IDIAP RESEARCH REPORT

DESIGNING SECOND ORDER RECURRENT NEURAL NETWORKS FOR PROSODY MODELLING

François Marelli

Idiap-RR-16-2018

NOVEMBER 2018

Centre du Parc, Rue Marconi 19, P.O. Box 592, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

UMONS
FACULTY OF ENGINEERING

MASTER'S THESIS

Designing second order recurrent neural networks for prosody modelling

Author:
François MARELLI

Supervisor:
Pr. Thierry DUTOIT[a]
Pr. Hervé BOURLARD[b c]
Dr. Philip N. GARNER[b]

[a]UMONS
[b]Idiap Research Institute
[c]EPFL

A thesis submitted in fulfilment of the requirements for the degree of Master in Electrical Engineering

Specialist Focus in Signals, Systems and BioEngineering

In collaboration with
Idiap Research Institute

August 2018

UMONS

Abstract

Faculty of Engineering
Idiap Research Institute

Master in Electrical Engineering

Designing second order recurrent neural networks for prosody modelling

by François MARELLI

Intonation in speech refers to the evolution of the fundamental frequency of the voice, also called pitch. In spoken language, it is used to carry additional information that only the words cannot convey, such as accents and emotions. It is therefore crucial in a speech synthesis system to properly handle this non-lexical feature in order not to alter the meaning of the synthesised utterances.

This thesis investigates the implementation of an artificial neural network that will be integrated into a physiologically plausible intonation synthesizer to enable end-to-end training. This system will be based on an existing atom-decomposition based intonation model, that represents the pitch as the sum of laryngeal muscle responses to nerve impulses.

We first explain how second order linear IIR filters can model muscle response, and how they can be implemented for training by gradient descent in order to integrate them in artificial neural networks. The derivation of the gradient back-propagation equations shows that gradient vanishing and explosion can cause convergence issues during the training process. We propose three techniques to solve these problems: loss function adaptation, gradient clipping and learning rate adaptation using Adam. These methods are experimentally validated and compared. We test the robustness of the second order linear recurrent units for different target systems and in noisy conditions.

We then detail how we integrate these filters in the atom-decomposition based intonation synthesis algorithm to enable end-to-end training. The transition from the old model to its new implementation is explained before detailing the initialisation and training procedure of the model. The performance of the end-to-end system is evaluated through objective scores. We describe how normalisation coefficients and a temporal L1 constraint must be imposed on the model to enforce physiologically plausible behaviours.

The model is compared to the one it is designed to mimic, showing that it has the characteristics sought at the outset. We analyse the behaviour of the internal signals of the system, showing that it is able to compensate for incorrect phrase component modelling, and giving an interpretation of the filter parameters that are learnt. Finally, the impact of the initialisation point on the performance and behaviour of the system is measured experimentally.

Keywords: Supervised learning (Machine learning), Neural networks (Computer science), Signal processing, Digital filters (Mathematics), Speech synthesis, Intonation (Phonetics)

Acknowledgements

I would like to thank Thierry Dutoit and Hervé Bourlard for giving me the opportunity to work on this Master's Thesis, and for encouraging me in the path of scientific research. I am also very grateful to Phil Garner for his advice in my beginnings in research, and for his availability.

I would like to thank Bastian Schnell for his contribution and help in this work, as well as for the great times we shared outside of work. Many other colleagues from Idiap provided me with advice and contributed to create a welcoming work environment. Some of them also involved me in activities outside of the research institute such as sport and cooking recipes exchanges. To cite only them, I would like to thank Banri, Jilt, Bogdan and Pavan.

Moving from Belgium to Switzerland was a big change for me, and I would like to thank Nadine Rousseau and Sylvie Millius for helping me to settle down in Martigny. Their kindness and their work have made the process much easier to me.

I would also like to thank the members of the Sei Mei Kan and the musicians of the Municipal Harmony of Martigny for their warm welcome. Thanks to them I discovered how to appreciate Valais and its culture.

Last but not least, I am grateful to my family and my friends that have supported me from Belgium throughout my stay in Switzerland. The many calls and messages we had helped me feel less alone when I was homesick.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
DNN	**D**eep **N**eural **N**etwork
CR	**C**ommand-**R**esponse
HMM	**H**idden **M**arkov **M**odel
IIR	**I**nfinite **I**mpulse **R**esponse
LSTM	**L**ong-**S**hort **T**erm **M**emory
MFCC	**M**el **F**requency **C**epstral **C**oefficient
MSE	**M**ean **S**quared **E**rror
RMS	**R**oot-**M**ean-**S**quare
RNN	**R**ecurrent **N**eural **N**etwork
SGD	**S**tochastic **G**radient **D**escent
SDM	**S**pring-**D**amper-**M**ass
SNR	**S**ignal to **N**oise **R**atio
V/UV	**V**oice/**U**n**V**oiced
WCAD	**W**eighted **C**orrelation **A**tom **D**ecomposition

List of Symbols

ζ	Damping ratio of a second order linear system
η	Learning rate
θ	Scale of a gamma atom
ρ	Pole modulus
σ	Sigmoid function
ϕ	Pole phase
ω_0	Natural frequency of a second order linear system
f_0	Fundamental frequency of speech
k	Discrete time step
K	Shape of a gamma atom
lf_0	Logarithm of the fundamental frequency of speech
s	Laplace transform variable
t	Continuous time
z	Z-transform variable
∇	Nabla operator
$\mathfrak{L}$	Loss function

Chapter 1

Introduction

1.1 Motivation

Spoken language carries much more information than only the words that are pronounced. Our identity, intentions and emotional state are also reflected in our speech. For example, the exact same sentence can have a drastically different meaning if said with irony. This additional information is transmitted in the way that we pronounce different words, stressing syllables and changing our intonation. Prosody refers to the non-lexical features that we use to communicate a message, describing *how* we say something rather than *what* we say (Santen, Mishra, and Klabbers 2008).

Speech synthesis is the process of computationally creating a spoken message to convey a specific meaning. With prosody carrying crucial parts of the information in speech such as punctuation, it is important to integrate it to synthesis algorithms. Moreover, the human auditory system is very good at spotting non natural sounding speech, and incorrect prosody in synthesised speech can highly degrade the perceived quality of a synthetic voice. For speech-to-speech translation systems (involving automatic recognition of speech in a source language, translation to a target language and then speech synthesis), it is very important to be able to capture prosodic features from the source, and then appropriately transfer them to the synthesis in order to convey the same meaning.

In this work, we will focus on the intonation of the speech, which is a major prosodic feature. Also referred to as intonation contour, it represents the evolution of the fundamental frequency of the voice. A high frequency corresponds to a high pitched voice, while a low frequency relates to a deep voice. This single characteristic can convey a lot of information by itself, as intonation contour defines the stressed parts of speech. A famous example is the following seven-word sentence, in which any word can be stressed to give a different meaning:

> "I never said she stole my money."

Changing only the intonation can lead to the following interpretations:

- *I never **said** she stole my money.* – But I could have written it.
- *I never said **she** stole my money.* – But I could have blamed someone else.
- *I never said she stole my money**?*** – I am not sure I said it.

This motivates the importance of appropriate intonation contour generation for speech synthesis, which constitutes the object of this thesis.

1.2 State of the art

Until recently, the state-of-the-art speech synthesis systems were based on unit selection and concatenative synthesis (Hunt and Black 1996). This technique assembles audio segments from a pre-recorded database to create sentences, varying the unit size in order to create the best sounding output. By carefully choosing the samples to concatenate, it is possible to obtain any intonation contour, assuming that the database is large enough to allow such selection.

This method generates very natural sounding speech, as long as the database used covers a wide range of sounds and has a good recording quality. However this model does not allow to define parameters to fine-tune the desired output: the synthesis is somehow limited to whatever is present in the database, and cannot really create new voices that were not recorded.

This can be achieved by parametric speech synthesis. This technique uses parametric models to generate a speech representation, such as prosodic features and MFCCs (**M**el **F**requency **C**epstral **C**oefficient). These features are then processed by a vocoder to generate synthesised speech. By correctly exploiting the parameters of the model, it is possible to change multiple characteristics of the synthesised voice, which can for example be used for speaker adaptation, i.e. tuning the synthesised voice to sound like a target speaker (Zen, Tokuda, and Black 2009). Parametric synthesis allows the explicit representation of the intonation contour, enabling precise control of the desired pitch curves.

The cutting-edge Wavenet system (Van Den Oord et al. 2016) provides the most natural-sounding synthesised speech without any explicit representation of the prosodic features. It is an end-to-end CNN (**C**onvolutional **N**eural **N**etwork) that directly generates raw audio outputs. Even though this system is able to generate significantly more natural sounding samples, the lack of explicit prosodic features in the model does not allow straightforward control of the intonation contour. The fundamental frequency of the synthesised voice is computed internally and not explicitly (Shen et al. 2017), and although it automatically matches punctuation, it provides no further tuning possibilities.

When it comes to modelling the intonation contour, there are two main categories of algorithms: implicit and explicit models. The first ones use standard probabilistic tools to infer a model from data without imposing an explicit behaviour on the algorithm. For example, it is possible to train a HMM (**H**idden **M**arkov **M**odel) to generate pitch curves (Tokuda, Zen, and Black 2002). That type of technique does not allow interpretation of the resulting model, nor does it enable easy tuning of its parameters to obtain different types of intonation curves.

On the other hand, explicit models focus on reproducing the mechanisms involved in intonation production as they are assumed to happen in the human body. To that extent, Fujisaki's CR (**C**ommand-**R**esponse) model represents intonation contours as a sum of phrase and accent contributions (Fujisaki and Hirose 1984). These are computed as the responses of second-order linear systems to command signals, the latter ones intended to give a representation of the brain action on intonation in speech. Trying to reproduce plausible processes allows a better interpretation of the model, but can also help to improve the naturalness of the generated speech by enforcing smooth transitions in the intonation. Fujisaki's physiologically plausible model is among the most popular ones for intonation curve modelling.

1.3 Scope of the thesis

Previous works at Idiap Research Institute showed that it is possible to build a well-performing generalised CR model based on atom decomposition (Gerazov, Honnet, et al. 2015). This model tries to reproduce the biological mechanisms impacting intonation variation, such as the laryngeal muscles and the subglottal pressure (Honnet et al. 2018). The command signals become impulses, and the responses are obtained by applying the muscles transfer function to these spikes.

Predicting the command signals with a RNN (**R**ecurrent **N**eural **N**etwork) allowed to integrate this model in an intonation synthesizer (Schnell and Garner 2018). In its current state, the network predicts the position and amplitude of the spikes that drive the muscles. These outputs must then be post-processed in order to generate the true control signals, compute the muscle response and reconstruct the intonation curve from the predicted atoms.

This intonation synthesis system can be exploited for parametric speech synthesis. Even though end-to-end synthesizers provide the best speech naturalness, we favour this option as we put strong emphasis on understanding the behaviour of the model. The Wavenet synthesizer is indeed a big CNN black box in which the intonation generation is near to impossible to analyse. In contrast, physiologically-plausible models tend to sound quite natural and have a very good behavioural interpretation.

Moreover, we can expect that using a physiologically plausible model would allow us to tune its different physiological parameters to obtain different intonation curves. Indeed, such results were noticed in the field of music synthesis with plausible physical models. Once properly set up, these models allow easy control of sound characteristics in an intuitive way thanks to the understanding of the underlying model (Drioli and Rocchesso 1998). Physical-like musical synthesis systems are also better at parameter identification for generalisation to multiple instruments for the same reason (Uncini 2002). If we can reproduce this behaviour on speech synthesis, it will be very useful in the field of emotional synthesis where fully annotated databases are difficult to build. Understanding a physiological model of intonation production may allow us to modify the emotional content of an utterance by tuning the different parameters of the model, without requiring huge amounts of training emotional data. This would represent a significant advantage on statistical models, that require important amount of samples to learn new features.

In the current atom-decomposition based intonation synthesis implementation, the need for a static post-processing operation prevents the model from being trained in an end-to-end manner. This does not allow the post-processing parameters to be trained along with the rest of the system to optimise its performance. The objective of this thesis is to design muscle models that can be trained using ANN (**A**rtificial **N**eural **N**etwork) frameworks, and to integrate these to the system as an alternative to the current post-processing algorithm, thus enabling end-to-end training strategy. We will first investigate the implementation of muscle models that can be trained like ANNs. Then, we will detail the integration of these models to the intonation synthesizer in order to suppress the need for an additional post-processing algorithm. The performance of the obtained system will be compared to the current results, and we will finally analyse the benefits brought by end-to-end training.

Chapter 2

Muscle model implementation

This chapter presents the design and implementation of muscle models that can be integrated in ANN training frameworks. We will first describe the type of model used before detailing the possible ways to implement it using state-of-the-art techniques. We will then propose our own implementation and analyse its behaviour in training. Finally, an experiment on idealised data will be used as a proof of concept of the proposed solution.

2.1 The Spring-Damper-Mass system

There are many different ways to model the transfer function of muscle tissues. By increasing complexity, we find the spring-damper-mass (second order) model, the Hill type (third order) model and even higher order models (Gerazov and Garner 2015). Although it has been measured that using higher order models can improve the synthesis quality (Prom-On, Xu, and Thipakorn 2009), we will focus on the simpler SDM (**S**pring-**D**amper-**M**ass) model in this work. Indeed, this choice is consistent with Fujisaki's assumption that command signals are filtered by second order systems. Moreover, it is possible to build any all-pole model with arbitrary complexity by combining multiple second order systems so this is not a restrictive choice.

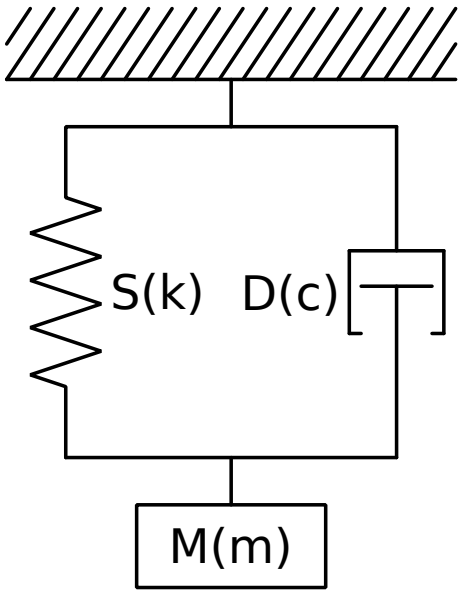


FIGURE 2.1: Spring-Damper-Mass system.

The SDM model, as shown in Figure 2.1, represents the basic mechanical properties of the muscles: elasticity, damping and inertia. Its transfer function in Laplace domain is given by (2.1), with k the spring coefficient, c the damping coefficient and m the mass.

$$Y(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2} X(s) \tag{2.1}$$

$$\zeta^2 = \frac{c^2}{4mk} \tag{2.2}$$

$$\omega_0^2 = \frac{k}{m} \tag{2.3}$$

Where $Y(s)$ represents the Laplace transform of the position of the mass with respect to the applied force $X(s)$. ω_0 is the natural frequency of the system and ζ is called the damping ratio. The system can be characterised by its damping:

- $\zeta > 1$: overdamped system with two distinct real poles[1]
- $\zeta = 1$: critically damped system with a double real pole
- $\zeta < 1$: underdamped system with two complex conjugate poles

The z-transform generic expression of the equivalent discrete-time system is given by (2.4); α, β and G being the parameters of the model.

$$Y(z) = \frac{G}{1 - \alpha z^{-1} - \beta z^{-2}} X(z) \tag{2.4}$$

This is also called a second-order all-pole discrete linear filter. It contains recurrence on its output, and is therefore characterised as an IIR (**I**nfinite **I**mpulse **R**esponse) filter. The stability of such a system implies and is implied by the fact that all its poles are inferior to 1 in modulus.

(2.4) can be rewritten as a recurrence relationship, given in (2.5) and illustrated in Figure 2.2[2]. k denotes the discrete time evolution.

$$y_{(k)} = Gx_{(k)} + \alpha y_{(k-1)} + \beta y_{(k-2)} \tag{2.5}$$

There are numerous closed-form filter extraction methods that can be used to compute the values of the parameters α and β. However, we cannot use traditional techniques in this case because we want to integrate the model in an ANN framework. This means that we must be able to learn these parameters using back-propagation, which is not the case in conventional filter extraction methods.

2.2 Recurrent Neural Networks

ANNs are machine learning algorithms that can approximate a function after having been trained on the target data. They are built using basic units called neurons. These perform a

[1] The roots of the denominator of the transfer function are called poles.

[2] In all diagrams using this notation, z^{-1} represents a time delay.

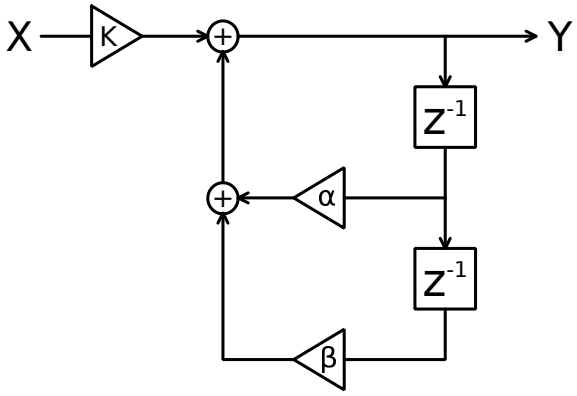


FIGURE 2.2: Second order linear IIR filter, with X the input and Y the output.

linear combination of their multiple inputs, followed by a nonlinear operation called activation function (e.g. sigmoid, hyperbolic tangent). Neurons sharing the same inputs are grouped to form a layer. The outputs of a layer of neurons can serve as inputs for other layers. Figure 2.3 illustrates how multiple layers can be stacked in this way to create a network. The number of layers in an architecture is called depth, and networks with more than 3 layers are usually called DNNs (**D**eep **N**eural **N**etwork).

Such systems contain many parameters that can be optimised using an iterative learning procedure. It uses an objective function, also referred to as cost function or criterion (e.g. mean squared error). During the training, data are fed to the inputs of the network, then the cost function is computed as a distance between the generated outputs and the ground-truth targets. After every iteration, the parameters of the network are updated using the gradients coming from the loss function, according to a gradient descent optimisation algorithm. These gradients are computed backwards in the network using the chain rule, which is known as back-propagation (Rumelhart, G. E. Hinton, and Williams 1985; LeCun, Touresky, et al. 1988). The gradients are

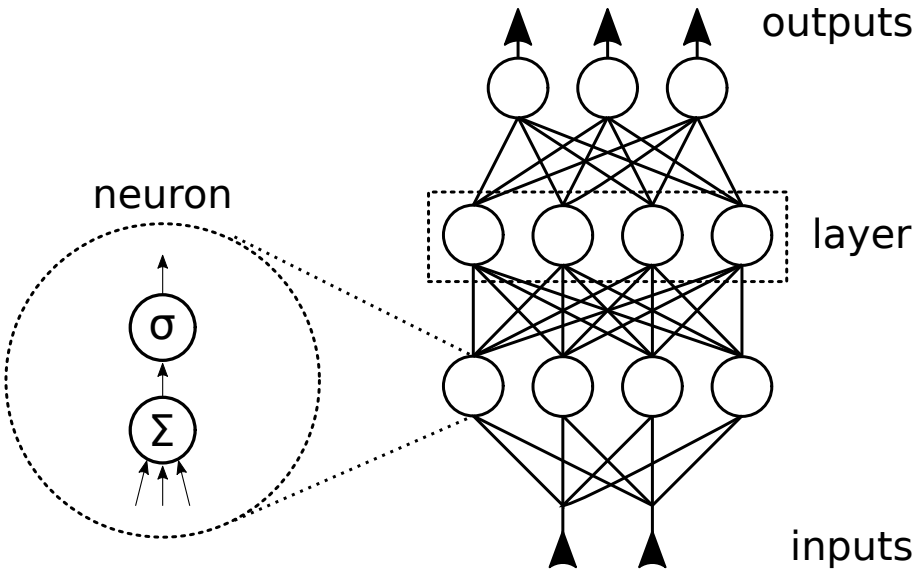


FIGURE 2.3: Artificial neural network general structure.

usually weighted before updating the parameters, in order to control the speed of convergence. This weighting can be global or specific to every parameter, and is called the learning rate. A comprehensive description of ANNs is out of the scope of this thesis, but a more detailed explanation can be found in the review by LeCun, Bengio, and G. Hinton 2015.

For this work, we are particularly interested in a specific type of ANN: the recurrent neural network. This kind of network integrates a time recurrence in its layers, making it the best fit to model a recurrent system. This structure is illustrated in Figure 2.4, where the delayed output is called the hidden state.

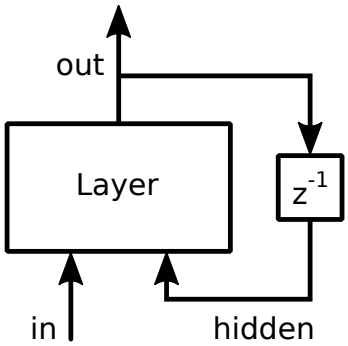


FIGURE 2.4: Recurrent layer layout.

There are two main ways to achieve second order recurrence in a RNN. On one hand, it is possible to stack multiple layers with first order recurrence, as cascading filters will result in a higher order system. On the other hand, it can be achieved by increasing the dimension of the hidden state. In that case, each hidden element can be used to represent a different time delay, and the resulting system will be at maximum of order equal to the dimension of the hidden state.

