A NEURAL NETWORK PRINCIPAL COMPONENT SYNTHESIZER FOR EXPRESSIVE CONTROL OF MUSICAL SOUNDS

BY

SYLVAIN LE GROUX

Submitted to Ircam Pedagogy Department on September 2, 2002 in partial fulfillment of the requirements for the Degree of *Diplome d'Etudes Approfondies* in Acoustics, Signal Processing, and Computer Science Applied to Music. This dissertation is a summary of research conducted at the Center for New Music and Audio Technologies from March 29, to August 15, 2002.

Thesis Supervisor: David Wessel
Title: Professor of Music and affiliate Professor of Psychology. Director of CNMAT.

ABSTRACT

This dissertation introduces a connectionist model that maps perceptual controllers to synthesis parameters to allow for an intuitive and powerful musical control of audio synthesis. This model, or system, allows the extraction, abstraction, reproduction and transformation of relevant features of a musician's style. All the information is deduced exclusively from audio. No prior knowledge of the musician's performance is implied. We will first define the general underlying principles of the model. After, the neural network specific requirements for the learning and synthesis of sound information will be presented. With this foundation, the many advantages of using a principal component based synthesis will be argued. This dissertation concludes with an overview of some musical applications, including the re-tuning of a melody, pitch shifting, time stretching, cross-synthesis and compression.

Acknowledgements

I would like to thank my supervisor David Wessel for inviting me to CNMAT and for making this research project happen. His advices, enthusiasm and insights on the field of computer music have been a great source of inspiration for me. I like his barbecue chicken too.

I want to thank Richard Andrews, administrator of CNMAT, for helping me with the organization of my stay at Berkeley and for his warm welcome. He helped me transform this research project into concrete reality.

Thanks to my office neighbour Ali Momeni for some nice moments and discussions in Berkeley and San Francisco, for his help on macintosh computers, and for his theories about French people. Thanks to Brian Vogel for his kindness and for taking time to answer my questions about PhD students' life. Thanks to Edmund Campion for his nice old style bike that carried me (almost) everyday to the lab.

I want to express my gratitude to Gerard Assayag and Michelle Castellengo, coordinator and responsible of the DEA Atiam at IRCAM, who supported my project of going to CNMAT for my *stage de DEA*.

This project would not have been possible without the financial support of CNMAT and a "bourse Ile de France" grant from Paris 6 University.

1 INTRODUCTION

1.1 Reasons Behind this Project

Until recently, the transformation and control of recorded sounds was a difficult task requiring intensive studio work. Computer technologies, with their huge computational power, now allow for a more efficient transformation of complex sounds. Nevertheless, a flexible, general tool for the refined control of audio synthesis is still to be desired. Electronic instruments remain unable to provide the musician with a variety of controllers and controllable sounds. The often standardized nature of the available tools limits the musician's expressive control.

This limit is particularly problematic since musicians are appreciated and recognized for their style. During many long hours of training, a unique relationship develops between the musician and his/her acoustic instrument. The physical properties of the instrument combined with the musician's individual personality, skills, and creativity, result in a specific playing style. This interaction is the foundation for a musician's identity, essential for listener recognition and enjoyment.

The model introduced in this dissertation, with its flexible controllers and possibility for abstraction of salient features of recorded sounds, provides a musician the opportunity to cultivate his/her style. A tool for greater expression is particularly relevant now when sound samples are commonly used in every type of music.

1.2 Literature

Several studies have explored the control of audio synthesis with perceptual parameters. The first detailed description of a connectionist model for real time control of synthesis was articulated by M. Lee and D. Wessel in [1]. These authors presented the problem of transformation of performance gestures into control parameters. A general "forward model" was analyzed, and the importance of both the pre-processing of gestural data as well as the perceptually based representation of sounds was underscored. C. Drame and D.Wessel [2] applying the ideas presented in the initial study [1], compared

a neural network model with a memory based machine learning technique. The network described in this second study received fundamental frequency and loudness in its input layer, and was trained to estimate amplitudes and frequencies of the sound partials. Additional Research on the modelling of speech and music signal was presented in [3]. While the Radial Basis Function neural network employed in this third study was particularly adapted for the prediction of time series, the system described lacked variety in choice of controllers. Although many studies have used artificial neural networks to predict or simulate harmony and melody in musical pieces, their application to the control of sound synthesis remains underdeveloped**.**

Another technique presented in [4] -- the Cluster-Weighted Modelling for non-linear function approximation – mapped physical movement to spectral sound representation. This mapping technique, which itself proved an interesting alternative to neural networks, was applied in [5] in a MAX-MSP [6] real time implementation. Schoner and Jehan borrowed the general ideas presented in [2] for the perceptual control and estimation of partials, but replaced this study's neural network with the Cluster Weighted Model. This Cluster Weighted Model approach is equivalent to the hierarchical mixture of expert model proposed by Jordan and Jacobs in [27].

1.3 The Approach of this Dissertation

This dissertation employs a connectionist model derived from control theory [7]. Since Artificial Neural Network theory is studied extensively, many algorithms and practical optimisations have already been devised and are easily accessible. The aim of this dissertation is to improve the system proposed in [1] and [2] with the use of more refined training algorithms and the creation of an optimised system architecture specifically designed for the synthesis of sounds. The models previously presented continue to lack efficiency and reliability. Using both the statistical tools provided by principal component analysis and enhanced learning algorithms, this dissertation proposes a new, compact representation of spectral information for an efficient sound synthesis determined by perceptual controllers. The general principles of this dissertation's model are described in the following section.

2 PRINCIPLES

2.1 Architecture of the System

The structure of this dissertation's model is straightforward and inspired by the "natural" system formed by a musician and his/her instrument. The controllers used by the musician to interact with the system are located on one side. The sound synthesis engine, which plays the role of the instrument, is located on the opposite side. The artificial neural network's task is to transmit information and coordinate proper communication between these two elements (Fig. 1). Its function is similar to that of a music teacher showing his/her students how to produce a nice sound out of their instrument's body (synthesis engine), from a given finger position (controllers).

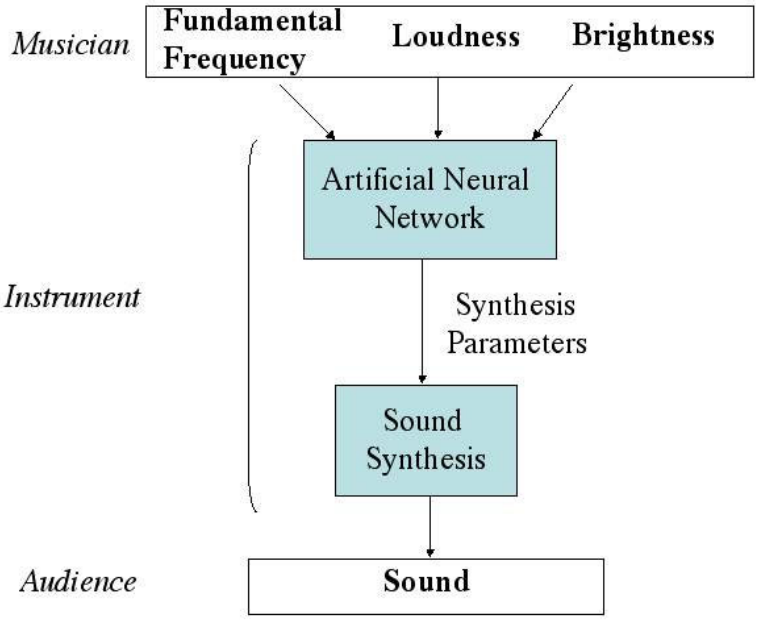


Figure1: Overview of the System

2.2 Smart Instrument

However, there is a limit (or extension depending on the point of view) to this music student/teacher analogy. Once the network is trained, the system has acquired a kind of intelligence. Any control information –provided it is in the range of the training data- will produce a reliable sound. This allows for experiments with any kind of controllers. This enhanced instrument adapts to the musician's control, it becoming possible to play the saxophone with flute control parameters and vice versa, for example. The relationship between the musician and his/her instrument is fundamentally changed. While the musician traditionally has had to adapt to the physical properties of his/her instrument, the system described in this dissertation gives the musician extended freedom of choice in gestural control of the instrument.

2.3 Modularity

Another important characteristic differentiating this system from acoustic instruments is that the controllers and the synthesis engine are two distinct part of the system. The synthesis engine, consisting of the neural network and the sound synthesis algorithms, may be physically separated from the control information. This distinctness makes this kind of synthesis suited to client server applications. The musician, via the controllers, plays the role of the client, and the synthesis engine that of the server. To give an example, a musician could remotely control this new instrument via Ethernet protocol.

2.4 Sequence of Operations

Another important aspect of our model, in addition to its specific structure, is the timing of the different processes taking place. Since the artificial neural network must first be trained before any attempt to use the system, it is necessary to carefully describe the sequence of the different processes (Fig. 2). Each step of the setup required for the system to give proper results is described below.

2.4.1 Choice of the Training Data Set

First, the audio samples are chosen to fit the needs of the musician. These samples should be as varied as possible and include the kinds of sonorities with which the musician wishes to play. To give an example, an accurate simulation of all the acoustical properties of an instrument would require a large number of training samples. These samples would be able to describe the main possibilities of interpretation, transition and playing modes of the specific instrument. The choice of the training data set is fundamental, since it will determine the overall results, the "colour" of the sound synthesis and finally, the musical result.

2.4.2 Analysis

Once the choice of the training data is done, the audio samples are passed through an additive harmonic analysis tool. This step allows modelling the sound with a set of parameters that will later be modified depending on the control information. We used *AddAn*, a sound analysis application distributed by IRCAM (a French institution for research and education in the fields of acoustics and computer science) as a part of Diphone Studio, an audio morphing and synthesis editor [8].

2.4.3 Extraction of the Control Information

The results from this analysis are used for the extraction and definition of the perceptual controllers (such as pitch, loudness, brightness…). These controllers are the only connection between the musician and his/her instrument and should be as convenient as possible from a musical point of view.

2.4.4 Training of the Neural Network

Finally, the third step involves the training of the neural network. The perceptual controllers information is sent to the input of the Artificial Neural Network (Fig. 2). The network is then trained to map these controls to the set of the output parameters needed for driving the sound synthesis algorithm. The quality of the sound produced by the system is highly dependant on the training performances. Once the training is considered satisfactory, the Neural Network Synthesizer is ready for use. The

parameters needed for the sound synthesis are estimated by the system, and the musician can play with it. Now that the procedure is explicit, the different parts of the system will be accurately defined.

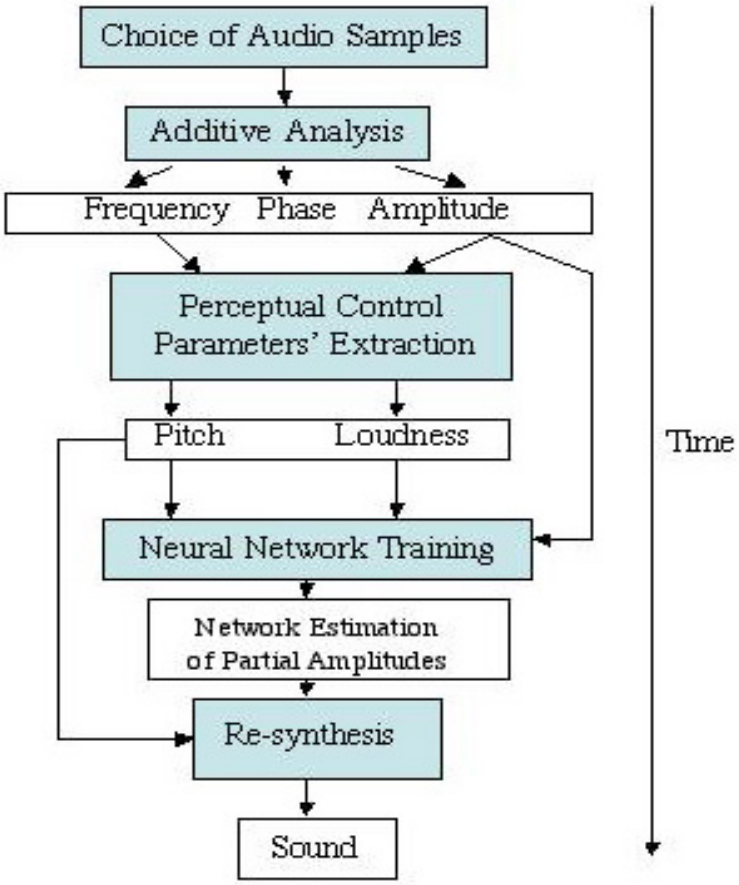


Figure 2: Sequence of Operations

3 ANALYSIS/SYNTHESIS PARADIGM

3.1 Overview

Many different sound synthesis techniques have already been devised. Each of them falls into four main categories [9].

3.1.1 Abstract Models

The first group of synthesis methods is called abstract algorithm models. They are usually simple and easy to implement. Nevertheless, they often sound artificial if compared to other more complex synthesis techniques. Abstract models includes synthesis schemes such as FM synthesis, particularly efficient for the synthesis of bell-like and metallic sounds, wave-shaping (or non-linear distortion) synthesis, and Karplus strong algorithm typically used for the synthesis of plucked string or percussive sounds.

3.1.2 Processed recording

The second group is based on processed recordings. It includes simple sampling synthesis, consisting in the play-back of short recording of sounds, wave-table synthesis, which principle is to store typical portions of an instrument sound in tables to be able to loop them, play them back, and shift the pitch. Finally, granular synthesis' idea is to represent the sound by elementary units or grains, which shape and temporal distribution change the synthesised sound.

3.1.3 Spectral Models

The spectral methods attempt to model the properties of sound accordingly to the perception of the listener. These synthesis techniques are more general, since any sound can be treated. Techniques based on the spectral model include additive synthesis, that models the signal by a sum of weighted sinusoids, the phase vocoder (that can be viewed either as a bank of filters or as a short term Fourier transform analyzer), source-filter synthesis, Mc Aulay-Quatieri algorithm and Spectral Modeling Synthesis

(SMS), that decompose any signal into a deterministic part and a stochastic part.

3.1.4 Physical Models

Finally, physical models continue to develop as they allow a better control over the synthetic instrument parameters. This final group can itself be divided into five categories [9], namely a numerical solving of partial differential equations, source filter modelling, vibrating mass-spring network, modal synthesis and waveguide synthesis.

3.2 The approach of this Dissertation

The neural network synthesizer described in this dissertation uses an additive analysis/synthesis model. This model is chosen for its ability to synthesize a large range of sounds while sounding quite natural (which unfortunately is sometimes not the case for abstract models). Unlike physical models, the additive model is based on the original recorded sounds. It doesn't require having a theoretical representation of the physical property for each different instrument. Since the concept of additive synthesis is quite old and has been often used in electronic music, theoretical and practical tools for additive analysis are reliable and well documented. The concept of additive synthesis is quite simple; nevertheless, it can require many parameters to get a satisfactory sounding result. Moreover, these parameters are not always adapted to a musical control of the synthesis. By introducing both a principal component analysis for the reduction of parameters, and a neural network for a flexible control of sound, this dissertation proposes a new and flexible scheme for musical control of sound.

3.3 Harmonic Additive Paradigm

3.3.1 Analysis

In this model, the sound is represented as a sum of weighted sinusoids at frequencies multiple of the fundamental frequency. This model presupposes the sound is quasi-harmonic, and the frequencies, amplitudes and phases are varying slowly. This is an important limitation, since most of the sounds encountered in nature contain inharmonicities, and since fast varying spectrum are common (in the attack part of an instrument for instance). Nevertheless, the results obtained with this type of analysis still prove satisfactory in a first approximation for the purpose of this paper.
A quasi-periodic tone can be generated by a sum of sine waves with time-varying amplitudes and frequencies.

$$y(n) = \sum_{k=0}^{N-1} a_k(n) \sin(2\pi f_k(n) + \phi_k) \qquad (1)$$

n is the time index
N is the number of harmonics in the synthetic signal
$a_k(n)$ is the time-varying amplitude of the k-th partial
$f_k(n)$ is the time-varying frequency of the k-th partial
$f_k = k.f0$ (harmonicity hypothesis)
ϕ_k is the corresponding phase
$a_k(t)$ *and* $f_k(t)$ are slowly time-varying

The information obtained from AddAn analysis software includes the time varying frequency of the fundamental (which gives the variation of the harmonics), the phase values, and the amplitude of each partial.

