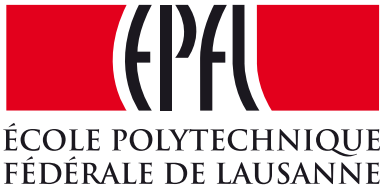
ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
SCHOOL OF LIFE SCIENCES



Master project in Life Sciences and Technology

MUSIC LEARNING WITH LONG SHORT TERM MEMORY NETWORKS

Done by

FLORIAN COLOMBO

Under the direction of

Prof. Wulfram Gerstner
Laboratory of Computational Neuroscience, EPFL

External Expert

Prof. Dr. Felix Gers
Online-Learning Laboratory, BHT Berlin

Lausanne, EPFL 2015

Acknowledgements

I sincerely thanks Alex Seeholzer, who supervised my work, for his more than helpful advices and ideas, Klaus Greff from IDSIA, who kindly answered some of my questions about the long short term memory networks and Pr. Wulfram Gerstner, who gave me the opportunity to join my studies with my passion for music.

Lausanne, January 2015 F. C.

"Again, it [the Analytical Engine] might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine . . . Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent."

Lovelace, Ada. Notes upon L. F. Menabrea's *Sketch of The Analytical Engine Invented by Charles Babbage.* 1843.

Abstract

Humans are able to learn and compose complex, yet beautiful, pieces of music as seen in e.g. the highly complicated works of J.S. Bach. However, how our brain is able to store and produce these very long temporal sequences is still an open question. Long short-term memory (LSTM) artificial neural networks have been shown to be efficient in sequence learning tasks thanks to their inherent ability to bridge long time lags between input events and their target signals. Here, I investigate the possibility of training LSTM networks to learn and reproduce musical sequences and eventually better understand some of the mechanisms neural networks deploy to learn and compose long time scale structures.

To be able to learn music with LSTM networks requires representing musical sequences in these networks. The musical representation developed for this work is inspired by the tonotopic representation of sounds in the auditory system. It is shown that LSTM networks are able to learn each note transitions of the monophonic and polyphonic versions of a simple song using a particular network architecture where both input and output of LSTM networks are musical notes in the developed network representation. However, this architecture for LSTM networks fail to learn longer and more complex musical sequences (e.g. the J.S. Bach cello suites). To solve this problem, I introduce the separation of time scales model, which consists in two connected LSTM networks, operating on different time scales. On one hand, trained slow time scale LSTM networks produce transitions between unique identifiers of musical patterns, which resemble a compressed memory of the pattern akin to neural memory. This gives the long time structure of music. On the other hand, trained fast time scale LSTM networks are producing the note-to-note transitions of each musical patterns. The latter receives as additional inputs the identifiers from slow time scale networks, akin to feed-forward input from memory regions of the brain. These unique identifiers bias fast time scale LSTM networks toward the production of the corresponding musical pattern. The most efficient identifiers of musical patterns are found to be a representation of how similar patterns are from one another. Finally, when unlearned pattern identifiers are given to trained fast time scale networks, novel musical patterns are created from the learned production rules.

I show that the introduction of a separation of time scales greatly improves the capacity of LSTM networks to learn a larger body of musical sequences. Finally, I demonstrate that previously unseen input biases can be used to induce the network into the generation of new musical sequences, akin but not similar to known patterns. This presents a possible first step towards the generalization of previously learnt musical knowledge to the creation and composition of new music by artificial neural networks.

Contents

Introduction

Can we come close enough to the mechanism of artistic creation in order to be able to understand how a composer, a musician or a director, chooses to combine particular notes or rhythms? This question is addressed among others in the book "Enchanted neurons" by the neurobiologist Pierre Changeux, the composer Pierre Boulez and the musicologist Philippe Manoury [1]. The work presented here, assesses this question by the means of artificial neural networks (ANNs). Can ANNs learn the mechanism of music composition and produce complex musical sequences?

Artistic creation is often considered as a third entity, which is not part of the physical world, but rather part of the spirit or the soul [1]. J.S. Bach compositions were even attributed to the voice of God. Indeed, many people could not find any rational explanations for how *only* a man could create such intrinsically complicated and yet beautiful compositions. In this regard, Hofstadter [2] relates the story of the musical offering: how can Bach's brain improvise in real time a six voices fugue, while others could not do it on paper? From the point of view of the neurosciences, Bach's brain is a neural circuits trained with past experiences. This circuit commands the hand of the composer to write a particular note at a particular place in the musical creation process. Thus, a neuroscientist would probably strictly say that Bach's brain was either better trained or had a special layout, particularly well suited for the task, or both. A computational neuroscientist would argue that the neural circuits of Bach's brain dedicated to music composition could be represented in an analytical form with interconnected units whose behavior obeys specific dynamical equations. The idea of using an analytical engine to mimic the musical creation process can be trace back as far as in 1843 when Ada Lovelace thought about the possibility of Charles Babbage's analytical engine to represent music in order to *compose elaborate and scientific pieces of music of any degree of complexity or extent* [3].

However, music is a very complex task to represent with analytical processes. Indeed, musical structures generally spread along very long time lags. In addition, musical sequences are highly non deterministic. On the note-to-note time scale, many different notes depending on the sequence place follow a particular note or chord. On a longer time scale, a given musical pattern is often followed by many other patterns, even in the same piece of music. Such non-deterministic sequences are here referred to as non-Markovian. Furthermore, music could have repetitive structures over many different time scales that range from the single

note to a symphony of more than one hour where the first theme is recalled during the last bars. Overall, although musical sequences are constrained by many rules, learning them is a very challenging task for ANNs.

Back to the neuroscientist point of view, as Bach's neurons operate on the millisecond time scale, solving such long time problems requires the storage of information. The brain possesses such storage capability in long- and short-term memory. When the fourteen years old Mozart's neurons received the *Miserere* music in Vatican, they were able to store it until Mozart wrote the polyphonic song entirely in musical notation the same evening. That Mozart's neurons, probably operating similarly to any regular human's neurons on the millisecond time scale, remembered the whole of this long musical sequence for such a long time is most likely a memory emerging from the collective activity of a whole network of recurrently connected neurons [4] – a network effect. Similarly, to improvise in real time a fugue with six Obligato part, a task that only Bach's neurons could do, Bach needed to efficiently store each previous time events to be able to keep a well organized structure that fit the highly constraining rule of the fugue. ANNs and especially recurrent neural networks (RNNs), inspired by the biological brain architecture, provide a numerical model for a kind of short-term memories in neural networks.

