* channel to a *distortion* channel. Some might think that the more distortion the amplifier has, the worst the PI will be, but this is not always the case. For example, the PI of the *MesaBoogie Mark V* is increasingly worse when choosing a channel of this amplifier with more nonlinearities, but this is not the case for the *MesaBoogie 550*, where the clean channel (i.e., the one with less nonlinearities) has the worst PI of the three channels.
- The *Blackstar* is the only "full" transistor amplifier of this set. Apparently, it is not better emulated than the others. This shows that the nonlinearities specific to vacuum tube amplifiers are just as well emulated than those in the transistor amplifier.

Table 4.13 The PI obtained for different amplifiers with the LSTM and DNN models and their relative differences:

Amplifier name	PI_{LSTM}	PI_{DNN}	ΔPI
Engl retro tube disto	35%	41%	6%
MesaBoogie 550 clean	36%	46%	10%
MesaBoogie 550 crunch	29%	31%	2%
MesaBoogie 550 disto	32%	37%	5%
Ibanez TSA15 crunch	25%	31%	6%
MesaBoogie Mark V clean	4%	7%	3%
MesaBoogie Mark V crunch	13%	23%	10%
MesaBoogie Mark V disto	21%	25%	4%
Blackstar HT5M	28%	39%	11%

Where: PI_{LSTM} and PI_{DNN} are the performance indices of the LSTM and DNN models. $\Delta_{PI} = PI_{DNN} - PI_{LSTM}$

4.4 Subjective evaluation of LSTM and DNN models by using listening tests

In Table 4.13, an evaluation of models 5 and 6 has been made for several amplifiers in terms of NRMSE. In order to select a model based on its perceptual quality, we have defined a perceptual performance index named the Probability of Audible Difference (PAD):

Probability of audible difference:

The PAD is the probability for a listener to be able to find an audible difference between the prediction and the target. The PAD is specific to a model and an amplifier (i.e., the PAD of the model 2 evaluated on the *Engl disto* amplifier is equal to ...). A PAD of 75% means that three listeners over four are able to hear a difference between the target and the prediction signal.

Two listening-tests have been realized to answer to the following questions:

- Can the number of parameters of a model be reduced in order to speed up its execution at the expense of the PI, but without a loss of perceptual accuracy (i.e., without increasing the PAD)?
- Is there a threshold in terms of PI, under which, the PAD does not evolve anymore (i.e., the model is sufficiently accurate and does not need to be improved anymore)?

The objective of these listening tests is not to rate the quality of the emulation on a continuous scale as in a classic ABX test [Rec, 2015]. Indeed, the objective is to know which percentage of listeners are still able to hear a difference between the *target* and the *prediction* signals. Consequently, this test only allows two possible answers: the *target* and the *test* signals are *the same* or *not the same*. The listening tests have been carried out using a *Seinnheiser HD25-1* headphone. However, in order to have more results, the listening test has also been put on-line [Schmitz, 2018a], where the control of the listening material is not possible. The sound signals are encoded in *wav* format with CD audio quality. The test is still ongoing and the results can be consulted in [Schmitz, 2018a].

4.4.1 The Progressive test: analysis of the PAD in function of the PI

Implementation of the Progressive test

The idea behind this first listening test, named the *Progressive test*, is to determine the threshold, in terms of PI, required to reach a PAD equivalent to the random choice probability (i.e., the

probability to be successful at the test by picking the answers randomly). Indeed, if listeners cannot distinguish the target from the prediction better than an algorithm that randomly associates the test signal to the target or to the prediction label, then, the model does not need to be improved anymore (unless to improve the CT). A quicker model using a smaller number of parameters can then be considered. The chosen model for this test is the LSTMConv model 5.

The test is organized as follows:

1. The prediction and the target audio signals are labeled and presented to the listener. The listener can play both sounds as many times as he wants until he thinks that he may retrieve the differences between the prediction and the target. This is the familiarization phase.

2. The amplifier MesaBoogie MarkV Disto has been chosen for this test since it is well emulated in terms of PI for such a strong level of distortion. Two sounds are presented to the listener: the first one is the target, the second one is either the same target or the prediction (randomly picked).

3. The listener has then to indicate if the two sounds are the same or not (as presented in Fig. 4.48).

4. The test starts with a model having a PI (i.e., the NRMSE) of 40% then 30%, 25% and finally 20%.

The different PI have been obtained by stopping the training phase of the neural network model when the PI threshold is reached (by limiting the number of epochs of the training phase).

Results of the Progressive test

The results of the Progressive test (with 243 participations) are presented in Fig. 4.49. The vertical axis represents the probability for of listener to be able to hear a difference between the prediction and the target (i.e., the PAD) and the horizontal axis represents the NRMSE of the LSTM model for the *MesaBoogie MarkV Disto* amplifier. The participants can be anybody without limit of age or any musical knowledges. As it can be seen, the PAD and the PI are not correlated in a linear way. Clearly, for a PI = 40%, the perceptual difference between the target and the prediction signals is strong, while the perceptual difference for models trained for a PI = 30, 25 and 20% is more subtle.

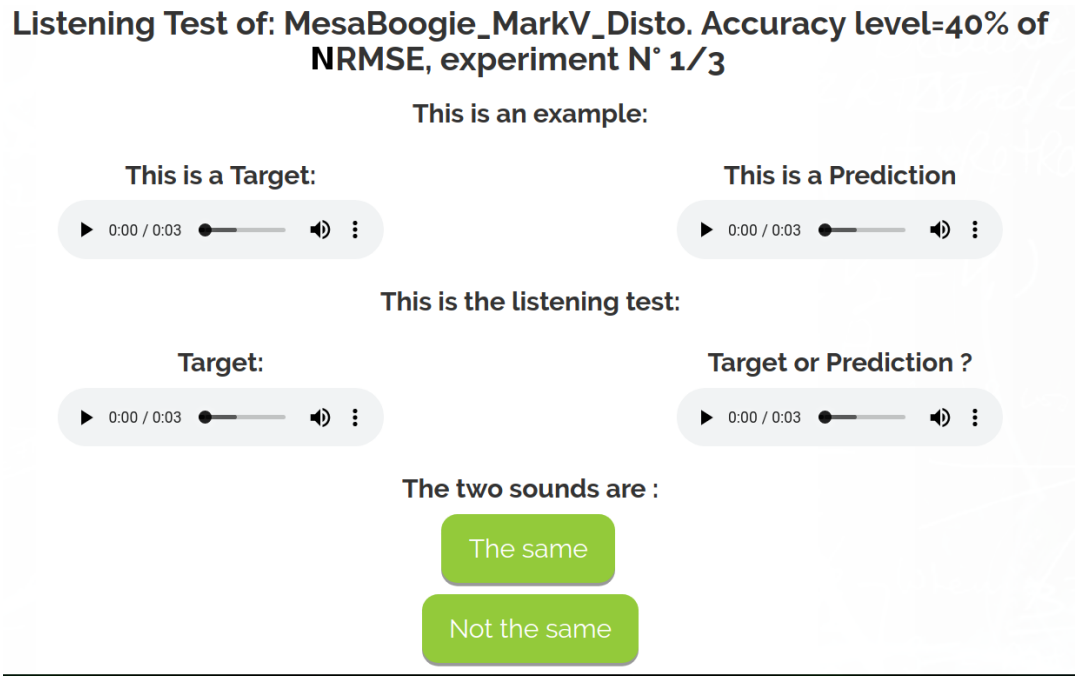


Figure 4.48 Listening test interface.

To reduce the possibility to give the correct answer by chance, we have asked each listener to make the same listening test 3 times (while changing randomly if the tested sound is a target or a prediction). The test is positive (i.e., the listener is able to hear a difference) if all the correct answers are given. The probability to pass the N_{exp} tests successfully by chance is given by:

$$\frac{1}{2^{N_{\text{exp}}}} = \frac{1}{2^3} = 12.5\%. \tag{4.16}$$

The model can be considered as "subjectively accurate" when the PAD is close to this random level. When analyzing the PAD, it is important to notice that 12.5% of tested listeners may be counted as being *able to hear a difference* while they just have made "good" random choices.

How can this bias be taken into account?

Considering N_p listeners making the N_{exp} experiments, these listeners can be divided into two classes X and $N_p - X$, where:

$$\begin{cases} X & \text{listeners who are really able to retreive the difference} \\ N_p - X & \text{the others} \end{cases} \tag{4.17}$$

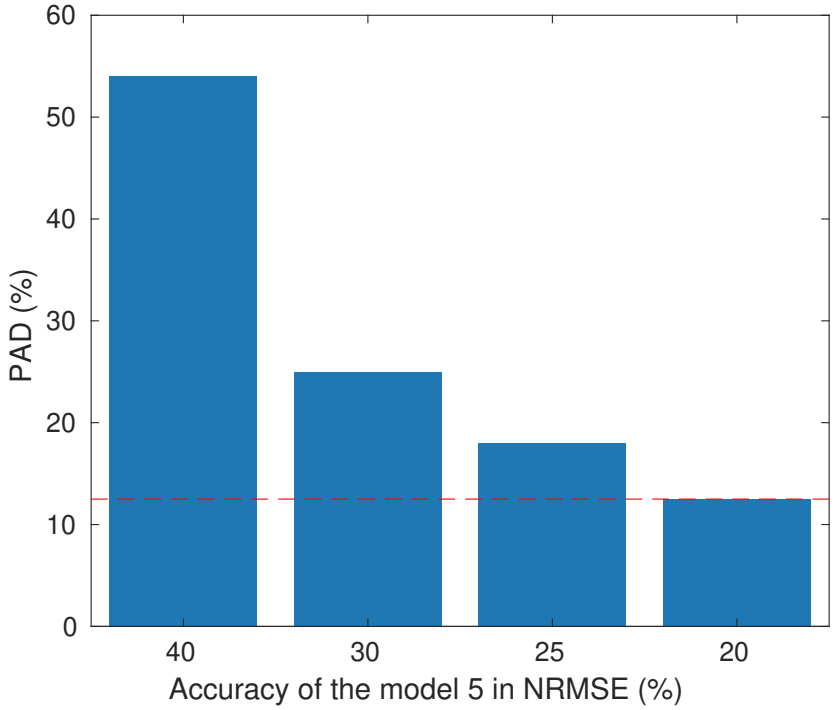


Figure 4.49 Progressive test: the percentage of listeners who hear a difference between the target and the prediction (*PAD*) when the LSTM model 5 is trained until some specific PI are reached. Emulation of the *MesaBoogie_MarkV_Disto* with the LSTM model 5.

We can guess that listeners belonging to the second class still have a probability of about 50% to correctly answer each question. Therefore, a proportion of $\frac{1}{2^{N_{\text{exp}}}}$ still gives the correct answers by chance. The total of correct answers is thus given by:

$$PAD \cdot N_p = X + \frac{N_p - X}{2^{N_{\text{exp}}}}. \tag{4.18}$$

$$\begin{aligned} PAD &= \frac{X}{N_p} + \frac{1 - \frac{X}{N_p}}{2^{N_{\text{exp}}}} \\ &= \frac{X}{N_p}(1 - \frac{1}{2^{N_{\text{exp}}}}) + \frac{1}{2^{N_{\text{exp}}}}. \end{aligned} \tag{4.19}$$

Finally, if the N_p listeners can be considered as a representative sample of the population, the percentage of listeners that are truly able to hear a difference between the prediction and the target signals is given by the Corrected Probability of Audible Difference (PAD_{corr}):

$$PAD_{\text{corr}} = \frac{X}{N_p} = \frac{2^{N_{\text{exp}}}.PAD - 1}{2^{N_{\text{exp}}} - 1}. \tag{4.20}$$

For example, if $N_{\text{exp}} = 3$ successive experiments have been made and that the experimental PAD = 25%, the estimated percentage of listeners truly able to hear a difference between the two signals (i.e., excluding those who give the correct answers by chance) is given by:

$$PAD_{\text{corr}} = \frac{2^3 \cdot \frac{25}{100} - 1}{2^3 - 1} \simeq 14.3\%. \tag{4.21}$$

The corrected results of the Progressive test using the PAD_{corr} are presented in Fig. 4.50. The vertical axis represents the probability for a listener to truly be able to hear a difference between the prediction and the target (i.e., the PAD_{corr}) and the horizontal axis represents the corresponding NRMSE of the LSTM model for the *MesaBoogie MarkV Disto* amplifier. As it

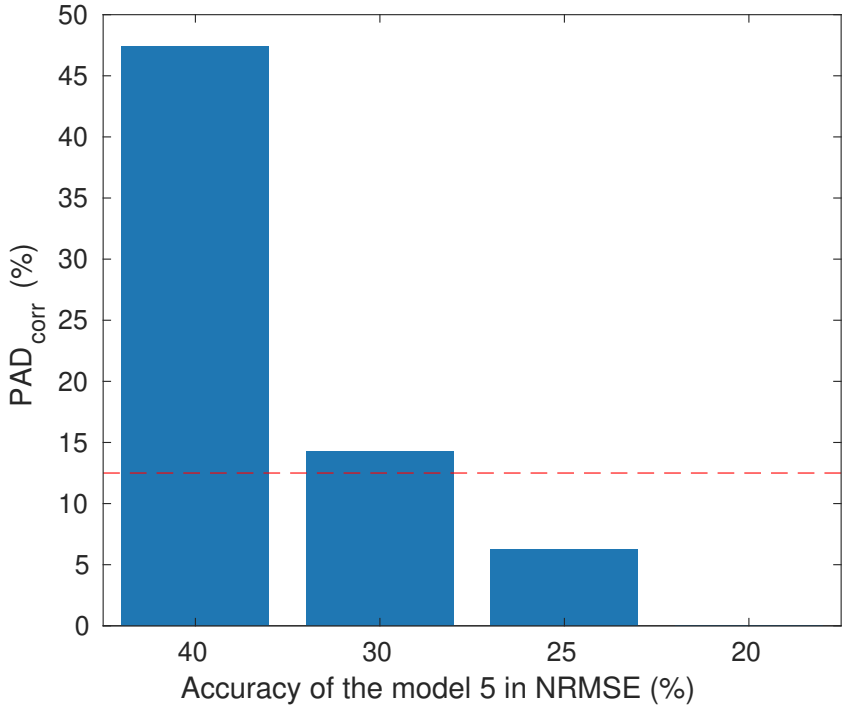


Figure 4.50 Progressive test: the percentage of listeners who *truly* hear a difference between the target and the prediction (PAD_{corr}) for different training PI measured in NRMSE. Emulation of the *MesaBoogie_MarkV_Disto* with the LSTM model 5.

can be seen in Fig. 4.50, statistically, nobody is able to hear a difference between the target and the prediction when the PI is lower than 20%.