Back-propagation in RNNs can be more complex than in standard ANNs. There are multiple ways to update the weights in such a system, depending on the assumptions about the network structure and the shape of the loss function (Pearlmutter 1995). Back-propagation through time (Werbos 1990) is an extension of the standard back-propagation algorithm that takes into account the time dimension. It consists of unfolding the recurrence as in (2.7) until the beginning of the signals is reached. The gradients can then be computed in a standard way as the recurrence does not appear in the obtained expression.

However, RNN are difficult to train because of well known gradient instability cases that may appear during back-propagation. For example, due to the multiple passes in each unit, the norm of the gradient can become infinite, which is known as gradient explosion (Bengio, Simard, and Frasconi 1994). RNN training must therefore be carefully supervised to prevent divergence, for example by adjusting the learning rate or by constraining the parameters. We are yet confident that a filter extraction method based on back-propagation can converge, as it has been shown that iterative methods are efficient for digital filter design problems (Howell and Gordon 2001).

$$y(k) = f\left(x_{(k)}, y_{(k-1)}\right) \tag{2.6}$$

$$= f\left(x_{(k)}, f\left(x_{(k-1)}, y_{(k-2)}\right)\right) \tag{2.7}$$

LSTM (**L**ong-**S**hort **T**erm **M**emory) is a specific architecture of RNN that is illustrated in Figure 2.5 (Hochreiter and Schmidhuber 1997). Its particularity is the presence of three gating effects, drawn with thick lines on the figure. From left to right, they represent:

- *input gate* (i): selects what information can enter the cell
- *forget gate* (f): selects if previous information must be kept or forgotten
- *output gate* (o): selects what information must be transmitted to the output

$\vec{x}$ represents the input signal, $\vec{c}$ the hidden state and $\vec{h}$ the output of the cell at the current sampling time t. $\vec{\omega}$, $\vec{v}$ and $\vec{v}$ are respectively the weights for the input, output and hidden state. ϕ and σ are nonlinear activation functions.

The forget gate makes this architecture particularly interesting in our case, as it creates a structure similar to a linear filter inside the LSTM cell (Gers, Schmidhuber, and Cummins 1999). Indeed, $\vec{c}_t$ can be seen as the output of a first order IIR filter with input $\tilde{c}_t$, the forget gate being assimilated to the pole of the system. If the forget gate is constant, this is identical to the first order IIR filter shown in Figure 2.6.

Even though the inner structure of a LSTM is similar to an IIR filter, this cell is not properly fitted to model a second order linear IIR filter. On one hand, LSTM cells include multiple non-linearities. These activation functions induce a behaviour quite different from a linear model. Second, the model is too complex for the target task. A muscle model has no need for input and output gates, and the forget gate should not be a variable value; this results in an over-parametrised system that will be more difficult to optimise. On the other hand, LSTMs are not intrinsically second-order systems. It can be obtained as mentioned earlier, but the model is

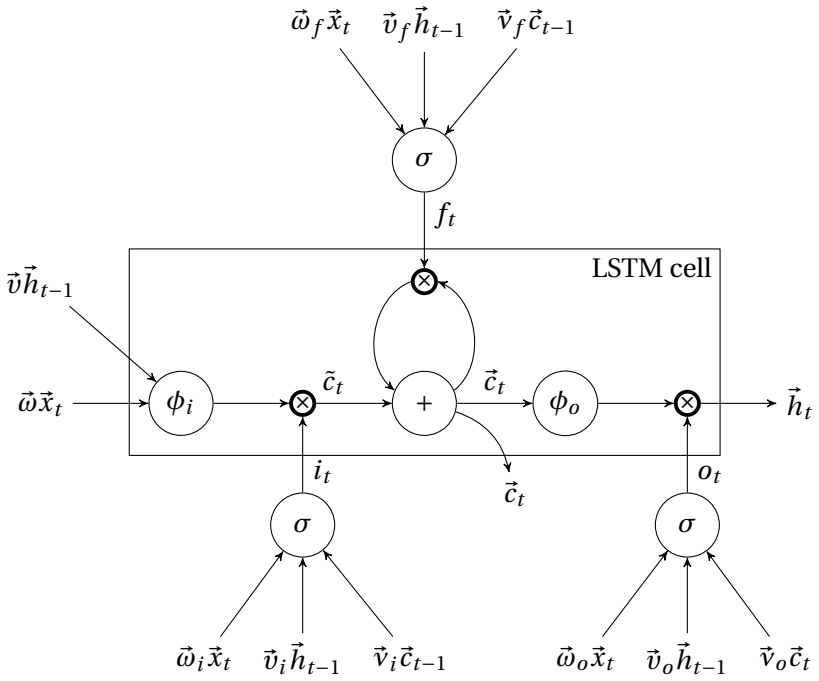


FIGURE 2.5: LSTM cell (courtesy of P.N. Garner), with $\vec{x}$ the input, $\vec{c}$ the hidden state and $\vec{h}$ the output. The input, forget and output gates are outlined in bold.

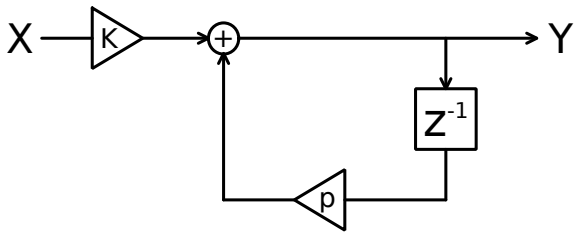


FIGURE 2.6: First order linear IIR filter, with gain K and pole p.

not guaranteed to converge to a second order solution. Globally, using LSTMs to implement muscle response would not be very efficient, and the obtained model would be less plausible and harder to interpret.

2.3 Neural Filter implementation

We propose our own implementation of a linear IIR filter to be trained with back-propagation algorithms. We will first analyse a first order implementation, before moving on to second order systems. We will investigate the stability of the proposed models. As the objective is to integrate filters in a neural network, we will call these models Neural Filters.

First order Neural Filter

The proposed implementation relies on (2.8). As compared to formulation 2.4, the gain G is set to 1 for simplicity. We do not need to learn this parameter for muscle modelling, but it is still possible to add a simple multiplication at the input or the output to integrate this parameter into the model if needed.

$$y_{(k)} = x_{(k)} + \sigma(a) \cdot y_{(k-1)} \tag{2.8}$$

Figure 2.7 illustrates the diagram corresponding to this model. σ represents the sigmoid function given by (2.9).

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{2.9}$$

The parameter p in (2.4) has been replaced with $\sigma(a)$, with a being the real parameter that we optimise. This is constrains the pole p between 0 and 1, so that we have the guarantee that the system is stable. Indeed, the only pole of this filter is p, and stability will be implied with a modulus less than 1. As this model is meant only for analysis, we do not need to have negative poles. However, it is possible to extend the model by replacing the sigmoid by a hyperbolic tangent to allow for negative values of p.

The expression of the loss function gradient with respect to the parameter can be computed using back-propagation through time, and is given by (2.10). The proof can be found in Appendix A.

$$\frac{\partial y_{(k)}}{\partial a} = \sum_{n=0}^{k-1} \left[y_{(k-1-n)} \cdot \sigma'(a) \cdot \sigma^n(a) \right] \tag{2.10}$$

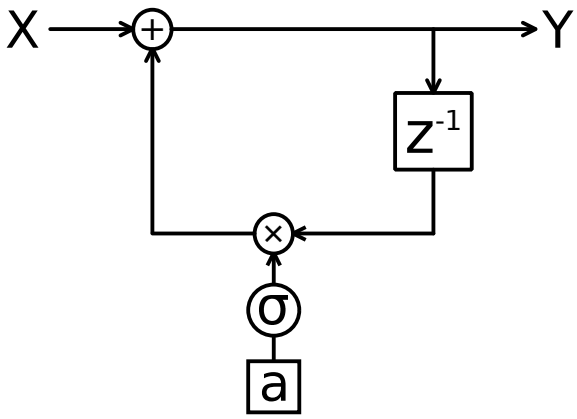


FIGURE 2.7: First order filter neuron, equivalent to a linear IIR filter. a is the only parameter.

The only term that can cause gradient issues is $\sigma^n(a)$. Indeed, if $|\sigma(a)| > 1$, it will quickly tend towards infinity and cause a gradient explosion. In contrast, with $|\sigma(a)| \ll$, it will quickly decrease to zero and the gradient will vanish (Pascanu, Mikolov, and Bengio 2013). Gradient explosion will cause the algorithm to have an unstable behaviour, which may prevent it to converge. Gradient vanishing will not strictly prevent convergence, but with extremely small gradients the training process is highly slowed down, and can possibly not converge in a reasonable amount of time.

Replacing α by $\sigma(a)$ prevents gradient explosion as the sigmoid is bounded between 0 and 1. However, it does not prevent gradient vanishing. Moreover, for $\sigma(a)$ close to one, the gradient can still reach high amplitudes as the summation increases for long input sequences. This can cause a partial gradient explosion. Therefore we will later detail what tools we can use to compensate for these effects.

Second order Neural Filter

The proposed implementation is based on (2.11), where we also set the gain to 1 for the same reasons. It involves two parameters: α and β that are unconstrained in this formulation. The corresponding z-transform is given in (2.12).

$$y_{(k)} = x_{(k)} + \alpha y_{(k-1)} + \beta y_{(k-2)} \tag{2.11}$$

$$Y(z) = \frac{1}{1 - \alpha z^{-1} - \beta z^{-2}} X(z) \tag{2.12}$$

The resulting back-propagation equations of this system (as developed in Appendix A) are given by 2.13-2.15.

$$K_n = \begin{cases} \alpha K_{n-1} + \beta K_{n-2} & \text{if } n > 0 \\ 1 & \text{if } n = 0 \\ 0 & \text{if } n < 0 \end{cases} \tag{2.13}$$

$$\frac{\partial y_{(k)}}{\partial \alpha} = \sum_{n=0}^{k-1} \left[y_{(k-1-n)} \cdot K_n \right] \tag{2.14}$$

$$\frac{\partial y_{(k)}}{\partial \beta} = \sum_{n=0}^{k-2} \left[y_{(k-2-n)} \cdot K_n \right] \tag{2.15}$$

To analyse the stability of the gradient, we must compute the stability of K_n. It is possible to write (2.13) as the impulse response of a discrete linear system, as shown in (2.16). The initial condition on K_0 is replaced by an impulse excitation signal $X_{(k)}$. This transformation allows the computation of the z-transform of the resulting system, given in (2.18).

$$K_{(k)} = X_{(k)} + \alpha K_{(k-1)} + \beta K_{(k-2)} \tag{2.16}$$

$$\begin{cases} X_{(0)} = 1 \\ X_{(k)} = 0 \quad \forall k \neq 0 \\ K_{(k)} = 0 \quad \forall k \leq 0 \end{cases} \tag{2.17}$$

$$K(z) = \frac{1}{1 - \alpha z^{-1} - \beta z^{-2}} X(z) \tag{2.18}$$

As the impulse response of this transfer function represents the evolution of K_n, proving the stability of this system induces the stability of the gradient. The transfer function (2.18) is identical to the one from the modelled system given in (2.12). This means that the stability of the target model implies the stability of the gradient. Gradient explosion will then be prevented. Vanishing and partial explosion of the gradient can still occur in the network when nearing the boundary values of the parameters, for the same reasons as mentioned previously.

However, there is no simple constraint that can be imposed on the system to ensure its stability. To achieve so, we must distinguish the different damping categories and implement a specific constraint for each case.

Overdamped second order Neural Filter

Overdamped systems have two distinct real poles. The easiest way to implement it and to guarantee its stability is to cascade two first order Neural Filters. The resulting filter will have two real poles with modulus constrained below 1.

Critically damped second order Neural Filter

Critically damped systems have one double real pole. It can be easily implemented by chaining twice the same first order Neural Filter. The resulting system will have one double real pole

with modulus lower than 1.

Underdamped second order Neural Filter

Underdamped systems have two complex conjugate poles. In order to control the modulus of the poles, it is easier to use polar notation. With ρ the modulus and ϕ the phase of the poles, (2.11) and (2.12) become (2.19) and (2.20).

$$y_{(k)} = x_{(k)} + 2\rho\cos(\phi)\ y_{(k-1)} - \rho^2\ y_{(k-2)} \tag{2.19}$$

$$Y(z) = \frac{1}{1 - 2\rho\cos(\phi)\ z^{-1} + \rho^2\ z^{-2}} X(z) \tag{2.20}$$

The Neural Filter implementation uses the parameters p and c, that we define as follows:

$$\sigma(p) = \rho \tag{2.21}$$

$$\tanh(c) = \cos(\phi) \tag{2.22}$$

This choice of parameters allows us to properly constrain the system to ensure its stability. Indeed, ρ will be bounded between 0 and 1 as the sigmoid, and $\cos(\phi)$ between -1 and 1 as the hyperbolic tangent. This leads to the implementation given by (2.23).

$$y_{(k)} = x_{(k)} + 2\ \sigma(p)\ \tanh(c)\ y_{(k-1)} - \sigma^2(p)\ y_{(k-2)} \tag{2.23}$$

Implementing a distinct model for every damping case allows us to ensure the stability of the system by directly constraining the modulus of its poles. However, it prevents us from using a single system to model all possibilities. If the target damping is known, this does not represent a problem. However, in some use cases where the damping ratio of the target system is unknown, this may be a limitation to use the proposed implementations. In this case, it is possible to implement a general second order filter directly based on (2.11), but such a system cannot be constrained to be stable. There is therefore no guarantee for the gradient stability in a general second order filter, and using it means taking the risk to face gradient explosion issues.

Critical damping can in practice be modelled with any of the three proposed implementations. A double real pole can indeed be seen as two equal real poles, or two complex conjugate poles with a phase of 0. The critically damped second order Neural Filter should only be used when critical damping must be imposed on the system. The two other models have a greater freedom of exploration of the possible solutions.

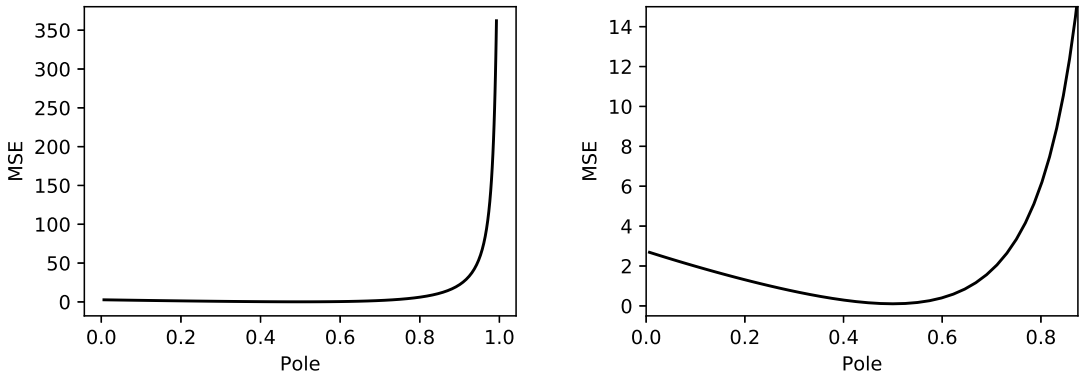
2.4 Gradient issues analysis

As stated in Section 2.3, back-propagation through time can cause the gradient to have odd behaviours such as vanishing or partial explosion. In this section, we take a closer look at how this can impact our system, and the different tools that we can use to reduce these unwanted effects.

In order to measure and analyse the performance of our model, we create the following experiment. For a given target filter, we select the Neural Filter of corresponding order and damping ratio. We build a database of 500 white noise samples with a size of 200 time steps each. Every sample is filtered by the target filter, and random noise is added to the obtained signal in order to obtain a SNR (**S**ignal to **N**oise **R**atio) of 20 dB. These noisy filtered signals constitute our reference outputs. We compare the outputs of the Neural filter for a white noise input to the corresponding reference output, by computing the loss function on these signals. The implementation uses PyTorch, a Python deep learning framework that provides tools for tensor computation, GPU acceleration and ANN design (Paszke et al. 2017).

It is possible to plot the evolution of the objective function with respect to the model parameters. This can be achieved by manually moving the parameters of the Neural Filter in a desired range and measuring the loss function at every step. The loss is averaged on the whole samples set for greater precision.

The standard error used for curve fitting tasks is the MSE (**M**ean **S**quared **E**rror). In the case of a first order Neural Filter with a target pole of 0.5, the loss function is a curve represented in Figure 2.8.



(A) Full view

(B) Zoom on the minimum

FIGURE 2.8: MSE loss for first order Neural Filter with target pole 0.5.

Gradient vanishing and explosion are well illustrated on Figure 2.8a. For low values of Neural Filter pole the loss is very flat, resulting in very small gradient vanishing to zero. It means that during iterative training, the parameter will move very slowly in that zone as it is driven by small gradients. This can highly slow down the learning process. On the other hand, the loss curve is very steep for high poles. Although it does not become infinite, the gradient reaches very high values, which we refer to as partial explosion. It will cause convergence problems in this system for the following reason: if the parameter reaches the partial explosion zone, the big gradient will immediately send it to the opposite side of the range. The parameter will then be stuck in the vanishing gradient zone. It also means that the algorithm will have difficulties to converge if the minimum is in the partial explosion zone, as too high gradients will possibly take the parameter away from the optimal position.

For second order filters, the loss is a surface showing similar properties, as seen in Appendix B, Figures B.1 and B.2. We will refer to parameter ranges where gradient vanishing or partial explosion may occur as instability zones.

We identify two possible situations in which gradient issues may occur during training:

- the parameter is initialised in an instability zone
- the target parameter is located in an instability zone

The first one is easily avoided by correctly choosing the initial value. For example, 0.5 is a good candidate for real poles. In the case of overdamped systems or when using multiple Neural Filters in parallel, it is important to add a random small deviation to this initialisation. If not, the parameter values will always stay identical as the gradients will never differ.

The second case is not easy to avoid. Indeed, we usually do not control the poles of the target system. The influence of the target pole can be measured by training the Neural Filter on a fraction of the reference outputs (called the training set) for a fixed number of epochs[3] and then measuring the average loss function achieved on the remaining samples (called the test set[4]). For this experiment, we use a test fraction of 20 % and train the model for 50 epochs with a standard SGD (**S**tochastic **G**radient **D**escent).

Figure 2.9 shows the measured MSE loss for different overdamped target systems. The gradient vanishing effects cannot be seen, as it only influences the speed of convergence and not the final result. 50 epochs is enough to let the algorithm converge even when taking into account the gradient vanishing problem. In contrast, the partial explosion of the gradient is easily noticeable for high values of target poles: the last two results on the graph have a very high loss as compared to the others.

[3] An epoch represents an iteration of the training algorithm on the whole training set.

[4] This separation between test and training sets is meant to distinguish good performance from overfitting on the training set.

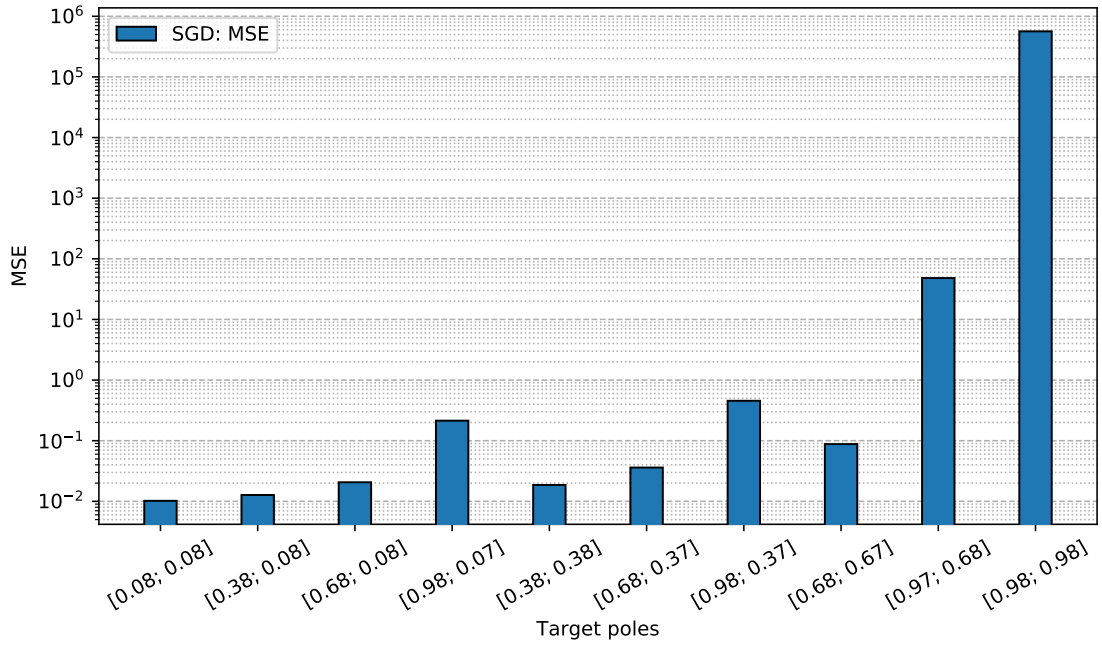


FIGURE 2.9: MSE on Neural Filter for second order overdamped targets. Gradient partial explosion can be seen for high pole modulus.