Among ANNs specialized in sequence learning, the long short-term memory (LSTM) network is a potential candidate for application to musical sequence learning and production. LSTM networks are RNNs that were introduced by Hochreiter and Schmidhuber in 1997 [5]. They are composed of three layers: an input, a hidden and an output layer. The hidden layer is composed of specialized units called the memory cells (MCs) that are able to store information for long time intervals [5]. Each MCs are part of a higher structure, the memory cell block (MCB). LSTM networks typically are composed of several MCBs, each of which operates on an intrinsic time scale. Gating variables associated with each MCB modulates the information flow through MCs. Thanks to its architecture and learning algorithm, LSTM networks have been shown to be efficient in learning complex sequences and have been applied to speech processing [6] and handwriting recognition [7].

The results obtained on complex sequence learning by LSTM networks, the internal mechanisms of LSTM networks (e.g. the different time scales of each memory cell blocks) and their capacity to bridge long time lags between input events and target signals ultimately make LSTM networks good candidates for the task of musical sequence learning. Moreover, the structure of LSTM networks, although more related to machine learning than biological neural networks, could be linked to the neural circuits of Mozart's brain retrieving the *Miserere*. Indeed, MC states can possibly be linked to neural networks capable of storing information and MCBs to short-term memory operational substructures of the brain, each working on different time scales [8]. For example, slow time scale subnetworks retaining the information about the global song structure (time entries of the different voices or how each musical phrase is ordered) and faster time scale subnetworks that have storing the melody or note transitions within each musical phrase. As a first step, these hypotheses on how musical sequences are

encoded in the brain can be tested with ANNs. However, experimental evidence for such time scale hierarchy has been reported in the neuroscience literature. For example, Kiebel *et al.* reviewed empirical evidence that time scale hierarchy is actually present in the cortex and build a model completely based on this concept. Their findings suggest that their temporal hierarchy model can naturally retrieve and reproduce bird songs [8].

Here, I therefore apply LSTM networks to the problem of musical sequences learning and production. Representation of music in ANNs is consequently a critical step. In physical terms, music is, like sounds, propagation of mechanical vibration in an elastic medium [1] and the more complete representation is consequently the sound spectrogram [9]. However, neurons are not directly coupled with the pressure wave of music propagating in the air. The transduction from mechanical pressure waves to action potentials, the neural communication mechanism, is done by the hair cells of the cochlea [10]. The cochlea is built in a manner that hair cells are frequency tuned (tonotopic representation). To be closer to what neurons are operating with, I chose to represent musical sequences in LSTM networks as pure tones perceived by hair cells, each pitch being associated with one cell in the input and output layers.

The results reported in this master thesis demonstrate that LSTM networks are able to predict every upcoming note from the previous one of simple musical sequences. In addition, a new model will be introduced that transforms trained LSTM networks into autonomous musical sequences composers, the generative model. The model used to learn the simple musical sequence is referred as the input-note-ouput-note (INON) model because both input and output to the network are notes only. While INON LSTM networks combined with the tonotopic music representation are efficient to learn a single musical sequence such as *Frère Jacques*, they fail to capture longer and more complex musical sequences (e.g. Bach's cello suites). To solve this problem, a novel model is presented that relies on the separation of time scales. The separation of time scales model architecture consists of one LSTM network trained to learn the slow time scale structure of the musical sequence that transfers its information to another LSTM network, which is producing the fast time scale structure. The musical sequence is severed into musical patterns of constant length, each of which is associated with unique identifiers. In this new model, the slow time scale LSTM network is then trained to predict pattern transitions, translated in transitions between pattern identifiers, while the fast time scale LSTM network is trained to predict note transitions from all musical patterns, given their corresponding unique identifiers. Furthermore, the musical pattern identifiers could be compared to a compressed memory of musical patterns that is taken advantage of by fast time scale LSTM networks in order to recall the high frequency note-to-note transitions from each of the pattern. This compressed memory of musical patterns is found to be very efficient for pattern recalling when it is taking into account similarities between patterns as the associative memory of Hopfield networks is [11].

Outline Chapter 1 summarizes the properties of LSTM networks and the learning algorithm of LSTM networks is introduced. Chapter 2 describes the tonotopic music representation in ANNs along with a tool to convert any MIDI files to the chosen representation. Chapter 3 presents the results obtained when LSTM networks are trained on the nursery rhyme *Frère Jacques* either in a monophonic or a polyphonic version. Both versions of the song are learned and reproduced by LSTM networks. Effects of the network topology, the learning rate and noise on learning are also discussed. Chapter 4 introduces the separation of time scale model and tests it on musical extracts from Bach's cello suites. Both the LSTM network dedicated to the fast and slow time scales are able to learn their corresponding time scale of Bach's cello suites. In addition, trained slow and fast time scale LSTM networks are transformed to generate their corresponding time scales. Finally, Chapter 5 explores the possibility of novel musical pattern production by trained fast time scale LSTM networks.

1 Long Short Term Memory

The specialized cells of long short-term memory (LSTM) networks have been shown to be able to bridge arbitrary time lags between input signals and their target outputs. LSTM networks were able to solve efficiently the non-trivial tasks that require very long time memory, e.g. the continuous embedded Reber grammar and continuous noisy sequence tasks [12]. LSTM networks have also outperformed other neural networks when applied to handwriting recognition [13], language learning [14] and more recently protein secondary structure classification [15]. Furthermore, they have been found to be able to learn non-Markovian sequences, which makes LSTM networks ideal candidates for learning of musical sequences.

This chapter begins with a short summary of artificial neural networks and present the backward propagation of the error signal learning algorithm. In the second part, I give introduction to the special properties of LSTM networks. Finally, the learning algorithm of LSTM networks (forward and backward passes) is presented and detailed.