The results of Addan analysis in the ascii format are represented as follows:

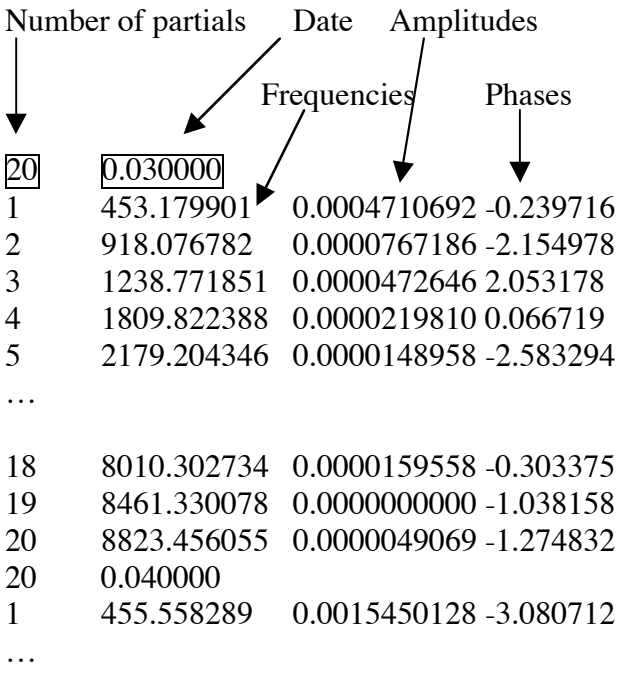


Fig 3: Addan Partial Information

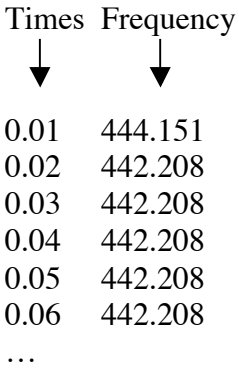


Fig 4: Addan Fundamental Analysis

In this dissertation, the result of the harmonic additive analysis for the partial amplitude is represented as a matrix A (eq. 2), of dimension *number of partial* by *number of frames*. The analysis parameters used for most of the sounds are: a FFT window size of 1024 with a Blackmann windows, a

fundamental frequency range of [200,1400Hz], a number of partial set to 20, and a sample rate of 44100Hz. Yet, these numbers are just an indication. They can be, and should be adjusted in regard to the nature of the sound to be analysed.

One row of the matrix represents the time-varying amplitude of a partial, and one column of the matrix represents the whole spectrum of the audio signal for one frame (typically, one frame is 0.01 or 0.02 s).

$$\begin{bmatrix} a_{1,1} & \cdot & a_{1,M} \\ \cdot & \cdot & \cdot \\ a_{N,1} & \cdot & a_{N,M} \end{bmatrix} \quad (2)$$

N is the number of partial
M is the number of frames

3.3.2 Synthesis

For the re-synthesis, the phase information, which is less perceptually relevant, is not taken into account, and the sample y(t) are reconstituted at the proper sample rate Fs by linear interpolation between each frame of the analysis matrix A to give the coefficients a_k(n). The procedure is similar for obtaining the coefficients f_k(n).
The analysis hop size, gives the number of samples represented by each column of the matrix:

$$hop = round(F_s \cdot frame\_duration) \quad (3)$$

Fs is the sample rate in Hertz
Frame _duration is in seconds

The interpolation scheme allows the calculation of the time varying coefficients between each frame:

$$a_k^j[n] = \frac{(a_{k,j+1} - a_{k,j})}{hop} \cdot n + a_{k,j} \quad (4)$$

where $a_k^j[n]$ is the time varying amplitude of the partial k between the frame j and j+1.
$a_{k,j}$ is the element of matrix A (Eq. 2) corresponding to the partial k and to the frame j

The same procedure is repeated for the interpolation of the fundamental frequency values between each frame, and the synthesised sound is calculated with Eq.1. The analysis information (time-varying fundamental, and partial amplitudes) provides a reliable abstract representation of the original audio samples. This information is used for the definition of control parameters.

3.4 High-level descriptors for Synthesis control

The control parameters are extracted from the analysis data and defined thereafter. They have been chosen in order to correspond to human perceptive criterions and are relevant for musical applications.

3.4.1 The Pitch

The pitch or time-varying fundamental frequency F0 is directly given by AddAnn. It is strongly recommended to carefully check the validity of the fundamental frequency estimation, since it is the primary cause of inaccuracy in the re-synthesis. It has been observed that an error of one octave often occurs in the detection of the fundamental frequency by Addan algorithms (Fig. 5 and Fig. 6).

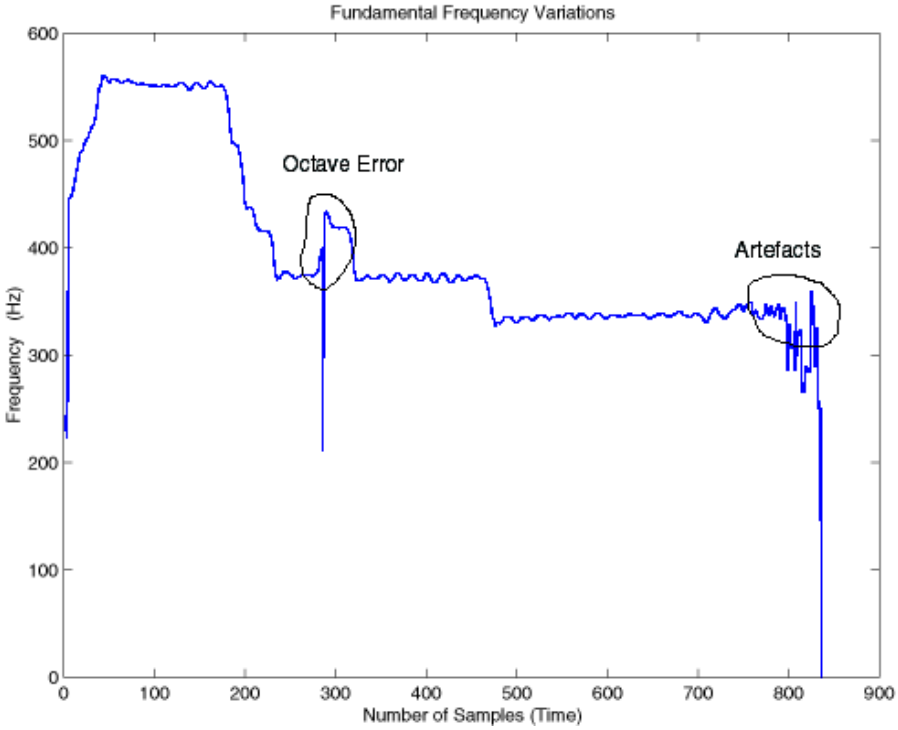


Fig 5: Original Fundamental Detection of a Singing Voice.

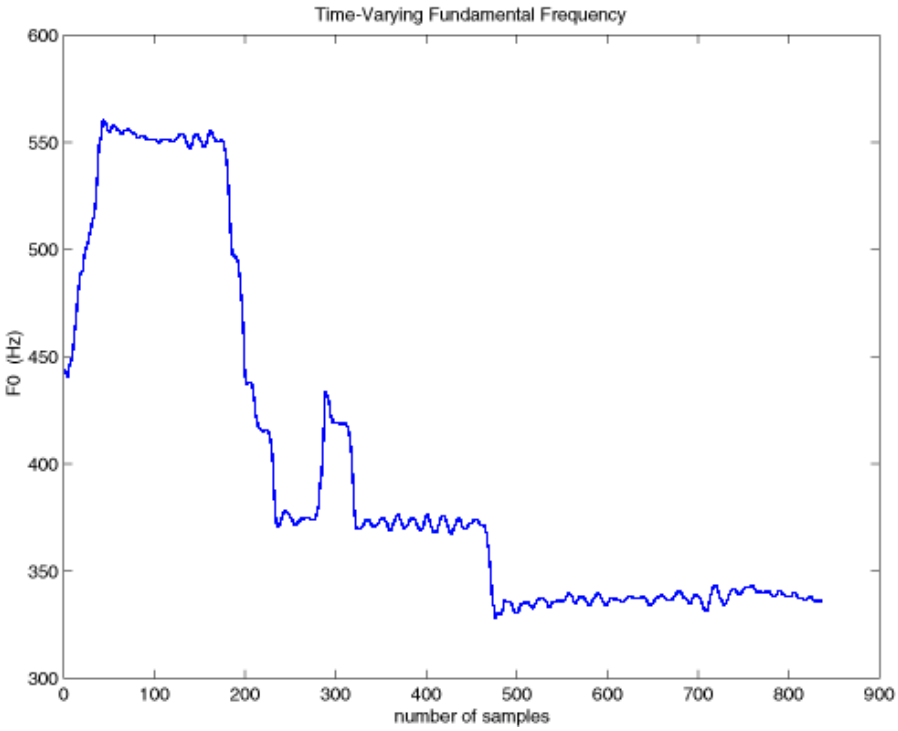


Fig 6: Corrected Fundamental Frequency

3.4.2 The Intensity Summary

The intensity summary $I_0(t)$ is given by:

$$I_0(t) = \sum_{k=1}^{N} a_k(t) \tag{5}$$

It gives a description of the intensity level of a sound.

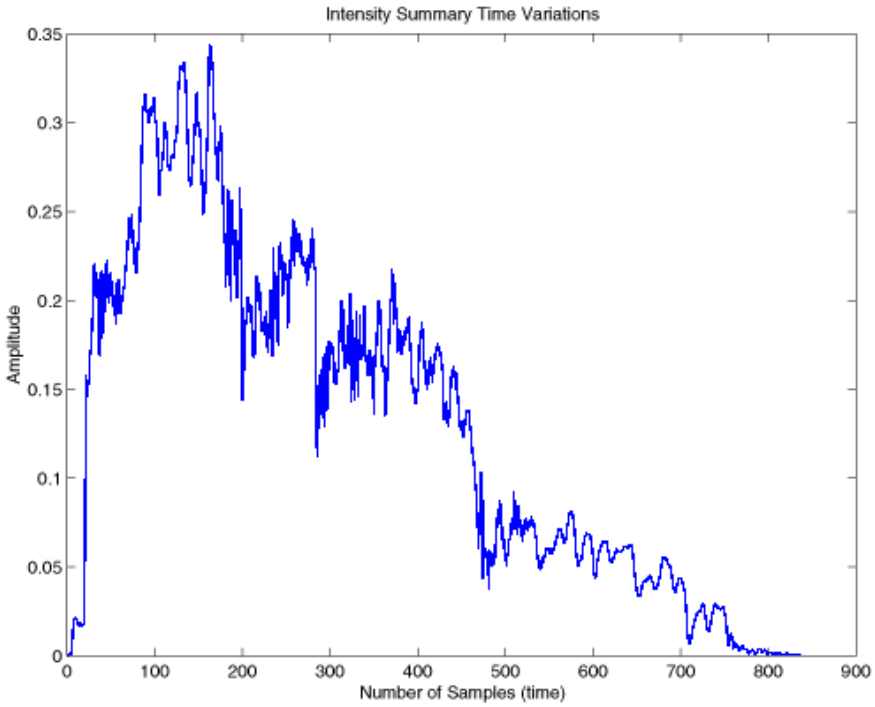


Fig 7: Intensity Summary Associated with the Fundamental Frequency of Fig. 6.

3.4.3 The Centroid

The Centroid C(t) [10] is calculated with:

$$C(t) = \frac{\sum_{k=1}^{N} a_k \cdot f_k}{\sum_{k=1}^{N} a_k}(t) \tag{6}$$

The centroid is often associated with the measure of the brightness of a sound. It is an estimation of the centre of gravity of a given spectrum.

Other parameters such as perceptual loudness, spectral flatness, noisiness or any other control inferred from the analysis data can be added according to the needs of the musician. In the next step, these control parameters are used to feed the input of the artificial neural network.

4 AN ARTIFICIAL NEURAL NETWORK ASSISANT FOR THE CONTROL OF SYNTHESIS

4.1 Overview

4.1.1 A Connectionist Approach

Artificial neural networks are an attempt to model the behaviour of biological neurons. They are constituted of a multitude of simple interconnected processing units. This kind of model is often referred to as connectionist or parallel distributed processing. The parallelism resides in the fact that the processing units interact simultaneously and are independent of each other. The model is distributed, since the knowledge resides in the strength of the connections that are adaptative. When exposed to a set of data, the network is capable of learning by adjusting the strength (the weights) of its connections.

4.1.2 Static/Dynamic

Two main different kinds of Artificial Neural Networks can be found. The first one is the static network, for which equations are memoryless. The output is a function of the current input only. One classical example of this category is the Multilayer Perceptron trained by the back-propagation error algorithm. The second kind, namely Dynamic Neural Networks, is making use of a memory, described by difference equations. One example of this kind is the Hopfield Network.

4.1.3 Supervised/unsupervised learning

Two different learning modes are possible. Supervised learning produces a desired output from a given input. It is thus supposed that the output is known in advance (C.f. the Perceptron). Unsupervised learning is more a way of extracting knowledge from the input data. The system is presented only with input data and is supposed to self organize and find the corresponding relevant output (For instance Kohonen networks).

The general advantages of neural networks are their learning capability, their generalization capacity, their content addressability and their noise tolerance.
On the other hand, Artificial Neural Networks usually offer poor explanation capabilities, they present difficulties with structured representations (concept, hierarchy, inferences…), and they are monolithic, and often frustrate analysis of an underlying mechanism.

4.2 The Approach of this Dissertation

For the purpose of this dissertation, we chose to use a Static Feed Forward Multilayer Neural Network with Supervised Learning. Such a network can be viewed as an implementation of a non-linear input-output mapping. Previous studies in [2] and [3] have demonstrated the efficiency of a static neural network, that doesn't require intense usage of computer memory. Efficiency is an important criterion for this study, the final purpose of which is to provide a reliable synthesis model for real-time interaction. The problem of memory requirements is even amplified by the fact that the implementation of the concepts presented in this dissertation has been realised in the Matlab programming language in which memory management is a sensitive problem… The learning of the neural network is done in a supervised fashion, since the principle of this dissertation's model is to reproduce a given target sound. This connectionist system is based on fairly simple algorithms, and the non linearity is learned from the training data set. We thereafter explain the principles and specificities of this complex system.

4.3 Artificial Neural Networks Fundamentals

4.3.1 Artificial Neuron

The artificial neuron is an attempt to model the behaviour of biological neurons that constitute the fundamental units of the nervous system. Biological neurons are organised and cooperate to accomplish complex tasks. The neuron's topology is composed of three main functional parts, namely the dendrites, the synapses, and the axon (Fig. 3 and 4). The dendrites are the incoming neuronal fibers that receive electrical signals from the other connected neurons. The synapses in which the electrical

transmission takes place can be either excitatory or inhibitory by respectively reducing or increasing the amplitude of the transmitted signal. The axon is the outgoing neuronal fiber. If the sum of the signals transmitted by the inhibitory and excitatory synapses is higher than a certain threshold, an action potential is fired through the axon towards other neurons, and there is a time delay called refractory period the neuron has to wait before it can fire again.