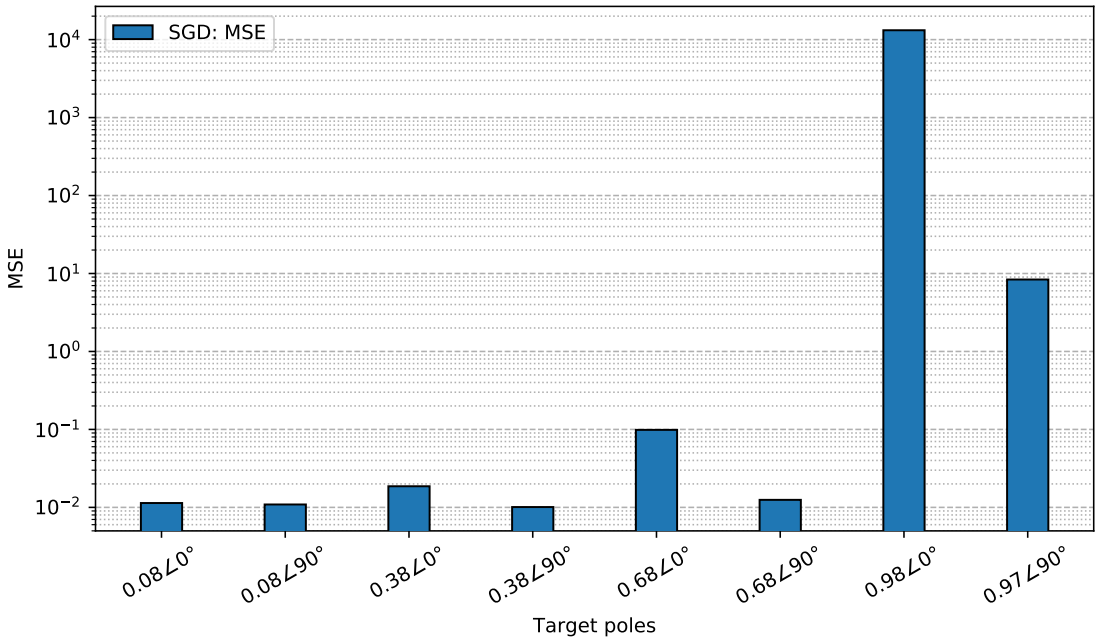


FIGURE 2.10: MSE on Neural Filter for second order underdamped targets. Gradient partial explosion can be noticed for high poles and low angles.

Similar conclusions can be drawn for underdamped Neural Filters, as shown on Figure 2.10. Convergence issue is less likely to happen with first order systems, due to the single recurrence. This can be seen on Figure B.3. However, gradient issues may still appear when using longer samples or with target poles very close to 1.

In order to understand why a gradient partial explosion can prevent convergence, we keep track of the evolution of the gradients in the model along the training process. The resulting graphs are shown in figures 2.11 and 2.12[5]. Based on these curves, we can describe the behaviour of the model when a partial explosion occurs.

The model starts training normally, until it enters the instability zone (for complex conjugate poles, it consists of modulus close to 1 and phase close to 0° or 180°). Then a partial explosion of the gradient happens (batch 22 on Figure 2.11, batch 3 on Figure 2.12), pushing the modulus to its boundary value of 0. With such a low modulus, the gradient vanishes extremely quickly to 0, preventing the system from learning anything else from that point.

It is the combination of gradient partial explosion and vanishing that leads to convergence issues with this model. As we cannot influence the target system to guarantee a good convergence behaviour, we must propose specific learning strategies to deal with gradient issues and reduce the risk of partial explosion and vanishing. We investigate three main ways to solve this problem:

- modifying the gradient itself
- using a different loss function
- exploiting the learning rate to compensate for the gradient variations

[5]A batch represent a chunk of 25 samples being run through the model.

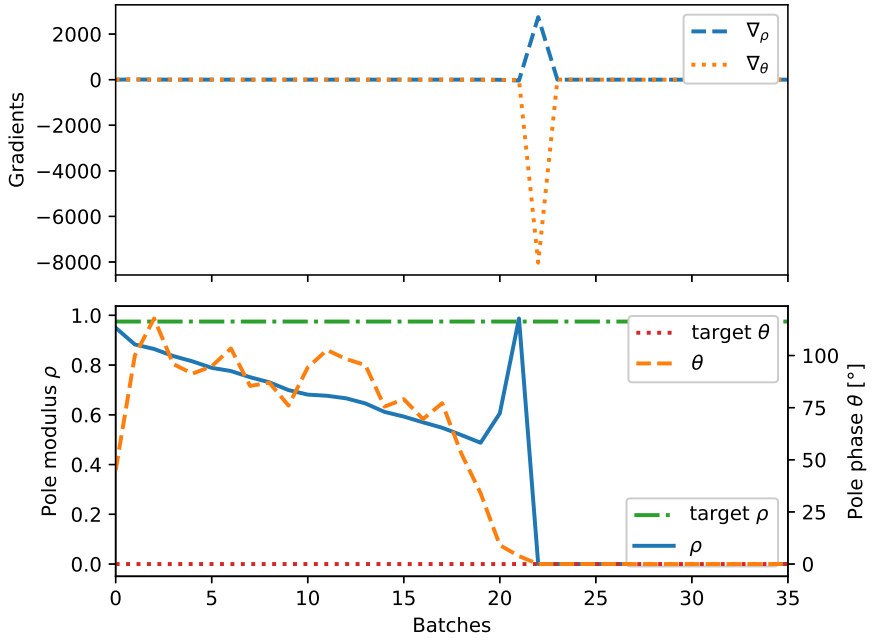


FIGURE 2.11: Gradient and parameters evolution while training Neural Filter on target poles $0.98\angle \pm 0°$.

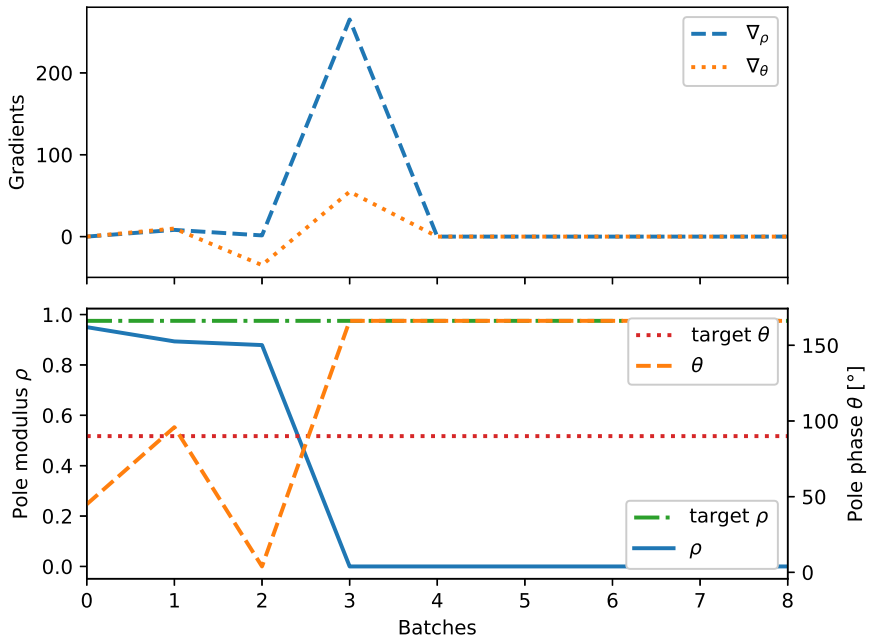


FIGURE 2.12: Gradient and parameters evolution while training Neural Filter on target poles $0.97\angle \pm 90°$.

Gradient clipping

To prevent the gradient from becoming unreasonably big, it is easy to manually impose a threshold that will truncate any higher value. This method, known as gradient clipping (Pascanu,

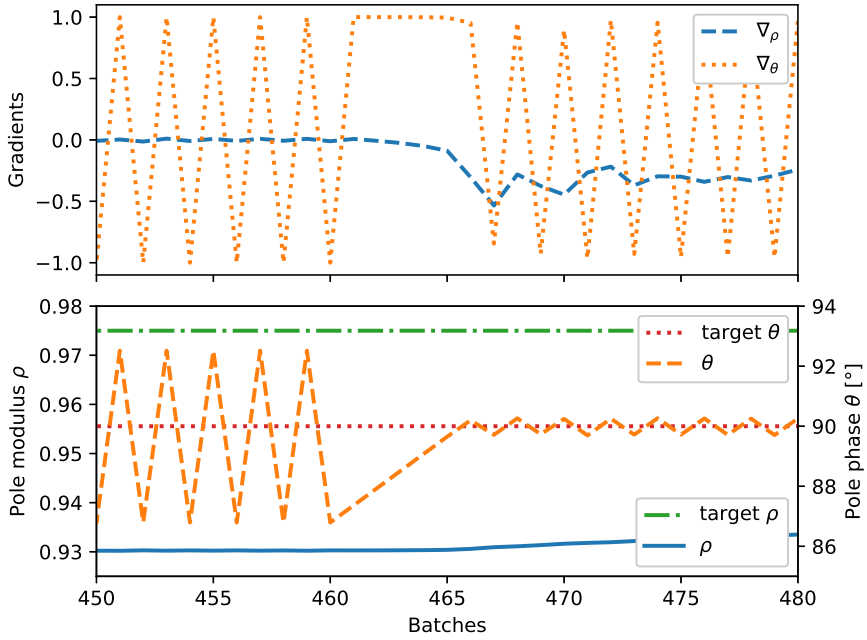
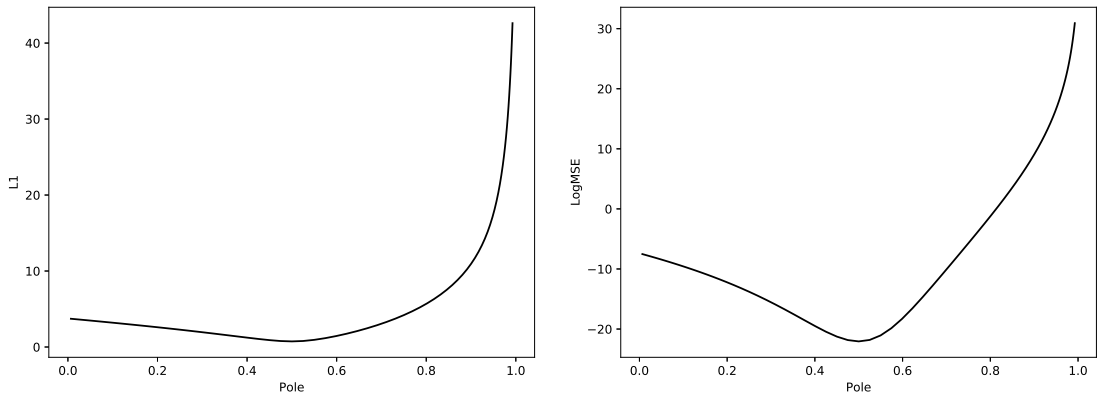


FIGURE 2.13: Gradient and parameters evolution while training Neural Filter on target poles $0.97\angle \pm 90°$ with gradient clipping to 1.

Mikolov, and Bengio 2012), is widely used in the field of RNNs. It consists in checking the gradient norm every time before updating the parameters, and thresholding it to a maximum value. Depending on the type of norm used, the direction of the gradient may or may not be kept. This solution is very easy to implement and is very efficient at preventing gradient explosion. However, clipping the norm of the gradient to a maximum value has major drawbacks:

- Clipping high gradients does not bring a solution to the vanishing problem, as this technique does not impact low gradients.
- The clipping operation induces a loss of information. It can be the direction if the thresholding is component-wise, or the smaller variations if the norm is applied to all the components at once.
- The clipping threshold introduces a new hyperparameter that must be correctly chosen to allow convergence. This value can depend on the type of model and on the learning rate and cannot be known a-priori.
- If the target is in the explosion zone, the gradient will trigger the clipping threshold at every step. This will lead to a saturated oscillating behaviour of the parameters around the optimum, as illustrated in Figure 2.13 (see the phase of the pole). Convergence will therefore be inefficient, and the final solution will possibly be sub-optimal. The only way to compensate for that oscillation is to progressively reduce the learning rate, which is done at batch 460 in Figure 2.13.



(A) L1 criterion (B) log(MSE) criterion

FIGURE 2.14: Alternative losses for first order Neural Filter with target pole 0.5.

Loss function adaptation

As the gradients are directly related to the loss function, we can adapt the criterion to obtain a better behaviour. A first possibility is to replace MSE (based on L2 norm[6]) by a L1 distance[7]. Indeed, we can expect that reducing the degree of the criterion will reduce the slope in the steep regions of the curve. As the gradient grows exponentially, we also propose to use the logarithm of the MSE as the objective function. Thanks to logarithm being a monotonic transformation, the optimum is not shifted. We expect that the logarithm will help compensate the effects of both vanishing and explosion, as they come from a power operation.

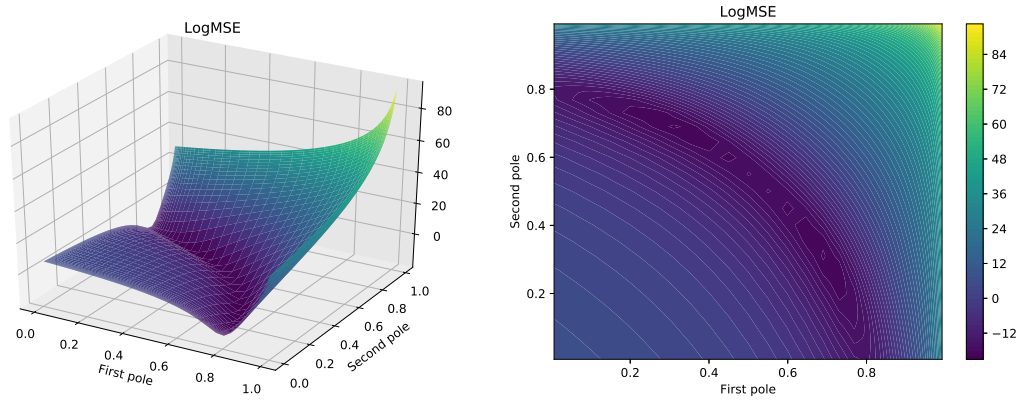
Figure 2.14 shows the obtained loss curves when adapting the criterion. In comparison to Figure 2.8a, we can notice that the flat region in low pole values is less flat, and that the slope for high values is less steep, meaning that the gradients are more even on the left and on the right edges. The logarithmic criterion has the most noticeable effect, and it can also be visualised for second order systems on Figures 2.15 and 2.16, and Figure B.4.

Although these adaptations of the criterion help to flatten the loss function and reduce both vanishing and explosion, they do not bring an definitive solution to the problem. Indeed, the boundary of the partial explosion zone is pushed further towards high pole values, but it is still possible to get gradient issues if the target pole modulus is extremely close to 1. Moreover, the logarithmic criterion is an ad-hoc proposition to solve the problem, and it does not rely on any statistical assumption or mathematical development that would prove its validity.

Smart learning rate adaptation

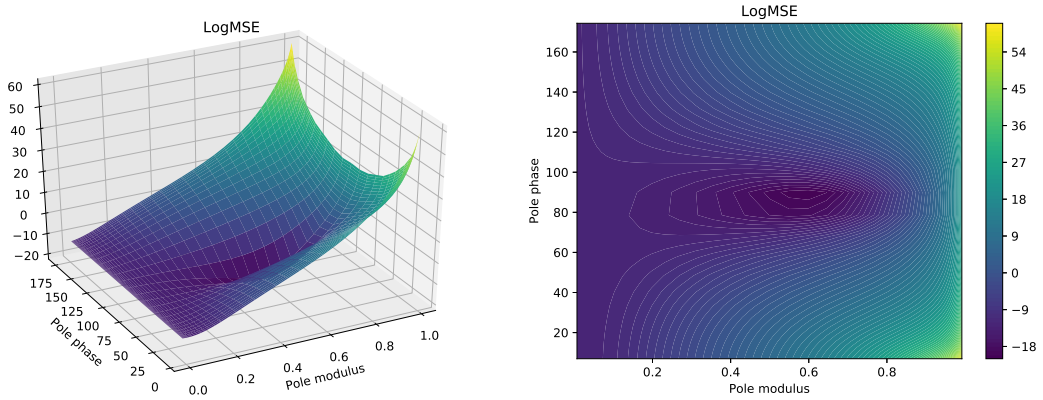
As stated in Section 2.2, the learning rate is a weighting applied to the gradient before updating the parameters of the model. It appears in the gradient descent in (2.24) as η. w_n represents the value of a parameter of the system at iteration n, with $\mathfrak{L}$ the loss function applied on the output of the system.

[6] Euclidean distance.
[7] Sum of absolute values.



(A) 3D view (B) Heatmap

FIGURE 2.15: log(MSE) criterion for second order Neural Filter with target poles 0.7 and 0.3.



(A) 3D view (B) Heatmap

FIGURE 2.16: log(MSE) criterion for second order Neural Filter with target poles $0.6\angle\pm 90°$.

$$w_{n+1} = w_n - \eta \left.\frac{\partial \mathfrak{L}}{\partial w}\right|_{w=w_n} \tag{2.24}$$

Using the learning rate to compensate for undesired gradient values can prevent convergence problems related to vanishing and partial explosion. On one hand, increasing the learning rate in gradient vanishing zones will scale up the incremental learning steps and prevent the convergence to slow down significantly. On the other hand, reducing the learning rate in partial explosion zones will scale down the incremental steps, and prevent the parameter values from suddenly jumping away from the optimum.

In order to obtain the desired effect, the learning rate adaptation must be applied in a smart way. It should be done on a per-dimension basis, as some components of the gradient may explode while others are vanishing. Moreover, the learning rate adaptation must be variable in time, as the parameters change along the training process, and may transition from explosion zones to vanishing ones, and conversely. This description is close to the Adam algorithm

(Kingma and Ba 2014), leading to the hypothesis that Adam is a good candidate to implement the learning rate adaptation. This optimisation algorithm is well known and widely used for RNN training. It uses first and second moment moving estimates of the gradient to compute a normalisation term used to adapt the gradient. This means that when the gradient tends to be small, the normalisation term will push the learning rate up; and when the gradient grows big, the normalisation term will pull the learning rate down. The fact that it relies on moving averages will create smooth transitions between vanishing and exploding zones, and will consistently compensate for the gradient when the parameters stay in one type of zone.

We must however keep in mind that scaling the gradient does not bring a solution to a real explosion problem. Also, the moving averages might not be reactive enough to compensate for a sudden change in gradient if the parameter suddenly goes from a vanishing zone to an explosion zone. Using the a-priori knowledge of the loss surface of our model, we expect that such situations will not appear during Neural Filter training.

Comparative experiments

In order to validate and compare the proposed solutions to gradient issues, we run the training algorithm on identical targets and compare the results obtained when applying the different methods. We also investigate the possibility of combining multiple solutions to improve the results.

As the experiments will involve significantly different objective functions, comparing the average loss on test set is not a good comparison criterion. This is why, after the training is complete, we measure the average MSE loss on the test set as a comparison criterion, regardless of the loss function used for training. This gives a common performance measure for the different methods that we can use for relevant efficiency comparison.

Figure 2.17 shows the measured MSE criterion obtained when using the loss adaptation method to reduce gradient issues. The first results use the MSE objective function for training and serve as baseline for the comparison. For small pole modulus, a slight improvement can be noticed but this effect is merely negligible. With higher poles, the improvement is more visible, and a 86 % improvement is achieved by adapting the criterion to L1 for the last experiment. This validates our assumption that flattening the loss function can reduce the effects of gradient partial explosion, and is further confirmed by the measurements shown in Figure B.5. However, this does not provide a robust solution to the problem as gradient can still explode if the parameters reach the edge of their allowed range, as seen when targeting poles $0.98\angle\pm0°$. For intermediary values of pole modulus, the loss adaptation does not affect the performance of the system.

Figure 2.18 shows the measured criterion when gradient clipping is applied during the learning process. The first bar is the baseline reference. It is clearly visible that gradient clipping improves the performance of the system for high pole modulus, with more than 90 % improvement on the last two experiments. As it only affects high gradient values, this method does not impact the performance of the training outside the gradient explosion zone. We can also see that combining gradient clipping and loss adaptation (the dotted and striped bars on the graph) does not improve the performance of the system, as it breaks the robustness of gradient clipping for edge values (see the before-last target).

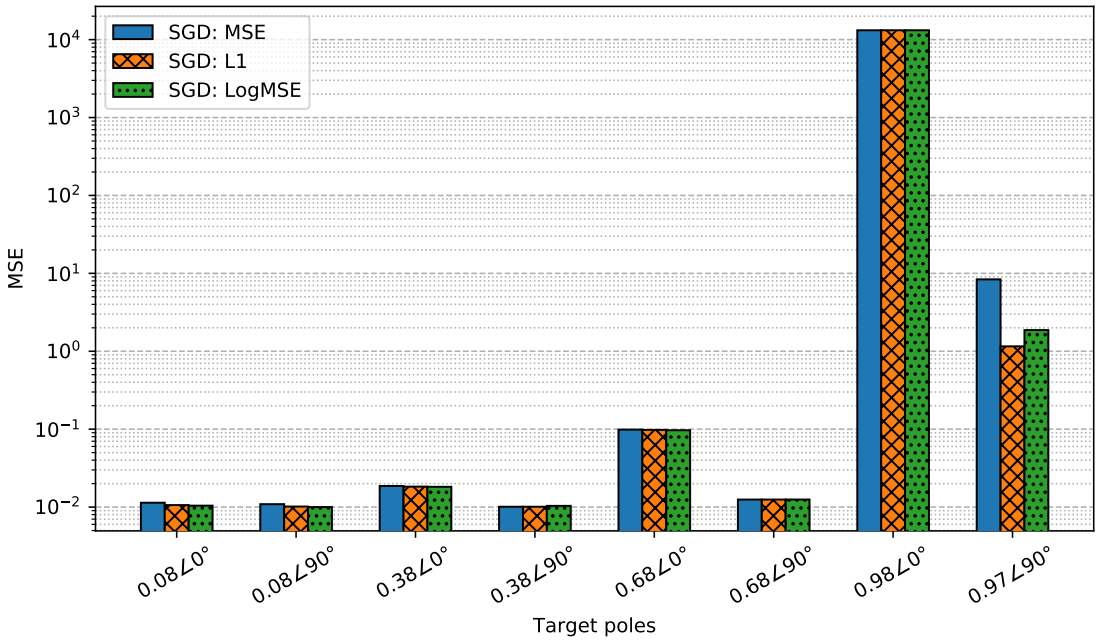


FIGURE 2.17: MSE criterion on Neural Filter when using the loss function adaptation method for training.