1.1 Artificial Neural Networks and Learning by Backward Propagation of the Error

Before introducing LSTM networks, I first summarize the learning mechanisms of standard artificial neural networks (ANNs) [16]. ANNs are typically multilayer networks composed of interconnected artificial neurons called units or cells. Like biological neurons, an artificial neuron j integrates the activation y_i of connected cells with a strength defined by the weight of the connection w_{ij} from neuron i to neuron j. Input units receive external stimulation that is then propagated along the network through the connections (forward pass). The objective of an ANN is to match the external stimulation signal to a target output through sequential update of the connection weights between each unit. The weight updates are made through backward propagation of the error signal observed at the output cells (backward pass). Recurrent neural networks (RNNs) are a particular class of ANNs where neurons are recurrently connected in addition to feedforward connections. Use of recurrent connections enables the network to use its own internal state to affect its output.

1.1.1 Forward pass

The input net_j to non-input neurons j in an ANN is given by the weighted sum of previous neuron activations y^i (equation 1.1). The input neurons receive the external input that is propagated along each of the connected units. The activation of each neurons y^j is then given by equation 1.2 where f is the activation function and is in general a non-linear monotonic and differentiable function such as the logistic function.

$$net_j = \sum_i w_{ij} y^i \tag{1.1}$$

$$y^j = f(net_j) \tag{1.2}$$

1.1.2 Backward pass

The error function is given by equation 1.3, where t^k is the target activation of output cell k and y^k the actual output unit activation. The gradient descent method gradually minimizes the error function by changing the connection weights with respect to the partial derivative of the error function $\frac{\partial E}{\partial w_{ji}}$ (equation 1.4).

$$E = \frac{1}{2} \sum_k ||t^k - y^k||^2 \tag{1.3}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y^j} \frac{\partial y^j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \tag{1.4}$$

The terms in the right hand side of equation 1.4 can be simplified. Indeed, the last term is y^i and the middle term is simply the derivative of the activation function evaluated at net_j. The first term is easy to evaluate for output neurons ($j = k$) and $\frac{\partial E}{\partial y^k} = y^k - t^k$. However, if j it is not an output neuron, the derivation is less trivial and for non-output neurons, due to the chain rule, one needs all higher derivatives $\frac{\partial E}{\partial y^j}$ until an output unit is reached, hence the name *backward propagation of the error*. The final equation for the partial derivative of the error with respect to the weights is given by equations 1.5 and 1.6. The local weight update to approach the minimum of the error function is given by equation 1.7 where α is the learning rate.

$$\frac{\partial E}{\partial w_{ij}} = \delta_j y_i \tag{1.5}$$

$$\delta_j = \begin{cases} f'(net_j)(y^k - t^k) & \text{for output neurons } (j = k) \\ f'(net_j)\sum_l \delta_l w_{jl} & \text{otherwise} \end{cases} \tag{1.6}$$

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}} \tag{1.7}$$

1.2 LSTM Network and Units

LSTM networks are RNNs composed of three layers. The input layer receives the external inputs and forwards information toward the hidden layer and output layer. In contrast to conventional RNNs, in LSTM networks, the hidden layer is entirely composed of memory cell blocks (MCBs). These special subunits are the central feature of LSTM networks and are described in more detail in section 1.2.2. A MCB is composed of a fixed number of separated memory cells (MCs) and three gate units (input, output and forget gates). The unique hidden layer of traditional LSTM networks has recurrent connections in addition to feedforward connections from input cells and toward output units.

LSTM networks are also trained using a gradient descent algorithm applied on the error signal at the output layer. The error signal is backward propagated through the hidden layer and its components to compute the weight changes needed to gradually approach the optimal solution. However, because MCs have their own activation function and mechanisms that differs from the model described in the previous section, the derivation of the error signal is different for those cells (see section 1.3).

The main objective of LSTM networks is to be able to store information relative to the network state of previous times until it is no longer necessary, thereby allowing to learn sequences with arbitrary long time lags. This is done through explicit gating of information flow and the inner cell state of each memory cell, which constitutes the memory of the network. More importantly, the learning algorithm enables the possibility to keep a constant flow of the error signal, which is crucial to solving long time lag problems [5].

1.2.1 Constant Error Flow

The conventional backward propagation through time (BPTT) algorithm (section 1.1) suffers from exponential blow up or vanishing of error signals in time [17]. The weight update being related with the error signal, if the flow of error signals grows exponentially, weights are updated with high values making them oscillate and eventually diverge. On the other hand, if the error signal is vanishing over time, learning to bridge long time lags is not possible. LSTM networks are addressing this problem by introducing the central feature of LSTM: the constant error carousel or inner cell state of MCs [5].

From the Hochreiter analysis [18], the local error back flow of a conventional BPTT algorithm for a single recurrently connected unit is given by equation 1.8. To force the error flow to a constant value across time ($\delta_j[n] = \delta_j[n+1]$) the equation 1.9 should be respected. The update rule of the state s_j from MC j is satisfying this condition by setting w_{jj} to 1 and $f(x) = x$ (equation 1.10).

$$\delta_j[n] = f'(net_j[n])\delta_j[n+1]w_{jj} \tag{1.8}$$

$$f'(net_j[n])w_{jj} = 1 \tag{1.9}$$

$$s_j[n+1] = f(net_j[n+1]) = f(w_{jj}s_j[n]) = s_j[n] \tag{1.10}$$

In other words, the inner cell state of MC s_j remains constant to be able to keep tract of the error signal. However, MCs are obviously not only connected only to themselves but to other cells too and truncation of the error signal leaving MCs and gates should be applied [5] (see section 1.3.2). The gates are introduced to be able to modulate the MC states and to resolve input and output weight conflicts (see update equation 1.21 for MC states).

1.2.2 Memory Cell Block

The fundamental structure of LSTM networks is the memory cell block (MCB). Figure 1.1 is a block diagram that summarizes the different components of a MCB. The hidden layer is entirely composed of MCBs, each of which contains several memory cells (MCs). Three gates are associated to each MCB. Their role is to modulate the information flow through the MCs. The gates and MCs receive input from the MCs in *every* blocks in addition to forward connections from all input units. The input gate (in) modulates delivery of information from MC inputs to their inner cell states. The output gate (out) modulates the access or sensitivity of connected cells to the information stored in MCs. The forget gate is an extension of original LSTM networks [5] introduced by Gers *et al.* [12]. The forget gate (φ) role is to discard useless memory from the inner cell states of MCs and has been shown to facilitate continual predictions [12].