Analyzing the subjective performance (PAD_{corr}) of a model in function of its PI can be a convenient way to select the fastest model that respects this performance. For example, if it is

considered that misleading 90% (PAD_{corr}=10%) of the listeners is enough, the parameters of the model can be relaxed to have a faster model that respects this percentage.

Example of parameter numbers reduction for a model

If the MesaBoogie Mark V Disto (whose the best PI is equal to 21% with a $PAD_{\text{corr}} = 0\%$) was emulated with a faster LSTM model using a smaller number of time steps N and a longer stride size S, a PI of 25% would still be reached, satisfying the condition $PAD_{\text{corr}} < 10\%$. The original model and the new shorter model are compared in Table 4.14. As it can be seen, the new model is 17% faster for a loss of 7% in the PAD_{corr}.

Table 4.14 Comparison of two LSTM models in terms of PI and Computational Time:

	N	S_1	S_2	PI	CT	PAD_{corr}
LSTM (model 5)	150	4	3	21%	4.1ms	0%
LSTM shorter	100	4	5	25%	3.5ms	7%

Where: N is the number of input time step S_l is the stride size of the l^{th} convolutional reduction layer.

4.4.2 The Hardest test

While the *Progressive test* was designed to compare the PAD in function of the PI, this second listening test (named the *Hardest test*) aims to compare the PAD of all amplifiers using the best reached PI obtained for the LSTM and the DNN models. It gives information on which amplifiers are sufficiently well emulated by the LSTM or the DNN models and which ones are not. There are 196 participants for the LSTM Hardest test and 113 participants for the DNN Hardest test. This test is not reserved to musicians.

Implementation of the Hardest test

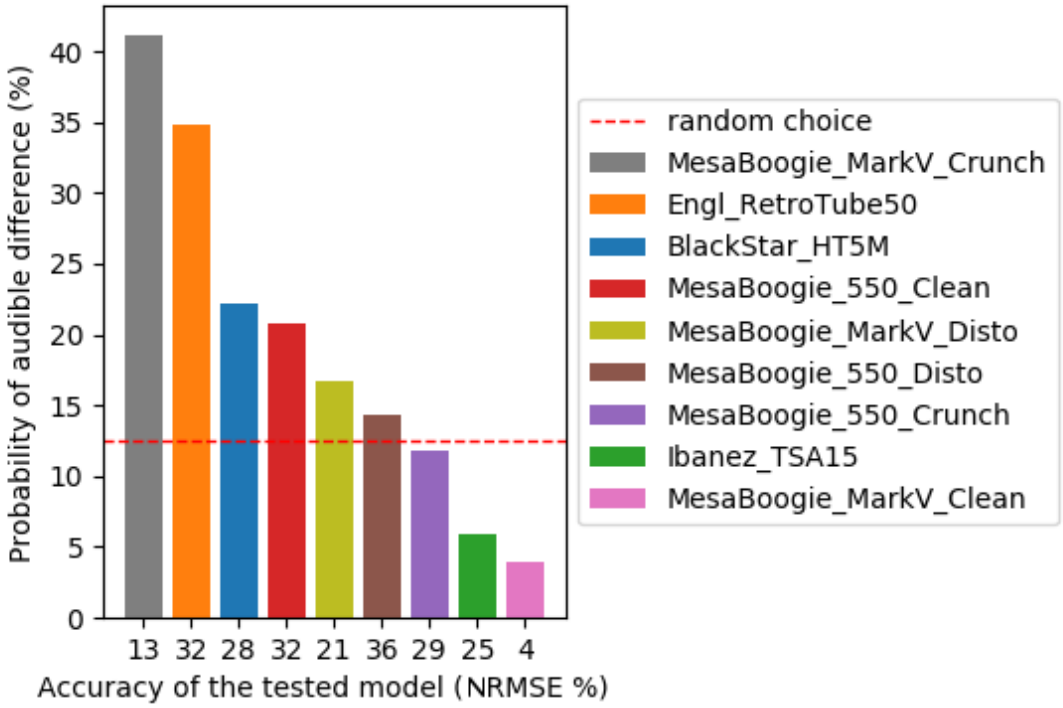
In this test, each listener has to choose between two different models before starting the listening test: the first one uses the LSTM model 5 and the second the DNN model 6. The test follows the same rules than for the Progressive test, except that the model type (LSTM or DNN) has to be chosen first.

Results of the Hardest test

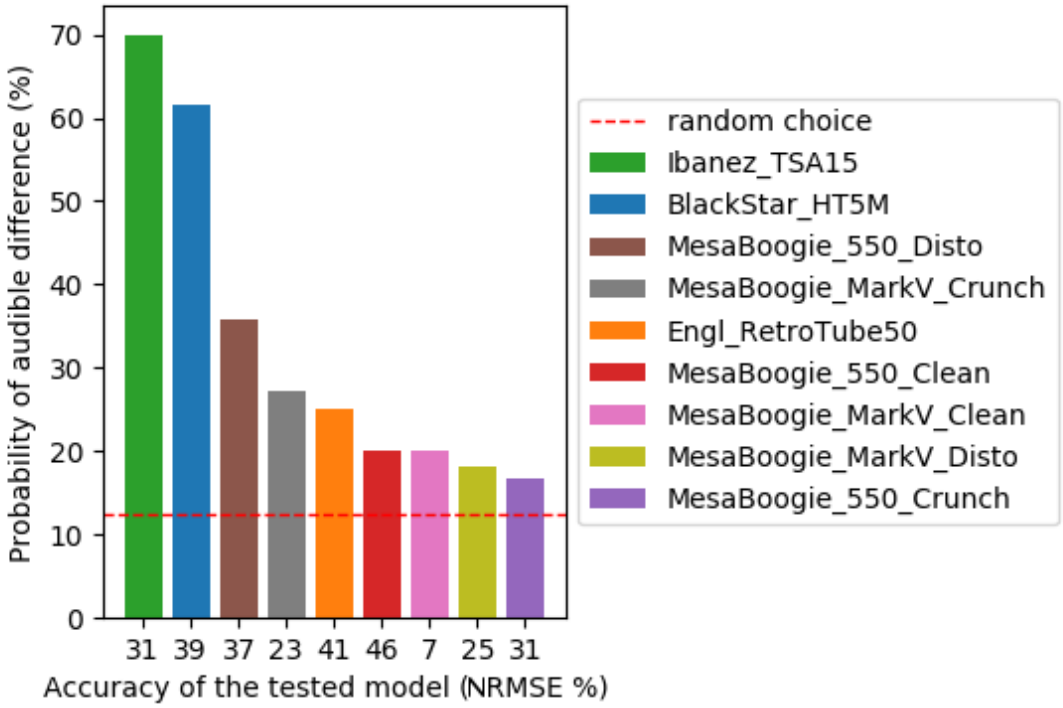
It has been shown that the Progressive test can help to choose the maximum PI that a model should have in order to respect a given PAD. However, the correspondence between the PAD and the PI significantly depends on the amplifier and on the model chosen. Consequently, a simple threshold on the PI is not sufficient to determine the perceived accuracy of a model. The PAD depends on the chosen model and on the selected amplifier as shown in Fig. 4.51 where the PAD between the target and the prediction for all amplifiers emulated with LSTM (a) and DNN (b) models are presented (the used PAD is the non corrected one).

The results of this test lead to several observations:

- Globally, the LSTM model has a better perceptual accuracy: the Mean(PAD_{LSTM}) = 18.8% while the Mean(PAD_{DNN}) = 30.6%. It also has a better PI: the $Mean(PI_{LSTM}) =$ 24.4% while the $Mean(PI_{DNN}) = 31.1\%$.
- Some amplifiers are very well emulated by LSTM model but not by the DNN model, e.g., the $PAD_{LSTM}(IbanezTSA15) = 7\%$ while the $PAD_{DNN}(IbanezTSA15) = 70\%$.
- On the contrary, some amplifiers are better emulated by the DNN model than by the LSTM model, e.g., the PAD_{LSTM}(*MesaBoogie Mark V crunch*) =46% while the PAD_{DNN}(*MesaBoogie Mark V crunch*) =27%.
- The PAD and the PI are not well correlated: this clearly appears in Fig. 4.51, since the different amplifiers are ranked by decreasing PAD, but their corresponding PI (on the horizontal axis) are not ranked in a decreasing order. This constitutes a fundamental barrier to train neural network models with a sound emulation purpose as discussed in Sec. 4.4.3.
- In Fig. 4.51a, some PAD are below the random probability line (in red), which can seem strange. Indeed, each amplifier has been tested by approximately 20 different listeners which is not enough to avoid some statistical fluctuations. Moreover, an additional explanation could be that the probability to have a target (or a prediction) as test signal is about 50% (randomly picked) which leads to a probability of 3 random correct answers of 12.5%. However, in the case where the same test signal is taken 3 times, human psychology could prevent the listener to answer three times the same answer reducing the probability of correct random answers.



(a) LSTM model.



(b) DNN model.

Figure 4.51 Hardest test, on the vertical axis, the probability of audible difference (PAD) between the target and the prediction. On the horizontal axis: the best normalized RMSE (PI) of the LSTM (a) and DNN (b) models.

The Table 4.15 lists the variation of performances in terms of PI and PAD when passing from the LSTM model to the DNN model. The following acronyms are used:

- PI_{LSTM} and PI_{DNN} are the performance indices of the LSTM and DNN models respectively.
- PAD_{LSTM} and PAD_{DNN} are the probabilities of audible difference between the target and the prediction for the LSTM and the DNN models respectively.
- $\Delta PI = PI_{DNN} - PI_{LSTM}$ is the loss of accuracy in terms of PI (i.e., the NRMSE) when the DNN model replaces the LSTM model. Finally, $\Delta PAD = PAD_{DNN} - PAD_{LSTM}$ is the perceived loss of accuracy when the DNN model replaces the LSTM model.

Table 4.15 The PI and the PAD obtained with the LSTM and DNN models and their relative differences:

Amplifier	PI_{LSTM}	PI_{DNN}	PAD_{LSTM}	PAD_{DNN}	ΔPI	ΔPAD
Engl retro tube disto	32%	41%	38%	25%	9%	-13%
MesaBoogie 550 clean	36%	46%	18%	20%	10%	+2%
MesaBoogie 550 crunch	29%	31%	13%	16%	2%	+3%
MesaBoogie 550 disto	32%	37%	18%	36%	5%	+18%
Ibanez TSA15 crunch	25%	31%	7%	70%	6%	+63%
MesaBoogie Mark V clean	4%	7%	4%	21%	3%	+17%
MesaBoogie Mark V crunch	13%	23%	46%	27%	10%	-19%
MesaBoogie Mark V disto	21%	25%	18%	18%	4%	0%
Blackstar HT5M	28%	39%	27%	62%	11%	+35%

The DNN model is better (i.e., it is perceived as more accurate) than the LSTM model if $\Delta_{PAD} < 0$. Moreover, if Δ_{PAD} is close to 0, the use of the DNN model may be preferred since it is still 2.5 faster than the LSTM model. This is the case for the amplifiers: *Engl retro tube disto*, *MesaBoogie 550 clean*, *MesaBoogie 550 crunch*, *MesaBoogie Mark V crunch*, *MesaBoogie Mark V disto*.

4.4.3 Choosing an appropriate objective function during the training phase

The two listening tests have shown that a PI based on the NRMSE is not strongly correlated with the perceptual accuracy of a model. This implies that:

- it is not appropriate to compare the perceptual quality of two different models based only on their NMRSE,
- the maximum PI respecting a chosen PAD can be studied in order to determine the fastest model to use. However, this maximum PI is still specific to a given model and to the chosen amplifier.

The optimized model (i.e., the fastest model respecting a given PAD) can be selected by using the following method:

1. For a given amplifier, compare the PAD of different model structures using the *Hardest* test as defined in Sec. 4.4.2.
2. Once the model type is selected, use a *Progressive* test to find the maximum PI verifying a chosen PAD as defined in Sec. 4.4.1. The models corresponding to the different PI thresholds can be selected by stopping the training phase when these PI are reached.
3. Compute an optimized model verifying that maximum PI but using a smaller number of parameters to speed up the CT of the model.

As it can be seen, computing an optimized model requires a lot of listening tests which is not ideal since it takes a lot of time and human resources. This is the result of an "imperfect" choice of the objective function. As described in Sec. 2.8.1, the PI does not correspond perfectly to the objective of the model, i.e., having the best perceptual accuracy as possible. Unfortunately, finding an objective measure that could be related to the subjective accuracy of a model is extremely difficult and could be the topic of research of an entire project.

Some systems, such as the Perceptual Evaluation of Audio quality (PEAQ) [Thiede et al., 2000; Zheng et al., 2012] and the Perceptual Objective Listening Quality Analysis (POLQA) [Beerends et al., 2013] try to give a PI score which tends to be correlated with the perceived difference between two audio signals. This PI is called the Objective Difference Grade (ODG). Although these systems seem well suited to measure the perceived accuracy without using listening tests, their reliability depends on the type of the tested signals [Zaunschirm et al., 2014; Creusere et al., 2008] as well as the type of differences that have to be retrieved (small distortion, crackling noise, white noise, harmonic content).