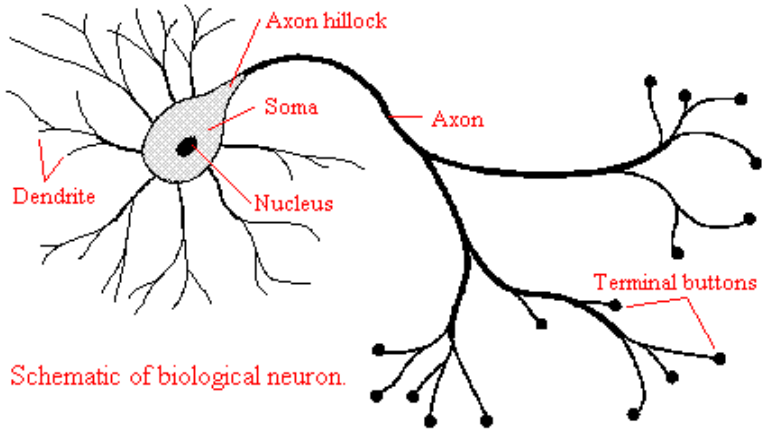


Figure 3: Biological neuron

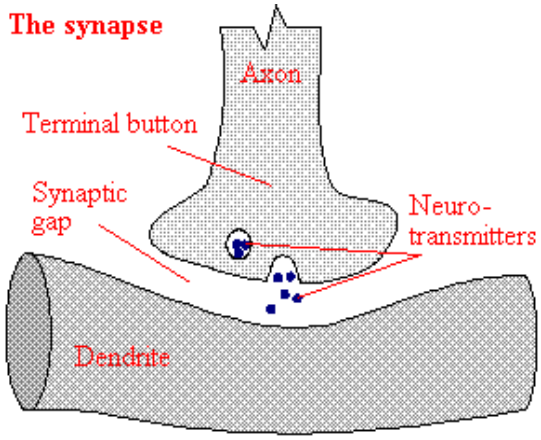


Figure 4: Synaptic connection

The artificial neuron is a mathematic representation of the schematised comportment of a biological neuron we described previously (Fig. 5).
Let **x(t)** be the input signal column vector of dimension n by 1, at time t. We suppose that the input is binary, i.e: $\mathbf{x}(t) \in \{0,1\}$. The inhibitory or excitatory effect of the synapses is modelled by a weighting vector **w** of dimension n by 1, applied on the inputs. Finally, a step function f(y) -or activation function- is used to set the output to zero or one at time t+1, depending if the sum of the input signals is higher than the neuron firing threshold.

$$f(y) = \begin{cases} 1 & if \ \ y \geq threshold \\ 0 & if \ \ y < threshold \end{cases} \tag{7}$$

The output of the artificial neuron is expressed by:

$$Out(t+1) = f\left(\sum_{i=1}^{n} w_i x_i(t)\right) = f(\mathbf{w}^t . \mathbf{x}(t)) \tag{8}$$

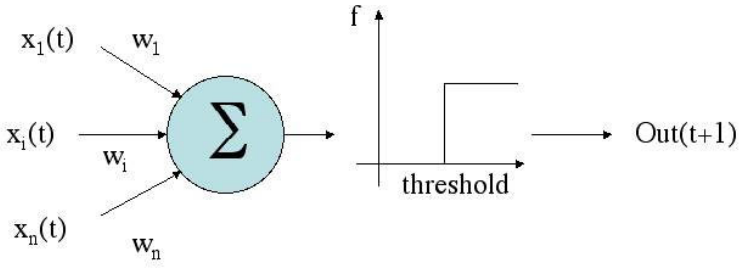


Fig 5: Artificial Neuron

If f is now centered around zero, equation (8) is equivalent to:

$$Out(t+1) = f(\sum_{i=1}^{n}(w_i x_i(t) - threshold)) = f(\sum_{i=0}^{n} w_i x_i(t)) = f(\mathbf{w}^t.\mathbf{x}(t)) \qquad (9)$$

The weight w_0 (or bias) is equal to the threshold's value and the imaginary input x_0 is set to the value 1.

One should note there still are many properties of the biological neuron this model doesn't take into account. For instance, biological neurons have a continuous response instead of a binary one, they use a non-linear weighting of the inputs, the refractory period can be different for each neuron, and the refractory/excitatory characteristics of a synapse is not always predictable. Yet, the artificial network model is still an efficient computational tool that allows basic Boolean operation including NOT, AND, OR.

4.3.2 Perceptron

One of the simplest kind of neural network architecture is called the perceptron. It allows only unidirectional feed forward connections between neurons. The network is constituted by one layer of p neural units connected to n inputs. In this architecture, the number p of outputs is the same as the number of neural units.

The output of the perceptron is given by:

$$out_i = f(\sum_{k=0}^{n} w_{ik}.x_k) \quad i = 1,\text{K}\ ,p \qquad (10)$$

Which in matrix notation gives:

$$\mathbf{out} = \mathbf{f}(\mathbf{w}^t.\mathbf{x}) \qquad (11)$$

Where f is a vector function such as the function fi is applied at the ith component of the vector and **w** is the weight matrix.

with the conditions mentioned in Equation (9).

where $\sum_{k=0}^{n} w_{ik} . x_k$ is the input of the unit i, and w_{ik} is the weight connecting the output unit i to the n inputs.

4.3.3 Multi-layer feed-forward Neural network

The perceptron architecture, although fundamental, can only solve a limited class of problem. It is unable, for example, to emulate the Boolean function XOR. More generally, a perceptron has been proved unable to solve non-linearly separable problems in [11]. This is the reason of the introduction of multi-layer feed forward networks (Fig. 6) in which the role of the first layers is to pre-process the inputs. Let suppose that the network has L layers. Each layer l has n(l) neuronal units connected to the layer l+1.

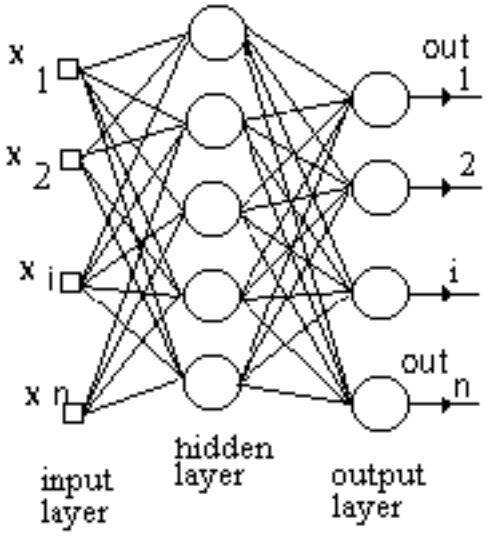


Figure 6: Feed Forward Neural Network with n inputs and n outputs.

The value at the output of each unit i in layer l is given by:

$$out_i = f_i\left(\sum_{k=0}^{n(l-1)} w_{ik}.out_k\right) \quad with \quad i = 1,\ldots,n(l) \tag{12}$$

The outputs of layer l-1 are the inputs of layer l and fi is the activation function of unit i.

4.4 Neural Network Architecture for Sound Synthesis

The neural network's architecture is determined by the nature and the size of its inputs and outputs, as well as by the specific tasks it has to handle.

4.4.1 Static Feed-Forward Neural Network

This paper uses a neural network for function approximation. Following previous research suggestions to look more closely at the use of static feed forward neural network with back-propagation error [1], [2], this model attempts to improve the network's architecture to make it more efficient and particularly suited for the control of sound synthesis. The number of controllers has to be the number of inputs in the network. In most of this dissertation's examples, the neural network will have two inputs corresponding to the fundamental frequency and the intensity summary respectively. Since two-layer networks have been proven of being able to approximate arbitrarily well any functional mapping [12], with a sufficient number of hidden units, this model is a two-layers network.

4.4.2 Activation Function

The activation function for the hidden-units is a "tanh" function (Fig.7), which in practice gives a faster convergence than the logistic function [13]. A linear activation function has been chosen for the output layer, thus making the range of the output limitless.

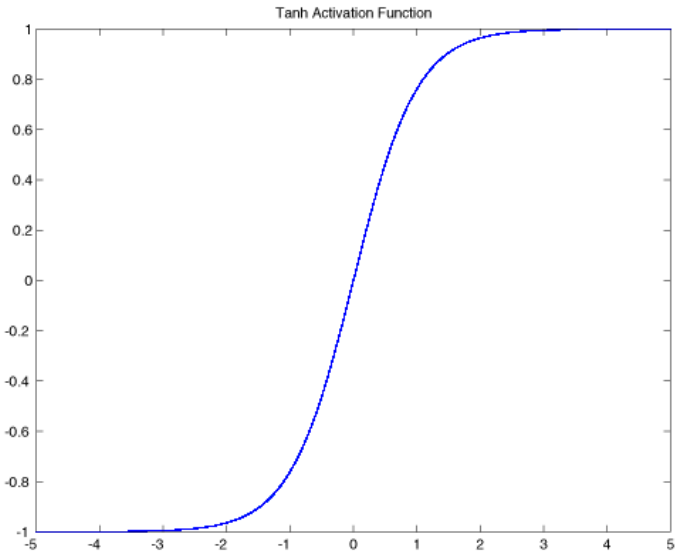


Fig 7: Activation Function

4.4.3 Units

The number of output units is variable and defined by the number of partials needed to perform the re-synthesis. (Experience has shown that we need at least twenty partials for a realistic additive re-synthesis).
The number of hidden units governs the complexity of the network. For determining the number of initial hidden units in our system, a practical rule of thumb proposed in [14] is used: the total number of weights in the network should approximately be the number of training points divided by ten.

4.5 Data Preprocessing

4.5.1 Normalization

In order to take into account the relative importance of the control parameters expressed in different units, a normalization of the data is necessary (Fig. 8 and 9). The mean and variance are calculated for the input vectors.

This yields the normalized input variable:

$$x_{norm}(n) = \frac{x(n) - \frac{1}{M}\sum_{i=1}^{M} x(n_i)}{\left(\frac{1}{M-1}\sum_{i=1}^{M}\left(x(n_i) - \frac{1}{M}\sum_{i=1}^{M} x(n_i)\right)\right)^{1/2}} \qquad (13)$$

M is the length of the input vector or number of frames
n is the index of the current input.

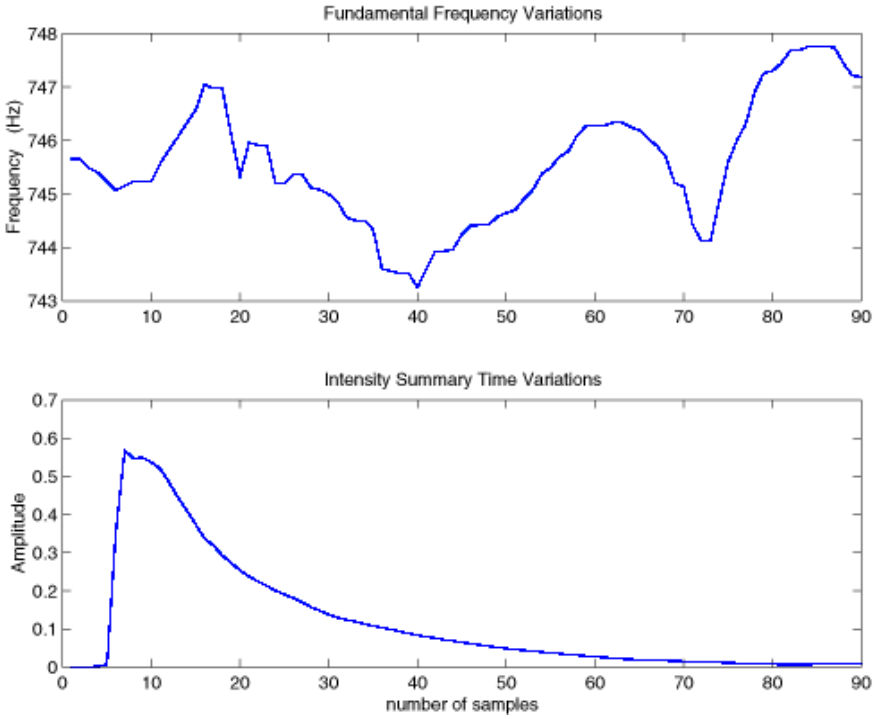


Fig. 8: Original Inputs

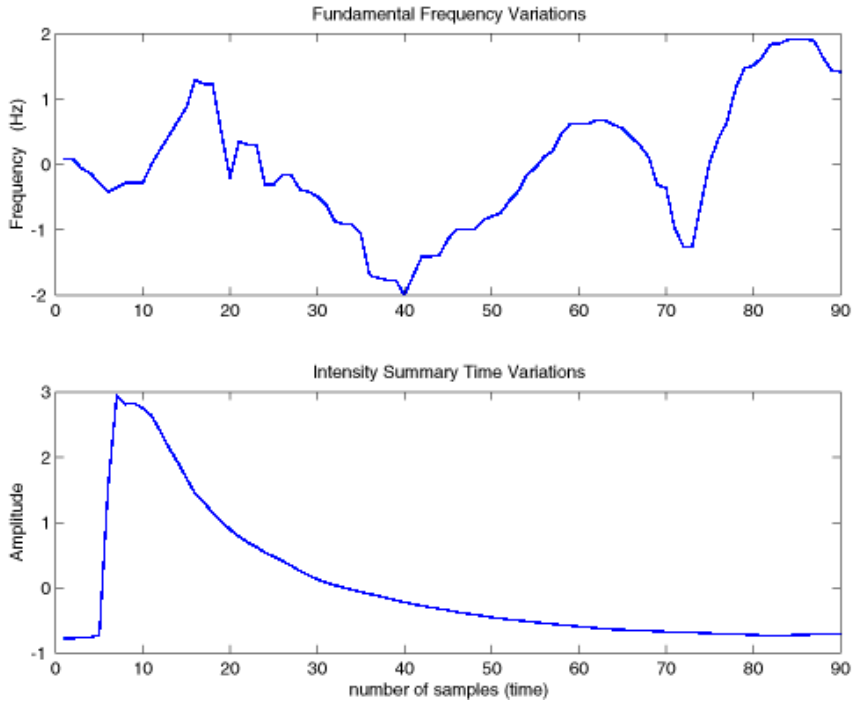


Fig 9: Normalised Inputs

4.5.2 Principal Component Preprocessing

A Principal Component Analysis [15] is then used to de-correlate the elements of the input vectors. This technique orthogonalizes the components of the input vectors and it orders the resulting orthogonal components so that those with the largest variation come first (Fig. 10). This technique can potentially reduce the size of the input vector by removal of data that is not statistically significant, but this reduction property is not used on the inputs. Reducing the dimension of the input vectors by too large a margin could actually become an obstacle for the function approximation. Information removed by the Principal Component Analysis technique, looking apparently irrelevant to the representation of the data itself can actually be of crucial importance for the regression process [11]. A safe approach is then to keep the maximum of information in the input data. The principal aim of using the principal component analysis in this step is to structure (de-correlate and order) the input data set, which helps the network in its regression task [11]. Nevertheless, the Principal Component Analysis ability to reduce the dimensionality of a space, if not recommended for the input space, is successfully used on the network's output space. This Principal

Component analysis of the output space actually contributes to a major improvement in the efficiency of the model, and is discussed in Chapter 5 to greater details.