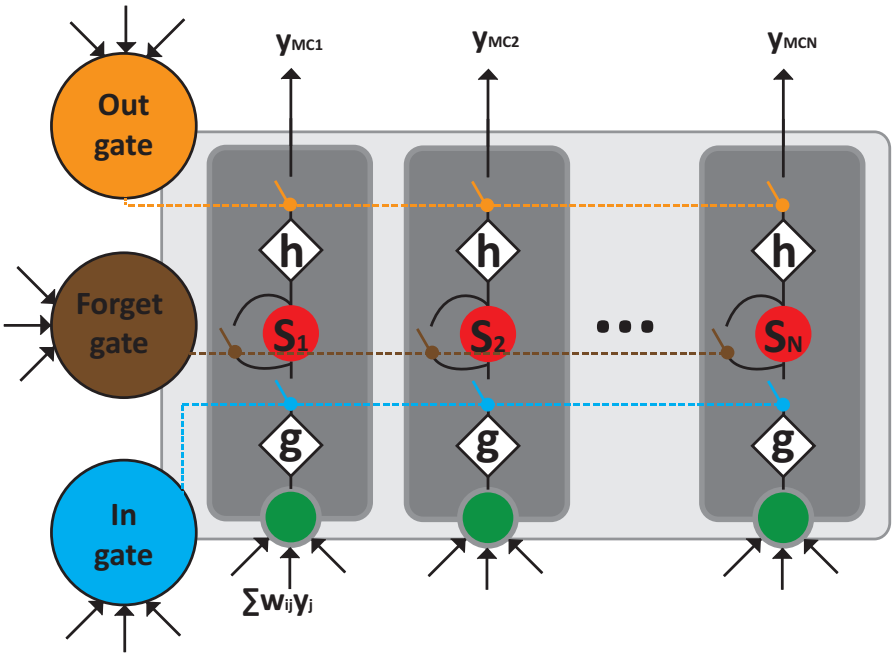


Figure 1.1 – Diagram of one memory cell block. S is the inner cell state, also called the constant error carousel, of each memory cell (see text for details). h and g are non–linear transfer functions common to all memory cells.

1.2.3 Gates

Here, I introduce and motivate gating of information flow within MCBs, while the mathematical formulation is given later in the learning algorithm section (section 1.3).

In gates The network uses the input gate to decide when to keep or override the information stored in memory cells. To avoid input weight conflict, the input gate controls the error flow to the MCs. The input weight conflict comes from the fact that learning will make the weights of the synapses from input units toward MCs participate in storing the new input but also protecting the information stored in MCs.

Out gates The output gate allows the network to decide when output units and other connected structures can access a set of MCs and when to prevent output units from being perturbed by these cells. Output weight conflicts can be prevented with the output gate by controlling the error flow from output units toward a set of MCs. The output weight conflict is the fact that learning will make the connections from MCs toward output units participate in accessing information stored in the MCs along with protecting output units from being perturbed by these information when they are not necessary.

Forget gates The original LSTM as presented by Hochreiter and Schmidhuber in 1997 [5] encounter issues when learning continuous streams of sequences. Indeed, the memory cell states are increasing linearly with time if there is no external reset of their values. The outputs

of the memory units are then saturated. Effects of this saturation are blocked incoming error signals and output equal to the value of the output gate. To *learn to forget*, Gers and his colleagues proposed the concept of forget gate [12]. These gates learn when to reset the memory cell states, consequently replacing an external reset and preventing saturation.

Using the gating of information flow and a constant flow of error signal, learning long time structures is possible. The figure 1.2 taken from Graves work [19] represents this effect. Indeed, when the first input is stored in the inner cell state, it remains in memory for as long as the forget gate is open and is not changed for as long as the input gate is closed. The output gate allows the output units to access this stored information whenever it is needed. By doing so, the network could theoretically bridge arbitrary time lags between input events and their target output signals.

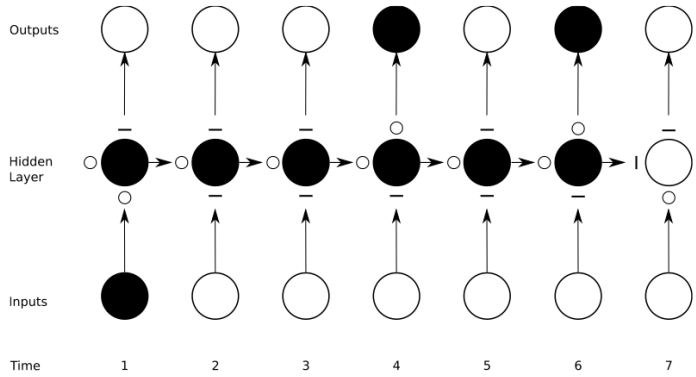


Figure 1.2 – Keeping information for long time lags is possible with memory cells. Open gates are schematized with circles and closed gates with lines. The bottom gate is the in gate, up is the out gate and on the left side is the forge gate.

1.3 LSTM Learning Algorithm

The algorithm of the forward pass (propagation of unit activations) and backward pass (back-propagation of error signals and weight updates) is described in this section. For all equations, the index k ranges over output units, j over memory cell blocks, v over memory cells in the block j and i over input units. in stands for in gates, out for out gates φ for forget gates and c for MCs. The mathematical equations given in this section are all adapted from the ones described in the paper from Gers *et al.* [12].

1.3.1 Activation Functions

The activation of gates and cells are computed by filtering the linear net input to the current units with a non linear sigmoid. Three of these sigmoids are used for this implementation of the long short term memory algorithm.

$$f(x) = \frac{1}{1+exp(-x)} \tag{1.11}$$

$$g(x) = \frac{4}{1+exp(-x)} - 2 \tag{1.12}$$

$$h(x) = \frac{2}{1+exp(-x)} - 1 \tag{1.13}$$

f is the activation function of the output units and gates. It is the logistic function and its output is in the range [0 1]. g is squashing the network input of memory cells into the range [-2 2] and h is squashing the inner cell states of MCs in the range [-1 1].