Indeed, this algorithm is based on the computation of audio features which are then combined together using a small feedforward neural network. The goal of the network is to match the subjective score given by listeners with the objective score computed using the audio features. The resulting model depends on the used dataset as well as the kind of audible

differences that were presented to listeners. Actually, this method was first developed to quantify the perceptual accuracy of audio-codecs. In other words, this method should be able to detect small distortions. Consequently, this objective is significantly too different from ours to give accurate results since our dataset is constituted of strong distorted signals. The context being different, the performance of this method is not guaranteed.

A test of the PEAQ system using the Matlab implementation from [Kabal et al., 2003] on our guitar signals is presented in Fig.4.52. The ODG is given by the PEAQ algorithm when it compares the target signal of the MesaBoogie MarkV Disto with it prediction at different levels of accuracy (NRMSE). Obviously, the ODG values are not correlated to the subjective evaluation of these signals (presented in Fig. 4.49) since the best PAD value is obtained with the PI=20% while the PEAQ evaluation associates the worst ODG value to this PI (as presented in Fig. 4.52). Moreover, the use of PEAQ algorithm in the training phase of the neural network is not trivial. Indeed, PEAQ recommends to use signal lengths comprised between 10 to 20 seconds in order to give a correct ODG score. This is a huge amount of data if it has to be used with the back-propagation algorithm [Chauvin and Rumelhart, 1995] during the training phase of the neural network. Therefore, we are not recommending the use of PEAQ as is, but we recommend to keep the NRMSE PI score, knowing that the comparison between models will not be straightforward. As future works, we propose to investigate other PI such as the Power Spectrum Model (PSM) and the Envelope Power-Spectrum Model (EPSM) [Biberger and Ewert, 2016], since they can also be used to account for psychoacoustical effects such as spectral masking and amplitude modulation masking.

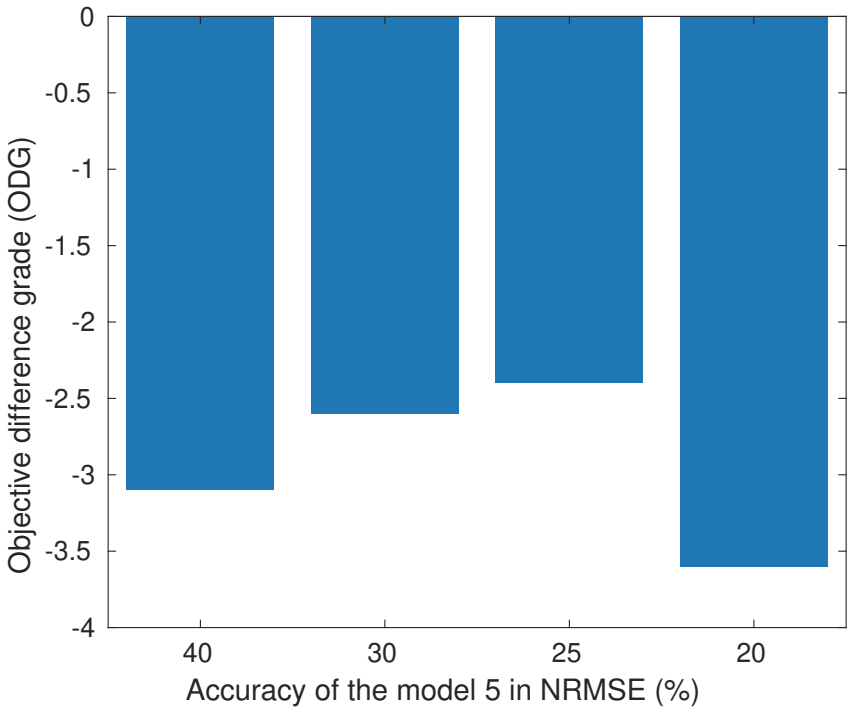


Figure 4.52 Objective Difference Grade given by PEAQ algorithm for the LSTM model having different NRMSE (Black Star amplifier).

Chapter 5

Conclusion and prospects

Contents

This research aimed to *accurately emulate in real time the nonlinear audio devices present in a guitar chain*.

5.1 Summary of work

Such digital emulations exist since many years, but their objective accuracy is hard to prove. Most existing methods are developed for a commercial use and their underlying technology is often kept secret. Moreover, their lack of perceptual accuracy is often reported by musicians. The objectives of this thesis can be summarized as follows:

- Improving the identification of nonlinear audio systems through black-box methods.
- Applying these methods to the emulation of the guitar signal chain while keeping the real-time constraint as low as possible.
- Providing objective and subjective measures of the emulation accuracy.

The results presented in this thesis are based on objective and subjective measures enabling the comparison in the time and the frequency domains of different models. The objective measures,

defined in Sec.2.8.1, are: the NMRSE, the SNR and the difference of harmonic content. While a perceptual measure based on the results of listening tests, i.e., the Probability of Audible Difference (PAD), is presented in Sec.4.4.
Two methods have been selected in response the nonlinear emulation of audio devices challenge:

- The first one (described in Chap. 3) is an improvement of an analytical method, originally developed in [Farina et al., 2001] which uses a sub-model of the Volterra series.
- The second method (described in Chap. 4) is original in the field of guitar chain emulation and is based on neural networks.

These two methods push the state of the art further, if compared with the reviews in [Pakarinen and Yeh, 2009; Oliveira et al., 2013].

As a first approach, a model based on the Volterra series has been selected. Indeed, the large class of nonlinear systems it can represent makes it suitable to model nonlinear audio devices. However, when the order of the system increases, the number of parameters increase exponentially, making difficult its real-time implementation. A subclass of this model, i.e., the polynomial parallel cascade of Hammerstein models, has been studied in Chap. 3. The possible sources of errors/limitations inherent to the method and the way to solve or circumvent them are studied in Sec. 3.3. The computation of the Hammerstein kernels (using the different corrections proposed) is extended to any order in Sec. 3.2.3. The final method, called the Hammerstein Kernels Identification by Sine Sweep (HKISS) method presents very accurate results. Some examples of nonlinear systems emulated through the HKISS method are presented in Sec. 3.4, the harmonic content accuracy measured at 1000 Hz is inferior to 0.3 dB for the 3 tested devices which is very accurate. The principal remaining limitation of the model (presented in Sec. 3.5.3) is that the accuracy of the model is no longer guaranteed if the amplitude of the signal to be emulated is different from the one selected during the measurement of the Hammerstein kernels. Nevertheless, we also have proposed a solution to this problem by introducing an appropriate amplitude function which changes the relative amplitudes of the Hammerstein kernels. This solution is adequate when the nonlinear system to model is close to an Hammerstein model. However, for nonlinear system too far from this ideal, another model should be used.

This leads us to consider more complex models by using a neural networks approach. Indeed, the results obtained with a recurrent neural network for this regression task were so promising that we have next focused on these kinds of identification procedures. One major advantage of neural networks is that the model can be trained using specific data related to the application. These data can be chosen close to the signal to emulate. For example, the neural networks have been trained using some guitar signals (see Appendix A.3) leading to models

that better generalize to these kinds of signals. A first neural network using Long Short Term Memory (LSTM) cells is presented in Sec. 4.2.1. This first network is already very accurate. It has been improved in terms of computational time, flexibility, accuracy through seven other models. In particular, the introduction of model 5 in Sec. 4.2.5 has enabled the emulation of tube amplifiers by means of LSTM neural networks with a very low latency. These models are compared in terms of computational speed and measured accuracy in Sec. 4.3 and two of them are compared in terms of perceptual accuracy in Sec. 4.4 for different guitar amplifiers. The Normalized Root Mean Square Error (NRMSE) and the Probability of Audible Difference (PAD) for these different amplifiers vary from 4 to 36% and from 4 to 46% respectively for the LSTMConv model 5. The DNN model 6 is the fastest model (processing a buffer of 400 samples in 2.2 ms) although its NRMSE and PAD vary from 7 to 46% and from 16 to 70% respectively. The harmonic content accuracy of the neural network models is lower than the harmonic content accuracy of the Hammerstein model, i.e., approximately 4 dB between the harmonic amplitude levels of the prediction and the target for the neural network models depending on the amplifier and on the model selected and less than 0.5 dB for the Hammerstein method (with a constant amplitude input signal). However, the neural network models generalize better to guitar signals (non constant amplitude level signals).

5.2 Future works

The Hammerstein kernels identification method is very accurate for sine signals. However, for signals with a variable amplitude, the prediction can deviate from the target. This method could be improved as follows:

- A criterion on the applicability of the method is needed. The criterion could be based on both the amplitude variability of the input signal and the Hammerstein-like behavior of the system to be modeled. The latter could be measured by the difference between the Hammerstein kernels evaluated for ESS of different amplitudes.
- The interpolation function between the Hammerstein kernels $H_m^{\alpha}(\omega)$ for different amplitudes of the ESS should be further developed.
- Finally, for models being too far from an ideal Hammerstein model, a truncated Volterra series still could be considered. Indeed, the use of modern GPU can help to compute the convolutions of the different kernels with the input signal since the convolution operation lends itself very well to parallelization. The HKISS method could be used to have a fast

first estimation of the main diagonal of the Volterra kernels; then, the flexibility of the model could be increased by adding some sub-diagonals terms to the Volterra kernels, until the desired accuracy is reached.

Neural network models can benefit from all the tremendous research activity of this field to improve the efficiency of the neural network itself: new cells, new regularization techniques, better initialization methods and better optimizers. However, from an acoustical point of view, the choice of the objective function seems to be a critical point. In Chap. 4, several performance indices have been used (i.e., the NRMSE, the SNR, the harmonic content accuracy) but none of them is perfectly correlated to the perceptual accuracy (measured in PAD). This is a fundamental barrier to the progress of neural networks for the emulation of audio devices. Moreover, the models cannot easily be compared to each other without listening tests which are very time consuming. These problems should be further studied by:

- Computing an evaluation function whose performance index is correlated to the PAD. This can be the task of a second neural network, taking some features as inputs and some PAD as targets. The features would be computed based on the difference between the prediction and the target signals of the audio device model. Consequently, once this second neural network is trained, it should be able to make a prediction of the PAD. This would enable the comparison of different models of an audio device directly in terms of PAD which is the real objective of their emulation.

- Choosing a better objective function (usually called the *loss* function) to use during the training phase of the neural network. Usually, for regression tasks, the mean square error is taken. But some other functions such as the Power Spectrum Model (PSM) and the Envelope Power-Spectrum Model (EPSM) [Biberger and Ewert, 2016] should be tested since their authors claim that these functions are able to take into account some psycho-acoustical effects such as spectral masking and amplitude modulation masking.

5.3 Contributions

The state of the art on the tube amplifier emulations has been pushed further in the following direction:

- The ESS method has been improved into the HKISS method. Indeed, this method simultaneously introduces different corrections that solve specific problems, sometimes underestimated by previous works. In particular, this improvement has enabled the time

comparison between the prediction and the target signals "sample by sample" which was not possible earlier.

- A toolbox allowing the computation of the Hammerstein kernels of an audio device and its emulation through nonlinear convolutions is proposed.
- We have introduced a new method to emulate tube amplifiers by proposing 8 different neural network models.
- Several optimizations on these models have been made to enable the real-time emulation of the guitar signal chain using neural networks and a GPU.
- Objective measures such has the NRMSE, the harmonic content accuracy, the SNR are given for each model, easing the (objective) comparison of the different models.
- A method to compare the accuracy of two models (e.g., LSTM model 5 and DNN model 6) including their perceptual accuracy is proposed.
- Finally, a dataset gathering the output signals of 9 different amplifier channels for different types of input guitar signals is provided. The output of each amplifier is available for ten positions of their *gain* parameter.

Part III

Appendix

Table of Contents

Appendix A

More information about:

A.1 Back propagation algorithm

Without this algorithm developed by [Rumelhart et al., 1988], ANN would probably have stayed in a dark era. To understand its importance, let us assume that a function $f : \mathbb{R}^{100} \rightarrow \mathbb{R}$ has to be modeled. The used neural network is a feed forward neural network composed of two layers of 1000 neurons. The number of weights between the input and the first layer is $N_1 = 100 * 1000$, the number of weights between the first and the second layer is $N_2 = 1000 * 1000$ and the number of weights between the second layer and the output is $N_3 = 1000$. This leads to a total number of weights $N = N_1 + N_2 + N_3 = 1101\ 000$. To compute the gradient of the Cost function C, the algorithm has to compute more than a billion of partial derivatives $\frac{\partial C}{\partial w}$, where w represents all the weights of the network. A simple way to compute the gradient of the cost function relatively to the parameters w_j is to approximate it by:

$$\frac{\partial C}{\partial w_j} \simeq \frac{C(w + \varepsilon e_j) - C(w)}{\varepsilon}, \tag{A.1}$$

where ε is a small positive number and e_j is a unit vector in the direction j. Computing the gradient of the cost function for N weights requires to compute the cost function $N+1$ times called the *forward passes*. The advantage of the back propagation algorithm is that it simultaneously computes all the partial derivatives in just one forward pass. Indeed, the back propagation algorithm provides a fast way to compute an error δ_j^l relative to a neuron j in a layer l and to relate it to the gradient $\frac{\partial C}{\partial w_j}$ of the cost function. Explaining the equations of the back propagation algorithm is beyond the scope of this introduction but understanding its principles gives a better insight into how the neural network can learn from the data. For interested readers, we propose the following references [Nielsen, 2015; Goodfellow et al., 2016,

P.204], where the computation of the gradient using the back propagation algorithm is well explained.