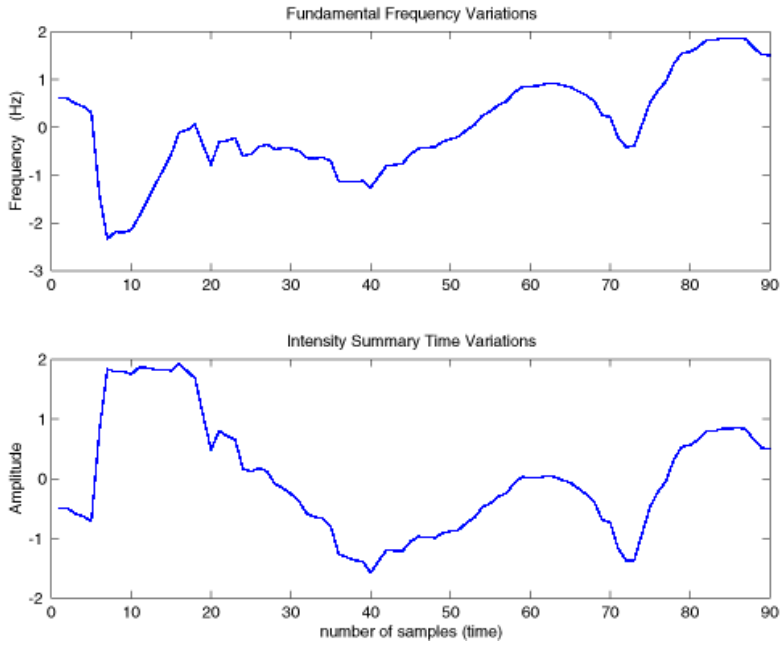


Fig 10: Normalisation and PCA

4.6 Training

The training of the network has been a crucial point in our project. The neural network is constituted of a set of neurons, or units, which weighted combination is trained to approximate time-varying partials.

4.6.1 Background

The standard learning algorithm for multi layers perceptrons is called the error back propagation algorithm. The principle is to compute the partial derivatives of the error function in the hidden units with respect to the output values. When the error and activation functions are differentiable, a gradient descent technique can be applied to find the weight matrix that minimise the error. The back propagation algorithm can be divided in two steps. First, the

input pattern $\mathbf{x}^q$, q={1,…,m} taken from a set of m training examples, is propagated through the network to the output. Then the error vector is back propagated from the output to the input of the network. The principle is to update the weight matrix at each step *u* in a way that minimises the error function $\varepsilon(W)$.

The sum of square error function is given by:

$$\varepsilon(\mathbf{W}) = \frac{1}{2}\sum_{q=1}^{m}\sum_{i=1}^{p}(out_i^q - t_i^q)^2 = \sum_{q=1}^{m}\varepsilon^q(\mathbf{W}) \qquad (13)$$

out is the output of the neural network
t is the target
m is the number of examples in the training set
p is the number of output units in the network

If the output error for the pattern is different from 0, the weight matrix is updated. The update is applied to every weight connecting unit k in layer l-1 to unit I in layer l.

$$\Delta w_{ik}^q = -\eta\frac{\partial\varepsilon^q}{\partial w_{ik}} \qquad (14)$$

We have:

$$\frac{\partial\varepsilon^q}{\partial w_{ik}} = \frac{\partial\varepsilon^q}{\partial o_i^q}\cdot\frac{\partial o_i^q}{\partial w_{ik}} \qquad (15)$$

Knowing that [17]:

$$\frac{\partial o_i^q}{\partial w_{ik}} = o_k^q \qquad (16)$$

And using

$$\delta_i^q = \frac{\partial\varepsilon^q}{\partial o_i^q} \qquad (17)$$

We obtain

$$\Delta w_{ik}^{q} = -\eta \delta_i^q o_k^q \tag{18}$$

The problem is now reduced to the calculation of δ_i^k. For the output units, i.e if the unit I is in the layer L, we obtain [18]:

$$\delta_i^q = \frac{\partial \varepsilon^q}{\partial o_i^q} = f^{'}(\sum_{k=0}^{n(l-1)} w_{ik}.out_k)(out_i^q - t_i^q) \tag{19}$$

for hidden units, the expression of δ_i^k. Is [18]:

$$\delta_i^q = f^{'}(\sum_{k=0}^{n(l-1)} w_{ik}.out_k) \sum_{j=1}^{n(l+1)} w_{ij}\delta_j^q \tag{20}$$

The principle of the backpropagation algorithm now appears more clearly. Once the δ_i^k is calculated for the output layer, the δ_i^k of layer l can be calculated with the value of δ_i^k in the layer l+1. The error is propagated from the output layer L to the input terminals.

4.6.2 Back-propagation with momentum

4.6.2.1 Theory

We first tried the standard back-propagation with the heuristic momentum algorithm, which was not very requiring but slow, and led to a certain lack of smoothness in the re-synthesized sound. The principle is to update the weight matrix at each step *u* in a way that minimises the error function ε(W). The gradient descent with momentum algorithm updates the network weights, taking into account the local gradient and the last weight change.

The formula is:

$$\Delta w_{ik}(u) = -\eta \frac{\partial \varepsilon(u)}{\partial w_{ik}} + \alpha \cdot \Delta w_{ik}(u-1) \qquad (21)$$

The w_{ik} are the element of the weight matrix.
η is the learning rate.
And $\frac{\partial \varepsilon(u)}{\partial w_{ik}} = Z^T \cdot \varepsilon$ is the gradient with Z being the Jacobian matrix.
u indicates the current cycle of the procedure over the whole training set.
α is the momentum term and is a means to avoid too fast changes in the weight updating, by keeping track of the step u-1. It avoids the network to stay stuck in a local minimum.

4.6.2.2 Results

In our application, the training is done in batch mode, where the weights and biases are updated after all the inputs have been presented to the neural network. The back-propagation with momentum method gives acceptable results, and doesn't need intensive computation, but it requires long training, and post-processing. The post-processing is necessary to eliminate the artefacts systematically heard in the end and in the beginning of the partial estimations (Fig. 11). We defined empirically that the points to be post-processed are the points where the partial value is smaller than the threshold given by the formula: *(mean(intensity summary))/factor*. Factor is an adjustable variable. The corresponding artefact values are then replaced by the value of the intensity summary at the same time index. (Fig.12: the 0 are replaced by X)

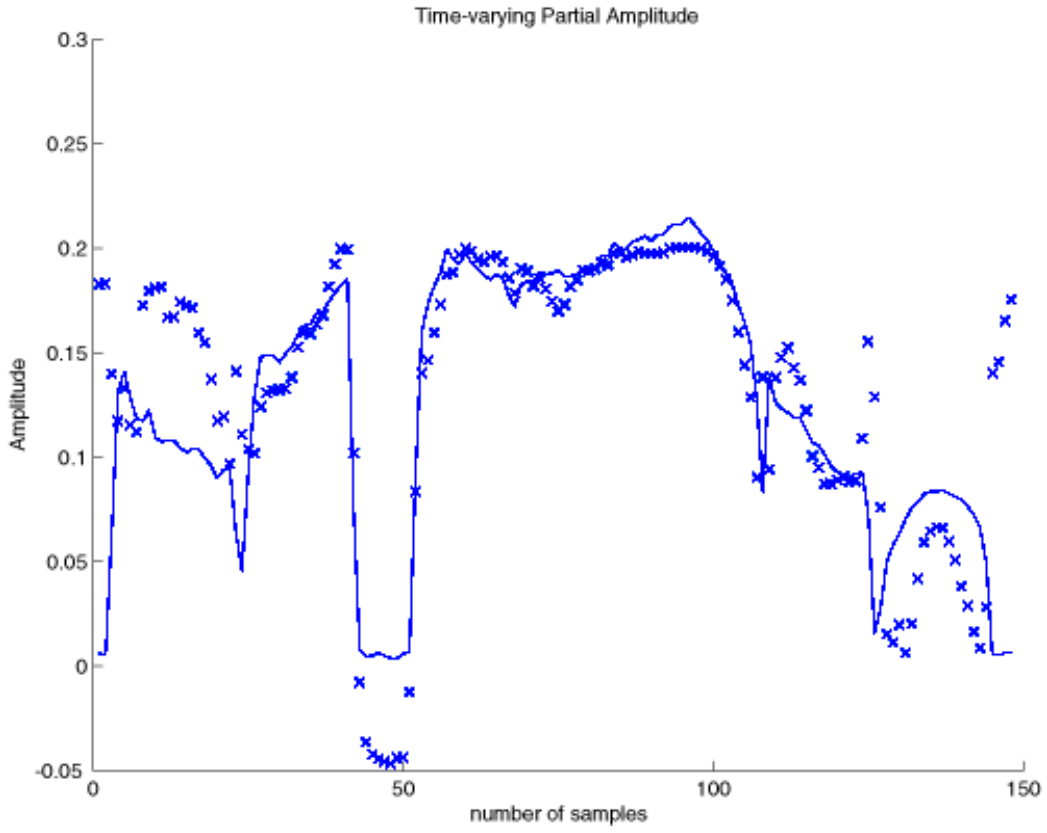


Fig 11: Neural Network Estimation (x) with Back-Propagation with Momentum and Original (-). First Partial of a Saxophone Sequence.

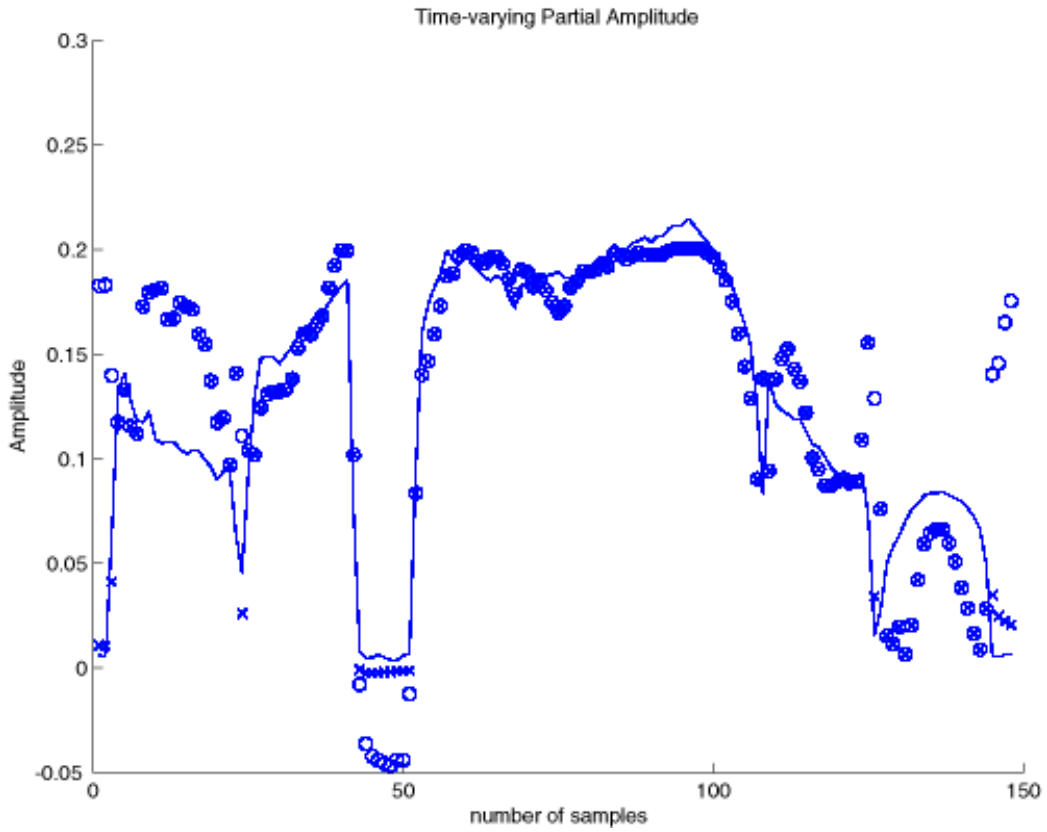


Fig 12: Network Estimation (o) with Smoothing Function(+) compared to the original (-)

4.6.3 Levenberg-Marquardt Algorithm

4.6.3.1 Theory

While some of the recurrent artefacts that appear in the end and in the beginning of the target partials (Fig. 11) can be successfully removed by post-processing (Fig. 12), an enhanced learning algorithm is proposed for even better results without any extra processing. This algorithm, namely the Levenberg-Marquadt optimisation algorithm, is specifically designed for minimizing the sum of square error [13]. It is faster than the gradient descent with momentum algorithm, but requires more computational power. The Levenberg-Marquardt algorithm is designed to approach second-order training speed without having to compute the Hessian matrix (second order derivative). The Hessian matrix is approximated by $H = Z^T \cdot Z$, where Z is the Jacobian (first derivatives of the network errors).

The formula for updating the weights becomes:

$$\Delta w_{ik}(u) = -(Z^T \cdot Z + \lambda I)^{-1} Z^T \varepsilon(u-1) \qquad (22)$$

where λ is the step size
ε is the error for the vector of samples.

4.6.3.2 Results

The speed of the convergence is considerably augmented with this algorithm. The response of the network also appears a closer fit with the partial amplitudes, yielding a better sounding result (Fig.13 and 14). In addition, the reproduction of a satisfactory sound doesn't require any filtering. While the response is reliable when the network is fed with the original inputs, the network's ability to generalise with new input data is discussed in the next section.

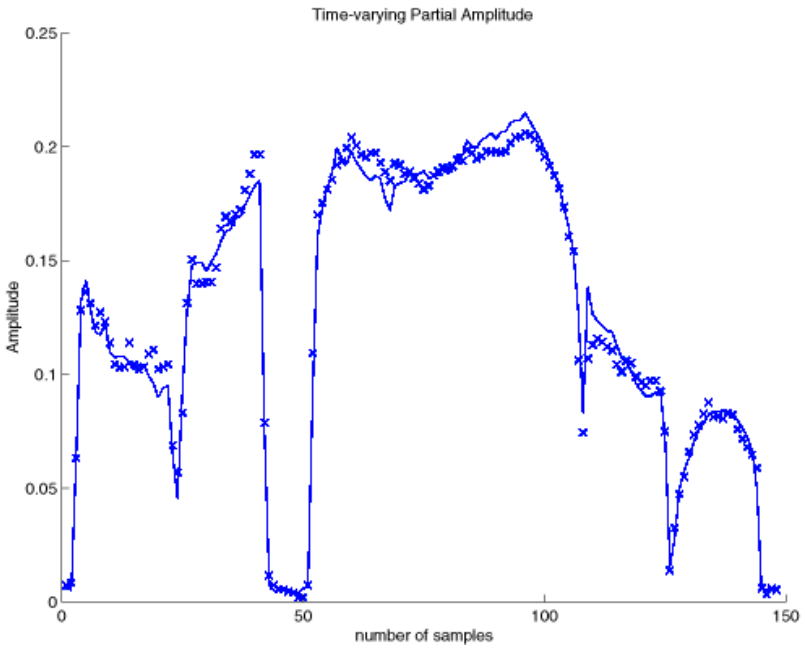


Fig 13: Estimation of First partial of saxophone with Levenberg Marquadt algorithm.(original: – estimation: x)

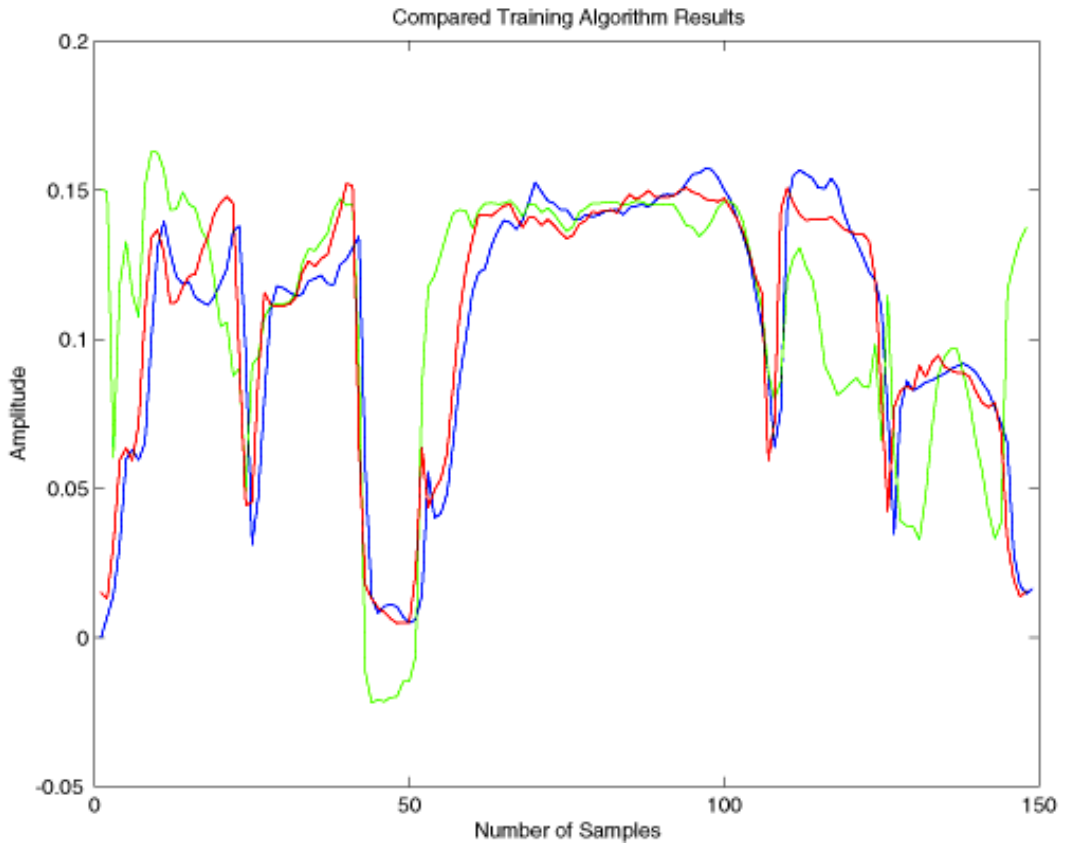


Figure 14: Comparison of the Training Algorithms for the Approximation of the Second Harmonic of a Saxophone Sample.
Backpropagation with Momentum in Green. Original in blue. Levenberg-Marquadt in red.