1.3.2 Forward Pass

The forward pass is computed at each time step. It consists in propagating the information from the input layer toward the output layer and computing the partial derivatives needed for the weight updates in the backward pass.

Unit Activations

The input unit y^i_{input} activations at time t are determined by the external input at this time.

$$y^i_{input}[n] = \text{external input}^i[n] \tag{1.14}$$

At the beginning of the forward pass, the activations of the previous time step is set to the current activities as a new step as begun (equation 1.15) . This is done after having set the

input unit activations to allow for easier computation of the information propagation.

$$y[n-1] = y[n] \tag{1.15}$$

The net input net_{in_j} and activation y^{in_j} of the in gate j are given by equations 1.16 and 1.17. In all activation equations, the $bias$ terms are additional inputs to the units that are typically fixed when the network is initialized (see **Network initialization** in section 3.1). Similar equations apply to forget gates (equations 1.18 and 1.19) and output gates (equations 1.22 and 1.23).

$$\begin{aligned}
net_{in_j}[n] &= \sum_m w_{in_j m} y^m[n-1] + w_{in_j}^{bias} y^{bias} && (1.16)\\
y^{in_j}[n] &= f(net_{in_j}[n]) && (1.17)\\
net_{\varphi_j}[n] &= \sum_m w_{\varphi_j m} y^m[n-1] + w_{\varphi_j}^{bias} y^{bias} && (1.18)\\
y^{\varphi_j}[n] &= f(net_{\varphi_j}[n]) && (1.19)
\end{aligned}$$

The net input of memory cells and their inner states are given by equations 1.20 and 1.21. The inner cell state s is additive with time and forget gates act as a *reset* function. The inner cell state is modulated when the in gate is open and depends on the memory cell input squashed by the sigmoid g.

$$\begin{aligned}
net_{c_j^v}[n] &= \sum_m w_{c_j^v m} y^m[n-1] && (1.20)\\
s_{c_j^v}[n] &= y_{\varphi_j}[n] s_{c_j^v}[n-1] + y^{in_j}[n] g(net_{c_j^v}[n]) && (1.21)\\
net_{out_j}[n] &= \sum_m w_{out_j m} y^m[n-1] + w_{out_j}^{bias} y^{bias} && (1.22)\\
y^{out_j}[n] &= f(net_{out_j}[n]); && (1.23)
\end{aligned}$$

The sensitivity of the output units to the memory cell activations is directed by the out gate activation and depends on the memory unit inner state $s_{c_j^v}$, which is squashed by the sigmoid h (see equation 1.24). Finally, the output unit activations are given by equations 1.25 and 1.26.

$$\begin{aligned}
y^{c_j^v}[n] &= y^{out_j}[n] h(s_{c_j^v}[n]) && (1.24)\\
net_k[n] &= \sum_m w_{km} y^m[n] + w_k^{bias} y^{bias} && (1.25)\\
y^k[n] &= f(net_k[n]) && (1.26)
\end{aligned}$$

Partial Derivatives

The partial derivatives needed to minimize the error function are truncated to ensure the constant error flow through the inner cell states of MCs by setting the error flow leaving MCs and gates to 0 (for detailed description of the truncation see [5]). Therefore, the remaining partial derivatives after truncation are the ones involving the inner cell state of each MC and are presented below.

The remaining partial derivatives of MCs are given by equation 1.27, while the input gates and forget gates partial derivatives are given by equations 1.28 and 1.29.

$$\frac{\partial s_{c_j^v}}{\partial w_{c_j^v m}}[n] = \frac{\partial s_{c_j^v}}{\partial w_{c_j^v m}}[n-1]\, y^{\varphi_j} + g'(net_{c_j^v}[n])\, y^{in_j}[n]\, y^m[n-1] \tag{1.27}$$

$$\frac{\partial s_{c_j^v}}{\partial w_{in_j m}}[n] = \frac{\partial s_{c_j^v}}{\partial w_{in_j m}}[n-1]\, y^{\varphi_j} + g(net_{c_j^v}[n])\, f'(net_{in_j}[n])\, y^m[n-1] \tag{1.28}$$

$$\frac{\partial s_{c_j^v}}{\partial w_{\varphi_j m}}[n] = \frac{\partial s_{c_j^v}}{\partial w_{\varphi_j m}}[n-1]\, y^{\varphi_j} + s_c^v[n-1]\, f'(net_{\varphi_j}[n])\, y^m[n-1] \tag{1.29}$$

1.3.3 Backward Pass

The backward pass consists in minimizing the output unit errors using a gradient descent algorithm. The error signal at each output unit is backward propagated toward the lower layers in order to update each weight (see section 1.1).

Errors and δs

The error at time t for each output unit is given by the difference between the current target of the output unit and its activation at the same time step (equation 1.30). The target t^k for each output unit k is set at each time step and depends on the external input; it is the value that should take the output unit given the input. The output units, out gates and inner cell states δs are given by equations 1.31, 1.32 and 1.33 respectively.

$$e_k[n] = t^k[n] - y^k[n] \tag{1.30}$$

$$\delta_k[n] = f'(net_k[n])\, e_k[n] \tag{1.31}$$

$$\delta_{out_j}[n] = f'(net_{out_j}[n]) \sum_{v=1}^{S_j} h(s_{c_j^v}[n]) \sum_k w_{kc_j^v} \delta_k[n] \tag{1.32}$$

$$e_{s_{c_j^v}}[n] = y^{out_j}[n]\, h'(s_{c_j^v}[n]) \sum_k w_{kc_j^v} \delta_k[n] \tag{1.33}$$

Weight Update

For all equations below, the weight update is applied with the learning rate α. Traditional gradient descent based weight update is applied to weights of connections toward output units and out gates as showed in equations 1.34 and 1.35.