A.2 Overview of the methods improving the learnability of neural networks

Actually, most of the work made in the neural networks field is about improving the learnability of the parameters by introducing some novel methods, such as:

- New network structures
  - Deep Neural Network [Schmidhuber, 2015]
  - Convolutional Neural Network [LeCun et al., 1998]
  - Recurrent Neural Network [Rumelhart et al., 1988; Schmidhuber, 1993]
  - Long Short Term Neural Network [Hochreiter and Schmidhuber, 1997]
- New activation functions:
  - Sigmoid [Goodfellow et al., 2016, p.67]
  - Rectified Linear Unit (ReLU) [Jarrett et al., 2009]
  - Leaky ReLU [Maas et al., 2013]
  - Hyperbolic tangent [Goodfellow et al., 2016, p.195]
  - Exponential Linear Unit (ELU) [Clevert et al., 2015]
- New initialization methods:
  - Xavier and He initialization [Glorot and Bengio, 2010]
- New regularization techniques:
  - Batch normalization [Ioffe and Szegedy, 2015]
  - Dropout [Srivastava et al., 2014]
  - Gradient clipping [Mikolov, 2012; Pascanu et al., 2013]
  - Early Stopping [Goodfellow et al., 2016, p.246]
  - l_1, l_2 regularization [Goodfellow et al., 2016, p.234]

  - Max-Norm regularization [Srebro et al., 2005]
  - Data augmentation [Goodfellow et al., 2016, p.240]

- New gradient descent optimizers:
  - Momentum [Polyak, 1964]
  - Nesterov accelerated gradient (NAG) [Nesterov, 1980; Sutskever et al., 2013]
  - AdaGrad algorithm [Duchi et al., 2011]
  - RMSProp [Tieleman and Hinton, 2012]
  - Adam [Kingma and Ba, 2014]

A.3 Dataset information

This section is largely inspired by our paper [Schmitz and Embrechts, 2018a].

A.3.1 Experimental setup

Five different types of guitar signals have been recorded using a guitar *PRS SE 245* and the *UR22 Steinberg* sound card (44100Hz sample frequency and float 32 bit depth). The 5 resulting musical samples have been concatenated in a *wav* file with a silence of two seconds inserted between them. Then, this *wav* file is sent through several guitar amplifiers (currently : the Ibanez TSA15 channel *Crunch*, the *Engl Retro Tube50* channel *Drive*, the *Mesa Boogie 5-50* channels *Clean*, *Crunch* and *Burn*, the *Mesa Boogie MarkV* channels *Clean*, *Crunch*, *Xtreme* and the *Blackstar HT5 Metal* channel *Disto*). The signal is then taken at the output of the power amplifier stage by a load box (*the Torpedo Reload* TwoNoteEngineering [2011]) reducing its amplitude to a line level. Finally, the load box sent the signal back to the sound card (as presented in Fig. A.1). A description of the 5 musical styles played is given hereafter.

Sample1: chromatic scale

In this sample, each note of the PRS guitar neck is played successively. Starting on the 6th string of the guitar (E low), the notes are played until the 4^{th} fret of the guitar neck. Then, the same pattern is applied from the 5^{th} to the 1^{st} string. When the 1^{st} string is reached, all the fingers are shifted of one fret to the right (as presented in Fig. A.2). This sample ends when the 22^{th} fret of the guitar neck is reached. A *tablature* is provided along with the partition.

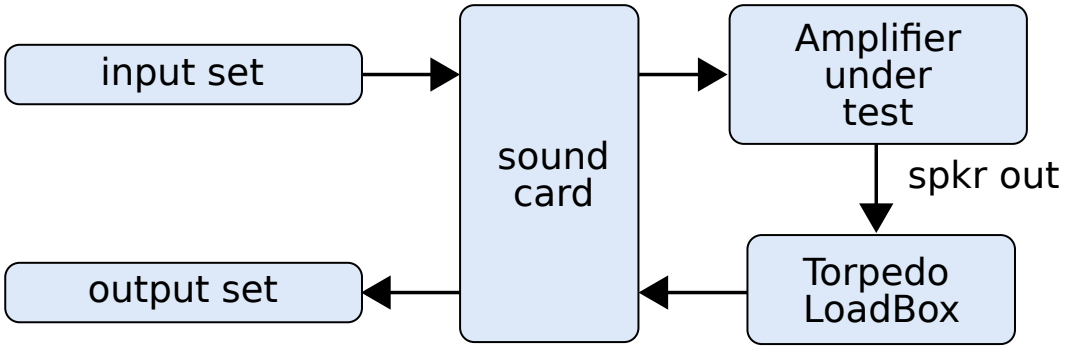


Figure A.1 Recording of the dataset, *the output set* is a scaled version of the output signal of the amplifier (speaker out)

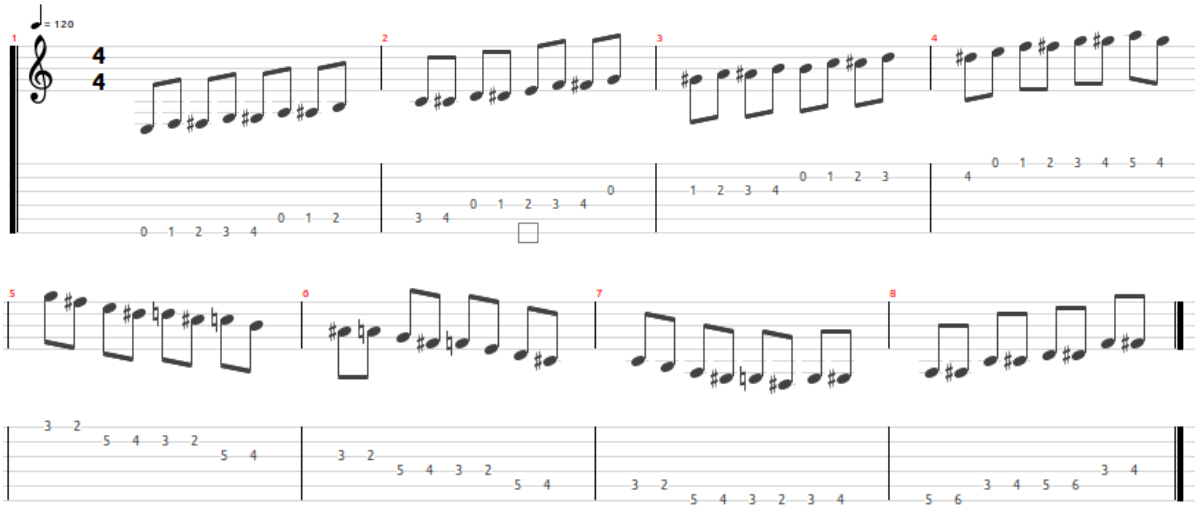


Figure A.2 Chromatic scale: all the notes of the guitar neck are played successively (sample1)

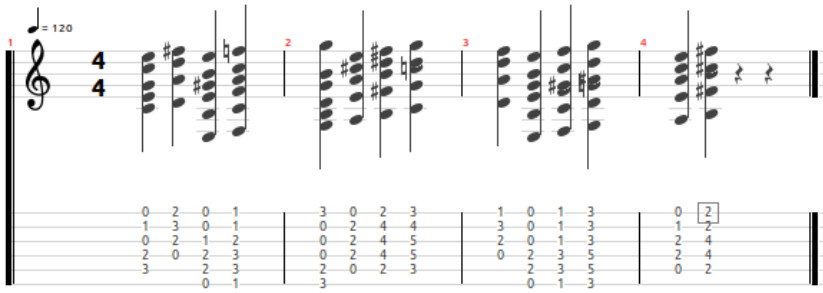


Figure A.3 Chords in Major scale then in Minor scale (sample2)

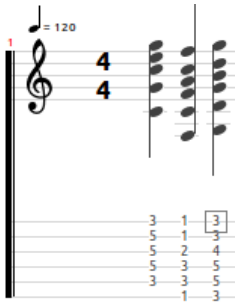


Figure A.4 Chords of the song *La Bamba* (sample3)

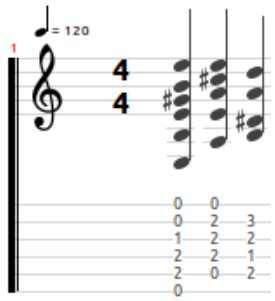


Figure A.5 Chords of a blues song (sample4)

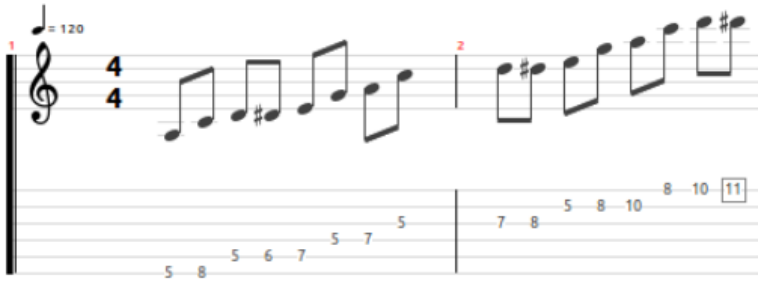


Figure A.6 Am blues scale (sample5)

A tablature depicts the six strings of the guitar. A number i corresponds to the i^{th} fret (see [Wikipedia, 2018] to get further information on *tablature*).

Sample2: chords

A set of chords has been played in this order: C, D, E, F, G, A, B in Major then in Minor. The chords are given in Fig. A.3.

Sample3: Bamba song

This sample is composed of a song of three chords: C, F, G as presented in Fig. A.4.

Sample4: blues song

This sample is composed of a classical blues pattern: E, A, B7 as presented in Fig. A.5.

Sample5: blues scale Am

This sample is a short solo of guitar in Am blues scale as presented in Fig.A.6.

A.3.2 Building a learning dataset

For each amplifier, the five musical samples are sent to the guitar amplifiers and recorded for ten different positions of the *Gain* parameter. A Matlab function is provided to help the user to build its *learning set* (train + test sets) easily:

$$[train, test] = getInOut(modelName, lStyle, lGain, ratio)$$

This function enables the user to choose which sample types, which amplifier and which gain(s) will be in the learning set. It returns two matrices (*train* and *test*) composed of the input sound signal in the first column, its corresponding output sound signal in the second column and the amplifier gain parameter used during the recoding in the third one. The parameters of the function are :

- *modelName*: the name of the *wav* file containing the output signal of a specific amplifier,
- *lStyle*: a list of numbers corresponding to the desired types of musical samples among the onces presented in the previous section (samples 1-5),
- *lGain*: a list of *Gain* parameter values (comprised between 1 and 10,
- *ratio*: the ratio of the length of the *training* set to the *test* set.

Example:

$$[train, test] = getInOut('Engl.wav', [5,3], [1,10], 0.8)$$

returns two matrices containing the concatenation of the sample 5 and 3 with the gain parameter 1 and 10 as it can be seen in Fig. A.7. The *train* matrix is used to train the model while the *test* matrix is used to evaluate the model on a sequence that has never been seen before. It ensures that the model does not over-fit the dataset. By default, each musical sample is sliced as follows: 80% of its content is sent to the *training* set while the remaining 20% are placed in the *test* set. The slices are chosen randomly under the constraint that they remain the same each time the user call the function *getInOut*.

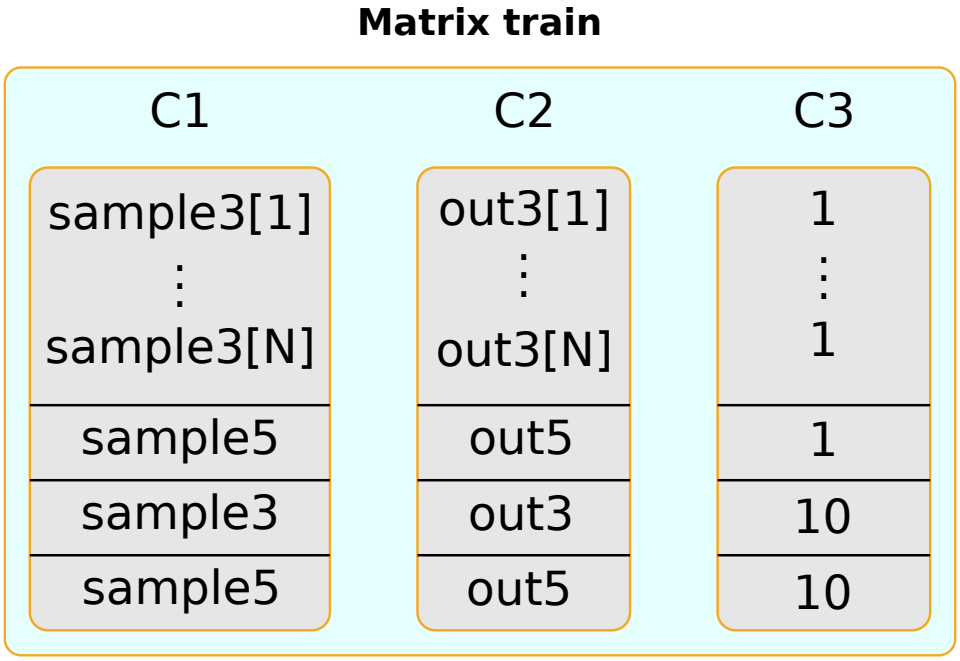


Figure A.7 The matrices *Train* or *Test* contain the desired musical input samples in the first column (C1), their corresponding output signals from the amplifier in the second column (C2). The gain parameter values used during the recording is in the third column (C3).

A.4 Vanishing or exploding gradient problem

In a recurrent neural network, it turns out that the gradient can vanish or become unstable along the different layers of the network [Hochreiter et al., 2001].

- A *vanishing* gradient occurs when the gradient decreases too much in the earlier layers. The former layers stop to learn.
- An exploding gradient occurs when the gradient increases too much in the former layers. The gradient becomes unstable.