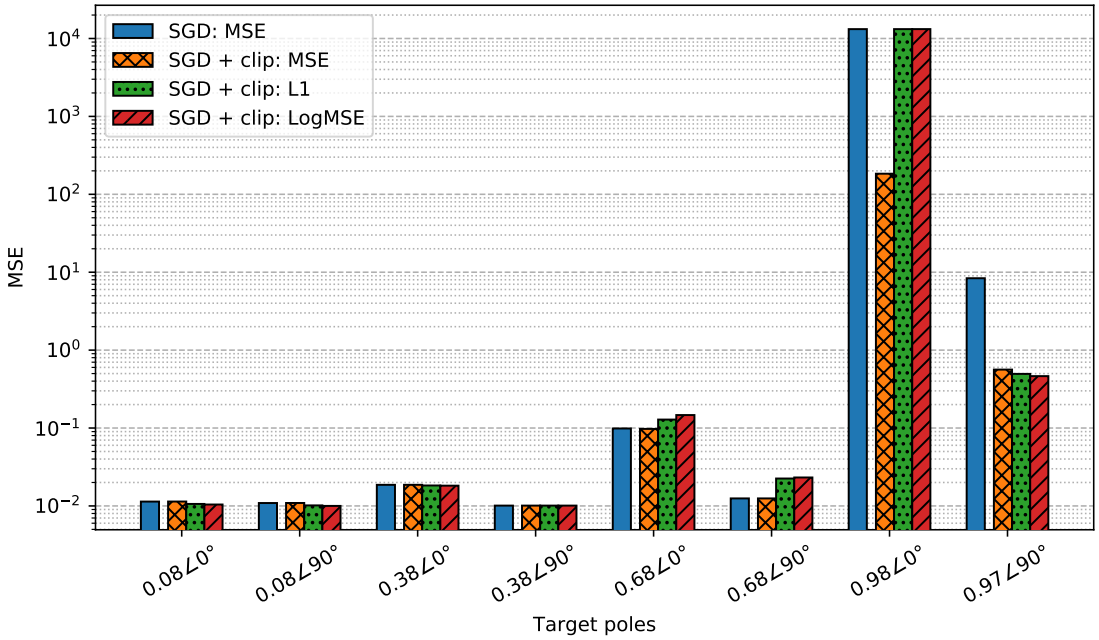


FIGURE 2.18: MSE criterion on Neural Filter when using the gradient clipping method for training.

Figure 2.19 shows the measured MSE obtained by using Adam algorithm for the training. The improvements of over 98 % measured on the two last experiments validate our hypothesis that adapting the learning rate using the Adam algorithm can compensate for the partial

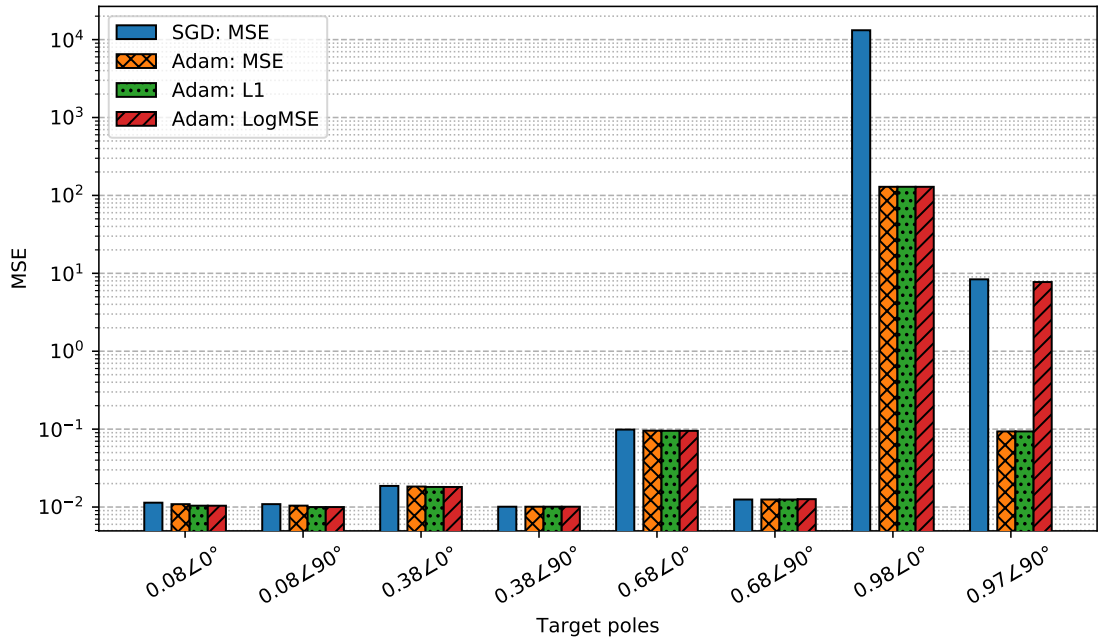


FIGURE 2.19: MSE criterion on Neural Filter when using the Adam algorithm for training.

explosion of the gradient. Negligible improvements are measured for low to mid-range pole modulus. The results show that combining Adam method and a loss adaptation (the dotted and striped bars on the graph) does not improve the performance of the system.

Figure 2.20 shows the comparison of all the proposed methods using only the best matching objective function. None of the investigated methods improves the performance for low to mid-range targets, as no gradient issues are measured in these regions. However, all the proposed methods help improving the final result when facing gradient partial explosion induced by high pole modulus. The Adam algorithm solution stands out for giving the best results in that situation, and this solution is validated as the most efficient one for our model. It will therefore be used for all further experiments. Figure B.6 shows the significant improvements obtained thanks to this algorithm when targeting poles in the partial explosion zone.

2.5 Validation experiments

To test the robustness of our model, we design and run multiple experiments with increasing complexity. These involve changing the target poles and gradually increasing the noise level on the target up to a SNR of 0 dB. The final experiment, exposed hereafter, is the most complex and summarises well the behaviour of the network.

For synthesis purposes, we want to be able to model a system consisting of the sum of multiple IIR linear filters. In order to show that the Neural Filter implementation is able to complete such a task, we design a proof-of-concept experiment. We create a system containing two second order IIR filters. This system has two inputs, that are filtered before being summed. White

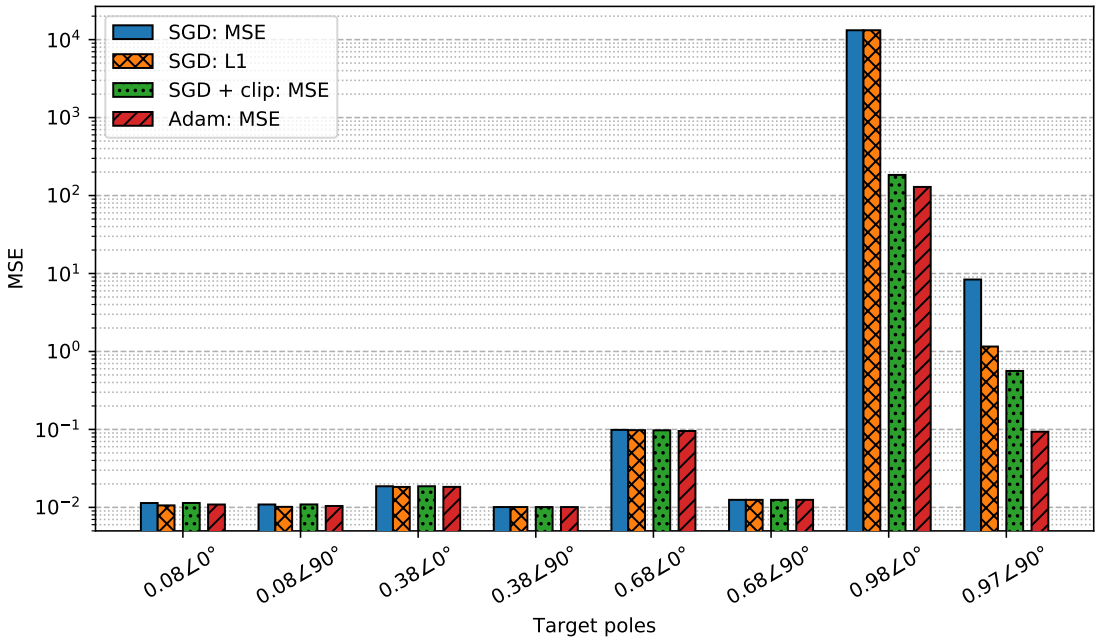


FIGURE 2.20: Compared MSE criterion on Neural Filter when using the proposed methods to solve gradient issue.

noise is then added to this signal in order to obtain a SNR of 0 dB, and the result constitutes the output of the system. This model is illustrated in Figure 2.21.

We build a database of 500 pairs of white noise samples with a size of 200 time steps each. These pairs are processed by the system to obtain the corresponding reference outputs. These input and reference samples are used to train a model containing the sum of two Neural Filters for 50 epochs. The resulting model is then evaluated by comparing the frequency responses of its inner filters to the ones from the target system. We run this experiment with multiple combinations of target inner filters.

Figure 2.22 shows the frequency responses of the systems for various pairs of target filters. In every situation, the Neural Filter model is able to reproduce the target frequency response with a high fidelity, thus confirming that this model can be used to learn a sum of arbitrary second order IIR filters. Moreover, the Neural Filters seem robust to noise since a SNR as low as 0 dB did not prevent it from correctly converging.

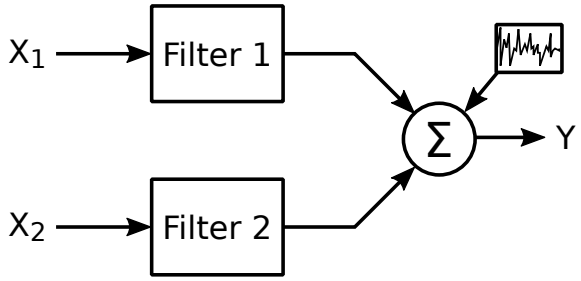


FIGURE 2.21: Sum of filters experimental system. The two input signals are filtered by different IIR models, then summed. White noise is added to the output.

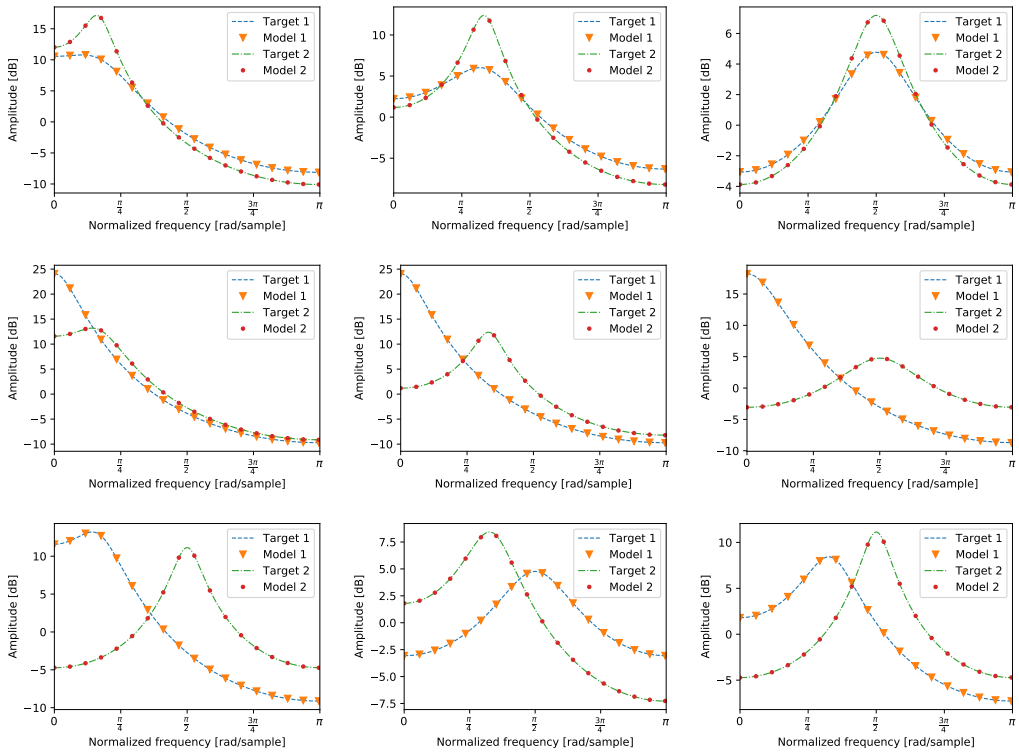


FIGURE 2.22: Frequency response of two-filters systems and their Neural Filter model after training.

2.6 Conclusion

The purpose of this chapter was to investigate the implementation of muscle models that can be trained as ANNs. After explaining why existing systems are not best fitted for this task, we introduced the Neural Filters, our own implementation of second order linear recurrent units with an IIR filter structure designed to model SDM systems. It is an adaptation of standard RNN units, and can be trained with iterative gradient descent. By deriving its gradient back-propagation equations, we identified potential convergence issues caused by gradient vanishing and partial explosion. We analysed these behaviours and proposed solutions based on objective function, gradient and learning rate adaptation. The analysis of the loss function profile led us to assume that the Adam optimiser would be an efficient solution to gradient issues. Comparative experiments allowed us to confirm this hypothesis, and to conclude that using the Adam algorithm gives better results than the other proposed methods in all situations.

Multiple experiments with different noise levels and various target systems were run to check the behaviour of the Neural Filters in all situations. The obtained results let us conclude that this system can efficiently model SDM systems with any arbitrary poles, and that it is robust to noise on the learning data. Additional tests allow us to believe that multiple Neural Filters can be used together to model systems of higher complexity, as shown by training a sum of filters.

Chapter 3

Intonation contour modelling

This chapter describes the current intonation synthesis system using atom decomposition, and how we want to modify its structure. We will first explain the principle of atom decomposition intonation modelling, before showing how it can be used for speech synthesis. We will finally define our evaluation criterion and the experimental setup used with this system.

3.1 Atom decomposition intonation modelling

Atom decomposition intonation modelling (Gerazov, Honnet, et al. 2015) tries to give a physiologically plausible way to model the intonation curve. Its principle is to reconstruct the logarithm of the fundamental frequency of speech by summing a finite number of elementary blocks that we call atoms. More specifically, these atoms are meant to represent the action of

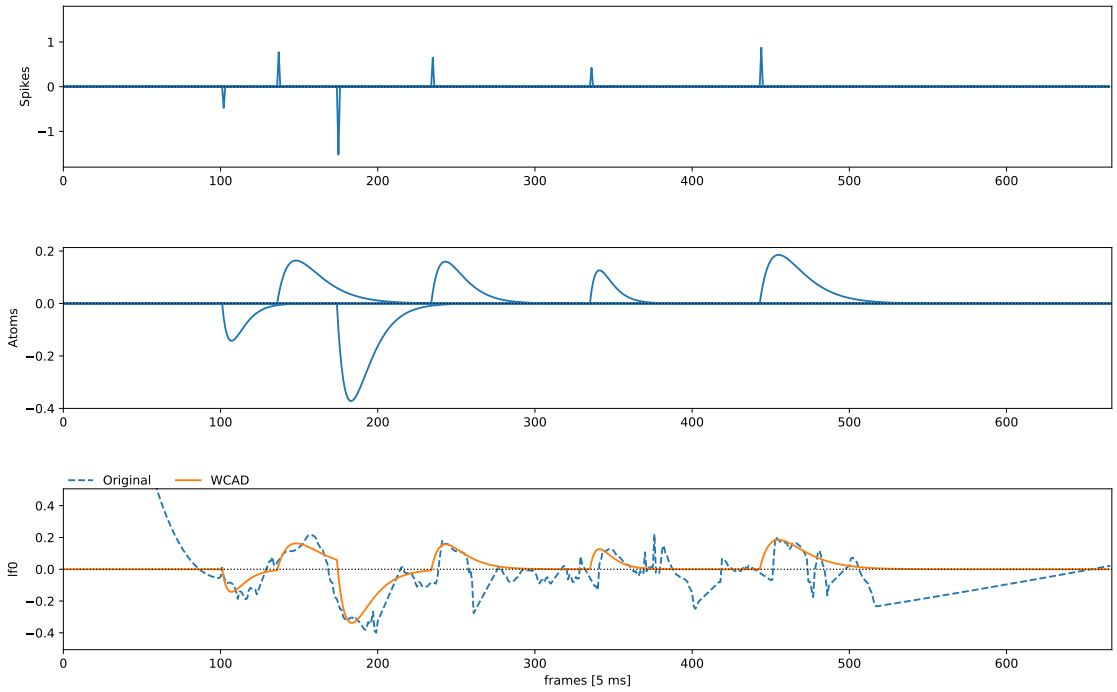


FIGURE 3.1: Atom decomposition intonation modelling. From top to bottom: 1. Spike command signals. 2. Generated muscle responses. 3. Intonation curve reconstruction (solid), and original (dashed).

laryngeal muscles in speech generation. They have the shape of the impulse response of muscle models. When creating a model, a chosen set of atoms of varying length are selected to constitute a dictionary. Reconstructing intonation curves is then achieved by summing correctly positioned and scaled atoms taken from the dictionary. The position and amplitude information of every atom in the representation is translated to spikes, giving a sparse discrete representation of the intonation curve. The physiological interpretation of the spiky signals is that they correspond to the nerve impulses that command the laryngeal muscles. This model is named WCAD (**W**eighted **C**orrelation **A**tom **D**ecomposition).

Figure 3.1 illustrates atom decomposition intonation modelling on a real speech utterance. The plot on the top contains the spikes that correspond to the atoms drawn in the middle plot. The bottom curves represent the logarithm of the fundamental frequency of a speech utterance, and its reconstruction obtained by summing the atoms. The fundamental frequency, also called pitch, is the inverse of the glottal closure period. We refer to the fundamental frequency as f_0 and to its logarithm as lf_0.

Atom decomposition intonation modelling is a generalised Fujisaki's CR model (Fujisaki and Hirose 1984). The spikes correspond to a control signal, that is filtered by muscle models and summed to reconstruct lf_0. The model targets the logarithm of the pitch according to Fujisaki's proof that intonation components are additive in logarithmic space. The atom summation is designed to generate intonation curves with a constant average. However, real speech has a slow pitch mean shift, that is supposedly coming from the lungs. Fujisaki refers to this as the phrase component in the CR model, and it can be modeled using a longer atom as shown in Figure 3.2. This phrase atom is not a muscle response and is not included in the atom dictionary.

Starting from a pitch curve, it is possible to extract the atom representation by using a matching pursuit algorithm. This iterative algorithm tries to fit the best atom from a static dictionary at the best position with the best amplitude to minimise the distance between the original curve and the reconstruction. A new atom is added at each iteration until the extracted

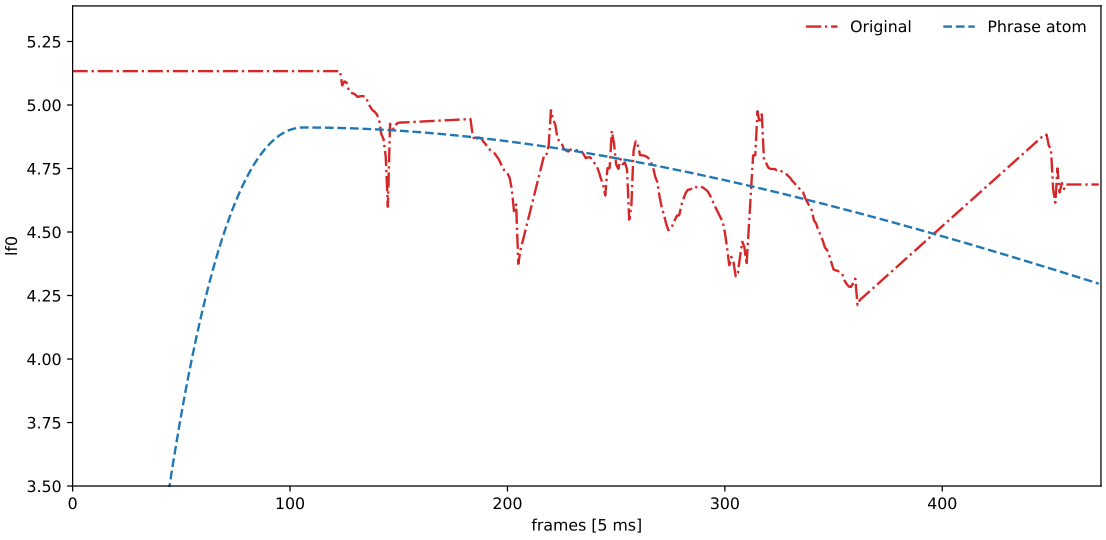


FIGURE 3.2: Phrase component (dashed) in speech lf_0 (dash-dotted).

amplitude falls below a threshold. The matching pursuit stops when the extracted atoms are too small in order to limit the representation to the relevant atoms. Prior to running the matching pursuit, the phrase component is extracted from the utterance by low-pass filtering it and fitting a single large atom on it.