$$\Delta w_{km} = \alpha \delta_k[n] y^m[n] \tag{1.34}$$

$$\Delta w_{out_j m}[n] = \alpha \delta_{out_j}[n] y^m[n-1] \tag{1.35}$$

Weights of connections toward in and forget gates are updated based on the error signal back propagation and the partial derivatives for all MCs in MCB j (equations 1.36 and 1.37). Finally, weights of connections toward memory cells are updated based only on the local partial derivative as described in equation 1.38.

$$\Delta w_{in_j m}[n] = \alpha \sum_{v=1}^{S_j} e_{s_j^v}[n] \frac{\partial s_{c_j^v}}{\partial w_{in_j m}}[n] \tag{1.36}$$

$$\Delta w_{\varphi_j m}[n] = \alpha \sum_{v=1}^{S_j} e_{s_j^v}[n] \frac{\partial s_{c_j^v}}{\partial w_{\varphi_j m}}[n] \tag{1.37}$$

$$\Delta w_{c_j^v m}[n] = \alpha \frac{\partial s_{c_j^v}}{\partial w_{c_j^v m}}[n] \tag{1.38}$$

Learning modes The learning mode consisting in update of the weights after each prediction or timestep n is called *online* learning. If the weight update is applied after a series of predictions, the learning mode is called *batch*. In the case of musical sequences learning, the network performances will be evaluated after a complete forward and backward pass on all the sequence (a trial). To evaluate correctly the network performances on the sequence, the weights should not change during a trial. It is therefore chosen the batch learning mode for musical sequences learning.

1.4 C++ Implementation

The LSTM model has been completely re-implemented in C++ in a object-oriented and flexible fashion. The main classes are listed below.

Cell The Cell super class is a generic class for every neurons in the ANN. Its attributes are the current and previous activations, the sum of weighted inputs and the bias.

Input cell The Input cell subclass inherits from Cell and possess only a function to externally set the input cell activations.

Output cell The output cell subclass inherits from Cell. Its attributes are the target output value, the δ_k and its activation function. Its public methods encompass the activation and the δ_k update functions.

Memory cell The Memory cell subclass inherits from Cell. Its attributes are the current and previous inner cell state, the local error and the two sigmoids needed to compute the activation of the MC. Its public methods are the inner cell state update given the activations of the forget and in gates, the activation update given the out gate activation and the local error update.

Memory cell block The Memory cell block class attributes are a collection of pointers to MCs and one pointer to each of the three gates.

Gate The Gate subclass inherits from Cell. Its attributes are the δ needed for the backward pass and the activation function. Its public methods encompass activation and $/delta$ updates.

Synapse The Synapse attributes are the synaptic weight, the pre and post synaptic cells and the partial derivatives needed for its weight update. A weight update public method is implemented in the Synapse class.

Sigmoid The Sigmoid class attributes are the three value needed to represent a sigmoid: the multiplication coefficient, the steepness and the offset. Its public methods are calculations of the activation and its derivative given the input.

Network The Network class reassembles every units and their connections in polymorphic collection of pointers. It is calling and computing the forward and backward passes.

2 Music Representation in Artificial Neural Networks

Music is a complex temporal sequence and consequently a candidate to be learned by long short-term memory (LSTM) networks. In order to be able to feed the network with a musical sequence, a specific encoding of the music should be elaborated. To characterize all information carried in a live performance of even a single instrument is a complex task. Indeed, depending on the level of representation, one should be able to encode, among others, the pitch, the tempo, the beat, the duration, the amplitude, the timbre of each note [20]. When the music is polyphonic or orchestral, the complexity increases further. The objective of representing music in artificial neural networks is to find a compromise that keeps as much information as possible but remains simple enough to be learned.

In the current chapter, I start by a review about musical sequence representation and then introduce a tool that allows to convert MIDI files in the proposed LSTM network representation.

2.1 Review of Music Representation

The bibliography for music representation encompasses the work of Todd [21], Eck and Schmidhuber [22], Franklin [23], Oliwa and Wagner [24], Boulanger-Lewandowski and Dannenberg [20]. Todd included in addition to pitch and duration the information about note beginning enhancing the possibility to discriminate between repeated notes and a single sustained one [21]. Franklin made effort in making the representation as simple as possible and proposed a representation that uses a single scale of 12 half tones or even a compressed version which he called the *Circles of Thirds* that compresses the 12 half tones scale into 7 bits, while keeping a certain musical meaning Franklin [23]. The representation of Eck and Schmidhuber [22] is of great interest for the current work because they used LSTM networks to learn blues chords and melodies in order to do blues improvisation. In their representation, they separated the chords from the melody in the sense that they first learn the chord structure and then, given the chord, the melody is displayed. This could make sense as they argue that in blues music, it is often the case that jazz musician improvise their way based on chord grids. However, this one-way relation between chords and melody does not hold for classical music and especially for monophonic musical sequences where no chords are implicitly present at each time. In addition, they restricted the possible outputs to the pentatonic scale, which is a way to ensure that the output will be pleasant to listen to. However, the main achievement they made that is promising for the current study, is that they were able to mimic and reproduce in slightly different ways famous blues melodies that were used as a training set. Their representation of music in LSTM networks was similar to the one from Todd [21]. Boulanger-Lewandowski used the piano roll representation for unconstrained polyphonic music [25]. This representation is relevant for a better generalization capability because it allows a two way interaction between chords and melody. It will be used as a starting point for the music representation used in this study. In addition, this representation can be linked with the neurophysiology of the hearing system, which is organized in a tonotopic manner. In particular, auditory neurons in the cochlea are tuned in a frequency specific manner [10]. The piano roll representation, where each pitch is represented by one cell of an ANN, either active or inactive, is consequently highly generalizable since it is unconstrained and closer to biology than other representations.

In the following section I will introduce how I am able to represent any musical instrument digital interface (MIDI) files in LSTM networks.

2.2 From MIDI to LSTM

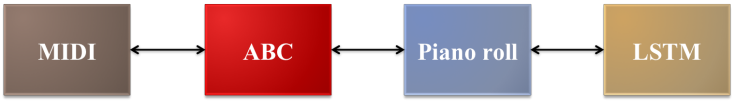


Figure 2.1 – Flow chart of the musical sequences conversion from MIDI file to LSTM representation.