The following example helps to understand the core of the problem. Supposing a deep neural network composed of H hidden layers with only one neuron in each layer with a sigmoid σ activation function and using a standard initialization of weights (zero mean and a standard deviation of 1). The repercussion of the variation of a bias b_l in the cost function C can be approximated by [Nielsen, 2015, Eq. (120)]:

$$\frac{\Delta C}{\Delta b_l} \simeq \frac{\partial \sigma(z_l)}{\partial b_l} w_l \cdot \frac{\partial \sigma(z_{l+1})}{\partial b_{l+1}} w_{l+1} \cdot \ldots \cdot \frac{\partial \sigma(z_H)}{\partial b_H} w_H \cdot \frac{\partial C}{\partial a_H}, \tag{A.2}$$

where z_l is the weighted input of the neuron in the layer l:

$$z_l = w_l a_{l-1} + b_l \tag{A.3}$$

and $a_l = \sigma(z_l)$ is the output of the neuron in the layer l. The sigmoid activation function is given by:

$$\sigma(z) = \frac{1}{1+e^{-z}}. \tag{A.4}$$

Consequently, its derivative is:

$$\frac{d\sigma(z)}{dz} = \frac{e^z}{(1+e^z)^2}, \tag{A.5}$$

which has a maximum in z=0 such that:

$$\frac{d\sigma(0)}{dz} = 0.25 \tag{A.6}$$

The gaussian initialization of the weights gives weights which globally satisfy $|w_h| < 1$ such that:

$$\left| \frac{\partial \sigma(z_h)}{\partial z} w_h \right| < 0.25 \ \forall h \in [1, H]. \tag{A.7}$$

Therefore, analyzing Eq. A.7 and Eq. A.2, it can be seen that the variation of the cost function in regards of the variation of a bias b_l situated in a layer l tends to decrease when the number of layers between the layer l and the output layer H increases. Similarly, an exploding gradient occurs if $\left| \frac{\partial \sigma(z_h)}{\partial z} w_h \right| > 1$.

The only way that the gradient does not explode or vanish is that all the product terms are close to balancing out. Several techniques have been created to obtain such results [Glorot and Bengio, 2010], such as: proper initialization, non-saturating function, batch normalization.

A.5 Structure of the neural network models in Tensorflow

During this thesis, we have chosen to build our neural network models with the `Tensorflow` framework [Abadi et al., 2015].

Tensorflow

"TensorFlow is a machine learning system that operates at large scale and in heterogeneous environments. TensorFlow uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. It maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, generalpurpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs). This architecture gives flexibility to the application developer: whereas in previous "parameter server" designs the management of shared state is built into the system, TensorFlow enables developers to experiment with novel optimizations and training algorithms. TensorFlow supports a variety of applications, with a focus on training and inference on deep neural networks. Several Google services use TensorFlow in production, we have released it as an open-source project, and it has become widely used for machine learning research. In this paper, we describe the TensorFlow dataflow model and demonstrate the compelling performance that TensorFlow achieves for several real-world applications."[Abadi et al., 2015]

In this section, the code of the neural network models [1-8] is provided, Tensorflow models are build in the form of dataflow graphs:

- The first step is to describe the graph of the model.
- The second step is to feed the input nodes of the graph with data.
- The third step is to compute the gradient of the cost function and to update the weights of the network to minimize that cost function.
- The fourth step is to use the network for inference.

The first step is described here, the entire code of the models can be found in [Schmitz, 2019b].

A.5.1 Notation and basic informations:

- In the codes below, the symbol "\" cut a line too long into two lines.
- In Tensorflow, the input nodes are specials and are called *placeholders*.
- In python, the length of an element can be *None*, it means that the size will be defined by the data shape fed into this node, this is useful since the variables of the model can have

different sizes without the need to compute several models, e.g., a placeholder can have a a size given by the minibatch size during the training phase (depending of GPU capacity) and another size during inference, e.g., the buffer length of the sound card.

A.5.2 Code defining the structure of the graph for model 1

```
##############################
# Model parameters
##############################
num_step = 100
num_hidden = 150
num_out = 1
num_feature = 1
batch_size = 1000

#######################
# Graph Construction
#######################
G = tf.Graph()
with G.as_default():
  with tf.name_scope("placeHolder"):
    data = tf.placeholder(tf.float32, [None, num_step],\
      name ="data")
    target = tf.placeholder(tf.float32, [None, num_out],\
          name = "target")
    dataShaped = tf.reshape(data,[tf.shape(data)[0],\
      tf.shape(data)[1],num_feature])
    dataShaped = tf.transpose(dataShaped,[1,0,2])

  with tf.name_scope("LSTMLayer"):
    fusedCell = LSTMBlockFusedCell(num_hidden,
      use_peephole=False)
    val, state = fusedCell(dataShaped,dtype=tf.float32)

  with tf.name_scope("extractLastCelloftheLSTMLayer"):
    last_index = tf.shape(val)[0] - 1
    lastState = tf.gather(val,last_index)
```

```
32
33  prediction = fully_connected(lastState ,\
34    int(target.get_shape()[1]), activation_fn=tf.nn.tanh ,\
35    weights_regularizer=None, scope="FCPred")
```

Code A.1 for model 1

In the code the subscript are not allowed, therefore, some variable names had to be replaced. Their equivalent with the rest of the text (see Chap. 4 is given here:

- num_step is the number of input time steps N,
- num_hidden is the number of hidden units N_h in each LSTM cell, it corresponds to the length of the state vectors,
- num_out is the number of output samples for each instance,
- num_feature is the input size of each LSTM cell,
- in the code, the term batch_size refers to the number of instances treated in parallel by the GPU, indeed it refers to the *minibatch size* that is often shorten into *batch size*.

Comments:

- Line 16: *data* is the input node of shape $batch\_size \times num\_step$.
- Line 18: *target* is the output value at the amplifier $y[n]$ of shape $batch\_size \times 1$.
- Line 20, 22: the LSTM layer has to receive data shaped into $num\_step \times batch\_size \times num\_feature$.
- Line 27: the output of the entire LSTM layer is given by a tuple (*val*, *state*). *val* is the state vector h and *state* gathers the state vector h and the long term state vector c.
- Lines 29, 30: *val* gathers all the state vectors (i.e., one for each LSTM cell), only the last one is taken to compute the prediction $\hat{y}[n]$,
- Line 33: a fully connected layer maps the $lastState$ of shape $batch\_size \times num\_hidden$ into the *prediction* vector of shape $batch\_size \times num\_out$.

A.5.3 Code defining the structure of the graph for model 2

```
##############################
# Model parameters
##############################
num_step = 100
num_hidden = 150
num_out = 1
num_feature = 1
batch_size = 1000

#######################
# Graph Construction
#######################
G = tf.Graph()
with G.as_default():
  with tf.name_scope("placeHolder"):
    data = tf.placeholder(tf.float32,\
      [None, num_step], name ="data")
    target = tf.placeholder(tf.float32, [None, num_out],\
      name = "target")
    dataShaped = tf.reshape(data,[tf.shape(data)[0],\
      tf.shape(data)[1],num_feature])
    dataShaped = tf.transpose(dataShaped,[1,0,2])

  with tf.name_scope("LSTMLayer1"):
    fusedCell1 = tf.contrib.rnn.LSTMBlockFusedCell(num_hidden,\
      use_peephole=False)
    val, state = fusedCell1(dataShaped,dtype=tf.float32)
    with tf.name_scope("LSTMLayer2"):
        fusedCell2 = LSTMBlockFusedCell(num_hidden,\
          use_peephole=False)
        val2, state2 = fusedCell2(val,dtype=tf.float32)
  with tf.name_scope("extractLastCelloftheLSTMLayer"):
    last_index = tf.shape(val2)[0] - 1
    lastState = tf.gather(val2,last_index)

  prediction = fully_connected(lastState,\
```

```
37      int(target.get_shape()[1]),activation_fn=tf.nn.tanh,\
38      weights_regularizer=None,scope="FCPred")
```

Code A.2 for model 2

Refers to Sec. A.5.2 for presentation of variable names.

Comments:

- The only difference with the code from model 1 is for the lines 28-30, where the state vector of the previous LSTM layer is sent into a second LSTM layer. One can notice that the input shape of the second layer is $num\_step \times batch\_size \times num\_hidden$ while the input shape of the first layer was $num\_step \times batch\_size \times num\_feature$. Each element of the state vector can be seen as an input feature of the second LSTM layer.

A.5.4 Code defining the structure of the graph for model 3

```
1  ##############################
2  # Model parameters
3  ##############################
4  num_step = 100
5  num_hidden = 150
6  num_out = 5
7  num_feature = 1
8  batch_size = 1000
9
10 ########################
11 # Graph Construction
12 ########################
13 G = tf.Graph()
14 with G.as_default():
15   with tf.name_scope("placeHolder"):
16     data = tf.placeholder(tf.float32, [None, num_step],\
17       name ="data")
18     target = tf.placeholder(tf.float32, [None, num_out],\
19       name = "target")
20     dataShaped = tf.reshape(data,[tf.shape(data)[0],\
21       tf.shape(data)[1],num_feature])
22     dataShaped = tf.transpose(dataShaped,[1,0,2])
23
```

```
24  with tf.name_scope("LSTMLayer1"):
25    fusedCell1 = LSTMBlockFusedCell(num_hidden,use_peephole=False)
26    val, state = fusedCell1(dataShaped,dtype=tf.float32)
27    last_index = tf.shape(val)[0]-1
28    lastVals = tf.gather(val,\
29      tf.range(last_index-num_out,last_index))
30  with tf.name_scope("ReshapeOutputofLSTMSeq"):
31    stacked_val = tf.reshape(tf.transpose(lastVals,[1,0,2]),\
32      [-1,num_hidden])
33    stacked_out = fully_connected(stacked_val,1,\
34      activation_fn=tf.nn.tanh,\
35      weights_regularizer=None,scope="stackedPred")
36
37  prediction = tf.reshape(stacked_out,[-1,num_out],\
38    name="FCPred")
```

Code A.3 for model 3

Refers to Sec. A.5.2 for presentation of variable names.

Comments:

This model is similar to model 1 except for

- Line 28: instead of taking the output of the last LSTM cell only, *num_out* outputs are taken. *lastVals* has a size given by $num\_out \times batch\_size \times num\_hidden$.
- Line 30: *lastVals* is reshaped into $batch\_size.num\_out \times num\_hidden$.
- Line 32: the *num_hidden* values of the $batch\_size.num\_out$ instances are mapped by a fully connected layer into a node of shape $1 \times batch\_size.num\_out$.
- Line 36: the prediction vector is reshaped into $batch\_size \times num\_out$.
- Lines 30-36 are sort of "trick" to use only one fully connected layer for the *num_out* outputs.

A.5.5 Code defining the structure of the graph for model 4

```
1  ##############################
2  # Model parameters
3  ##############################
```

```
4  num_step = 100
5  num_hidden = 150
6  num_out = 1
7  num_feature = 2
8  batch_size = 1000
9
10 ########################
11 # Graph Construction
12 ########################
13 G = tf.Graph()
14 with G.as_default():
15   with tf.name_scope("placeHolder"):
16     data = tf.placeholder(tf.float32, [None, num_step,\
17       num_feature], name ="data")
18     target = tf.placeholder(tf.float32, [None, num_out],\
19       name = "target")
20     dataShaped = tf.transpose(dataShaped,[1,0,2])
21
22   with tf.name_scope("LSTMLayer"):
23     fusedCell = LSTMBlockFusedCell(num_hidden)
24     val, state = fusedCell(dataShaped,dtype=tf.float32)
25
26   with tf.name_scope("extractLastCelloftheLSTMLayer"):
27     last_index = tf.shape(val)[0] - 1
28     lastState = tf.gather(val,last_index)
29
30   prediction = fully_connected(lastState,\
31     int(target.get_shape()[1]), activation_fn=tf.nn.tanh,\
32     scope="FCPred")
```

Code A.4 for model 4

Refers to Sec. A.5.2 for presentation of variable names.

Comments:

- The code is identical to the code for model 1, except that the input placeholder *data* is of shape $batch\_size \times num\_step \times num\_feature = 2$ (see Line 16) instead of shape $batch\_size \times num\_step$ in model 1. The second input feature is for the gain of the amplifier.

A.5.6 Code defining the structure of the graph for model 5

```
##############################
# Model parameters
##############################
num_step = 150
num_hidden = 150
num_out = 1
num_feature = 1
conv_chan = [35]
conv_strides = [4,3]
conv_size = 12
batch_size = 1000

########################
# Graph Construction
########################
G = tf.Graph()
with G.as_default():
  with tf.name_scope("placeHolder"):
    data = tf.placeholder(tf.float32, [None, num_step],\
      name ="data")
    target = tf.placeholder(tf.float32, [None, num_out],\
      name = "target")
    dataShaped = tf.reshape(data,[tf.shape(data)[0],num_step,1,1])

  with tf.variable_scope("ConvLayers"):
    dataReduced = tf.layers.conv2d(inputs = dataShaped,\
      filters = conv_chan[0], kernel_size = (conv_size,1),\
      strides=(conv_strides[0],1), padding = "same",\
      activation = tf.nn.elu, name = "C1")
    dataReduced = tf.layers.conv2d(inputs = dataReduced,\
      filters = conv_chan[0], kernel_size = (conv_size,1),\
      strides=(conv_strides[1],1), padding = "same",\
      activation = tf.nn.elu, name="C2")
  dataReduced = tf.reshape(dataReduced,[tf.shape(data)[0],\
    tf.shape(dataReduced)[1],conv_chan[0]])

```

```
37  fusedCell = LSTMBlockFusedCell(num_hidden, use_peephole=False)
38  dataReduced = tf.transpose(dataReduced,[1,0,2])
39
40  with tf.name_scope("extractLastValueLSTM"):
41    val, state = fusedCell(dataReduced, dtype=tf.float32)
42    last_index = tf.shape(val)[0] - 1
43    last = tf.gather(val, last_index)
44
45  prediction = fully_connected(last, int(target.get_shape()[1]),\
46    activation_fn=tf.nn.tanh, scope="FCPred")
```

Code A.5 for model 5

Refers to Sec. A.5.2 for presentation of variable names previously presented. The others are:

- *conv_chan* is the number of maps C in a Convolutional Reduction (CR) layer.
- *conv_strides* is a vector that gathers the different stride sizes S of the different CR layers.
- *conv_size* is a vector gathering the lengths of kernels for the CR layers.