4.7 Regularization

The audio quality of the results is highly dependant on the inaccuracies encountered in the approximated partial amplitudes. A large error in the approximation of the partials occurs when the network is over-fitting the data during the learning process and is presented with a new set of inputs. A way to avoid over-fitting is to reduce the size of the network but it is hard to know in advance what is the ideal size of a network for a specific application. The technique used in this paper is Bayesian regularization [20]. Early-stopping regularisation methods [21] have been put aside because they don't give the smooth response required to satisfy human's sensitive ears.

Bayesian regularisation puts a control on the smoothness of a mapping function, by adding a variable extra-term in the expression of the error function. This term is used to evaluate the degree of smoothness.

The error function ε becomes:

$$\varepsilon' = \gamma\varepsilon + (1-\gamma)\varepsilon'' \tag{23}$$

where $\varepsilon'' = \frac{1}{n}\sum_{j=1}^{n} w_j^2$ is the sum of square of the network weights.
And γ is called the performance ratio.

This error function forces the network to have small weights, which entails a smoother response, less likely to over-fit. The determination of the optimum performance ratio is not an easy task. If γ is too large, the network can over-fit and if it is too small, the network will not fit the data at all. To solve this problem, we used an automated Bayesian Regularization algorithm [20] that automatically finds the optimised parameters for regularization and gives the effective number of weights and bias necessary for the neural network. This information is highly relevant for the optimisation of the network architecture, so that improved efficiency may be attained. Despite the many improvements achieved on network performances, the size of the audio samples to be processed is highly limited by the computational cost of dealing with too many data.

5 IMPROVING THE MODEL WITH PRINCIPAL COMPONENT SYNTHESIS

5.1 Introduction

One problem encountered in the realization of this project is the high computational cost of training the network on large audio samples. A tradeoff between memory requirement and speed efficiency has to be defined for each training data set. This seriously limits the generalization capacities of the network by restraining the length of the training data. One possible way to circumvent this problem is to somehow reduce the dimension of the output space necessary to represent the audio data. This is equivalent in our model to finding a compact representation of the time-varying partial amplitudes. The Principal Component Analysis tools here are particularly useful in reducing the computational cost associated with successful neural network functioning.

5.2 Analysis

The PCA is based on the data matrix extracted thanks to AddAn analysis algorithms (fig. 15 and eq. 2). The figure 15 represents a matrix of 20 rows (partials) and 150 columns (frames of 0.01 seconds).

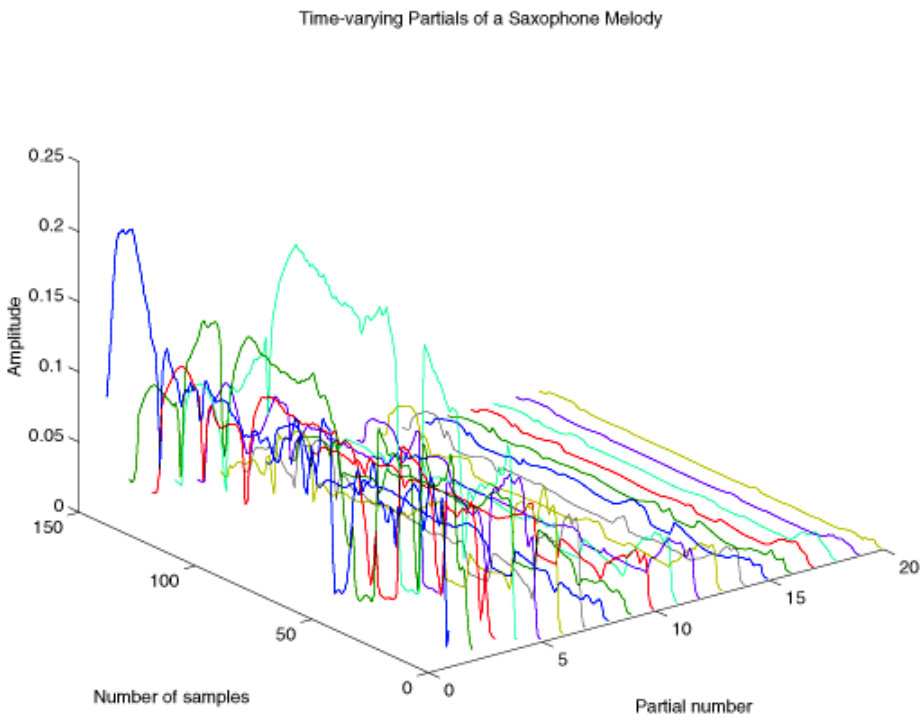


Figure 15: 3D Representation of the Additive Analysis Matrix A of a Saxophone Phrase by Coltrane.

The PCA techniques are applied in each frame of the analysis matrix successively. The covariance matrix is first calculated, the element of which are given by:

$$c_{l,c} = \frac{1}{M}\sum_{i=1}^{M}(a_{l,i} - E(a_l))(a_{c,i} - E(a_c)) \qquad (24)$$

Where N is the number of partials,
c, 1= 1,2…N,

E is the expected value:

$$\mathrm{E}(\mathrm{x_k}) = \frac{1}{M}\sum_{i=1}^{M} a_{l,c}$$

M is the number of frames

From a symmetric matrix such as the covariance matrix, an orthogonal basis can be calculated once eigenvalues and eigenvectors are found. The eigenvectors e_i and the corresponding eigen-values λ_i are the solutions to the equation:

$$C_x e_i = \lambda_i e_i \tag{25}$$

These values can be found once the solutions of the characteristic equation are known:

$$|C_x - \lambda I| = 0 \tag{26}$$

where the I is the identity matrix having the same order than Cx, and the notation |.| represents the determinant of the matrix.

The trace of the matrix (or sum of its eigen-values) is the variance of the data set. By placing the eigenvectors according to the descending order of eigenvalues (largest first), an orthogonal basis is created in which the first eigenvector lies in the direction of largest data variance. It becomes possible to simplify the representation of the analysis matrix, and avoid losing large amounts of information. By choosing the eigenvectors with the largest eigenvalues, only the smallest amount of information is lost in the mean-square sense. A fixed number of eigenvectors (or Principal Component basis) and their respective eigenvalues can be chosen while a consistent representation (or abstraction) of the data is obtained [21]. The principal component analysis of the saxophone sample shows (fig. 16) that only 6 PC basis associated with the Eigen-Values accounting for more than 99% of the total variance have been kept.

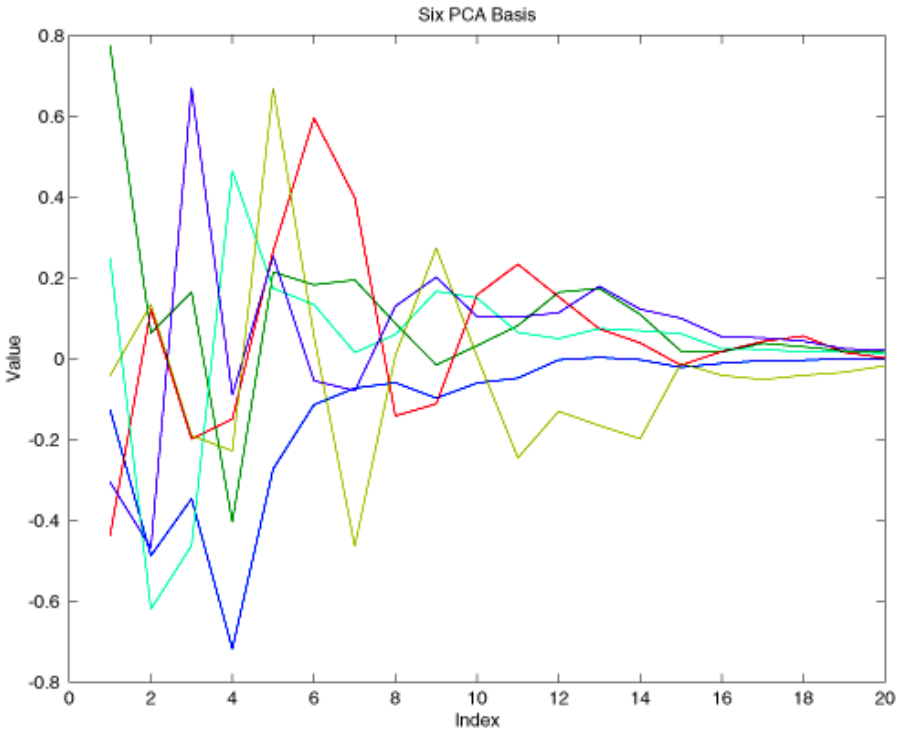


Figure 16: 6 PC Basis Obtained with the PCA of Matrix A.

Consequently, an efficient tool for creating and choosing a variable number of Principal Component Basis can be used. Yet, it is still necessary to include the time-varying weights that correspond to the PC basis in order to obtain an approximation of the original partials. This weight matrix of smaller dimension than the original matrix of partial amplitudes now serves as the new and smaller target of the neural network.

5.3 Principal Component Synthesis

5.3.1 Modeling the Problem

In order to match the original sound spectrum, the set of new PC basis must be varied in time. The method is to multiply the PC bases by a time-varying envelope (fig. 17).

The problem is represented by the following over-determined system of equations:

$$\begin{bmatrix} pca_{1,1} & . & pca_{1,L} \\ . & . & . \\ pca_{N,1} & . & pca_{N,L} \end{bmatrix} . \begin{bmatrix} T_{1,1} & . & T_{1,N} \\ . & . & . \\ T_{L,1} & . & T_{L,N} \end{bmatrix} \approx \begin{bmatrix} a_{1,1} & \cdot & a_{1,M} \\ \cdot & \cdot & \cdot \\ a_{N,1} & \cdot & a_{N,M} \end{bmatrix} \quad (27)$$

L is the number of principal component basis
N is the number of partials
M is the number of samples
A is the matrix obtained from the additive analysis
PCA is the matrix of Principal Component basis deduced from A
T is the matrix of the time-varying envelopes (fig. 17)

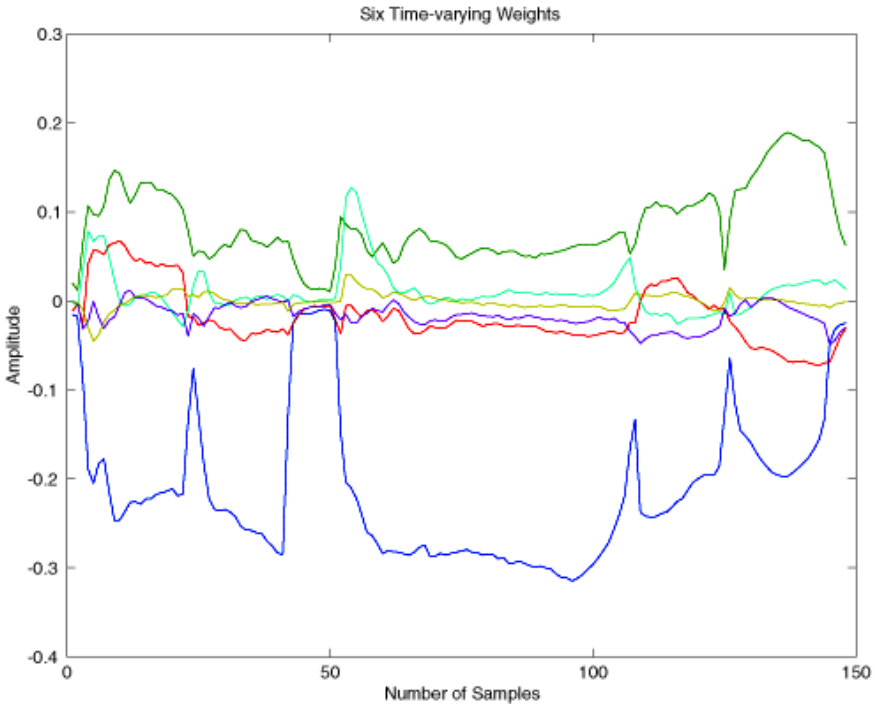


Figure 17: Time-Varying Envelope Corresponding to the 6 PC Basis of the Saxophone Sample.

Note that $L \leq N$ is desirable if the task of the network is to be eased by reducing its target matrix A of dimensions *N by M,* to a new target matrix T of lower dimension *L by M.*

5.3.2 Solving the problem

To solve this over-determined system of equations, where there are more rows or equations than unknowns, we seek the minimum norm least squares envelope solution T that minimizes:

$$\|A - pca.T\|_2 = \sum_{k=1}^{N} (\sum_{j=1}^{M} pca_{k,j} T_{j,r} - A_{k,r})^2 \qquad (28)$$

The matrix T solution of this problem is calculated with the standard Cholesky decomposition technique [28]. Any number of PC bases can now be chosen to re-synthesize the audio signal. The more bases we keep the better the quality of the audio rendering. If the number of bases is equal to the number of frames in the analysis matrix, the synthetic spectrum is exactly the same as the original sound spectrum. In the case of the saxophone sample, we can now replace the previous target matrix A of the neural network (dimension 20 by 158) by the matrix T of dimension 6 by 158. The training of the network on this matrix W is more efficient, accurate, and less expensive in terms of computation since the training data is more than 3 times smaller than before. The retrieval of the time-varying amplitudes for the re-synthesis is obtained by a simple multiplication of the matrix PCA by the matrix T. In a real-time situation, the calculation of partials in response to control information is then simple and efficient, provided the network is well trained. The PC bases are calculated beforehand and stored in memory. Fig.18 shows a 3D representation of a time-varying spectrum in a basis of 3 principal components. Additional graphical representation showing the efficiency of the PCA neural network synthesizer can be found in the appendix B.