MIDI is a widely used format of music encoding and multiple music pieces in MIDI format (.mid) can be freely downloaded from internet. MIDI files principally contains the information about when and for which duration a pitch is active and is consequently an ideal candidate as dataset for the piano roll, or tonotopic, representation of music in ANNs. Furthermore, MIDI files can be easily visualized in the musical notation. The Figure 2.2 shows the musical notation obtained by reading the *Frère Jacques* MIDI file with the free software MuseScore.

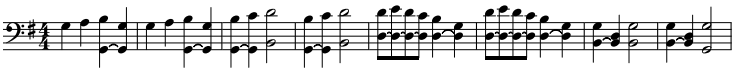


Figure 2.2 – The musical notation of the song *Frère Jacques*.

To convert MIDI files into the tonotopic representation that will be used by LSTM networks I first use a MIDI to abc converter (download and documentation at http://abc.sourceforge.net/abcMIDI/) as depicted on Figure 2.1. This converter extract the MIDI information about pitches and note durations and convert them into a text file (see Figure 2.3). The header include the converted MIDI file name (T), the time signature of the song (M), the note basic duration (L), the speed in beats per minute (Q) and the key (K). Each bar are separated by the '|' tag and pitches are associated to the corresponding letter in the english musical notation. The ',' indicates on which octave the pitch stands and the number after each note stands for the note duration as a factor of the specified basic duration in the header (L).

The ABC notation is then parsed to extract a vector in the piano roll representation for each sixteenth note (second step in Figure 2.1). The piano roll representation is a vector of size 84 that ranges all possible notes (tempered) from C1 to C8. When a note is active, its corresponding index in the piano roll representation is activated (value of 1). In addition, the converter from ABC to piano roll representation transposes every song in the common tonality C Major. For each musical sequence to be learned by LSTM networks, the LSTM representation cuts off every non-displayed note in the training set from the piano roll representation (last step in Figure 2.1). This is not a mandatory step, however, as the input units that are dedicated to a note that is never active would always be null in the training phase, their contributions to the network would always be also be null. In order to reduce the computation time and remove useless connections, it is chosen to apply this processing step. It could also be validated against biology, indeed, the neurons that are tuned to frequencies that are never displayed in a repertoire of sequences would never be activated and can thus be discarded from the neural circuit that is being studied. Figure 2.4 shows the input and target of the neural network for the

```
X: 1
T: from NurseryRhymes/frere_jacques.mid
M: 4/4
L: 1/8
Q:1/4=120
K:G % 1 sharps
%%MIDI program 24
G,2 A,2 [B,2G,,2-] [G,2G,,2]| \
G,2 A,2 [B,2G,,2-] [G,2G,,2]| \
[B,2G,,2-] [C2G,,2] [D4-B,,4-]| \
[B,2G,,2-] [C2G,,2] [D4-B,,4-]| \
[DD,-][ED,-] [DD,-][CD,] [B,2D,2-] [G,2D,2]| \
[DD,-][ED,-] [DD,-][CD,] [B,2D,2-] [G,2D,2]| \
[G,2B,,2-] [D,2B,,2] [G,4-B,,4-]| \
[G,2B,,2-] [D,2B,,2] [G,4-B,,4-]| \
```

Figure 2.3 – The abc notation of the song *Frère Jacques*.

harmonized nursery rhyme *Frère Jacques* and the prelude of the first cello suite from J.S. Bach. Each coordinate in the y axis represent a pitch or its corresponding input/output cell in the network representation. It should be mentioned the pitch order in the LSTM representation (y axis) is not the same as the order in which pitches appears on the frequency scale, rather the order in which each pitch appears in the converted musical sequences.

Note duration The information about note length are lost using this representation. There are several possibilities to introduce this information such as adding the information about note beginning or end [21]. However, this only works for monophonic representations. If it is chosen to have this information in a polyphonic song, each voices should have their own network and connections between the networks could then be constructed to link them. The main goal of this study being to predict and compose note transitions, note duration information is discarded.

Note For the rest of the study the term *note* represents the vector that characterizes the active and inactive pitches in the network representation for a duration of a sixteenth note.

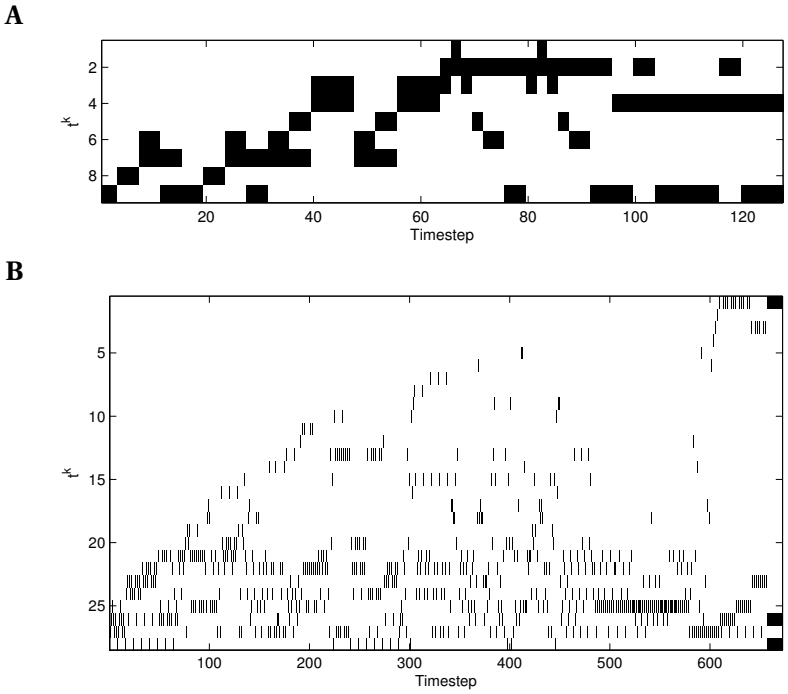


Figure 2.4 – The LSTM representation of (**A**) the song *Frère Jacques* in the polyphonic version and (**B**) the prelude of Bach first cello suite.