3.2 Atom decomposition intonation synthesis

In order to use this model for text-to-speech synthesis, the spikes must be generated by a dedicated system. It is possible to achieve this by training a RNN to reproduce the WCAD extracted spikes (Schnell and Garner 2018). Its inputs are HTK labels[1] and binary features generated from the text to synthesise, and its training targets are the spikes extracted using matching pursuit with a given dictionary. This system does not directly predict spikes, instead it splits position and amplitude information as two distinct signals. The first output of the system is the position flag, that represents the occurrence of spikes. The second output of the system is a vector with the same dimension as the atom dictionary, containing an amplitude value related to each entry in the dictionary. The last output of the system is a V/UV (**V**oice/**U**n**V**oiced) flag, that represents whether the vocal folds are vibrating or not. This information is later needed by the vocoder[2] to synthesise the voice.

As the RNN does not directly generate intonation curves, its outputs must be processed before being used for synthesis. The process is the following: when a peak in the position flag occurs above a given threshold, a spike is generated. The amplitude vector is sampled at that precise time, and its maximum amplitude (or minimum in the case of a negative spike) determines which atom is fired as well as its amplitude. The spikes are then translated to atoms, and all generated atoms are finally summed to reconstruct the lf_0 curve. Figure 3.3 shows an example of spikes prediction. The top plot is the amplitude vector, and the second one is the position flag. The third plot represents the corresponding spikes after post-processing of the outputs, and the last one shows the lf_0 curves from which the phrase component has been subtracted.

The RNN does not directly generate spikes because it is not able to learn both position and amplitude information. It is possible to train a RNN to directly generate the intonation contour (Zen and Sak 2015), however such a system does not have a physiologically plausible interpretation.

The current WCAD-based intonation synthesis system has a major drawback: as the extracted spikes serve as reference for the training, their parameters cannot be learned by the network. The choice of a dictionary is arbitrary and made prior to the training. WCAD uses that static dictionary to extract spikes that are thus also static. Having the ability to train both the RNN and the dictionary parameters would allow the system to learn the optimal atom shape and the best spike positions during the training process in order to improve the reconstruction performance. Moreover, automatically adapting the dictionary parameters could help figure out the combination of atoms that works the best for intonation modelling, and allow us to analyse the size and shape of the obtained atoms. This is why we want to modify the system in order to enable end-to-end training.

[1] Text based representation of linguistic features.
[2] A vocoder is a system that synthesises human voice.

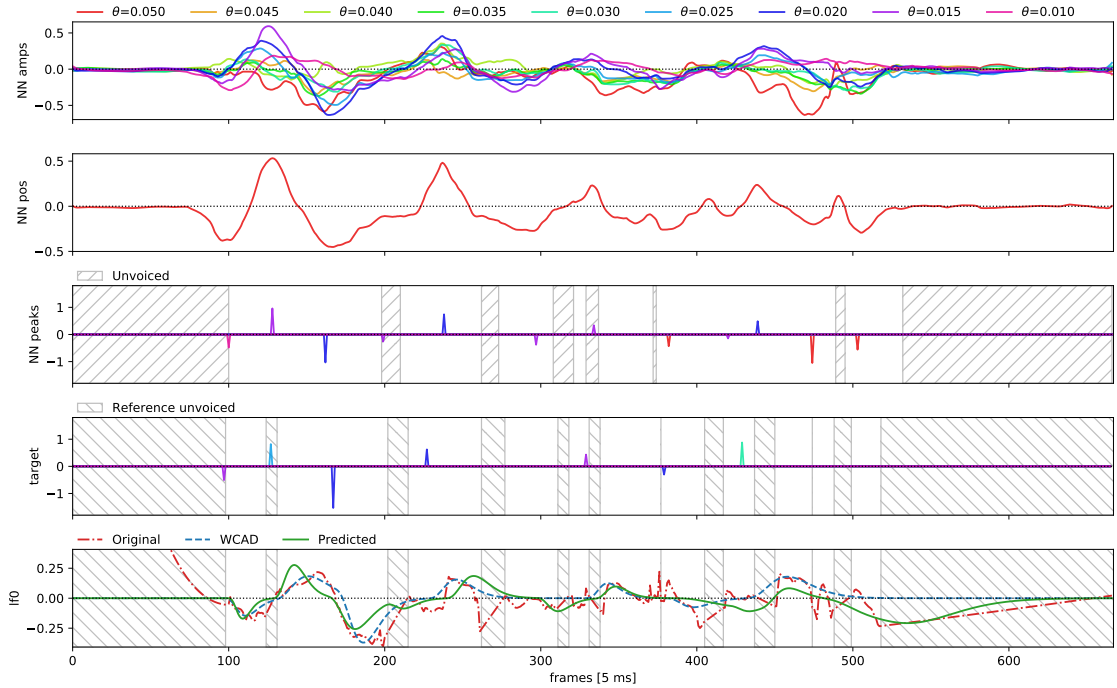


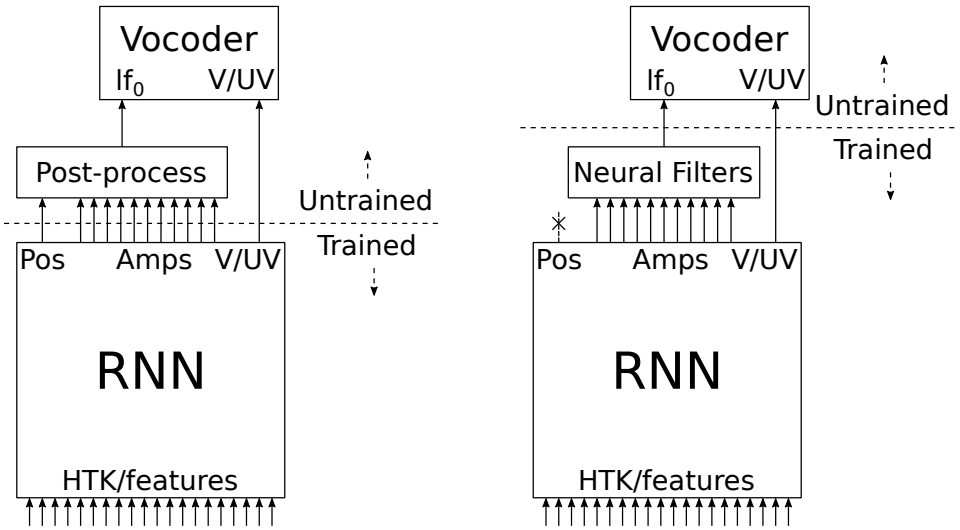
FIGURE 3.3: Atoms prediction with RNN. From top to bottom: 1. Amplitude prediction outputs. 2. Position prediction output. 3. Post-processed predicted spikes. 4. Target spikes. 5. Original intonation curve (dash-dotted), reconstruction using extracted atoms (dashed) and system prediction (solid).

The new architecture that we propose replaces the post-processing using static muscle models by a layer of Neural Filters. The outputs of this layer will be scaled and summed to reconstruct the intonation curve. Among the outputs of the RNN, we want to drop the position flag and connect only the amplitude vector to the inputs of the Neural Filters. This way, we want to force the RNN to learn both timing and amplitude of the spikes in one signal, and we believe this can be achieved with the new model thanks to the greater flexibility brought by end-to-end training. This is supported by the fact that amplitude curves in the current system already contain some temporal information, as seen in Figure 3.3. The comparison between the current architecture and the target one is shown in Figure 3.4. It clearly shows which parameters of the system can be learnt in each case, and we see that only the target system allows end-to-end training from the HTK labels and text features to the intonation curve.

3.3 Evaluation criterion

The main focus of this thesis is the design and the validation of a new synthesizer architecture. In order to compare the performance of intonation synthesis systems, we must define an appropriate measure of the quality of the produced lf_0. There are two main possibilities:

- *Subjective listening tests*: multiple audio samples of the same utterance synthesised using different techniques are ranked by participants based on their subjective listening appreciation.



(A) Current (B) Target

FIGURE 3.4: WCAD intonation synthesizer structure comparison.

- *Objective scores*: a criterion based on the synthesised samples is computed according to a well-defined equation.

Subjective scores give a very good measure of the quality level of audio samples. Indeed, multiple voice factors such as voice naturalness are difficult to define with an equation, but very easy to detect through listening. This is why subjective listening tests are widely used to evaluate speech synthesis systems. However, this type of test takes time to set up and gathering enough participants to obtain high confidence levels is not easy. Moreover, measuring intonation quality through subjective testing can be tricky as listeners tend to evaluate the global quality of the audio samples, and not only the intonation.

Objective criteria are much easier to get, as the score can be computationally measured for every synthesised sample. It is therefore very fast to use and can be applied directly on the intonation curve. However, it is difficult to spot the features that make a speech sample good or bad, and objective scores may not always give a good estimation of the quality of a synthesizer.

Due to time constraints, subjective testing is not an available option for this work. We will only use objective criteria to compare the different systems. This is not a problem for us as our objective is not to measure the quality improvement of the synthesised audio.

We will use the same criteria as the ones defined for the current atom-decomposition based system. The first one is the error rate of the V/UV flag, computed on the whole utterance and expressed in percent. The second one is the RMS (**R**oot-**M**ean-**S**quare) error on the frequency curve (not in the logarithmic space), computed only on the voiced frames (based on the ground-truth V/UV flag) and expressed in Hertz.

3.4 Experimental setup

The samples that we use for training and testing the system are part of the 2008 Blizzard Challenge database (Karaiskos et al. 2008). We use a subset (carroll, arctic, theherald 1,2,3) of the

Roger native English recordings, which is around 6.5 hours long on a 16 kHz sampling rate. 5% of the samples constitute the test database and another 5% form the validation set.

The HTK labels are computed from text using the Festival framework (Black et al. 1999). A framework derived from Merlin (Wu, Watts, and King 2016) is used for text alignment and generation of the 425 binary features used as inputs. The fundamental frequency of the samples and V/UV flag are extracted using the WORLD vocoder, D4C edition (Morise, Yokomori, and Ozawa 2016; Morise 2016). The obtained frames have a period of 5 ms. The lf_0 is interpolated and normalised before the training process.

Chapter 4

End-to-end intonation modelling

This chapter details the integration of Neural Filter muscle models to the WCAD intonation synthesizer system. We will first analyse how the existing muscle models can be translated to Neural Filters. We will then investigate how to take advantage of the new implementation to train the system in an end-to-end manner and describe the different experiments conducted on the system. Every experiment is preceded by a statement of its theoretical basis and assumptions.

4.1 Atom dictionary adaptation

The WCAD synthesizer does not use second order IIR filters as muscle models. Instead, it uses gamma-shaped atoms to model critically damped muscle responses of any order, with an impulse response given by (4.1). This is a continuous time formulation with t representing time. These curves have two parameters: the shape K and the scale θ. The matching pursuit spike extraction implemented in the WCAD framework can only accept these parameters to configure the shape of the gamma atoms.

$$f(t;K,\theta) = \frac{t^{K-1}\,\exp\left(\frac{-t}{\theta}\right)}{\theta^K\,\Gamma(K)} \tag{4.1}$$

$$F(s;K,\theta) = \frac{1}{\theta^K}\frac{1}{\left(s+\frac{1}{\theta}\right)^K} \tag{4.2}$$

In order to replace these atoms by Neural Filters, we must find the relationship between the parameters of the gamma atoms and the poles of an equivalent IIR filter. The Laplace transform of a gamma-shaped atom is given in (4.2). In the particular case where $K = 2$, the expression is identical to a critically damped second-order IIR filter transfer function. This continuous system has a double real pole P_s in $-1/\theta$. As we use the impulse response of the filters to reconstruct the intonation, we discretise this system using an impulse-invariant transform to obtain the equivalent Neural Filter parameters. The poles P_z of the corresponding discrete system are given by (4.3), with T_s the sampling period used for the discrete system.

$$P_z = \exp\left(P_s \cdot T_s\right) = \exp\left(\frac{-T_s}{\theta}\right) \tag{4.3}$$

The amplitude relationship between gamma atoms and Neural Filters does not have to be analytically computed. Indeed, the gamma atoms used for WCAD extraction are normalised

with respect to their L2 norm. The corresponding Neural Filters will have the same amplitude if normalised with respect to the L2 norm of their impulse response. We decided to use an underdamped Neural Filter to model the gamma atoms. Indeed, underdamped systems can be used to model critically damped systems. However, choosing not to constrain the system to be critically damped will allow more flexibility during the training and may improve the results. The underdamped model is preferred to the overdamped one because in practice muscle models are expected to be either critically damped or slightly underdamped (Gerazov and Garner 2016).

We can build a dictionary layer by summing multiple Neural Filters, each affected by their respective normalising amplitude factor. Such a structure is represented in Figure 4.1, where ϕ_n represents a Neural Filter and G_n the associated gain. The Neural Filters parameters and gains can be set up to exactly correspond to any WCAD dictionary. In that case, the output of this layer for a sequence of WCAD spikes will be the exact WCAD reconstructed lf_0.

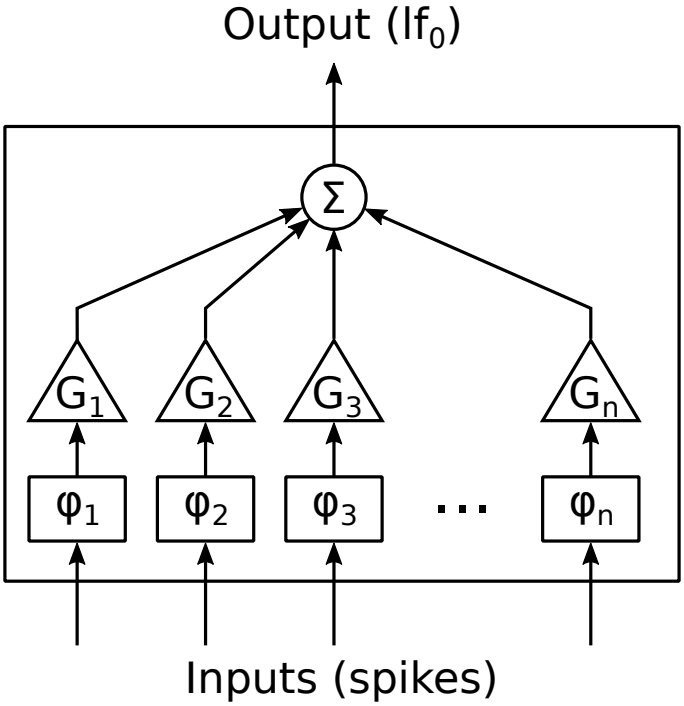


FIGURE 4.1: Neural Filter dictionary layer. The Neural Filters φ_n are associated to a gain G_n and summed to reconstruct the intonation curve.

If we train the dictionary layer starting from that initialisation point, with WCAD spikes as inputs and lf_0 as target, we expect that the parameters of the Neural Filters will not change. Indeed, the matching pursuit algorithm generates the best fitting spikes for a given atom dictionary to approximate the intonation curve. Any other dictionary is not expected to perform better with the same spikes, and therefore the initial parameters should constitute an optimum of the dictionary layer. This assumption can be experimentally verified.

Experiment

The objective of this experiment is to confirm that a dictionary layer initialised on the basis of a WCAD dictionary will not move away from its initialisation point if trained with the corresponding WCAD spikes as inputs. We first discuss the dictionary that is used before giving the results.

Current studies with WCAD do not use second order gamma-shaped atoms, but instead use an atom shape $K = 6$. In order to translate gamma atoms into Neural Filters, we must first define a WCAD dictionary with $K = 2$. We choose to use the same number of atoms as what is defined in previous works, and pick θ values that will generate a dictionary as similar as possible to the one used in the latest work. Thanks to this, a synthesizer based on our new WCAD dictionary will have a similar behaviour to the existing system. Figure 4.2 shows the comparison between the old dictionary ($K = 6$, on the left) and the one that will be used for this experiment ($K = 2$, on the right). The proposed θ is a trade-off between same peak amplitude, same peak position and same atom spread as the old atoms.

For all the samples in the database, we subtract the phrase component and then normalise the intonation curve to zero mean and unit standard deviation. Matching pursuit is then used to extract the WCAD spikes based on our second order dictionary. These extracted spikes constitute our input dataset, and the reference output contains the normalised lf_0 curves.

After initialising the Neural Filters dictionary layer using the parameters derived from the WCAD dictionary, we train it using the described inputs and references. MSE is used as the objective function of the system. The training stops when the test loss variation falls below a threshold of 0.0001 for more than 10 epochs. The experiment results are averaged on 10 different random seeds. We compare the parameters after training to their initialisation point.

The maximum θ deviation measured is less than 1 %, with negligible standard deviation. This confirms our hypothesis that the system is already at an optimal point when initialised with the values corresponding to the WCAD dictionary.

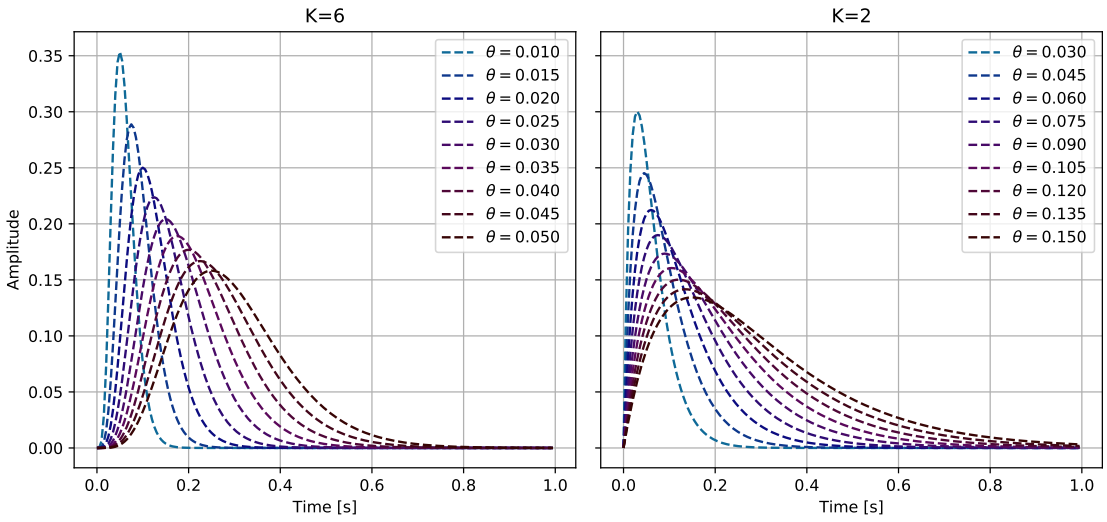
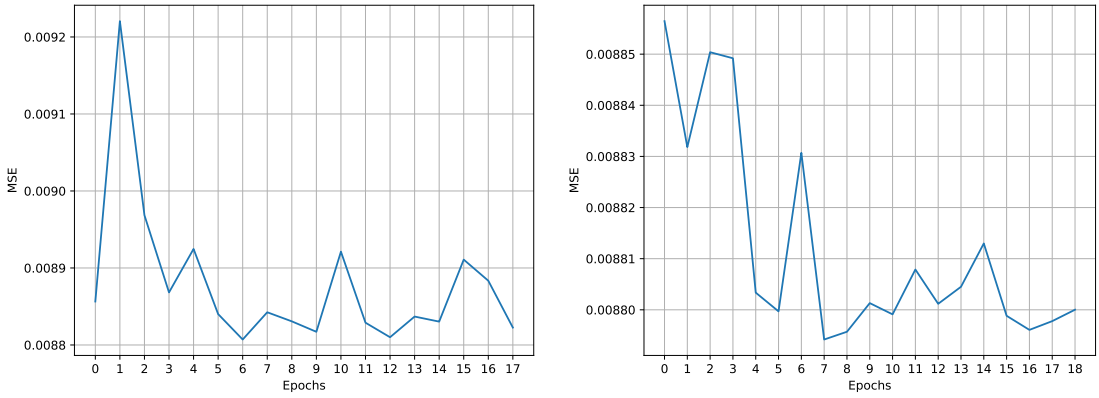


FIGURE 4.2: WCAD atom dictionaries comparison. The plots show the normalized impulse responses of the muscle models.



(A) Adam side-effect

(B) Side-effect compensated

FIGURE 4.3: MSE loss when training the dictionary layer.

This experiment also illuminates a side-effect of the Adam optimiser. During the first iterations, Adam does not know the mean and variance of the gradient, and the learning rate is not accordingly scaled. This can lead to a brief gradient partial explosion happening during the first iterations, as can be noticed when plotting the loss function evolution in Figure 4.3a. The loss increases significantly during the first epoch, reducing the overall performance of the training. However this can be easily avoided, either by pre-training the model without updating the weights to correctly initialise the mean and variance estimates, or by reducing the learning rate to prevent jumps. The loss increase is then avoided as shown in Figure 4.3b. This behaviour is not a major problem, as the impact on performance is very low. We can measure it in this situation because the initialisation point is already the optimum, and any deviation from that point causes a degradation of the loss.

We can add a random perturbation to the initialisation point to check the stability of the system. With the biggest perturbation, a uniform distribution with a maximum value equal to the step between θ values, the system does not move further than a maximum of 10 % away from the WCAD optimum. The final loss variation is negligible, with less than 0.25 %. This means that this optimum is stable, as the system gets back to that point even when small perturbations are applied to its initialisation point.

These experiments have shown that the original WCAD dictionary parameters constitute an optimum of the Neural Filters equivalent representation, and that this optimum is quite stable. This allows us to conclude that the proposed dictionary layer based on Neural Filters correctly mimics the behaviour of the WCAD dictionary, and can therefore replace it.

4.2 End-to-end training

In order to enable end-to-end training from the input text features to the lf_0 curve, we must completely get rid of the post-processing operation and replace it by a trainable system. We propose to achieve this by directly using the amplitude vector coming out of the RNN as input for the dictionary layer. Our objective is to train the system so that the RNN produces sparse and spiky amplitude signals that will serve as excitation for the muscle models, thereby conserving the physiological interpretation of the WCAD intonation model.