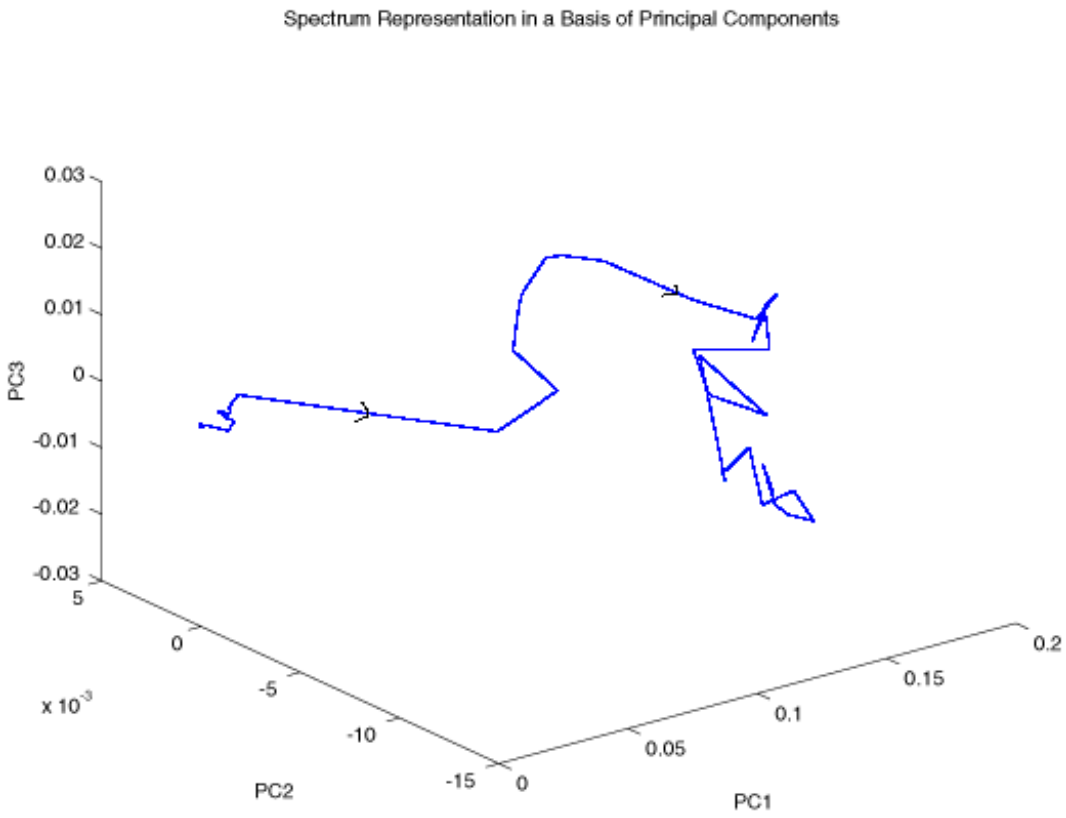


Fig 18: Time Varying Spectrum in a 3 PC Basis Representation

6 APPLICATIONS

6.1 Playing a New Melody

One of the most powerful features of this system is its ability to change the original melody, while preserving the musical identity of the audio sequence on which the network has been trained. The figure 7 shows the spectrogram of the saxophone melody estimated from the PCA neural, network synthesizer. The difference between the audio signal reconstituted from the target spectrum (represented by the additive analysis matrix A), and the synthesised signal built from the response of the network is not audible. The network can be fed with pitches and loudness in a different order, and still gives convincing results, if the new controllers values are in the same range as the training data set (Fig. 19 and 20). The spectral properties of the sound are preserved when the control information varies.

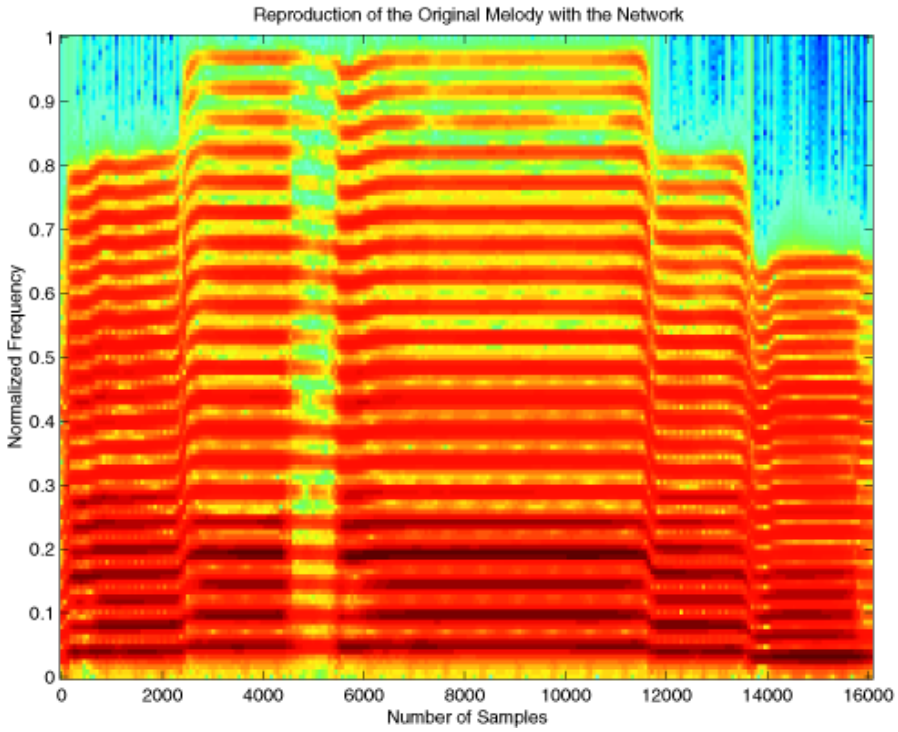


Fig. 19: Original Melody Played by Coltrane

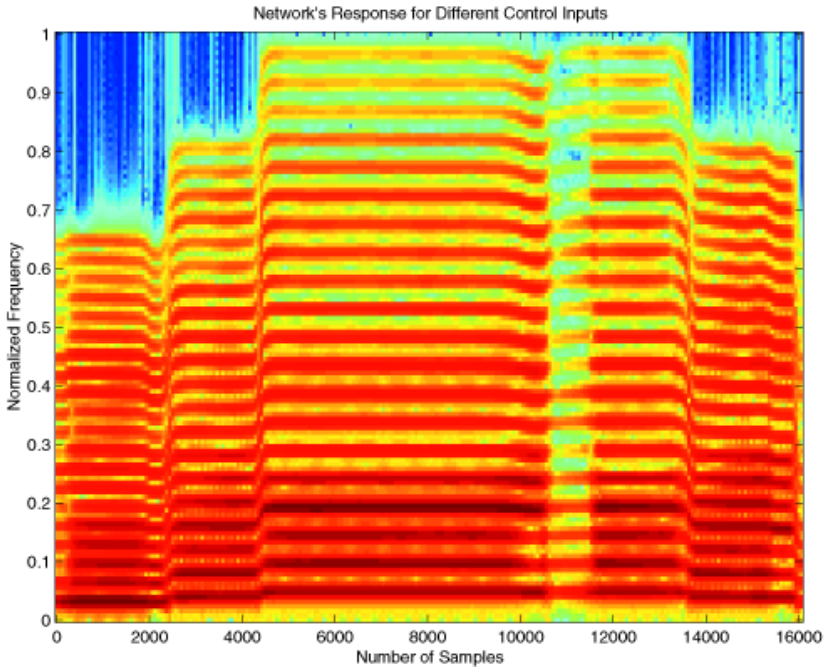


Fig. 20: Modified Melody preserving Coltrane's Style.

6.2 Pitch Shifting/ Time Stretching

Common applications such as time stretching and pitch shifting also give satisfactory results.

Two different techniques for pitch shifting are possible with the neural network principal component synthesiser. The first type of technique is linked to the previous paragraph. It takes advantage of the generalisation properties of the network. The musician can change the note he/she wants to play by entering directly the desired frequency as a control input. The limitation of this technique is the range of the data on which the network has been trained. The results will be poor if the frequencies, given as an input to the system, are too different from the frequencies encountered in the training data set.
Another method, that tackles the previous problem, consists in doing the pitch processing on the output of the network. Pitch shifting is realised by accessing the output values at different rates during playback.

6.3 New Controllers

The choice of controllers is left to the will of the musician. The model we propose is applicable to any other kind of controller that can be inferred from an audio sample. For instance, controllers for noisiness or vibrato could be implemented too.

6.4 Cross-Synthesis

6.4.1 Type 1

The structure of the model makes it suited for cross-synthesis where the control parameters of one instrument can be reused on a network trained with another type of instrument.

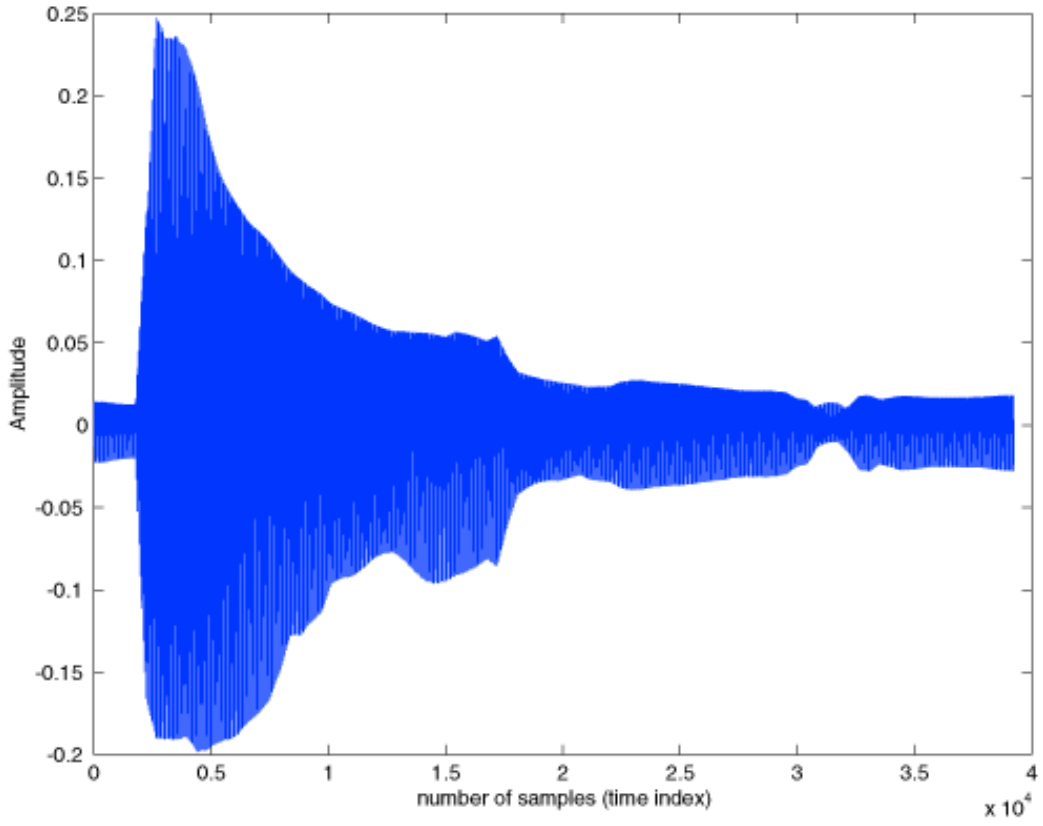


Fig. 21: Network Trained on a Trumpet Controlled with Piano Parameters

6.4.2 Type 2

The option of keeping the controllers and the network from a specific instrument while doing the synthesis using PC bases from another instrument can be considered too. All the results presented are applicable to any kind of musical instrument that can be easily described by additive analysis.

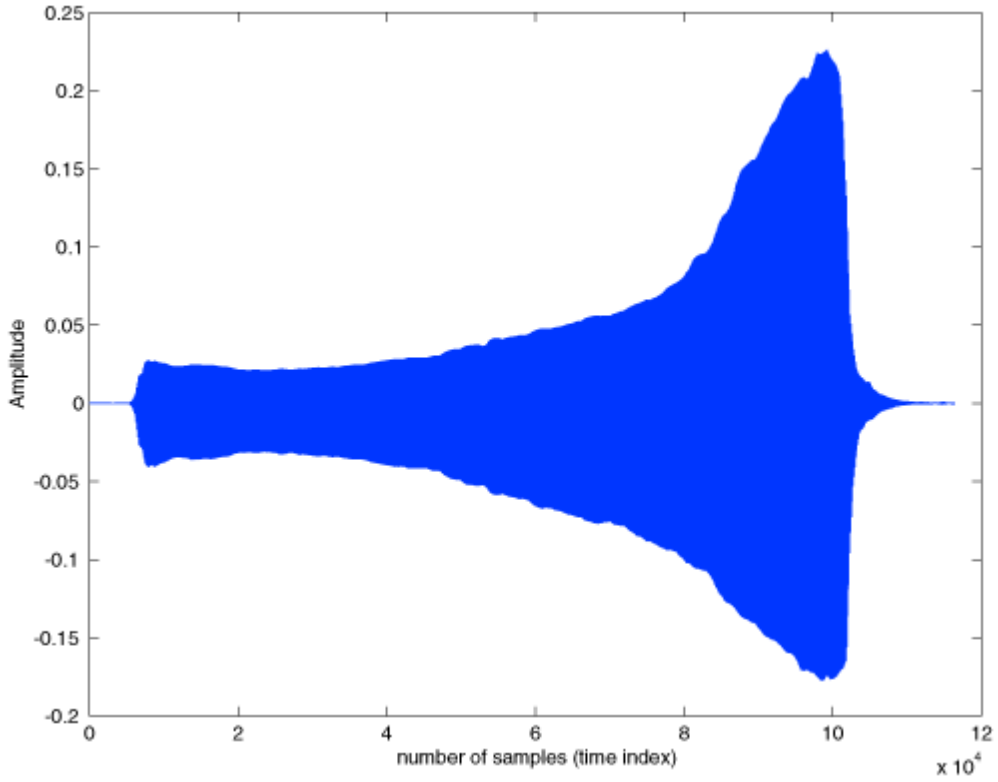


Fig. 22: Network and Controls from a Trumpet but Principal Components from a Piano

6.5 Studio Corrections

The system we defined is equally interesting for studio work. If a musician's performance contains a small mistake in the interpretation, or in the melody, the neural network synthesizer allows for its correction directly from the recorded material. It preserves the original musician's style and avoids a second recording take.

Fig. 23 and 24 show the spectrogram of two different interpretations of the same note, obtained by varying the control information.