Comments:

- Lines 26-29: describe the convolutional reduction (CR) layer. The input tensor should have a shape given by $batch\_size \times length \times width \times depth$ which gives in our case $batch\_size \times num\_step \times 1 \times 1$.
- Lines 30-33: describe the second CR layer, this time, the depth is the number of maps *num_chan* of the first CR layer. The input tensor has shape:

$$batch\_size \times ceil(num\_step/conv\_strides[0]) \times 1 \times num\_chan$$

  and the output tensor has shape:

$$batch\_size \times ceil(ceil(num\_step/conv\_strides[0])/conv\_strides[1]) \times 1 \times num\_chan.$$

- Lines 34-38: reshape the output of the second CR layer into $num\_step \times batch\_size \times num\_chan$ before to send it to the LSTM layer. The rest of the model is then identical to the model 1.

A.5.7 Code defining the structure of the graph for model 6

```
1  ##############################
2  # Model parameters
3  ##############################
4  num_step = 150
5  num_hidden = [1000,500]
6  num_out = 1
7  num_feature = 1
8  batch_size = 1000
9
10 ######################
11 # Graph Construction
12 ######################
13 G = tf.Graph()
14 with G.as_default():
15   with tf.name_scope("placeHolder"):
16     data = tf.placeholder(tf.float32, [None, num_step],\
17       name ="data")
18     target = tf.placeholder(tf.float32, [None, num_out],\
19       name = "target")
20
21   with tf.variable_scope("DNNLayers"):
22     FC1 = fully_connected(data,num_hidden[0],\
23       activation_fn=tf.nn.relu, scope="FC1")
24     FC2 = fully_connected(FC1,num_hidden[1],\
25       activation_fn=tf.nn.relu, scope="FC2")
26
27   prediction = fully_connected(FC2,int(target.get_shape()[1]),\
28     activation_fn=tf.nn.tanh,scope="FCPred")
```

Code A.6 for model 6

Refers to Sec. A.5.2 for presentation of variable names previously presented.

Comments:

- Line 22: maps the input tensor *data* of shape $batch\_size \times num\_step$ to a feed forward layer of 1000 hidden units.
- Line 24: maps the output tensor from layer 1 to a feed forward layer of 500 hidden units.

- Line 27: maps the output tensor from layer 2 into a prediction of shape $batch\_size \times 1$.

A.5.8 Code defining the structure of the graph for model 7

```
1  ##############################
2  # Model parameters
3  ##############################
4  num_step = 150
5  num_hidden = 1000
6  num_out = 1
7  conv_chan = [35 , 70]
8  conv_size = 12
9  size_poll = 3
10 batch_size = 1000
11
12 ########################
13 # Graph Construction
14 ########################
15 G = tf.Graph()
16 with G.as_default():
17   with tf.name_scope("placeHolder"):
18     data = tf.placeholder(tf.float32, [None, num_step],\
19       name ="data")
20     target = tf.placeholder(tf.float32, [None, num_out],\
21       name = "target")
22
23   dataShaped = tf.reshape(data,[tf.shape(data)[0],num_step,1,1])
24   dataShaped = tf.transpose(dataShaped,[0,3,1,2])
25
26   with tf.variable_scope("ConvLayers"):
27     conv1 = tf.layers.conv2d(inputs = dataShaped,\
28       filters = conv_chan[0],kernel_size = (conv_size,1),\
29       strides=(3,1), padding ="same", activation=tf.nn.relu,\
30       data_format='channels_first', name="conv1")
31
32     pool1 = tf.nn.max_pool(conv1, ksize=[1, 1, size_poll, 1],\
33       strides=[1, 1, 2, 1], data_format='NCHW', padding='same')
```

```
34
35    conv2 = tf.layers.conv2d(inputs = pool1,\
36      filters = conv_chan[1], kernel_size = (conv_size/2,1),\
37      strides=(1,1), padding="same", activation=tf.nn.relu,\
38      data_format='channels_first',name="conv2")
39
40    pool2 = tf.nn.max_pool(conv2, ksize=[1, 1,size_poll, 1],\
41      strides=[1, 1, 2, 1],data_format='NCHW', padding='same')
42
43  dataShape = int(np.prod(pool2.get_shape()[1:]))
44  dataReshaped = tf.reshape(pool2, [-1, dataShape])
45  FC1 = fully_connected(dataReshaped, num_hidden,\
46    activation_fn=tf.nn.relu, weights_regularizer=None,scope="FC1")
47
48  prediction = fully_connected(FC1,int(target.get_shape()[1]),\
49    activation_fn=tf.nn.tanh,scope="FCPred")
```

Code A.7 for model 7

Refers to Sec. A.5.2 for presentation of variable names previously presented. The others are:

- *conv_chan* is the number of maps C in the convolutional layers
- *conv_size* is the length of the convolutional kernels.
- *size_poll* is the length of the receptive field where the polling is made.

Comments:

- Line 27: the input shape of the first convolutional layer is $batch\_size \times num\_chan \times num\_step \times num\_feature = 1$. There are $num\_chan[0]$ kernels of length *conv_size* computed with a stride size of 3.
- Line 32: a pooling of size *size_pool* is made along the num_step dimension with a stride size of two.
- Line 35: the second convolutional layer has the same principle than the first one but the length of the kernels are divided by two, the number of maps is multiplied by two and the stride size is equal to 1.

- Line 40: the second pooling layer is the same than the first one. The output of *pool2* has shape $batch\_size \times num\_chan[1] \times N_{out} \times num\_feature = 1$, where N_{out} can be computed as proposed in Eq. (4.12).
- Lines 43, 44: the output of *pool2* is flatten to obtain a shape $batch\_size \times num\_chan[1].N_{out}$
- Line 48: the data are mapped in a prediction vector $\hat{y}[n]$.

A.5.9 Code defining the structure of the graph for model 8

```
1  ##############################
2  # Model parameters
3  ##############################
4  num_step = 150
5  num_hidden = 150
6  num_out = 1
7  num_feature = 1
8  conv_chan = [35]
9  conv_strides = [4,3]
10 conv_size = 12
11 batch_size = 1000
12
13 ########################
14 # Graph Construction
15 ########################
16 G = tf.Graph()
17 with G.as_default():
18   with tf.name_scope("placeHolder"):
19     data = tf.placeholder(tf.float32, [None, num_step],\
20       name ="data")
21     target = tf.placeholder(tf.float32, [None, num_out],\
22       name = "target")
23     dataShaped = tf.reshape(data,[tf.shape(data)[0],num_step,1,1])
24
25   with tf.variable_scope("ConvLayers"):
26     dataReduced = tf.layers.conv2d(inputs = dataShaped,\
27       filters = conv_chan[0], kernel_size = (conv_size,1),\
28       strides=(conv_strides[0],1), padding = "same",\
```

```
29        activation = tf.nn.elu, name = "C1")
30      dataReduced = tf.layers.conv2d(inputs = dataReduced,\
31        filters = conv_chan[0], kernel_size = (conv_size,1),\
32        strides=(conv_strides[1],1), padding = "same",\
33        activation = tf.nn.elu, name="C2")
34    dataReduced = tf.reshape(dataReduced,[tf.shape(data)[0],\
35      tf.shape(dataReduced)[1],conv_chan[0]])
36
37    RNN = tf.keras.layers.SimpleRNN(num_hidden, activation='tanh',\
38      return_sequences=False, return_state=False,unroll=True)
39
40    with tf.name_scope("extractLastValueLSTM"):
41      val = RNN(dataReduced)
42
43    prediction = fully_connected(val,int(target.get_shape()[1]),\
44      activation_fn=tf.nn.tanh,scope="FCPred")
```

Code A.8 for model 5

Refers to Sec. A.5.2 for presentation of variable names previously presented. The others are:

- *conv_chan* is the number of maps C in a Convolutional Reduction (CR) layer.
- *conv_strides* is a vector gathering the different stride sizes S of the different CR layers.
- *conv_size* is a vector gathering the lengths of the kernels for the different CR layers.

Comments:

The code is the same than for model 5 except for:

- Lines 37: the function simpleRNN defines a simple RNN network, the parameter *return_sequences=False* enables to return the last state vector of the *num_step* RNN cells.
- Line 41: uses the RNN by sending to it a tensor of shape $batch\_size \times num\_step \times conv\_chan$. It returns the last state of the RNN of shape $batch\_size \times num\_hidden$.

Appendix B

Additional developments for the third chapter

B.1 Development of Eq. (3.7)

The phase grows exponentially as:

$$\phi[n] = \omega_1 \frac{R}{f_s}.(e^{\frac{n}{R}} - 1) \qquad \forall n \in [0, N-1]. \tag{B.1}$$

Assuming that:

$$\begin{cases} B = \dfrac{\omega_1.R}{f_s}, \\ \Delta_m = R.\log(m). \end{cases} \tag{B.2}$$

A multiple m of the phase can be written such that:

$$\begin{aligned} m.\phi[n] &= m.B(e^{\frac{n}{R}} - 1) \\ &= B\left(e^{\frac{n}{R}} e^{\log(m)} - m\right) \\ &= B\left(e^{\frac{n+R.log(m)}{R}} - m\right) \\ &= B\left(e^{\frac{n+\Delta_m}{R}} - m\right) + B - B \\ &= B\left(e^{\frac{n+\Delta_m}{R}} - 1\right) - B(m-1) \\ &= \phi[n+\Delta_m] - B(m-1). \end{aligned} \tag{B.3}$$

B.2 Development of Eq. (3.9)

Considering the following ESS with a unit amplitude:

$$\begin{aligned} ss[n] &= \sin(\phi[n]) \\ &= \sin\left(\omega_1 \frac{R}{f_s}\left(e^{\frac{n}{R}} - 1\right)\right), \end{aligned} \tag{B.4}$$

with $R = (N-1).log\left(\frac{\omega_2}{\omega_1}\right)^{-1}$. In the following, it is more convenient to consider the sine sweep in the continuous time domain, such that:

$$\begin{cases} ss(t) = A\sin\left(\omega_1 L\left(e^{\frac{t}{L}} - 1\right)\right) \\ \text{with } L = \dfrac{R}{f_s}, \end{cases} \tag{B.5}$$

Let the analytic signal z(t) of ss(t) be:

$$\begin{aligned} z(t) &\triangleq -\mathscr{H}\{ss(t)\} + jss(t) \\ &= cs(t) + jss(t) = es(t) \\ &= e^{j\omega_1 L\left(e^{\frac{t}{L}}-1\right)}. \end{aligned} \tag{B.6}$$

where:

- the *Hilbert Transform* $\mathscr{H}\{.\}$ [Zwillinger, 2002; Qin et al., 2008] has the following property:
$$\mathscr{H}\{\sin(u(t))\} = \sin(u(t)) * \hbar(t) = -\cos(u(t)), \tag{B.7}$$
with $\hbar(t) = \frac{1}{\pi t}$.
- $cs(t) = \cos(\phi(t)) = \cos\left(\omega_1 L\left(e^{\frac{t}{L}} - 1\right)\right)$.

The Fourier transform of the analytic signal can then be written as:

$$\begin{aligned} Z(\omega) &= \int_{-\infty}^{+\infty} e^{j\omega_1 L\left(e^{\frac{t}{L}}-1\right)} e^{-j\omega t} dt \\ &= \int_{-\infty}^{+\infty} e^{j\left(\omega_1 L\left(e^{\frac{t}{L}}-1\right)-\omega t\right)} dt \\ &= e^{-j\omega_1 L} \int_{-\infty}^{+\infty} e^{j\zeta(t)} dt, \end{aligned} \tag{B.8}$$

where:

$$\zeta(t) = \omega_1 L e^{\frac{t}{L}} - \omega t.$$

The approximation of the integral has been analytically computed in [Novak et al., 2015] using Taylor series and the *stationary phase approximation* [Courant and Hilbert, 1953]. The Fourier transform of the sine sweep analytic signal is then given by:

$$\begin{aligned} Z(\omega) &= e^{jL\left(\omega\left(1-\log\left(\frac{\omega}{\omega_1}\right)\right)-\omega_1\right)}\sqrt{\frac{2\pi L}{\omega}}e^{j\frac{\pi}{4}} \\ &= \sqrt{\frac{2\pi L}{\omega}}e^{j\left[L\left(\omega\left(1-\log\left(\frac{\omega}{\omega_1}\right)\right)-\omega_1\right)+\frac{\pi}{4}\right]}. \end{aligned} \tag{B.9}$$

It is also recalled [Novak et al., 2015] that the Fourier transform of real signal and the Fourier transform of its analytic signal are linked by the following properties:

$$\begin{cases} z(t) \triangleq jss(t) - \mathscr{H}\{ss\}(t) \\ \mathscr{F}\{z\}(\omega) = Z(\omega) = jSS(\omega) - \mathscr{F}\{\mathscr{H}\{ss\}(t)\}(\omega), \end{cases} \tag{B.10}$$

where:

$$\mathscr{F}\{\mathscr{H}\{ss\}(t)\}(\omega) \triangleq -j\, sgn(\omega)\mathscr{F}\{ss(t)\}(\omega) \tag{B.11}$$

and

$$sgn(\omega) = \begin{cases} 1, \text{ if } \omega > 0 \\ 0, \text{ if } \omega = 0 \\ -1, \text{ if } \omega < 0, \end{cases} \tag{B.12}$$

which gives:

$$Z(\omega) = 2j.SS(\omega) \text{ for } \omega > 0. \tag{B.13}$$

Therefore, the Fourier transform of the ESS is equal to:

$$SS(\omega) = \sqrt{\frac{\pi L}{2\omega}}e^{j\left[L\left(\omega\left(1-\log\left(\frac{\omega}{\omega_1}\right)\right)-\omega_1\right)-\frac{\pi}{4}\right]}. \tag{B.14}$$

Finally, the Fourier transform of the inverse ESS is given by:

$$\overline{SS}(\omega) = \frac{1}{SS(\omega)} = \sqrt{\frac{2\omega}{\pi L}}e^{-j\left[L\left(\omega\left(1-\log\left(\frac{\omega}{\omega_1}\right)\right)-\omega_1\right)-\frac{\pi}{4}\right]}. \tag{B.15}$$

B.3 Development of Eq. (3.15)