The end-to-end model will be trained on the text features as inputs and the normalised lf_0 (phrase atom subtracted) and the V/UV flag as targets. The spiky signals are now an internal representation of the intonation, and are completely trainable. The training procedure differs from the previous model in the following characteristics.

Initialisation point

Previous works have shown that learning a spiky representation is difficult. We will take advantage of the existing WCAD synthesis system to provide a sensible initialisation point for the training of our end-to-end system. We will first train a conventional WCAD synthesizer based on our second order dictionary according to the state-of-the-art procedure. The parameters of the obtained RNN will be used as initialisation point for the RNN present in the end-to-end system. The dictionary layer will be initialised to match the WCAD dictionary used for training. We assume that using this sensible initialisation point instead of randomised values will improve the overall performance of the system.

Objective function

As the target is now the lf_0 curve instead of its spiky WCAD representation, the loss function used for training must be adapted. The criterion that we use contains two contributions: one for the lf_0 curve and one for the V/UV flag.

The loss function related to the intonation curve is a MSE criterion. However, in order to stay consistent with the RMS evaluation criterion, it is computed only on the voiced frames. The reference V/UV flag is used to identify voiced frames. A MSE loss function is also used to penalise the predicted V/UV flag. This is computed on all the frames of the samples. The two terms are then summed to get the global objective function, taking into account both lf_0 and V/UV accuracy. It is possible to assign different weights to these contributions to alter their relative importance in the optimisation. For this work, we will weight them equally.

Training results

The traditional WCAD synthesizer is first trained for 50 epochs. We compute the evaluation score of this system on the validation set defined in Section 3.4, which will serve as our baseline for comparison. It is also used to initialise the end-to-end system, which is then trained for 50 epochs. We compute the evaluation score of the new system and compare it to the baseline in Table 4.1.

Model	**f_0 RMSE**	**V/UV error**
Baseline	28.0 Hz	26.5 %
End-to-end	28.0 Hz	22.2 %

TABLE 4.1: Evaluation score of the end-to-end model.

These results show that the end-to-end system is able to reproduce the performance of the old model, and that it can even improve the V/UV accuracy. However, this model does not

have the behaviour that we are targeting. In order to show that, we plot the internal signals of the system in Figure 4.4. The plot on the top represents the command signals coming out of the RNN, and the middle plot represents the responses that come out of the Neural Filters, multiplied by their respective gains. The last plot represents the original lf_0 and the intonation curve generated by the system.

In the plots, we can see that the command signals are clearly not sparse. This system therefore loses its physiological interpretation as a muscle model controlled by nerve impulses, as the latter ones are supposed to be composed of spikes. Another problem can be spotted in the middle plot: all the filtered curves but one have a very small amplitude, and do not have a big influence on the reconstructed curve. Only the shortest filter (smallest θ value) has a noticeable contribution on the lf_0 curve. This prevents the interpretation of the dictionary as being a set of muscles, because only one Neural Filter has a significant contribution for the reconstruction. The last thing we want to point out in this graph is that the correlation between the command signals and the corresponding filtered curves does not appear clearly. Indeed, the command signals seem to be all active on the whole utterance with similar amplitudes, but only one filtered signal really remains significant. This makes it harder to interpret the command signals as nerve impulse and the filtered ones as the associated muscle responses.

These issues can also be noticed on longer utterances, as shown in appendix Figure B.7. They have different causes, and we will solve them separately in order to make the behaviour of the system physiologically plausible again.

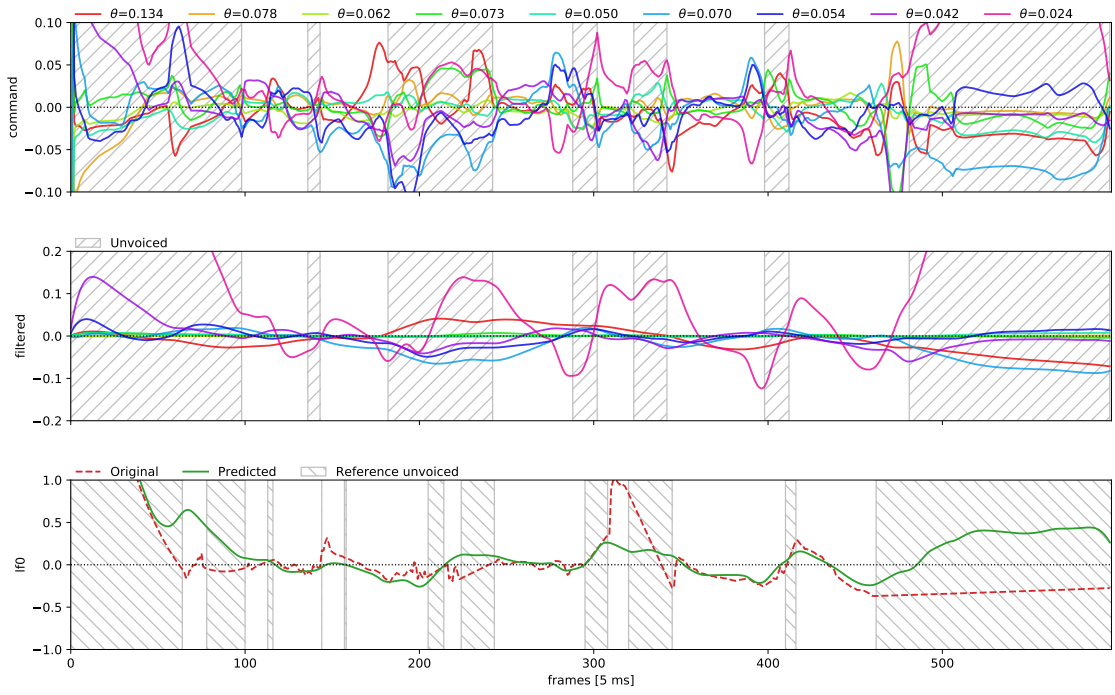


FIGURE 4.4: End-to-end synthesizer internal signals [Roger 6561]. From top to bottom: 1. Command signals. 2. Muscle responses. 3. Reconstructed (solid) and original (dashed) intonation curves.

4.3 Normalisation of the Neural Filters

We will first analyse why the filtered signals have a different amplitude distribution than the command signals they are related to. As can be seen in Figure 4.4, all command signals have similar amplitude ranges across the utterance, but after filtering and multiplying by the gains, most of the signals have a very small amplitude range compared to the last one. If we want to be able to interpret the model using physiological properties such as nerve impulses and muscle models, the distribution of amplitude of the responses must match those of the command signals.

The causes of this discrepancy are the gains associated to each Neural Filter. Indeed, after training the system, the last filter has a gain 10 times higher than the others, as shown on Figure 4.5. This explains why the amplitude ratio of the filtered signals is not the same as for the command signals.

In order to solve this, we want to prevent the network from learning the gains by itself, thereby preventing it from unequally weighting the different contributions. We replace the gain parameters by normalisation factors, which will ensure that the impulse responses of the Neural Filters are always normalised with respect to their L2 norm. To guarantee the unitary L2 norm at every iteration, these coefficients will be automatically updated when the parameters of the Neural Filters are modified. Thanks to this, we expect that a given amplitude distribution on the command signals will always generate a matching distribution on the filtered signals. This differs from the previous implementation where the normalisation of the filters was only guaranteed at the initial iteration.

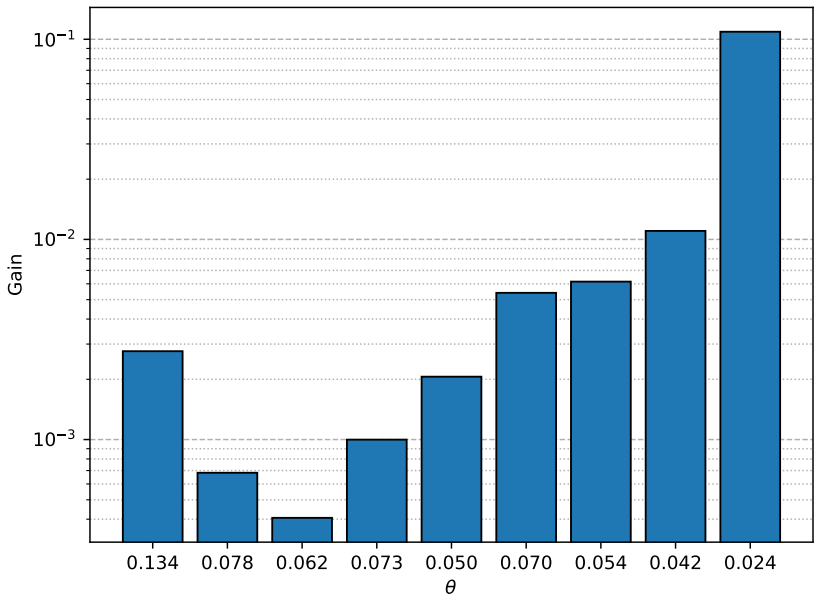


FIGURE 4.5: Neural Filter gains obtained after training.

Normalisation coefficients

The normalisation factors directly depend on the poles of their associated Neural Filter. In order to correctly compute the back-propagation of the gradient through the dictionary layer, this relationship must be expressed mathematically. This formulation is difficult to derive analytically, and its complexity would bring an unnecessary computation load in the system.

In order to reduce the computational impact of the normalisation factors on the system, we opt for a numerical approximation to implement this relationship. We determine heuristically the model to use for the numerical approximation, as a trade-off between accuracy and complexity, as given in (4.4), where G represents the normalisation factor and the A_i are the parameters of the model.

$$G = A_0 + A_1\rho + A_2\rho^2 + A_3\exp(\rho) + A_4\exp(\rho)^2 \tag{4.4}$$

We compute the parameters of this model using a linear least squares method. The target curve is generated by computing the normalisation factor for filters with equally spaced θ values ranging from 0.01 to 0.35. This range encompasses all the values used for the initial dictionary, with a sufficient margin to comprise all the plausible values that the system could take. The real normalisation factor and its numerical approximation are shown in Figure 4.6.

This numerical model has a limited complexity, and it can approximate the the real normalisation factor with a sufficient fidelity. As seen in Figure 4.7, the relative error on the normalisation factor only reaches 5 % for boundary values, and is below 1 % elsewhere. This deviation is acceptable, as even in the worst case the filters will still have a very even weighting.

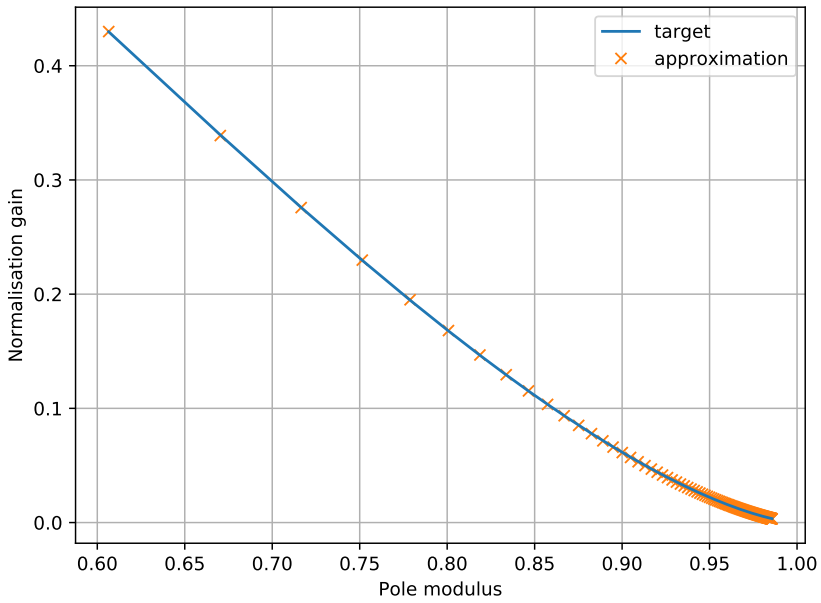


FIGURE 4.6: Normalisation factor of the Neural Filters and numerical approximation.

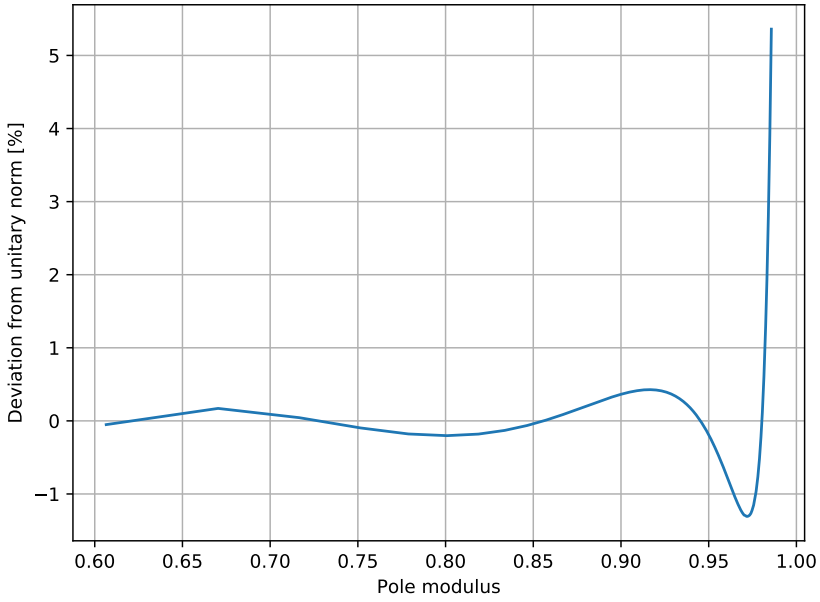


FIGURE 4.7: Numerical normalisation factor relative deviation.

Training results

We use the obtained numerical model to implement the normalisation factors in the dictionary layer, as a replacement for the gain parameters. The obtained normalised end-to-end system is trained for 50 epochs, starting from the same initialisation point as in Section 4.2. The evaluation score of this system is given in Table 4.2, and shows that this system is still able to match the performance of the baseline system.

Model	**f_0 RMSE**	**V/UV error**
Baseline	28.0 Hz	26.5 %
Normalised end-to-end	27.4 Hz	26.4 %

TABLE 4.2: Evaluation score of the normalised model.

Figure 4.8 shows the internal signals of the normalised model. As we can see, the implementation of the normalisation factors has the expected effect of matching the amplitude distributions before and after filtering. Indeed, only one command signal has a non negligible amplitude, and only the associated filtered signal is activated. This validates our solution to the problem of non matching amplitudes, but the two other issues described in Section 4.2 remain.

4.4 Temporal sparsity by regularisation

We will now tackle the non spiky command signals. In order to increase the sparsity of these signals, we propose to add a constraint in the loss function that will draw the command signals towards a sparse representation. Our objective is to let the RNN generate spiky outputs, that we will be able to interpret through an analogy with nerve impulses.

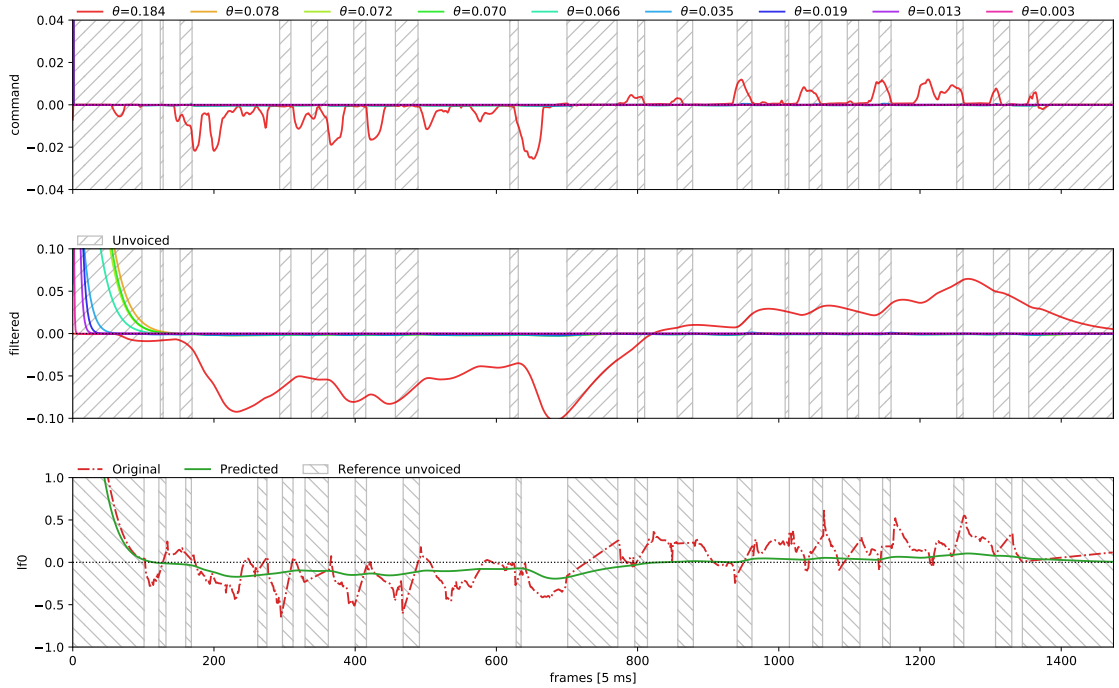


FIGURE 4.8: Normalised end-to-end synthesizer internal signals [Roger 7701]. From top to bottom: 1. Command signals. 2. Muscle responses. 3. Reconstructed (solid) and original (dashed) intonation curves.

The L1 regularisation constraint (also known as Lasso) is known to induce sparse representations (Tibshirani 1996). It involves summing the absolute values of all the penalised parameters and adding this term to the loss function. This means that the optimisation algorithm aims at minimising the absolute values of the penalised parameters. Thanks to the shape and the properties of the L1 norm, the parameters favour being close to zero, resulting in sparsity in the constrained dimension.

In deep learning, the L1 regularisation constraint is usually applied on the weights of the neurons, in order to sparsify the matrices that represent a model. In our case, we do not want the weights to be sparse, but we want to induce temporal sparsity on the outputs of the RNN. This is not a common use case, and we are not aware of other works investigating this problem. To obtain the temporal sparsity, we propose to apply the L1 constraint directly to the outputs of the RNN, for each utterance that goes through the model.

In this system, the RNN outputs a matrix for each utterance. Its first dimension is time, and its second dimension corresponds to the size of the dictionary used. We sum the absolute values of all the elements in this output matrix, which can be seen as first summing across time and then along the filters that compose the dictionary. The summation in the time dimension is meant to induce sparsity on the temporal dimension. We then sum over the dictionary because we want all the command signals to be equally sparse.

The temporal L1 constraint constitutes a new term that is weighted and added to our objective function. The obtained criterion $\mathfrak{L}$ is expressed by (4.5). The w parameters represent the weights associated to each contribution in the loss function. They can be tuned to adjust the relative importance of lf_0 fidelity, V/UV accuracy and temporal sparsity when optimising

the system.

$$\mathfrak{L} = w_{\mathrm{lf}_0} \cdot \mathrm{MSE}_{\mathrm{lf}_0} + w_{V/UV} \cdot \mathrm{MSE}_{V/UV} + w_{\mathrm{L1}} \cdot \mathrm{L1} \tag{4.5}$$

By introducing the temporal L1 constraint in the criterion, we expect the outputs of the RNN to be more spiky. We believe that this constraint will also force the system to use more than a single filter of the dictionary. Indeed, using only one filter results in a non sparse command signal. If sparsity is imposed, such a behaviour will be impossible for the system. We expect that the RNN is going to use large filters to represent the slow evolution of pitch across the sentence, and shorter filters to match the intonation details. In this case, the command signals will be impulses distributed on the different filters of the dictionary, providing both sparse command signals and a good lf_0 reconstruction. If these assumptions are confirmed, the L1 temporal constraint will solve the issues of non sparse command signals, and of a single filter being used for the reconstruction.

Training results

The L1 constraint is introduced in the loss function, and the contributions of lf_0 fidelity, V/UV accuracy and temporal sparsity are weighted equally. The normalisation factors from Section 4.3 are used in the dictionary layer. The system is trained for 50 epochs, starting from the same initialisation point as in Section 4.2.

The obtained evaluation score is given in Table 4.3. We can see that the constrained model using normalisation factors and temporal L1 regularisation is able to reproduce the baseline performance. A slight improvement on the V/UV error can also be noticed.

Model	**f_0 RMSE**	**V/UV error**
Baseline	28.0 Hz	26.5 %
Constrained end-to-end	28.0 Hz	22.4 %

TABLE 4.3: Evaluation score of the constrained model.

Figure 4.9 shows the internal signals of the system. The control signals have become more spiky, allowing us to validate the use of L1 constraint to induce temporal sparsity. Moreover, we can see that the system is now using multiple filters to reconstruct the intonation curve. This confirms our second hypothesis that implementing the temporal L1 constraint prevents the system from using a single filter.

This system fulfils the objectives of this thesis, as it enables end-to-end training while keeping similar performance to the previous WCAD synthesizer and conserving a physiologically plausible interpretation. We will now analyse its behaviour more in detail.