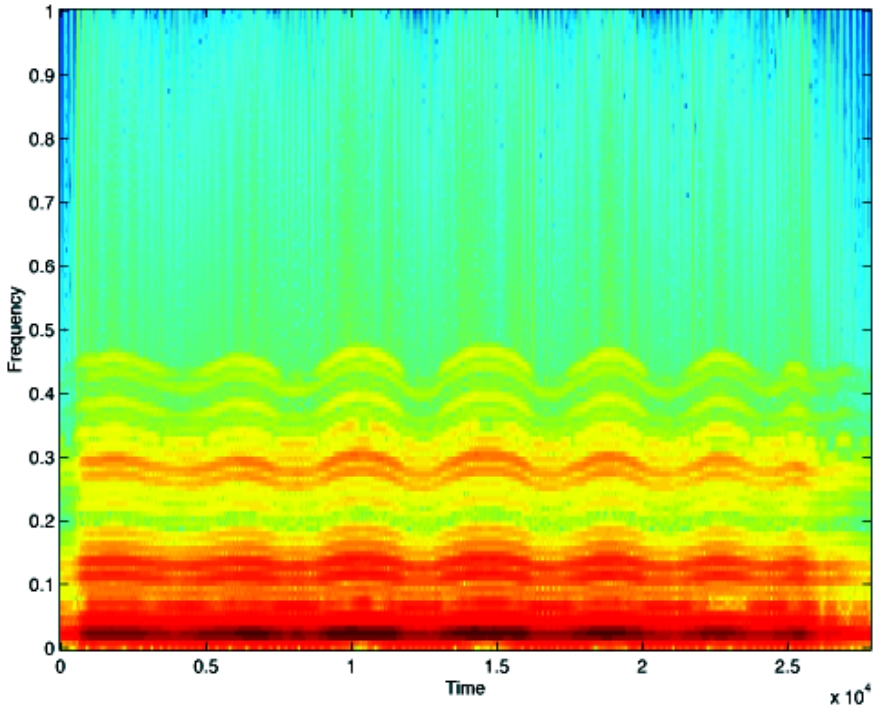


Fig. 23: Singing Voice

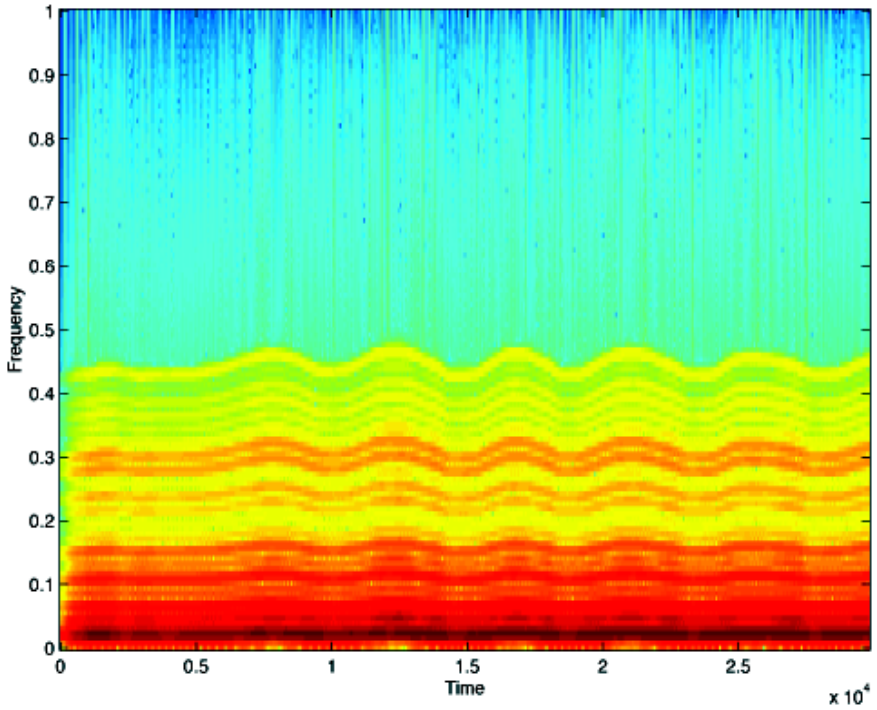


Fig. 24: Same Note but Different Control Information

6.6 Scalable Synthesis

In addition, due to the characteristics of the PC synthesis, our model has an interesting property of scalability. The number of useful PC bases can be chosen depending on the quality of sound required at the moment. This comportment is particularly interesting in networked applications, where the bandwidth available is variable. The quantity of data to be transmitted (number of time varying envelopes) can be modified in regard to the availability of the bandwidth, which make the Neural Network Principal Component Synthesizer an interesting synthesis scheme in the context of MPEG-4 [22], [23] and [24].
This model definitely offers a flexible control on sound which purpose is to stimulate musical creativity.

7 CONCLUSION

7.1 Summary

This paper used recorded audio samples to present a model for flexible control of sound synthesis. An additive analysis/synthesis paradigm was first chosen for its generality. Unlike other specific models, this model was able to simulate many different sounds with satisfactory results. An efficient design of an artificial back-propagation neural network for partial approximation was then described. The Levenberg-Marquadt training algorithm (associated with Bayesian Regularization) provided an accurate estimation of the time-varying partials needed for the re-synthesis of a musical sound. The powerful statistical methods of Principal Component Analysis improved the model's flexibility and scalability. These methods also reduced the quantity of data necessary for the representation of sound parameters, and proved particularly suited for the neural network's approximations of harmonics. Applications of the original design for the control of sound samples led to interesting new ways of interacting with sounds.

7.2 Future Directions

There are several improvements to be made on this model, which can be achieved through further research. An extra input for the neural network should be added to take into account the noise parameter, not included in this study's additive synthesis paradigm. A promising additive analysis-synthesis technique called Loris, that uses a noise model in addition to sinusoidal tracks, has been developed by Kelly Fitz [25]. It could replace the analysis/synthesis paradigm of this dissertation, without any major change in the global architecture of our system. The work presented in this dissertation did not include a memory mechanism for previous states of an instrument. We know, for example, that in a rapid passage, a given note on a wind instrument will sound different depending on whether it was preceded by a higher or lower pitch. Future work should take these temporal dependencies into account. Prior state can be introduced into the network architecture. The synthesis models presented are causal and can be implemented in real time. Future research and musical application will benefit greatly from real-time interaction between the musician and this new "smart instrument". Another possible future direction that shows considerable promise is the application of Kernel Principal Component Analysis [26]. By the use of Mercer kernels, one can efficiently compute principal components in high-dimensional feature spaces, related to input space by some nonlinear map.

This paper's modular model is easily adaptable to future changes. It is possible to add new inputs, and refined synthesis models while preserving the general architecture of the original system. This flexibility ensures a reliable tool for the development of creative expression.

8 APPENDIX A

Descriptive List of the Main Functions Written in Matlab for the Neural Network Principal Component Synthesizer

This code has been tested on a desktop macintosh PowerPC G3 upgraded to G4 at 500MHz with 384 MB of ram, and with virtual memory on.
Matlab version 5.2.1
Neural Network Toolbox version 3.0
Signal Processing Toolbox version 4.1

- **addread.m**

Syntax:

part = addread(file, n_synthpartial);

Description:

Reads Ircam's *Addan* partial analysis file in Ascii format and returns the matrix of time-varying partials.

Inputs:

file: Name of *addan* analysis file for the partials in ascii format.

n_synthpartial: Number of partial desired in the re-synthesis

Outputs:

part: Time-varying partials matrix. (dimension number of partials by number of analysis frames)

➢ centroid.m

Syntax:

bright = centroid(part, f0)

Description:

Calculate the centroid of a time varying spectrum.

Inputs:

part: analysis matrix of partial's amplitude (dimension number of partials by number of analysis frames)

f0: vector of time varying fundamental frequency values (dimension one by number of frames)

Outputs:

bright: centroid of the time-varying spectrum

➢ cumul.m

Syntax:

[added,nbr] = cumul(b,thres);

Description:

In the context of Principal Component Analysis, gives the successive sum of the eigen values and the number of principal component corresponding to a desired precision.

Inputs:

b: matrix of the time-varying partials of the signal to be pca-analyzed (dimension number of partials by number of analysis frames)

thres: parameter controlling the accuracy of the synthesized sound (percentage of the total variance kept) 0<thres<1

Outputs:

added: vector representing the successive sum of the eigen-value

nbr: number of principal component basis necessary to attain the precision required

➢ f0read.m

Syntax:

[f0,time] = f0read(file_f0)

Description:

Imports fundamental frequency analysis files from *addan* in ascii format, and returns a time-varying frequency vector in matlab.

Inputs:

File_f0: *addan* analysis f0 file in ascii format

Outputs:

f0: vector of time varying fundamental frequency (dimension one by number of analysis frames)

time: vector giving analysis frame time sequence

➢ fltr.m

Syntax:

[filter,start,stop] = fltr(p,factor)

Description:

Removes artifacts in the beginning and in the end of the partials estimation of a neural network trained with back-propagation and momentum technique.

Inputs:

p: Neural network input matrix. The first line corresponds to the fundamental frequency. The second line corresponds to the intensity summary. The third line corresponds to the centroid. (dimension three by number of training samples)

factor: Factor controls the threshold value (mean of the signal/factor) under which the signal is considered as an artifact.

Outputs:

Filter: "filtered" partial matrix.
Start: First sample index where the amplitude of one partial is not zero.
Stop: Last sample index where the amplitude of one partial is not zero.

➢ **lsq.m**

Syntax:

x = lsq(a, b);

Description:

Least square solution of over-determined system of equation a.x = b. Solved with cholesky decomposition of the normal equation.

Inputs:

a: matrix with n_row > n_column;
b: row vector;

Outputs:

x: matrix of the least-square solutions.

➢ **main.m**

Syntax:

[out, out_s, net, tr, p, t, synth, synth_part, noise, n, a, synth_w] = main(file_add, file_f0, n_partial, epochs, goal, norm, pca, thres, fs, awin);

Description:

Main script. Performs the principal component analysis of the data, the training of the network and the sound re-synthesis.

Inputs:

file_add:	*addan* harmonics analysis file in the ascii format
file_f0:	*addan* fundamental frequency analysis file in the asccii format
n_partial:	number of partial desired for the re-synthesis
epochs:	number maximum of iteration in the training process
goal:	wanted error value
norm:	performs the normalization on the inputs and outputs of the neural network when norm = 1
pca:	decorrelates and orders the input vectors of the network if pca=1 (pca without any reduction of the dimensionality)
thres:	sets the number of principal component to synthesize (if integer value). Calculates the number of principal component to synthesize in order to retain a certain percentage of the total variance of the data (if percentage value) (c.f. the function cumul.m)
fs:	sample rate in Hertz
awin:	size of the analysis window in seconds

Outputs:

Out:	synthesized sound waveform calculated from the outputs of the neural network.
out_s:	smoothed version of the output, filtered with the function fltr.m. Only necessary when the network is trained with simple back propagation with momentum algorithm.
net:	trained network
tr:	record of training performances
p:	Neural network input matrix. The first line corresponds to the fundamental frequency. The second line corresponds to the intensity summary. The third line corresponds to the centroid. (dimension three by number of training samples)
t:	target matrix of the neural network (normalized partials)
synth:	synthesized sound waveform calculated directly from the analysis partials (not from the estimation of these partials by the network)
synth_part:	partial matrix as estimated by the network

noise:	difference between synth and out (difference between the target and the network estimation).
n:	number of initial units in the neural network
a:	matrix of PC basis. (dimension number of partials by number of principal component basis)
synth_w:	synthesized time-varying envelopes (c.f. chap5 improving the model with principal component synthesis).

➢ **nn_io.m**

Syntax:

[ptrans, pn, tn, p, t, meanp, stdp, meant, stdt, a] = nn_io(file_add, file_f0, n_partial, norm, pca, thres);

Description:

Pre-processes the neural network inputs and outputs. Normalizes, de-correlates and calculates the principal components of a training data set.

Inputs:

file_add:	*addan* harmonics analysis file in the ascii format
file_f0:	*addan* fundamental frequency analysis file in the asccii format
n_partial:	number of partial desired for the re-synthesis
norm:	performs the normalization on the inputs and outputs of the neural network when norm = 1
pca:	decorrelates and orders the input vectors of the network if pca=1 (pca without any reduction of the dimensionality)
thres:	sets the number of principal component to synthesize (if integer value). Calculates the number of principal component to synthesize in order to retain a certain percentage of the total variance of the data (if percentage value) (c.f. the function cumul.m)
fs:	sample rate in Hertz
awin:	size of the analysis window in seconds

Outputs:

ptrans:	de-correlated network input
pn:	normalized network input (mean of 0 and variance 1)
tn:	normalized target of the network
p:	Neural network input matrix. The first line corresponds to the fundamental frequency. The second line corresponds to the intensity summary. The third line corresponds to the centroid. (dimension three by number of training samples)
t:	network target matrix of time-varying partials. (dimension number of partials by number of analysis frames)
meanp:	mean of the input data
stdp:	standard deviation of the input data
meant:	mean of the target data
stdt:	standard deviation of the target data
a:	matrix of the principal component (dimension number of partials by number of principal components)

➢ **pca.m**

Syntax:

[base,value] = pca(b);

Description:

Performs principal component analysis on the analysis matrix of time-varying partials

Inputs:

b:	Matrix of data for analysis. Each row of b represents a variable (in this project a row corresponds to a time-varying partial amplitude) each column of b represents an observation (spectrum for an analysis frame).

Outputs:

base:	PCA basis matrix.
value:	vector made of the eigen-values sorted by decreasing order.

➢ Synthadd.m

Syntax:

Out = synthadd(part, f0, n_partial, fs, awin)

Description:

Additive synthesis with linear interpolation between the analysis frames to match the right sampling rate.

Inputs:

Part:	Matrix of time-varying partials. (dimension number of partials by number of analysis frames).
f0:	Vector of time-varying fundamental frequency values (dimension one by number of frames)
n_partial:	number of partial desired for the re-synthesis
fs:	sample rate in Hertz
awin:	length of the analysis window in seconds.

Outputs:

Out:	synthesized waveform.

All the other functions used for this project (for cross-synthesis, pitch shifting, or for displaying graphics, etc) are based on the fundamental functions we just described.

crossynth.m	performs cross-synthesis of type 1 (c.f chap 6 applications)
crossynth2.m	performs cross-synthesis of type 2 (c.f chap 6 applications)
pcadd.m	performs principal component analysis on an additive analysis file from *addan* in ascii format
plotpart3d.m	plots a 3D representation of the time-varying partials

plotpca3d.m spectrum representation in a basis of principal components

plotting.m displays principal component basis, loudness, time-varying envelopes, time-varying partials, target and synthesized sound waveform, target and synthesized spectrogram.

plot_input_norm.m

displays the normalized inputs of the neural network

plot_part_smooth_target.m

displays time-varying amplitude of partials. Compares the network estimation, the target and the "smoothed" estimation.

start.m launch the whole analysis synthesis process for a specific sound sample and save the results in a record file.