The output of the nonlinear system is expressed in Eq. (3.14) as:

$$\begin{aligned} y[n] = {} & ss[n] * h_1[n] \\ & + \frac{A}{2}\left(ss[n+\Delta_2]\cos(B) * \hbar[n] - ss[n+\Delta_2]\sin(B)\right) * h_2[n] \\ & + \frac{A^2}{4}\left(3ss[n] - ss[n+\Delta_3]\cos(2B) - ss[n+\Delta_3]\sin(2B) * \hbar[n]\right) * h_3[n]. \end{aligned} \tag{B.16}$$

The transition into the frequency domain is obtained using the property of the Fourier transformation of the Hilbert transform (B.11). The Eq. (B.16) can then be expressed in the frequency domain (for $\omega > 0$) by considering that:

$$\begin{aligned} \mathscr{F}\left\{\mathscr{H}\left\{ss[n+\Delta_2]\right\}\right\}(\omega) &= -j\mathscr{F}\left\{ss[n+\Delta_2]\right\}(\omega) \\ &= -je^{j\omega\Delta_2/f_s}SS(\omega), \end{aligned} \tag{B.17}$$

this gives:

$$\begin{aligned} Y(\omega) = {} & SS(\omega).H_1(\omega) \\ & + \frac{A}{2}\left(SS(\omega)e^{j\omega\Delta_2/f_s}\cos(B).(-j) - SS(\omega)e^{j\omega\Delta_2/f_s}\sin(B)\right).H_2(\omega) \\ & + \frac{A^2}{4}\left(3SS(\omega) - SS(\omega)e^{j\omega\Delta_3/f_s}\cos(2B) - SS(\omega)e^{j\omega\Delta_3/f_s}\sin(2B).(-j)\right).H_3(\omega). \end{aligned} \tag{B.18}$$

Then, using euler formula : $e^{jx} = \cos(x) + j\sin(x)$, Eq. (B.18) becomes:

$$\begin{aligned} Y(\omega) = {} & SS(\omega).H_1(\omega) \\ & - \frac{A}{2}\left(SS(\omega).j.e^{-jB}.e^{\frac{\omega\Delta_2}{f_s}}\right).H_2(\omega) \\ & + \frac{A^2}{4}\left(3.SS(\omega) - SS(\omega).e^{-j2B}e^{\frac{jw\Delta_3}{f_s}}\right).H_3(\omega). \end{aligned} \tag{B.19}$$

B.4 Matrix of phase correction presented in Sec. 3.3.3

The correction to apply to each power of the sweep is immediate if it is noticed that the phase property given in Eq. 3.6 applied to a sine gives:

$$\sin(m.\phi[n]) = \sin(\phi[n+\Delta_m] - B(m-1)). \tag{B.20}$$

In the frequency domain, it can be written:

$$\mathscr{F}\{\sin(m.\phi[n])\} = e^{-jB(m-1)}\mathscr{F}\{\sin(\phi[n+\Delta_m])\}. \tag{B.21}$$

Each power of the ESS given by Eq. 3.24 can then be corrected to take this B factor into account.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Abel, J. S. and Berners, D. P. (2006). A technique for nonlinear system measurement. In *Audio Engineering Society Convention 121*. Audio Engineering Society.

Abramowitz, M. and Stegun, I. A. (1965). *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55. Courier Corporation.

Aiken, R. (2006). What is blocking distortion? *2006, Internet Article*.

Araya, T. and Suyama, A. (1996). Sound effector capable of imparting plural sound effects like distortion and other effects. US Patent 5,570,424.

Ashby, W. R. (1957). An introduction to cybernetics. pages 86–107.

Banks, H. T. and Kunisch, K. (2012). *Estimation techniques for distributed parameter systems*. Springer Science & Business Media.

Barbour, E. (1998). The cool sound of tubes [vacuum tube musical applications]. *IEEE Spectrum*, 35(8):24–35.

Beerends, J. G., Schmidmer, C., Berger, J., Obermann, M., Ullmann, R., Pomy, J., and Keyhl, M. (2013). Perceptual objective listening quality assessment (polqa), the third generation itu-t standard for end-to-end speech quality measurement part i—temporal alignment. *Journal of the Audio Engineering Society*, 61(6):366–384.

Bendat, J. S. (1998). *Nonlinear systems techniques and applications*. Wiley.

Benedetto, S., Biglieri, E., and Daffara, R. (1979). Modeling and performance evaluation of nonlinear satellite links-a volterra series approach. *IEEE Transactions on Aerospace and Electronic Systems*, (4):494–507.

Berners, D. and Abel, J. (2004). Ask the doctors! *Universal Audio WebZine*, 2(6).

Biberger, T. and Ewert, S. D. (2016). Envelope and intensity based prediction of psychoacoustic masking and speech intelligibility. *The Journal of the Acoustical Society of America*, 140(2):1023–1038.

Billings, S. and Leontaritis, I. (1982). Parameter estimation techniques for nonlinear systems. *IFAC Proceedings Volumes*, 15(4):505–510.

Billings, S. A. (1980). Identification of nonlinear systems–a survey. In *IEE Proceedings D (Control Theory and Applications)*, volume 127, pages 272–285. IET.

Billings, S. A. (2013). *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons.

Boyd, S. and Chua, L. (1985). Fading memory and the problem of approximating nonlinear operators with volterra series. *IEEE Transactions on circuits and systems*, 32(11):1150–1161.

Boyd, S. P. (1985). Volterra series: Engineering fundamentals. *Doctoral Thesis, University of California at Berkley*.

Bunge, M. (1963). A general black box theory. *Philosophy of Science*, 30(4):346–358.

Chatterjee, A. (2010). Identification and parameter estimation of a bilinear oscillator using volterra series with harmonic probing. *International Journal of Non-Linear Mechanics*, 45(1):12–20.

Chauvin, Y. and Rumelhart, D.-E. (1995). *Backpropagation: Theory, Architectures, and Applications*. Psychology Press.

Chen, S. and Billings, S. A. (1989). Representations of non-linear systems: the narmax model. *International journal of control*, 49(3):1013–1032.

Cheng, C., Peng, Z., Zhang, W., and Meng, G. (2017). Volterra-series-based nonlinear system modeling and its engineering applications: A state-of-the-art review. *Mechanical Systems and Signal Processing*, 87:340–364.

Chun, C. J., Moon, J. M., Lee, G. W., Kim, N. K., and Kim, H. K. (2017). Deep neural network based hrtf personalization using anthropometric measurements. In *Audio Engineering Society Convention 143*.

Ćirić, D. G., Marković, M., Mijić, M., and Šumarac-Pavlović, D. (2013). On the effects of nonlinearities in room impulse response measurements with exponential sweeps. *Applied Acoustics*, 74(3):375–382.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Cohen, I. and Helie, T. (2009). Simulation of a guitar amplifier stage for several triode models: examination of some relevant phenomena and choice of adapted numerical schemes. In *127th Convention of Audio Engineering Society*.

Courant, R. and Hilbert, D. (1953). Mathematical methods of physics.

Covert, J. and Livingston, D. L. (2013). A vacuum-tube guitar amplifier model using a recurrent neural network. In *2013 Proceedings of IEEE Southeastcon*, pages 1–5. IEEE.

Creusere, C. D., Kallakuri, K. D., and Vanam, R. (Jan 2008). An objective metric of human subjective audio quality optimized for a wide range of audio fidelities. *IEEE Transactions on Audio, Speech, and Language Processing*.

Csáji, B. C. (2001). Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Damskägg, E.-P., Juvela, L., Thuillier, E., and Välimäki, V. (May 2019). Deep learning for tube amplifier emulation. In *International Conference on Acoustics, Speech and Signal Processing*. IEEE.

Dempwolf, K., Holters, M., and Zölzer, U. (2011). A triode model for guitar amplifier simulation with individual parameter fitting. In *Audio Engineering Society Convention 131*. Audio Engineering Society.

Dieudonné, J. (2013). *Foundations of modern analysis*. Read Books Ltd.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Eby, R. L. (1970). Solid state substitute for a dual triode electron tube. US Patent 3,531,654.

Engl (2015). Engl retro tube 50 head amplifier. https://www.englamps.de/portfolio/retro-tube-50-e762/.

Farina, A. (2000). Simultaneous measurement of impulse response and distortion with a swept-sine technique. In *Audio Engineering Society Convention 108*. Audio Engineering Society.

Farina, A., Bellini, A., and Armelloni, E. (2001). Non-linear convolution: A new approach for the auralization of distorting systems. *Journal of the Audio Engineering Society*.

Gelb, A. and Vander Velde, W. E. (1968). Multiple-input describing functions and non-linear system design.

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.".

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Guo, L., Billings, S. A., and Coca, D. (2010). Identification of partial differential equation models for a class of multiscale spatio-temporal dynamical systems. *International Journal of Control*, 83(1):40–48.

Guo, Y., Guo, L., Billings, S., Coca, D., and Lang, Z. Q. (2014). A parametric frequency response method for non-linear time-varying systems. *International Journal of Systems Science*, 45(10):2133–2144.

Gustafsson, F., Connman, P., Oberg, O., Odelholm, N., and Enqvist, M. (2004). System and method for simulation of non-linear audio equipment. US Patent App. 10/872,012.

Harris, F. J. (1978). On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83.

Hélie, T. (2010). Volterra series and state transformation for real-time simulations of audio circuits including saturations: Application to the moog ladder filter. *IEEE transactions on audio, speech, and language processing*, 18(4):747–759.

Hirvonen, T. (2015). Classification of spatial audio location and content using convolutional neural networks. In *Audio Engineering Society Convention 138*.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Holters, M., Corbach, T., and Zölzer, U. (2009). Impulse response measurement techniques and their applicability in the real world. In *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx-09)*.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Hélie, T. and Hasler, M. (2004). Volterra series for solving weakly non-linear partial differential equations: application to a dissipative burgers' equation. *International Journal of Control*, 77(12):1071–1082.

Hélie, T. and Roze, D. (2008). Sound synthesis of a nonlinear string using volterra series. *Journal of Sound and Vibration*, 314(1-2):275–306.

Ibanez (1978). Tube screamer overdrive/distotion pedal effect. https://www.ibanez.com/usa/products/model/tube_screamer/.

Ibanez (2015). Tsa15h ibanez amplifier. https://www.ibanez.com/eu/products/detail/tsa15h_03.html.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Irving, A. (2008). Dynamical hysteresis in communications: A volterra functional approach. *IET Signal Processing*, 2(2):75–86.

Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. (2009). What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision (ICCV)*, pages 2146–2153. IEEE.

Jeffreys, H. and Jeffreys, B. (1999). *Methods of mathematical physics*. Cambridge university press.

Jing, X., Lang, Z., and Billings, S. A. (2009). Determination of the analytical parametric relationship for output spectrum of volterra systems based on its parametric characteristics. *Journal of Mathematical Analysis and Applications*, 351(2):694–706.

Juditsky, A., Hjalmarsson, H., Benveniste, A., Delyon, B., Ljung, L., Sjöberg, J., and Zhang, Q. (1995). Nonlinear black-box models in system identification: Mathematical foundations. *Automatica*, 31(12):1725–1750.

Kaastra, I. and Boyd, M. (1996). Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236.

Kabal, T. et al. (2003). An examination and interpretation of itu-r bs. 1387: Perceptual evaluation of audio quality.

Kadlec, F., Lotton, P., Novak, A., and Simon, L. (2008). A new method for identification of nonlinear systems using miso model with swept-sine technique: Application to loudspeaker analysis. In *Audio Engineering Society Convention 124*. Audio Engineering Society.

Karjalainen, M. and Pakarinen, J. (2006). Wave digital simulation of a vacuum-tube amplifier. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 5, pages V–V. IEEE.

Karlik, B. and Olgac, A. V. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122.

Katayama, T. and Serikawa, M. (1997). Reduction of second order non-linear distortion of a horn loudspeaker by a volterra filter-real-time implementation. In *Audio Engineering Society Convention 103*. Audio Engineering Society.

Kemp, M. J. (2006). Audio effects synthesizer with or without analyzer. US Patent 7,039,194.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirkeby, O. and Nelson, P. A. (1999). Digital filter design for inversion problems in sound reproduction. *Journal of the Audio Engineering Society*, 47(7/8):583–595.

Kon, H. and Koike, H. (2018). Deep neural networks for cross-modal estimations of acoustic reverberation characteristics from two-dimensional images. In *Audio Engineering Society Convention 144*.

Koren, N. (1996). Improved vacuum tube models for spice simulations. *Glass Audio*, 8(5):18–27.

Korenberg, M. J. (1991). Parallel cascade identification and kernel estimation for nonlinear systems. *Annals of biomedical engineering*, 19(4):429–455.

Krantz, S. G. and Parks, H. R. (2002). *A primer of real analytic functions*. Springer Science & Business Media.

Krenz, W. and Stark, L. (1991). Interpretation of functional series expansions. *Annals of biomedical engineering*, 19(4):485–508.

Kuleshov, V., Enam, S. Z., and Ermon, S. (2017). Audio super resolution using neural networks. *CoRR*, abs/1708.00853.

Leach Jr, W. M. (1995). Spice models for vacuum-tube amplifiers. *Journal of the Audio Engineering Society*, 43(3):117–126.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Leontaritis, I. and Billings, S. A. (1985). Input-output parametric models for non-linear systems part i: deterministic non-linear systems. *International journal of control*, 41(2):303–328.

Lester, M. and Boley, J. (2007). The effects of latency on live sound monitoring. In *Audio Engineering Society Convention 123*.

Li, H.-X. and Qi, C. (2010). Modeling of distributed parameter systems for applications—a synthesized review from time–space separation. *Journal of Process Control*, 20(8):891–901.

Lin, J.-N. and Unbehauen, R. (1990). Adaptive nonlinear digital filter with canonical piecewise-linear structure. *IEEE Transactions on Circuits and Systems*, 37(3):347–353.

Ljung, L. (1998). *System identification*. Springer.