Behavioural analysis

Thanks to end-to-end training, the model is able to optimise the parameters of its filter dictionary. It is therefore possible to analyse the final parameters and to compare them to the dictionary that was used to initialise it. The first noticeable fact is that all the filters of the trained dictionary remained critically damped after the training. Even though the system has

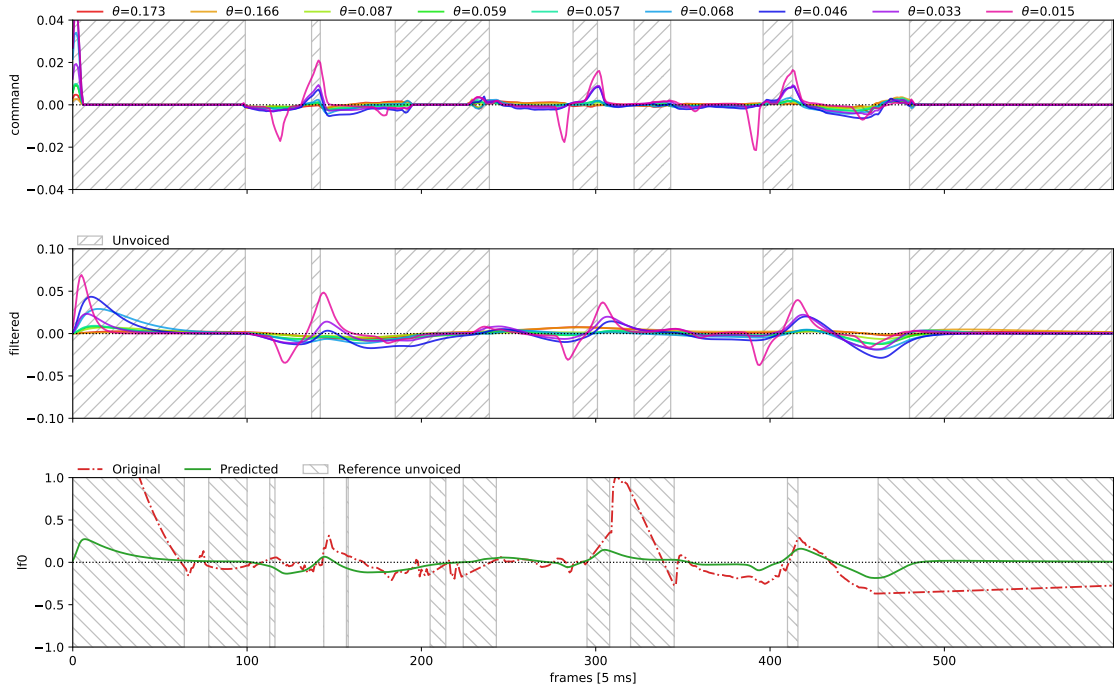


FIGURE 4.9: Constrained end-to-end synthesizer internal signals [Roger 6561]. From top to bottom: 1. Command signals. 2. Muscle responses. 3. Reconstructed (solid) and original (dashed) intonation curves.

the possibility to use underdamped filters, in its final state the dictionary only contains double real poles. This allows us to represent the dictionary using the scale θ associated to the corresponding gamma atom, which facilitates the comparison with the initial point. Table 4.4 lists the scales of the initial and final dictionaries in increasing order.

Initial dictionary	0.030	0.045	0.060	0.075	0.090	0.105	0.120	0.135	0.150
Trained dictionary	0.015	0.033	0.046	0.057	0.059	0.068	0.087	0.166	0.173

TABLE 4.4: Dictionaries of the initial and trained models, by increasing θ value.

We can see that the final dictionary contains a wider range of θ values. The lowest scale is smaller than in the initial setup, resulting in a shorter filter impulse response. The highest scale is larger than in the initial setup, resulting in a longer filter impulse response. This allows the system to model both fine details and slow changes in the intonation curve.

Figure 4.10 shows the impulse responses of the filters that compose the trained dictionary. Apart from the shortest one and the two longest ones, the length of the filters is around 0.25 s. This allows us to think that they carry information relative to the syllables in the utterances, as the average syllable rate in human speech is 4 Hz (Hermansky and Morgan 1994). The shortest filter is most likely related to local accents in the intonation.

The two longest filters are directly related to the phrase component of the intonation. Indeed, their length is much bigger than the average syllable duration. Figure 4.11 shows the internal signals of the system for a longer utterance. In the middle plot, we can see that the two

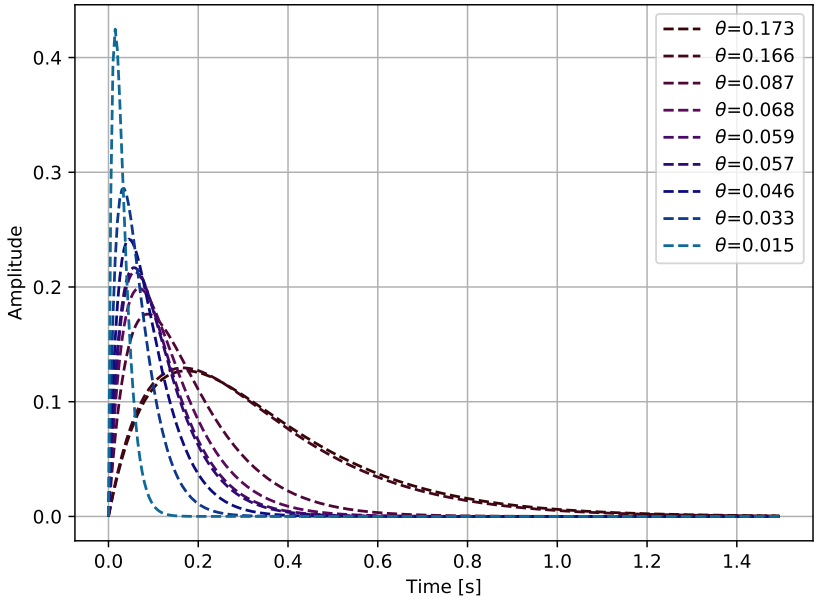


FIGURE 4.10: Normalized impulse responses of the filters in the final dictionary.

long filters (the two first ones in the legend) are used by the system to model the slow evolution of pitch across the utterance. However, they are not used in shorter utterances, such as shown in Figure 4.9. This is due to an imperfect phrase component modelling in the framework.

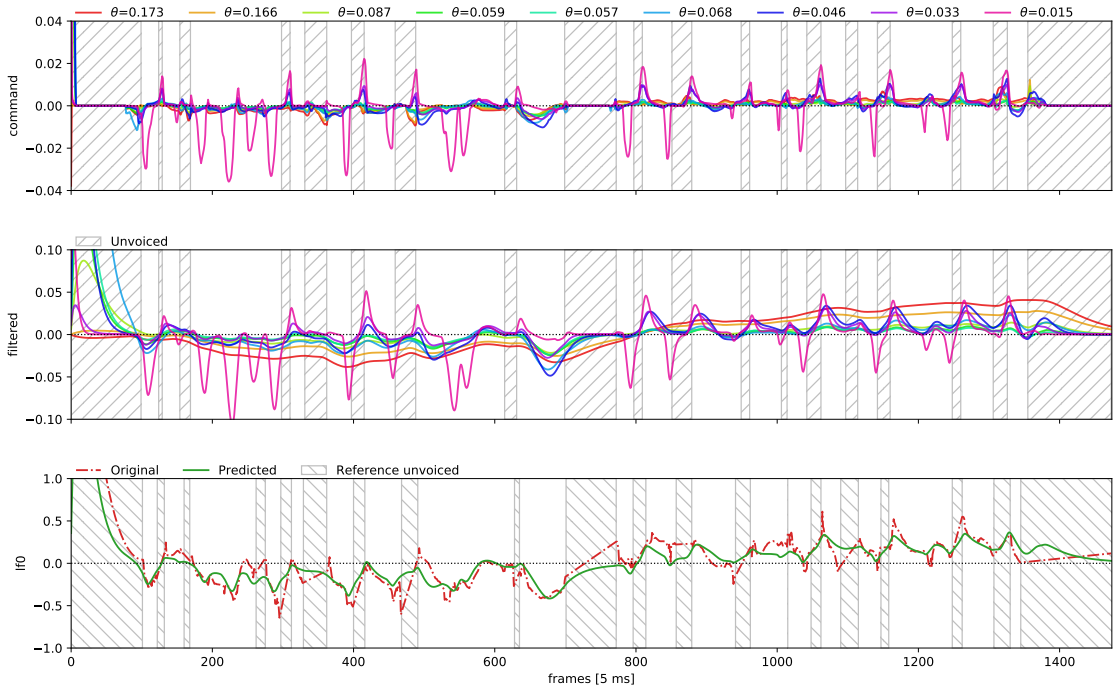


FIGURE 4.11: Constrained end-to-end synthesizer internal signals [Roger 7701]. From top to bottom: 1. Command signals. 2. Muscle responses. 3. Reconstructed (solid) and original (dashed) intonation curves.

The WCAD extraction algorithm assumes that there is only one phrase atom in each sample. This works well for short utterances, but longer ones often contain more than one phrase atom. It happens when the speaker breathes in before the end of the sample. In that situation, modelling the phrase component with only one long atom does not correctly represent the average evolution of the intonation curve. The difference can be noticed on the original lf_0 curves (phrase atom subtracted) in figures 4.9 and 4.11: the shorter utterance has a mostly flat mean, while the average of the longer one seems to slowly increase across the utterance. If the phrase component had been correctly modelled, both curves would have been flat.

This explains why the long filters are not used in short utterances: as the phrase component is correctly modelled and removed, there is no slow pitch evolution in these samples. In longer utterances, the system uses these long filters to compensate for the phrase component modelling error.

When analysing the command signals curves, it appears that some filters have correlated behaviours. It is possible to plot the command signals in clusters to clearly see the different contributions in the intonation curve. We identified three distinct behaviours, leading us to cluster the filters as follows: the shortest filter alone, the two longer filters together and all the other filters in the last cluster.

Figure 4.12 shows the internal signals of the system as clusters, characterised by their mean θ value. These curves allow to clearly see the contributions of the phrase component and the shorter accents in the intonation curve.

The fact that the system generates command signals in clusters can have different causes. On one hand, it is possible that we use too many filters, and that the system needs less than

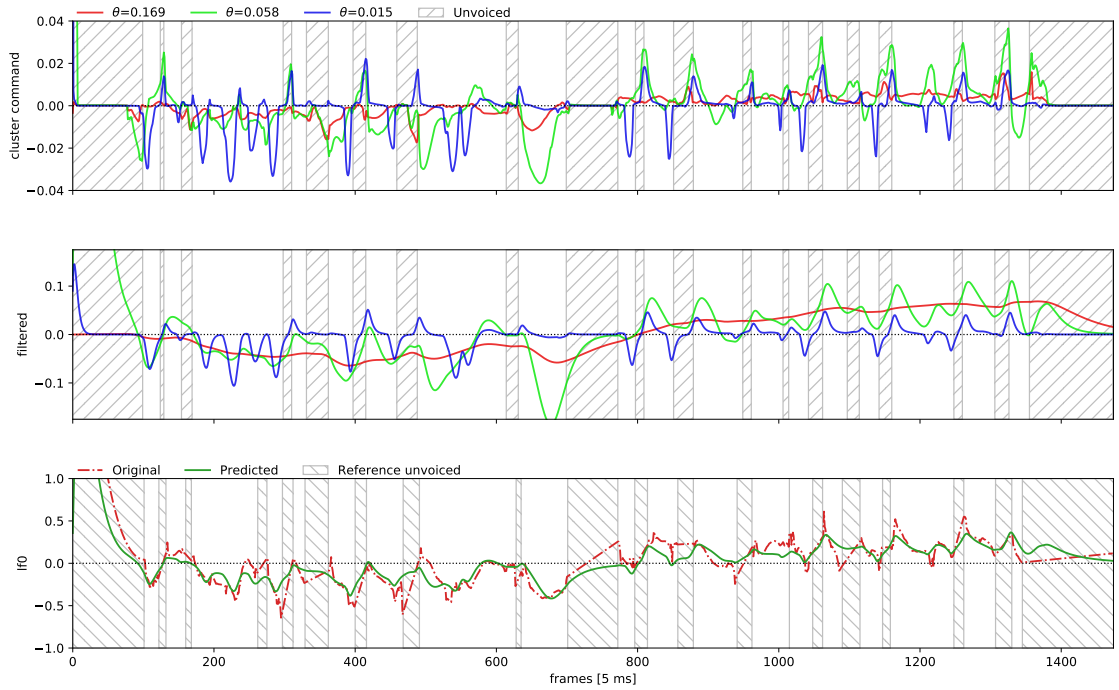


FIGURE 4.12: Constrained end-to-end synthesizer internal signal clusters [Roger 7701]. From top to bottom: 1. Command signals. 2. Muscle responses. 3. Reconstructed (solid) and original (dashed) intonation curves.

9 filters to correctly reconstruct the intonation curve. The clustered filters could then be removed and replaced by a single filter with the same effect. On the other hand, it is possible that the system needs muscle models of higher complexity to correctly reconstruct the intonation curve. In that case, the clustering would be a clue that the system is summing multiple second order filters to generate more complex models.

The current experiments and results do not allow us to favour one of these hypotheses. Further measurements would be needed to understand the clustering behaviour of the system.

4.5 Influence of the sensible initialisation

For the previous experiments, the RNN was initialised by training it using the procedure described in previous WCAD works. This initial point was chosen to facilitate the learning process of the network, following the assumption that the pre-trained RNN is closer to the optimum than a completely random initialisation.

We can test this hypothesis by training the system from a totally random initialisation of its parameters. We train the system for 150 epochs, with the normalisation coefficients and the temporal L1 constraint. More epochs are needed for the training because the system is not pre-trained. The evaluation score of the obtained system is given in Table 4.5.

Model	**f_0 RMSE**	**V/UV error**
Baseline	28.0 Hz	26.5 %
Constrained end-to-end	28.0 Hz	22.4 %
Randomised end-to-end	40.1 Hz	33.6 %

TABLE 4.5: Evaluation score of the randomised model.

We can see that the performance of the system is significantly worse when the sensible initialisation point is not used. This confirms our hypothesis that pre-training the RNN helps the system to find a better optimum and improves its evaluation score.

This experiment is also the opportunity to analyse the behaviour of a randomly initialised system to check if it can learn a physiologically plausible interpretation.

Figure 4.13 shows the internal signals of the trained system. Its behaviour is very similar to the one analysed in Section 4.4. It has spiky command signals and uses multiple filters to reconstruct the intonation curve. The phrase component contribution is easily noticeable, and the command signals have a clustered behaviour. The major difference is that the dictionary of this system is much wider than the previous ones, including much shorter filters as well as much longer ones.

The smallest and largest θ values in this wide dictionary are on the edges of the range that was used in Section 4.3 to compute the numerical approximation of the normalisation coefficients. As shown before, the deviation on the normalisation factors is small enough for such values, and no significant weighting unbalance can be noticed between the top and the middle plot.

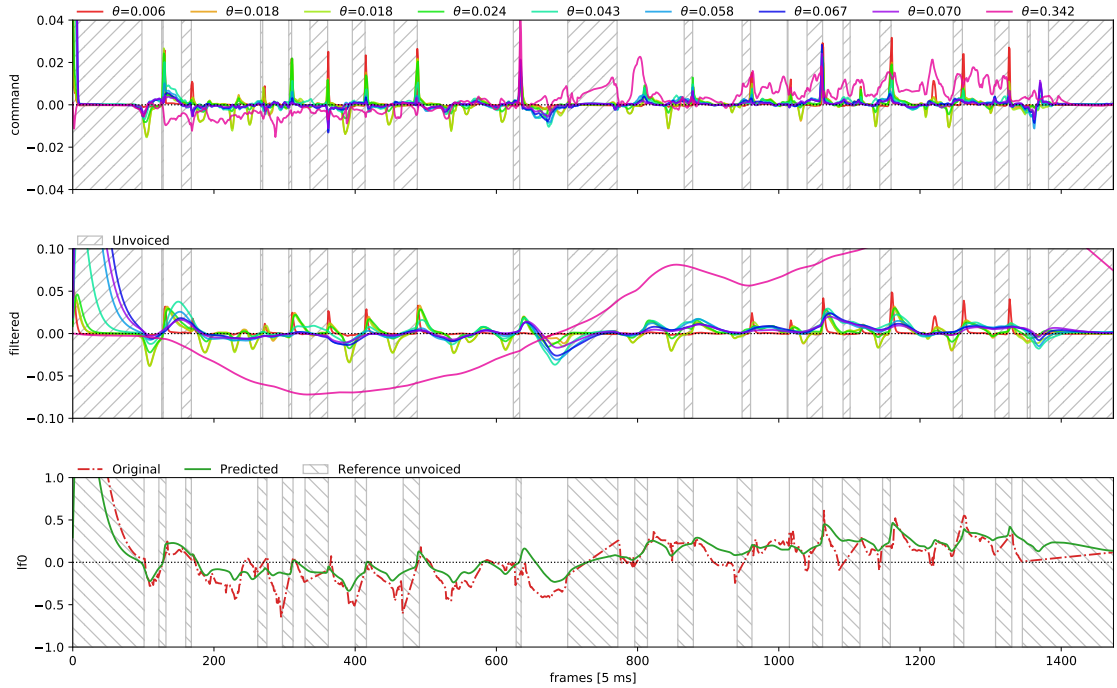


FIGURE 4.13: Randomly initialised end-to-end synthesizer internal signals [Roger 7701]. From top to bottom: 1. Command signals. 2. Muscle responses. 3. Reconstructed (solid) and original (dashed) intonation curves.

These observations allows us to state that a randomly initialised model is able to learn the same physiologically plausible behaviour as a pre-trained system. It mainly differs in its very wide dictionary range, but can be interpreted similarly to a pre-trained system.

4.6 Conclusion

The objective of the experiments in this chapter was to modify the existing WCAD intonation synthesizer in order to allow end-to-end training while conserving its physiologically plausible interpretation. We first assumed that the muscle responses could be efficiently modelled by Neural Filters. We therefore explained the choice of the filters used to build a dictionary, and showed that replacing gamma-shaped atoms by Neural Filters did not impact the optimum of the system. This allowed us to conclude that Neural Filters are a valid replacement for WCAD muscle models, as they correctly mimic their behaviour.

We explained how we integrate the Neural Filters in the implementation of the synthesizer, and how we initialise and train the obtained system in an end-to-end way. These results showed us that the end-to-end system is able to reconstruct intonation curves. However, plotting the internal signals of the system allowed us to show that it does not have a physiologically plausible behaviour. The three issues that we identified were non sparse command signals, unevenly weighted filters and single filter reconstruction of the intonation.

We assumed that implementing normalisation coefficients in the filters would solve the unequal weighting of the different contributions. This hypothesis was tested and confirmed experimentally. We proposed an implementation of a temporal L1 constraint as a solution to

enforce both sparsity in the command signals and the use of multiple filters to reconstruct the pitch. Experimental results confirmed the efficiency of this solution.

Finally, we stated the hypothesis that using a sensible initialisation of the model before training improves the performance of the system. Analysing a randomly initialised system showed that it was still able to learn a physiologically plausible behaviour, but that its performance was significantly reduced. This allowed us to validate the impact of a sensible initialisation on the performance of the system.

The pre-trained end-to-end system including normalisation coefficients and temporal L1 constraint is able to match the performance of the original WCAD synthesizer, while also having a physiologically plausible behaviour. This system meets our objectives, and allows us to conclude that neural networks can be used to model physiologically plausible behaviours in intonation synthesis.

Chapter 5

Conclusions

5.1 Conclusion

In this thesis, we have investigated the implementation of second order linear recurrent units to serve as muscle models, and we have integrated them to an intonation synthesizer based on atom decomposition to build an end-to-end system with a physiologically plausible behaviour. We have studied the problem from the ANN point of view, with the goal to propose a synthesizer able to achieve the same objective evaluation score as the WCAD synthesizer on which this work is based.

In the first part, we have shown that it is possible to design digital IIR filters to be trained by gradient descent in ANN frameworks. We have discussed the gradient issues that can occur in these units, and explained why using the Adam algorithm improves convergence by compensating the gradient instabilities. We have introduced the name of Neural Filters to refer to these filtering units.

In the second part, we have described how to integrate Neural Filters in an atom decomposition intonation synthesizer. We have shown that it is possible to enforce this system to have physiologically plausible behaviours by implementing normalisation and sparsity constraints in the model. We have explained how a temporal L1 constraint can be applied to the outputs of a RNN in order to make them sparse and spiky. We have finally trained the obtained system in an end-to-end manner, and shown that it is able to match the performance of the WCAD system and that its behaviour has a physiologically plausible interpretation.

5.2 Future directions

By analysing the behaviour of the end-to-end intonation synthesizer, we have noticed that it is able to compensate for wrong phrase component modelling. This suggests that the system could be able to completely model the intonation curves, including the phrase component. This would represent an improvement on the WCAD model, where the phrase atom is not integrated in the synthesizer. We expect that the end-to-end system would perform well if trained on raw lf_0 curves including the phrase component.

Plotting the internal command signals of the end-to-end system showed that the different muscle models behave similarly in clusters. We have formulated two hypotheses to explain this phenomenon. The first one is that our implementation of the synthesizer includes too many filters. Some of them would therefore be redundant, explaining the similarities in the command signals. Our second hypothesis is that the system combines multiple second order units to

create more complex muscle models. This would mean that second order is not sufficient to properly model muscle responses, and that more complex units should be integrated in the synthesizer. Further experiments would be needed to validate these hypotheses, but we expect that the clustering behaviour of the system results from a combination of these two causes.

We believe that the end-to-end system can improve the perceived quality of the synthesised intonation. Subjective listening tests would give a better criterion to evaluate the performance of the system than the objective scores used in this thesis. Although the RMS objective evaluation criterion has been used to compare the different results in this work, it may not be an exact measure of the quality as estimated by subjective listening. It would be interesting to replace this criterion by a more physiologically plausible metric such as weighted correlation and to measure the performance of the system according to that new criterion. It would also be sensible to use this metric as the objective function for the end-to-end training to go one step further in physiological plausibility.