synthmelod.m re-synthesize with a different succession of notes

synthpca.m pca synthesis from addan .f0 and .add files

synthsmooth.m smoothen time-varying partials

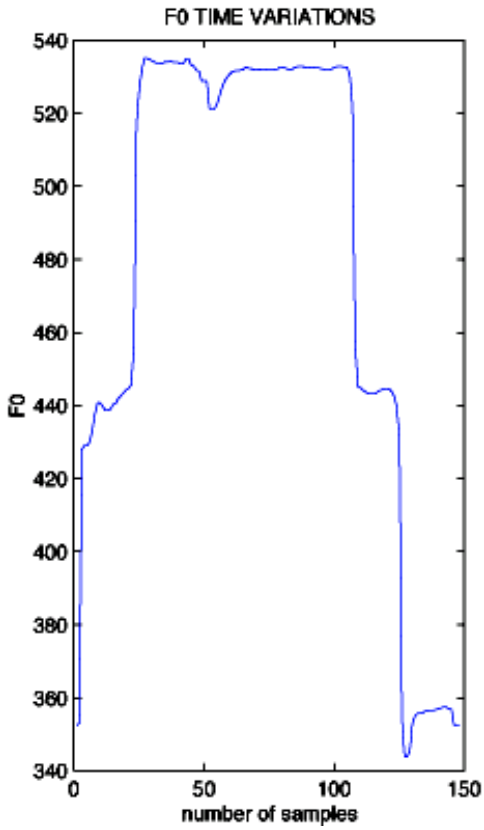
synthstretch.m performs time-stretching

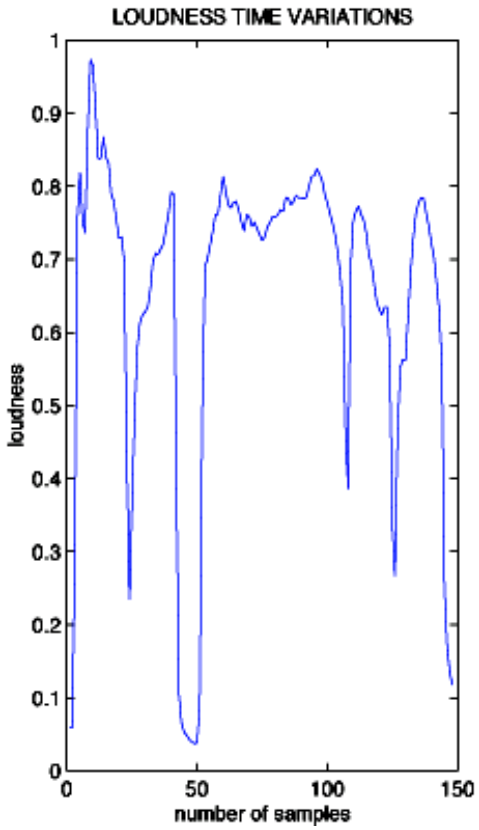
9 APPENDIX B

Graphical Tools

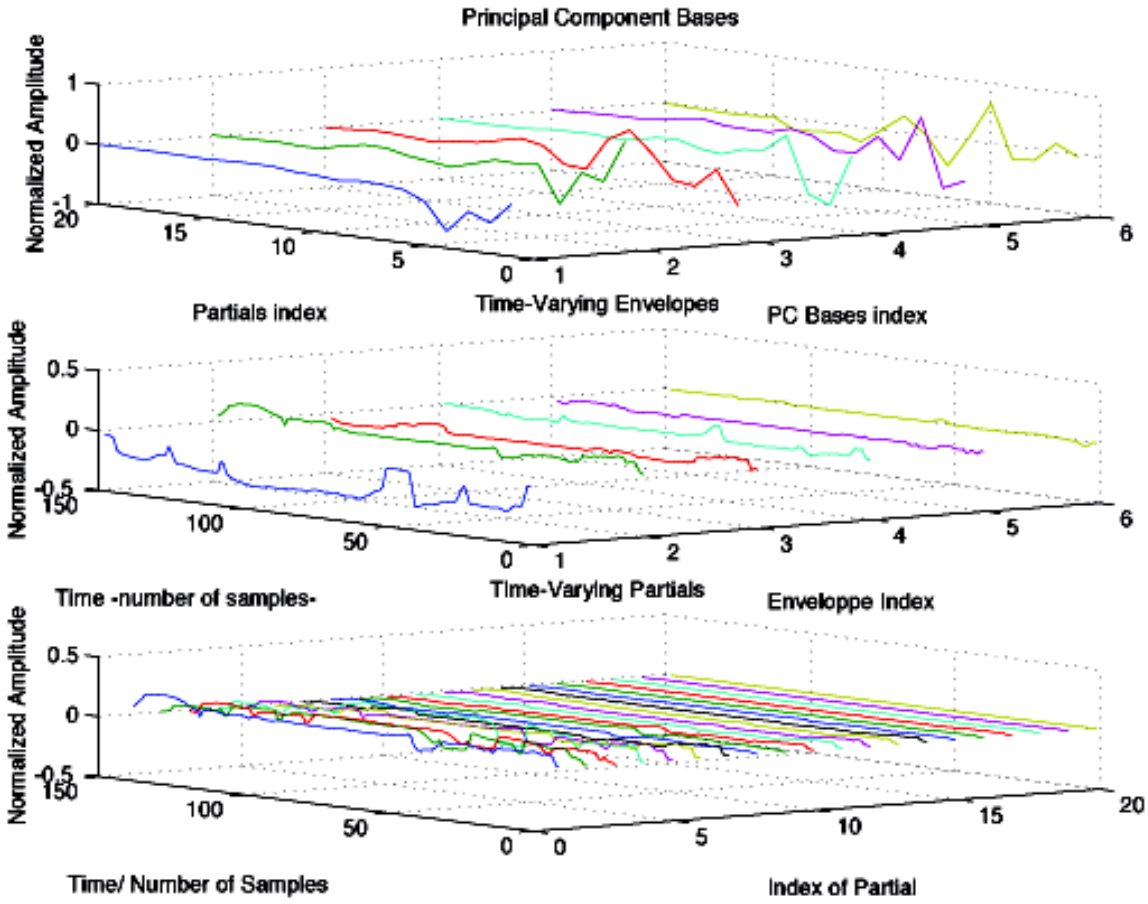
During this project, graphical tools appeared to be very useful. Examples of typical displays plotted during an analysis and synthesis session are proposed in this appendix. The following displays come from the same saxophone analysis synthesis session as in chapter 5.

Inputs:

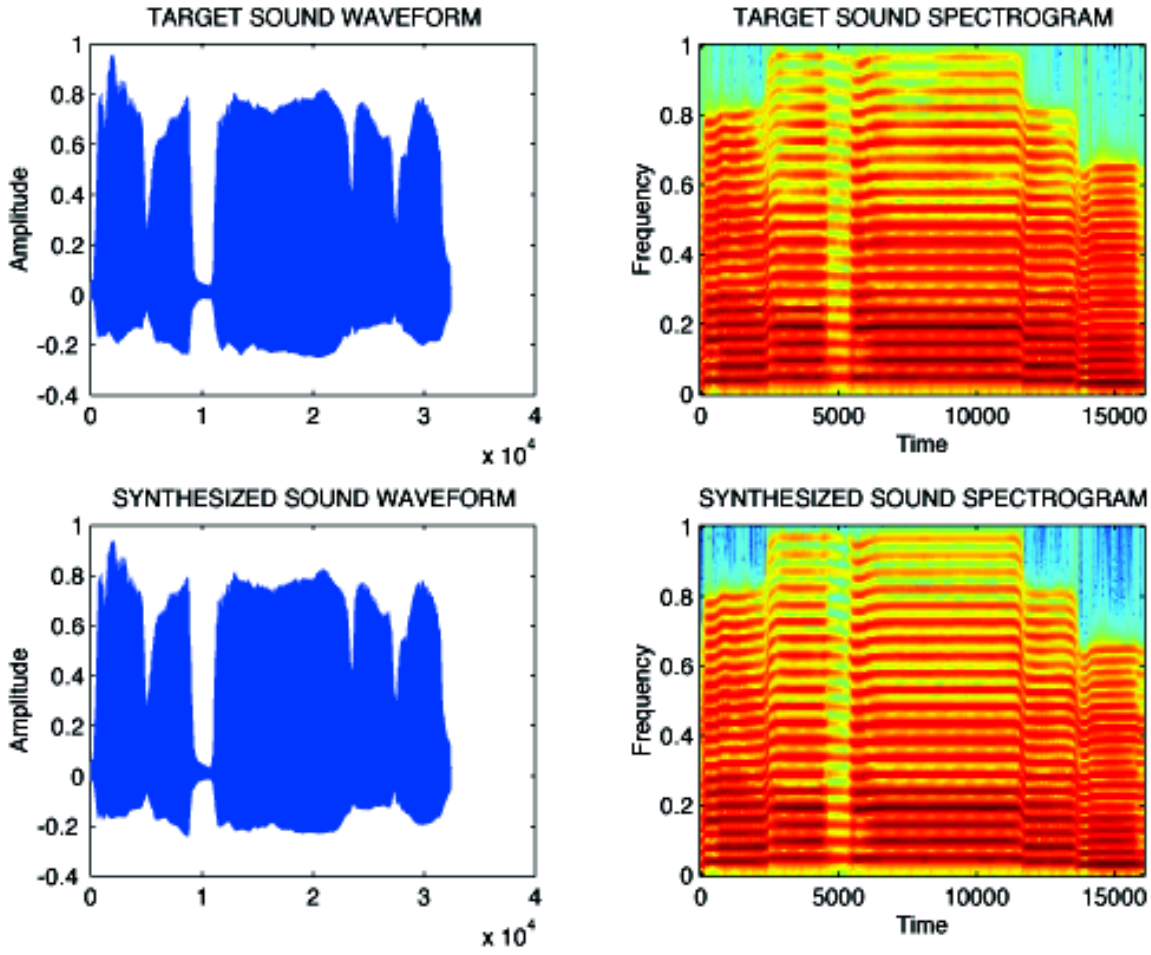




Outputs:

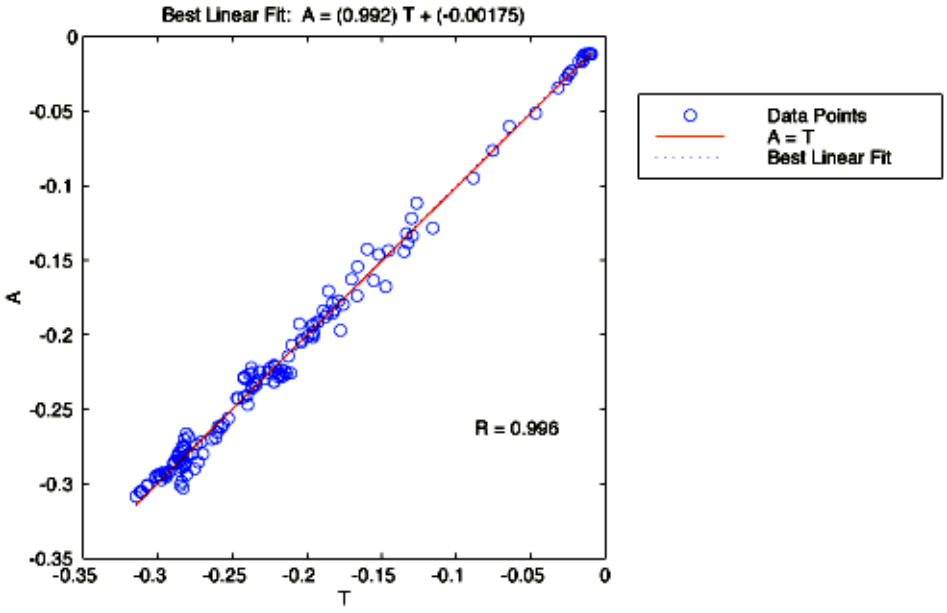

Principal Component Bases
Normalized Amplitude
Partials index
Time-Varying Envelopes
PC Bases index
Time -number of samples-
Time-Varying Partials
Enveloppe Index
Time/ Number of Samples
Index of Partial

Qualitative Description of the Results

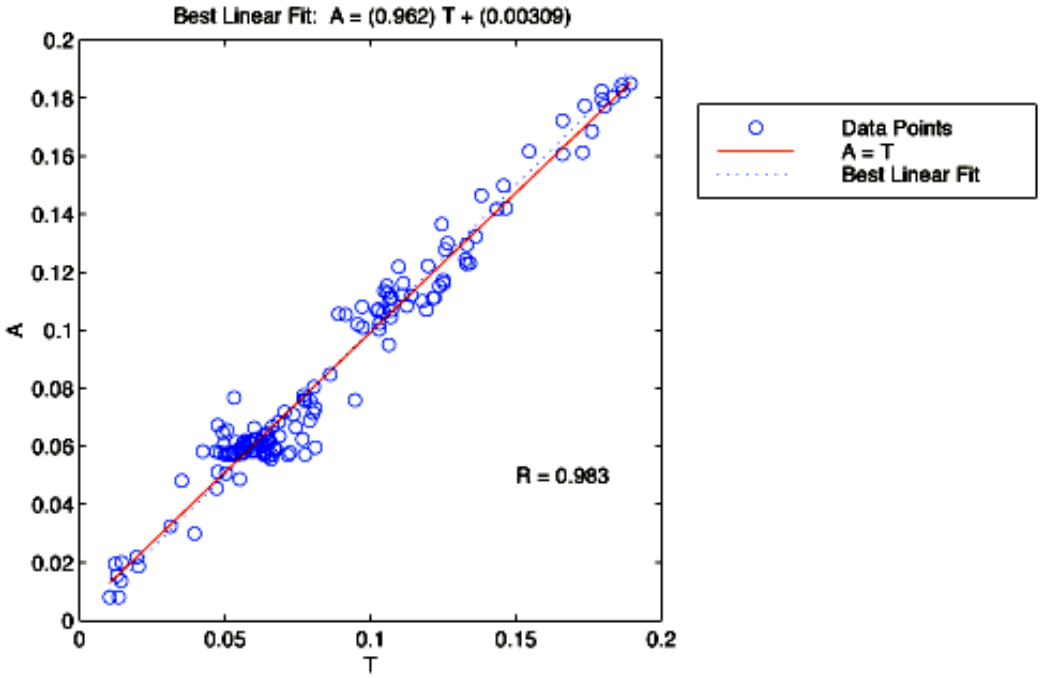


Quantitative Description of the Results:

Linear regression between the target (time-varying envelope) and the output of the network.



Time-varying envelope 1. Coefficient of regression R: 0.996



Time varying envelope 2. Coefficient of regression R: 0.983

10 BIBLIOGRAPHY

[1] M. Lee, D.Wessel, "Connectionists Models for Real-Time Control of Synthesis and Compositional Algorithms", *ICMC Proceedings* (1992).

[2] D. Wessel, C. Drame and M. Wright, "Removing the Time Axis from Spectral Model Analysis-Based Additive Synthesis", *ICMC* (Univ. of Michigan, Ann Arbor, USA, 1998)

[3] A. Robel, "Neural Network Modeling of Speech and Music Signals" In Neural Information Processing Systems 9 (NIPS 96), pages 779--785, 1997.

[4] B. Schoner, C. Cooper, C. Douglas, N. Gershenfeld. "Data-driven Modeling of Acoustical Instruments." Journal for New Music Research, Vol 28(2), 1999.

[5] T. Jehan and B. Schoner. "An Audio-Driven, Spectral Analysis-Based, Perceptual Synthesis Engine." Audio Engineering Society, Proceedings of the 110th Convention, Amsterdam, The Netherlands, 2001.

[6] What is Max-Msp? [http://www.cycling74.com/products/maxmsp.html]

[7] W. Thomas Miller, III, Richard S. Sutton, and Paul J. Werbos (Eds.) Neural Networks for Control. Cambridge, Massachusetts: The MIT Press. 1990.

[8]Diphone Studio, [http://www.ircam.fr/produits/logiciels/diphone-e.html]

[9] Tolonen, Tero, Välimäki, Vesa & Karjalainen, Matti. Evaluation of Modern Sound Synthesis Methods. Report no. 48 / Helsinki University of Technology, Department of Electrical and Communications Engineering, Laboratory of Acoustics and Audio Signal Processing. TKK, Otaniemi, 1998.

[10]W. M. Hartmann, *Signals, Sound, and Sensation* (AIP Press. Woodbury, NY, USA, 1997)

[11] M. Minsky and S. Papert, *Perceptrons*. MIT Press, Cambridge, MA,1969.

[12] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control,Signals and Systems*, 2:303 – 314, 1989.

[13] C. M. Bishop, *Neural Networks for Pattern Recognition* (Oxford University Press. New York, USA, 1995), pp 126-127.

[14] R.O. Duda, P. E. Hart, D. G. Stork, *Pattern Classification* (Wiley Interscience Publication, New York, USA, 2001), p 310.

[15] S. Haykin, *Neural Networks, a Comprehensive Foundation* (Prentice Hall. Upper Saddle River, NJ, USA, 1999), pp 396-405.

[16] C. M. Bishop, *Neural Networks for Pattern Recognition* (Oxford University Press. New York, USA, 1995), p 310.

[17] C. M. Bishop, *Neural Networks for Pattern Recognition* (Oxford University Press. New York, USA, 1995), p 143.

[18] M. Berthold and D. J. Hand, *Intelligent Data Analysis* (Springer Press. 1999), pp. 229-230.

[19] C. M. Bishop, *Neural Networks for Pattern Recognition* (Oxford University Press. New York, USA, 1995), p 290.

[20] MacKay, D. J. C., "Bayesian interpolation," *Neural Computation*, vol. 4, no. 3, pp. 415-447, 1992.

[21] S. Le Groux, "A PCA-Based Mechanism for Scalable Sound Synthesis" *MSc Dissertation*, King's College of London, 2001.

[22] Scheirer, E.D. *MPEG-4 standard*. MIT Media Laboratory. Machine Listening Group, MIT Media Laboratory.

[23] Scheirer, E.D. (1999) *External documentation and release notes for saolc*. MIT Media Laboratory. Machine Listening Group, MIT Media Laboratory.

[24] Scheirer, E.D. *SAOL: the MPEG-4 structured audio orchestra*

language. MIT Media Laboratory. Machine Listening Group, MIT Media Laboratory.

[25] The Reassigned Bandwidth-Enhanced Additive Model, [http://www.cerlsoundgroup.org/Loris/]

[26] Scholkopf, B., Smola, A. J., and Mtiller, K.-R. Nonlinear component analysis as a kernel eigenvalue problem. *In Neural Computation*, volume 10, pages 1299-1319, 1998.

[27] M. Jordan and R. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, volume 6, pages 181-214,1994.

[28] G. Strang, *Linear Algebra and its Application* (Saunders College Publishers. Orlando, Florida, USA. 1988).