Low, S. and Hawksford, M. J. (1993). A neural network approach to the adaptive correction of loudspeaker nonlinearities. In *Audio Engineering Society Convention 95*.

Maachou, A., Malti, R., Melchior, P., Battaglia, J.-L., Oustaloup, A., and Hay, B. (2014). Nonlinear thermal system identification using fractional volterra series. *Control Engineering Practice*, 29:50–60.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3.

Macak, J. and Schimmel, J. (2010). Real-time guitar tube amplifier simulation using an approximation of differential equations. In *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx'10)*.

Mastromauro, R., Liserre, M., and Dell'Aquila, A. (2007). Frequency domain analysis of inductor saturation in current controlled grid converters. In *IECON 2007-33rd Annual Conference of the IEEE Industrial Electronics Society*, pages 1396–1401. IEEE.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Meng, Q., Sen, D., Wang, S., and Hayes, L. (2008). Impulse response measurement with sine sweeps and amplitude modulation schemes. In *2008 2nd International Conference on Signal Processing and Communication Systems*, pages 1–5. IEEE.

Mikolov, T. (2012). Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 80.

Mitchell, T. (1997). Machine learning, mcgraw-hill higher education. *New York*.

Mülling, K., Kober, J., Kroemer, O., and Peters, J. (2013). Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279.

Nesterov, Y. (1980). A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Sov math Dokl*, 27(2).

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:.

Norcross, S. G., Soulodre, G. A., and Lavoie, M. C. (2002). Evaluation of inverse filtering techniques for room/speaker equalization. In *Audio Engineering Society Convention 113*. Audio Engineering Society.

Novak, A., Lotton, P., and Simon, L. (2015). Synchronized swept-sine: Theory, application, and implementation. *Journal of the Audio Engineering Society*, 63(10):786–798.

Novak, A., Simon, L., Kadlec, F., and Lotton, P. (2009). Nonlinear system identification using exponential swept-sine signal. *IEEE Transactions on Instrumentation and Measurement*, 59(8):2220–2229.

Novak, A., Simon, L., and Lotton, P. (2010a). Analysis, synthesis, and classification of nonlinear systems using synchronized swept-sine method for audio effects. *EURASIP Journal on Advances in Signal Processing*, 2010(1):793816.

Novak, A., Simon, L., Lotton, P., and Gilbert, J. (2010b). Chebyshev model and synchronized swept sine method in nonlinear audio effect modeling. In *Proc. 13th Int. Conference on Digital Audio Effects (DAFx-10)*.

Nugraha, A. A., Liutkus, A., and Vincent, E. (2016). Multichannel audio source separation with deep neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(9):1652–1664.

Ogunfunmi, T. (2007). *Adaptive nonlinear system identification: The Volterra and Wiener model approaches*. Springer Science & Business Media.

Oliveira, T. C. d. A., Barreto, G., and Pasqual, A. M. (2013). Review of digital emulation of vacuum-tube audio amplifiers and recent advances in related virtual analog models. *INFOCOMP*, 12(1):10–23.

Oppenheim, A. V. and Schafer, R. W. (2004). From frequency to quefrency: A history of the cepstrum. *IEEE signal processing Magazine*, 21(5):95–106.

Oppenheim, A. V., Willsky, A. S., and Nawab, S. (1996). *Signals and Systems (Prentice-Hall signal processing series)*. Prentice Hall Englewood Cliffs, New Jersey.

Orcioni, S. (2014). Improving the approximation ability of volterra series identified with a cross-correlation method. *Nonlinear Dynamics*, 78(4):2861–2869.

Orcioni, S., Carini, A., Cecchi, S., Terenzi, A., and Piazza, F. (2018). Identification of nonlinear audio devices exploiting multiple-variance method and perfect sequences. In *Audio Engineering Society Convention 144*. Audio Engineering Society.

Pakarinen, J. and Karjalainen, M. (2010). Enhanced wave digital triode model for real-time tube amplifier emulation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(4):738–746.

Pakarinen, J. and Yeh, D. T. (2009). A review of digital techniques for modeling vacuum-tube guitar amplifiers. *Computer Music Journal*, 33(2):85–100.

Palm, G. (1979). On representation and approximation of nonlinear systems. *Biological Cybernetics*, 34(1):49–52.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.

Peng, Z., Lang, Z., Billings, S., and Lu, Y. (2007a). Analysis of bilinear oscillators under harmonic loading using nonlinear output frequency response functions. *International Journal of Mechanical Sciences*, 49(11):1213–1225.

Peng, Z., Lang, Z.-Q., and Billings, S. (2007b). Resonances and resonant frequencies for a class of nonlinear systems. *Journal of Sound and Vibration*, 300(3-5):993–1014.

Pfeiffer, R. R. (1970). A model for two-tone inhibition of single cochlear-nerve fibers. *The Journal of the Acoustical Society of America*, 48(6B):1373–1378.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Qin, S., Qin, Y., and Mao, Y. (2008). Accurate instantaneous frequency estimation with iterated hilbert transform and its application. In *Proceedings of the 7th WSEAS International Conference on Signal Processing, Robotics and Automation*, pages 165–170. World Scientific and Engineering Academy and Society (WSEAS).

Rajbman, N. (1976). The application of identification methods in the ussr—a survey. *Automatica*, 12(1):73–95.

Ran, Q., Xiao, M. L., and Hu, Y. X. (2014). Nonlinear vibration with volterra series method used in civil engineering: The bouc-wen hysteresis model of generalized frequency response. In *Applied Mechanics and Materials*, volume 530, pages 561–566. Trans Tech Publ.

Rébillat, M., Hennequin, R., Corteel, E., and Katz, B. F. (2011). Identification of cascade of hammerstein models for the description of nonlinearities in vibrating devices. *Journal of sound and vibration*, 330(5):1018–1038.

Rec, I. (2015). Bs. 1116-3: Methods for the subjective assessment of small impairments in audio systems.

Reed, M. and Hawksford, M. (1996). Identification of discrete volterra series using maximum length sequences. *IEE Proceedings-Circuits, Devices and Systems*, 143(5):241–248.

Reiss, J. D. and McPherson, A. (2014). *Audio effects: theory, implementation and application*. CRC Press.

Rudin, W. et al. (1964). *Principles of mathematical analysis*, volume 3. McGraw-hill New York.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Schmidhuber, J. (1993). Habilitation thesis: System modeling and optimization. *Page 150 demonstrates credit assignment across the equivalent of 1,200 layers in an unfolded RNN.*

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

Schmitz, T. (2012). Modélisation du comportement non-linéaire d'un haut-parleur pour sonorisation instrumentale. http://hdl.handle.net/2268/236626.

Schmitz, T. (2018a). Blind test on the emulation of several tube amplifiers. http://pc-dsp.montefiore.ulg.ac.be/choosingTest.

Schmitz, T. (2018b). Dataset of guitar amplifiers for emulation purpose. http://www.montefiore.ulg.ac.be/services/acous/STSI/downloads.php.

Schmitz, T. (2019a). Matlab implementation of the hammerstein identification by sine sweep toolbox. https://github.com/TSchmitzULG/Thesis/tree/master/Hammerstein.

Schmitz, T. (2019b). Tensorflow implementation of neural networks for real-time emulation of nonlinear audio system. https://github.com/TSchmitzULG/Thesis/tree/master/NeuralNetwork.

Schmitz, T. and Embrechts, J.-J. (March 2018c). Real time emulation of parametric guitar tubes amplifier with long short term memory neural network. In *the 5th International Conference on Signal Processing (CSIP2018)*.

Schmitz, T. and Embrechts, J.-J. (March 2019). Objective and subjective comparison of several machine learning techniques applied for the real-time emulation of the guitar amplifier nonlinear behavior. In *the 146th Convention of the Audio Engineering Society*.

Schmitz, T. and Embrechts, J.-J. (May 2013). Nonlinear guitar loudspeaker simulation. In *the 134th Convention of the Audio Engineering Society*, Rome Italy.

Schmitz, T. and Embrechts, J.-J. (May 2014). Improvement in non-linear guitar loudspeaker sound reproduction. In *the IWSSIP 2014 Proceedings*, pages 103–106. IEEE.

Schmitz, T. and Embrechts, J.-J. (May 2018b). Nonlinear real-time emulation of a tube amplifier with a long short term memory neural-network. In *the 144th Convention of the Audio Engineering Society*.

Schmitz, T. and Embrechts, J.-J. (October 2018a). Introducing a dataset of guitar amplifier sounds for nonlinear emulation benchmarking. In *the 145th Convention of the Audio Engineering Society*.

Schmitz, T. and Embrechts, J.-J. (September 2016). A new toolbox for the identification of diagonal volterra kernels allowing the emulation of nonlinear audio devices. In *the 22th International Congress on Acoustics*, Buenos-Aires Argentina.

Schmitz, T. and Embrechts, J.-J. (September 2017). Hammerstein kernels identification by means of a sine sweep technique applied to nonlinear audio devices emulation. *Journal of the Audio Engineering Society*, 65(9):696–710. https://doi.org/10.17743/jaes.2017.0025.

Schoukens, M. (2015). Identification of parallel block-oriented models starting from the best linear approximation. *Brussels, Belgium: Vrije Universiteit Brussel (VUB)*.

Shuvaev, S., Giaffar, H., and Koulakov, A. A. (2017). Representations of sound in deep learning of audio features from music. *arXiv preprint arXiv:1712.02898*.

Sondermeyer, J. C. (1983). Circuit for distorting an audio signal. US Patent 4,405,832.

Sondermeyer, J. C. and Brown Sr, J. W. (1997). Multi-stage solid state amplifier that emulates tube distortion. US Patent 5,619,578.

Sontag, E. D. (1993). Neural networks for control. In *Essays on Control*, pages 339–380. Springer.

Spiegel, M. R. (1968). *Mathematical handbook of formulas and tables*. McGraw-Hill.

Srebro, N., Rennie, J., and Jaakkola, T. S. (2005). Maximum-margin matrix factorization. In *Advances in neural information processing systems*, pages 1329–1336.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stover, C. (2019). Uniformly continuous.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

Thiede, T., Treurniet, W. C., Bitto, R., Schmidmer, C., Sporer, T., Beerends, J. G., and Colomes, C. (2000). Peaq-the itu standard for objective measurement of perceived audio quality. *Journal of the Audio Engineering Society*, 48(1/2):3–29.

Thoidis, I., Vrysis, L., Pastiadis, K., Markou, K., and Papanikolaou, G. (2019). Investigation of an encoder-decoder lstm model on the enhancement of speech intelligibility in noise for hearing impaired listeners. In *Audio Engineering Society Convention 146*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Torras-Rosell, A. and Jacobsen, F. (2011). A new interpretation of distortion artifacts in sweep measurements. *Journal of the Audio Engineering Society*, 59(5):283–289.

Tronchin, L. (2013). The emulation of nonlinear time-invariant audio systems with memory by means of volterra series. *Journal of the Audio Engineering Society*, 60(12):984–996.

Tronchin, L. and Coli, V. L. (2015). Further investigations in the emulation of nonlinear systems with volterra series. *Journal of the Audio Engineering Society*, 63(9):671–683.

TwoNoteEngineering (2011). Torpedo reload. https://www.two-notes.com/fr/torpedo-reload.

ULG (2019). Indicateur bibliographique.

Vanderkooy, J. and Thomson, S. (2016). Harmonic distortion measurement for nonlinear system identification. In *Audio Engineering Society Convention 140*. Audio Engineering Society.

Wang, F.-Y., Zhang, J. J., Zheng, X., Wang, X., Yuan, Y., Dai, X., Zhang, J., and Yang, L. (2016). Where does alphago go: From church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120.

Westwick, D. T. and Kearney, R. E. (2003). *Identification of nonlinear physiological systems*, volume 7. John Wiley & Sons.

Wiener, N. (1966). Nonlinear problems in random theory. *Nonlinear Problems in Random Theory, by Norbert Wiener, pp. 142. ISBN 0-262-73012-X. Cambridge, Massachusetts, USA: The MIT Press, August 1966.(Paper)*, page 142.

Wikipedia (2018). How to read tablatures. https://en.wikipedia.org/wiki/Tablature.

Yeh, D. T., Abel, J. S., Vladimirescu, A., and Smith, J. O. (2008). Numerical methods for simulation of guitar distortion circuits. *Computer Music Journal*, 32(2):23–42.

Yeh, D. T. and Smith, J. O. (2008). Simulating guitar distortion circuits using wave digital and nonlinear state-space formulations. *Proc. Digital Audio Effects (DAFx-08), Espoo, Finland*, pages 19–26.

Yu, W., Sen, S., and Leung, B. (1997). Time varying volterra series and its application to the distortion analysis of a sampling mixer. In *Proceedings of 40th Midwest Symposium on Circuits and Systems. Dedicated to the Memory of Professor Mac Van Valkenburg*, volume 1, pages 245–248. IEEE.

Yu, W., Sen, S., and Leung, B. H. (1999). Distortion analysis of mos track-and-hold sampling mixers using time-varying volterra series. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(2):101–113.

Zaunschirm, M., Frank, M., Sontacchi, A., and Castiglione, P. (October 2014). Audio quality: comparison of peaq and formal listening test results. In *6th Congress of Alps-Adria Acoustics Association*.

Zheng, J., Zhu, M., He, J., and Yu, X. (2012). Peaq compatible audio quality estimation using computational auditory model. In *Neural Information Processing*. Springer Berlin Heidelberg.

Zhu, A., Draxler, P. J., Hsia, C., Brazil, T. J., Kimball, D. F., and Asbeck, P. M. (2008). Digital predistortion for envelope-tracking power amplifiers using decomposed piecewise volterra series. *IEEE Transactions on Microwave Theory and Techniques*, 56(10):2237–2247.

Zölzer, U. (2011). *DAFX: digital audio effects*. John Wiley & Sons.

Zulkifli, H. (2018). Understanding learning rates and how it improves performance in deep learning. *Software testing fundalmentals*.

Zwillinger, D. (2002). *CRC standard mathematical tables and formulae*. Chapman and Hall/CRC.