Expressive speech synthesis could take advantage of this system. Indeed, it is difficult to infer the differences in expressiveness between two speech utterances by only looking at the pitch curves. However, the sparse representation of the command signals may carry easier to read information, such as the frequency of spikes, the type of filters activated or the amplitude of the spikes. This could be exploited in emotional speech synthesis, by changing the properties of the command spikes to create different sounding emotions in the utterances.

Appendix A

Gradient derivation proof

First order Neural Filter

With y the output of the filter, a the optimized parameter, σ the sigmoid function and x the input of the filter (sampling time k):

$$y_{(k)} = x_{(k)} + \sigma(a) \cdot y_{(k-1)}$$

$$\begin{aligned}
\frac{\partial y_{(k)}}{\partial a} &= \frac{\partial y_{(k)}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial a} + \frac{\partial y_{(k)}}{\partial y_{(k-1)}} \cdot \frac{\partial y_{(k-1)}}{\partial a} \\
&= y_{(k-1)} \cdot \sigma'(a) + \sigma(a) \cdot \frac{\partial y_{(k-1)}}{a} \\
&= y_{(k-1)} \cdot \sigma'(a) + \sigma(a) \cdot \left(y_{(k-2)} \cdot \sigma'(a) + \sigma(a) \cdot \frac{\partial y_{(k-2)}}{a} \right) \\
&\vdots
\end{aligned}$$

$$\frac{\partial y_{(k)}}{\partial a} = \sum_{n=0}^{k-1} \left[y_{(k-1-n)} \cdot \sigma'(a) \cdot \sigma^n(a) \right]$$

Second order Neural Filter

With y the output of the filter, α and β the optimized parameters, and x the input of the filter (sampling time k):

$$y_{(k)} = x_{(k)} + \alpha \cdot y_{(k-1)} + \beta \cdot y_{(k-2)}$$

$$\begin{aligned}\frac{\partial y_{(k)}}{\partial \alpha} &= y'_{(k)} + \frac{\partial y_{(k)}}{\partial y_{(k-1)}} \cdot \frac{\partial y_{(k-1)}}{\partial \alpha} + \frac{\partial y_{(k)}}{\partial y_{(k-2)}} \cdot \frac{\partial y_{(k-2)}}{\partial \alpha} \\ &= y_{(k-1)} + \alpha \cdot \frac{\partial y_{(k-1)}}{\partial \alpha} + \beta \cdot \frac{\partial y_{(k-2)}}{\partial \alpha}\end{aligned}$$

$$\begin{aligned}\frac{\partial y_{(k)}}{\partial \beta} &= y'_{(k)} + \frac{\partial y_{(k)}}{\partial y_{(k-1)}} \cdot \frac{\partial y_{(k-1)}}{\partial \beta} + \frac{\partial y_{(k)}}{\partial y_{(k-2)}} \cdot \frac{\partial y_{(k-2)}}{\partial \beta} \\ &= y_{(k-2)} + \alpha \cdot \frac{\partial y_{(k-1)}}{\partial \beta} + \beta \cdot \frac{\partial y_{(k-2)}}{\partial \beta} \\ &\vdots\end{aligned}$$

$$K_n = \begin{cases} \alpha K_{n-1} + \beta K_{n-2} & \text{if } n > 0 \\ 1 & \text{if } n = 0 \\ 0 & \text{if } n < 0 \end{cases}$$

$$\frac{\partial y_{(k)}}{\partial \alpha} = \sum_{n=0}^{k-1} \left[y_{(k-1-n)} \cdot K_n \right]$$

$$\frac{\partial y_{(k)}}{\partial \beta} = \sum_{n=0}^{k-2} \left[y_{(k-2-n)} \cdot K_n \right]$$

Appendix B

Additional figures

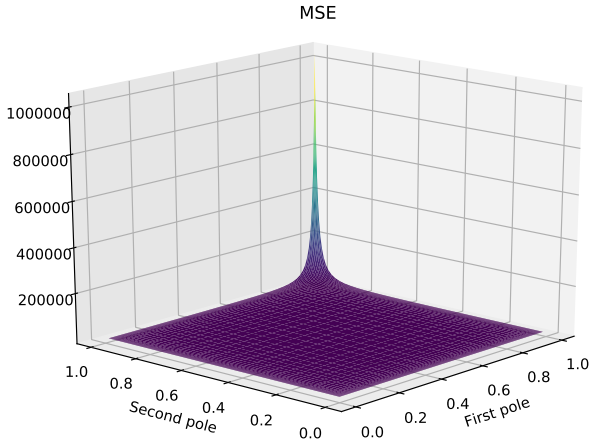


FIGURE B.1: MSE loss for second order Neural Filter with target poles 0.7 and 0.3.

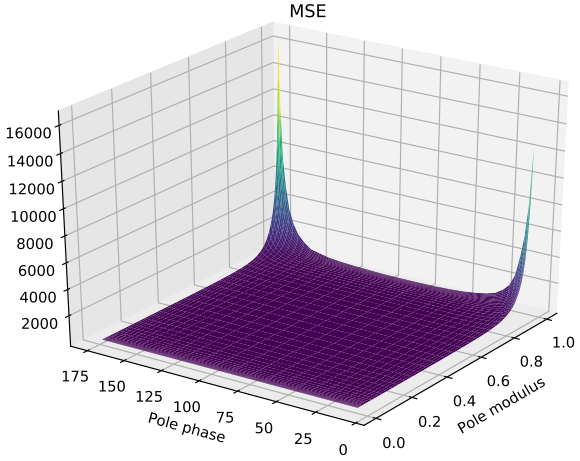


FIGURE B.2: MSE loss for second order Neural Filter with target poles $0.6\angle\pm90°$.

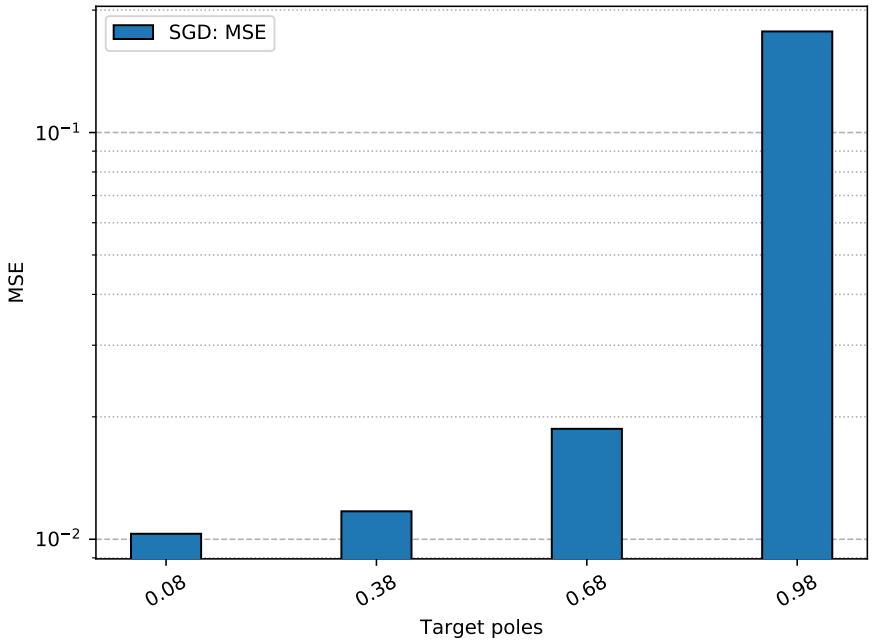
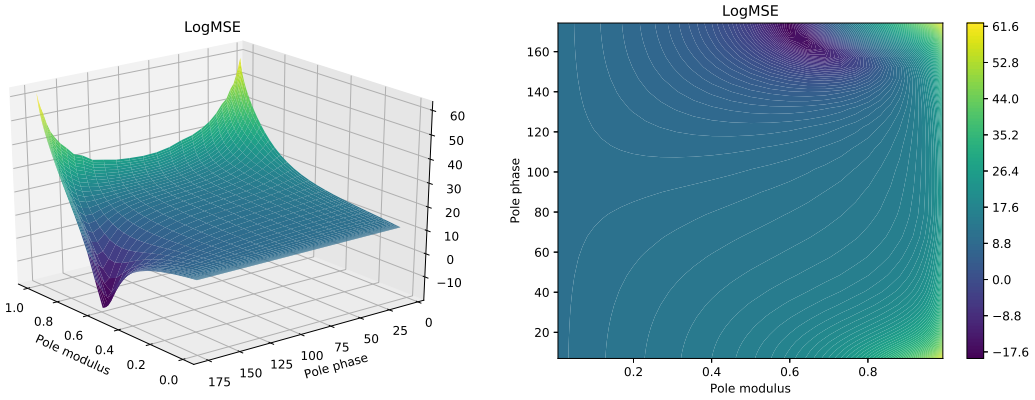


FIGURE B.3: MSE on Neural Filter for first order targets.



(A) 3D view

(B) Heatmap

FIGURE B.4: log(MSE) criterion for second order Neural Filter with target poles $0.6\angle \pm 180°$.

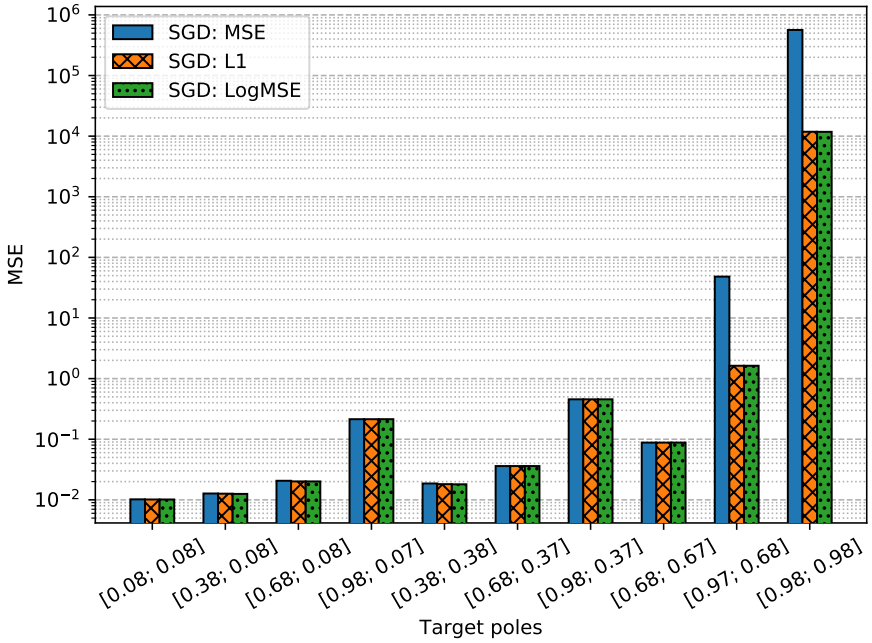


FIGURE B.5: MSE criterion on overdamped Neural Filter when using the loss function adaptation method for training.

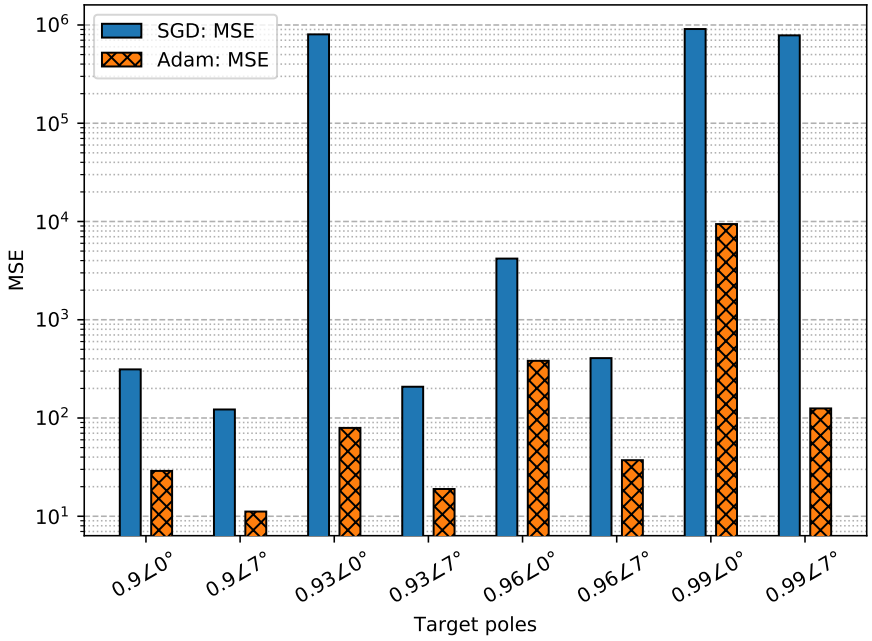


FIGURE B.6: MSE criterion improvement on Neural Filter when using the Adam algorithm for training on targets in partial explosion zone.

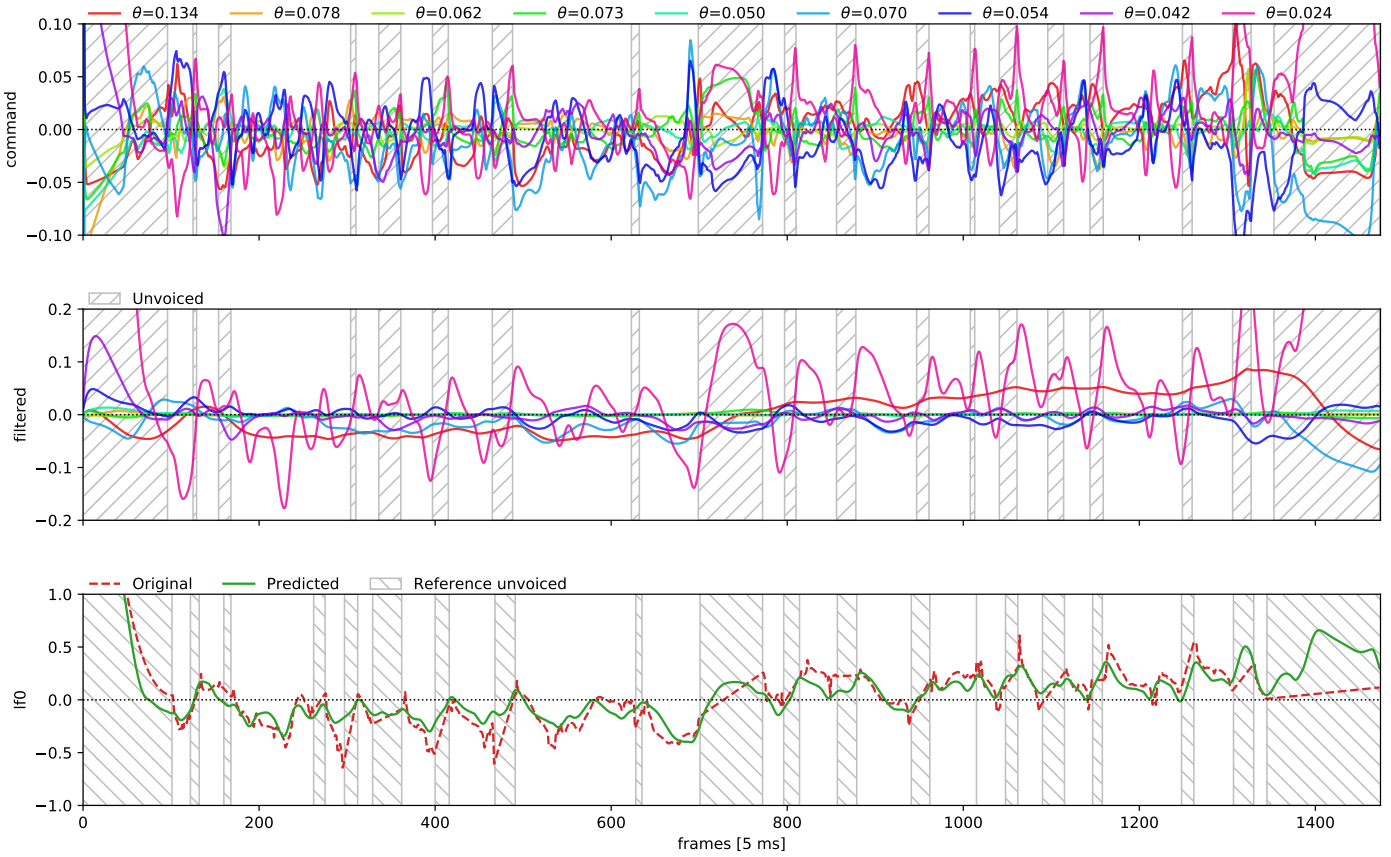


FIGURE B.7: End-to-end synthesizer internal signals [Roger 7701]. From top to bottom: 1. Command signals. 2. Muscle responses. 3. Reconstructed (solid) and original (dashed) intonation curves.

Bibliography

Yoshua Bengio, Patrice Simard, and Paolo Frasconi (1994). “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Alan W Black et al. (1999). *The festival speech synthesis system*.

Carlo Drioli and Davide Rocchesso (1998). “Learning pseudo-physical models for sound synthesis and transformation”. In: *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*. Vol. 2. IEEE, pp. 1085–1090.

Hiroya Fujisaki and Keikichi Hirose (1984). “Analysis of voice fundamental frequency contours for declarative sentences of Japanese”. In: *Journal of the Acoustical Society of Japan (E)* 5.4, pp. 233–242.

Branislav Gerazov and Philip N. Garner (2015). “An investigation of muscle models for physiologically based intonation modelling”. In: *Telecommunications Forum Telfor (TELFOR), 2015 23rd*. IEEE, pp. 468–471.

— (2016). “An agonist-antagonist pitch production model”. In: *International Conference on Speech and Computer*. Springer, pp. 84–91.

Branislav Gerazov, Pierre-Edouard Honnet, et al. (2015). “Weighted correlation based atom decomposition intonation modelling”. In: *Sixteenth Annual Conference of the International Speech Communication Association*.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins (1999). “Learning to forget: Continual prediction with LSTM”. In:

Hynek Hermansky and Nelson Morgan (1994). “RASTA processing of speech”. In: *IEEE transactions on speech and audio processing* 2.4, pp. 578–589.

Sepp Hochreiter and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.

Pierre-Edouard Honnet et al. (2018). “Intonation modelling using a muscle model and perceptually weighted matching pursuit”. In: *Speech Communication*.

MN Howell and TJ Gordon (2001). “Continuous action reinforcement learning automata and their application to adaptive digital filter design”. In: *Engineering Applications of Artificial Intelligence* 14.5, pp. 549–561.

Andrew J Hunt and Alan W Black (1996). “Unit selection in a concatenative speech synthesis system using a large speech database”. In: *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*. Vol. 1. IEEE, pp. 373–376.

Vasilis Karaiskos et al. (2008). “The blizzard challenge 2008”. In: *Proc. Blizzard Challenge Workshop, Brisbane, Australia*.

Diederik P Kingma and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *nature* 521.7553, p. 436.

Yann LeCun, D Touresky, et al. (1988). "A theoretical framework for back-propagation". In: *Proceedings of the 1988 connectionist models summer school*. CMU, Pittsburgh, Pa: Morgan Kaufmann, pp. 21–28.

Masanori Morise (2016). "D4C, a band-aperiodicity estimator for high-quality speech synthesis". In: *Speech Communication* 84, pp. 57–65.

Masanori Morise, Fumiya Yokomori, and Kenji Ozawa (2016). "WORLD: a vocoder-based high-quality speech synthesis system for real-time applications". In: *IEICE TRANSACTIONS on Information and Systems* 99.7, pp. 1877–1884.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio (2012). "Understanding the exploding gradient problem". In: *CoRR, abs/1211.5063*.

— (2013). "On the difficulty of training recurrent neural networks". In: *International Conference on Machine Learning*, pp. 1310–1318.

Adam Paszke et al. (2017). "Automatic differentiation in PyTorch". In: *NIPS-W*.

Barak A Pearlmutter (1995). "Gradient calculations for dynamic recurrent neural networks: A survey". In: *IEEE Transactions on Neural networks* 6.5, pp. 1212–1228.

Santitham Prom-On, Yi Xu, and Bundit Thipakorn (2009). "Modeling tone and intonation in Mandarin and English as a process of target approximation". In: *The Journal of the Acoustical Society of America* 125.1, pp. 405–424.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

Jan van Santen, Taniya Mishra, and Esther Klabbers (2008). "Prosodic processing". In: *Springer Handbook of Speech Processing*. Springer, pp. 471–488.

Bastian Schnell and Philip N. Garner (July 2018). *A Neural Model to Predict Parameters for a Generalized Command Response Model of Intonation*. Tech. rep. Idiap.

Jonathan Shen et al. (2017). "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions". In: *arXiv preprint arXiv:1712.05884*.

Robert Tibshirani (1996). "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288.

Keiichi Tokuda, Heiga Zen, and Alan W Black (2002). "An HMM-based speech synthesis system applied to English". In: *IEEE Speech Synthesis Workshop*, pp. 227–230.

Aurelio Uncini (2002). "Sound synthesis by flexible activation function recurrent neural networks". In: *Italian Workshop on Neural Nets*. Springer, pp. 168–177.

Aaron Van Den Oord et al. (2016). "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499*.

Paul J Werbos (1990). "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Zhizheng Wu, Oliver Watts, and Simon King (2016). "Merlin: An open source neural network speech synthesis system". In: *Proc. SSW, Sunnyvale, USA*.

Heiga Zen and Haşim Sak (2015). "Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis". In: *Acoustics, Speech*

and Signal Processing (ICASSP), 2015 IEEE International Conference on. IEEE, pp. 4470–4474.

Heiga Zen, Keiichi Tokuda, and Alan W Black (2009). "Statistical parametric speech synthesis". In: *Speech Communication* 51.11, pp. 1039–1064.