3 Learning a Single Musical Sequence – The INON Model

The objective of the current work is to produce music with LSTM networks. To approach this, I aim to extract production rules from musical examples. As a first step in this direction, I chose to learn the well known nursery rhyme *Frère Jacques* to evaluate the capacity of LSTM networks to learn music in the chosen representation and to find optimal network and learning parameters. Two versions of the song are fed to the network; one monophonic with the melody only and one polyphonic including harmonization of the song. Although the song has a quite simple structure, it is non Markovian: after many given notes there is not a unique possible transition. The correct transitions will rather depend on the history of notes played, i.e. the position of the notes in the musical sequence.

The chapter starts by introducing the model architecture and learning paradigm along with two operating modes: the predictive and the generative mode. I then show the results obtained on both version of the *Frère Jacques* song.

3.1 Network Architecture and Learning

Here, I first introduce the architecture of INON LSTM networks used for single sequence learning. Importantly, the network can operate in two modes, the predictive and the generative modes, with slightly different architectures. However, for both modes, the input and output of the ANN are notes in the network representation, hence I call this model the input-note-output-note (INON) model.

The training phase (predictive mode) consists in adapting the connection weights until the network has learned to predict every upcoming notes from the last one. On the other hand, the production phase (generative mode) consists in making a *trained* LSTM network to autonomously *re*produces a musical sequence thanks to the learned and fixed connection weights.

Network initialization Initialization of a LSTM network applied to learning and production of musical sequences involves several steps described below. It is created the same number of input units as the number of pitches that are present in the network representation of the musical sequence to be learned. The hidden layer units are created with B MCBs containing M MCs each and the three gates, where B and M are free parameters. Learning involves predicting the upcoming note from the notes history, hence each output unit also represents a pitch. The network is fully connected with connections from each input unit toward all MCs and gates, connections from each input unit toward all output units and recurrent connections from each memory cells toward all memory cells (including itself) and gates. No connection from gates toward other cells are set. Each connections have an initial random weight in the range [-0.2 0.2]. Bias activations from each biased cell are fixed to 1 (see equations for unit activations in the forward pass mathematics, section 1.3.2). The output units each have a random weight bias in the range [-0.2 0.2].

Gradual increase of gate biases To have an initial decreasing ability through memory cell blocks to, first, influence memory acquisition, second, forgetting, and finally, delivering stored information to the output units, the fixed biases across MCBs are not chosen randomly. Forget gate weight biases across MCB are serially increased: the forget gate of the first MCB has a weight bias of 0.5, the second 1, the third 1.5, ... By doing so, the forget gate of the last memory cell will initially be always open, making harder for the MCs of this MCB to forget memories about the sequence than for the MCs of the first block. Similarly, a decreasing weight bias for the in and out gates starting from -0.5 and decreasing with a step of 0.5 is applied. The first MCB will learn quicker to open and close its in and out gates than the last one which is initially biased toward having the in and out gates always open. By these mechanisms, it is ensured serial activations of MCs during the initial phase of training [5].

3.1.1 Learning – Predictive Mode

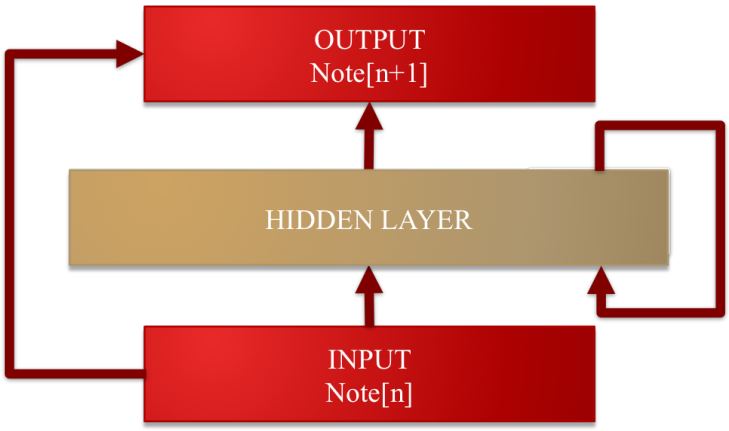


Figure 3.1 – INON Network architecture of the predictive mode.

To learn a single musical sequence the network is fed with this sequence until it can correctly predict it. The error is taken as the difference between the expected (target) and observed (output) output unit activations (see equation 1.30). The single sequence is looped so that the target note at the last time step is the first one. By doing so, the network will theoretically learn to generate an infinite sequence of the same song. The network state is consequently never externally reset. The main task of the network is then to learn the transitions between each note along with the note durations. The pseudocode of the learning algorithm is given in Algorithm 1.

In the training phase, the network is fed with a note in the song and should predict the upcoming one. Again, as both input and output are note in the network representation, this model is called the input-note-output-note (INON) model. In order to assess for non Markovianities in the sequence, the network has to find a way to keep track of useful information about the song structure. The long short term memory network can store these information into its memory cell states as explained in section 1.2.3.

Batch learning is applied, consequently weight updates are accumulated across the trial and the update is made at the end of the trial (see Algorithm 1). One trial is considered as the prediction of each and every note in the dataset. This allows to evaluate the network performance during training. If online learning would be applied, evaluation of the performances is needed to be performed at every time step or, alternatively, in a separate evaluation sequence with fixed weights.

Note selection from network output To infer if the corresponding pitch should be active, the output unit activations, real valued numbers comprised between 0 and 1, are filtered with a step function that projects every activations below 0.5 to 0 and every activations over 0.5 to 1 (see equation 3.2). Applying this filter allows to get a note both as an input and as an output of

the network as seen on the Figure 3.1.

$$\text{pitch}_k[n] = \chi(y_k[n]) \tag{3.1}$$

$$\chi(y_k[n]) = \begin{cases} 1 & \text{if } y_k[t] > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

$$\text{Accuracy}(T) = \frac{C(T)}{N} \tag{3.3}$$

Accuracy A prediction is then considered correct in the predictive mode if the output note has the exact same pitches activated as the next input note, the target. The accuracy is taken as the percentage of correctly predicted note in a given musical sequence. Equation 3.3 gives the mathematical formulation of the accuracy where T is the current trial, N the total number of notes in the sequence and C the number of correctly predicted note.

3.1.2 Reproducing – Generative Mode

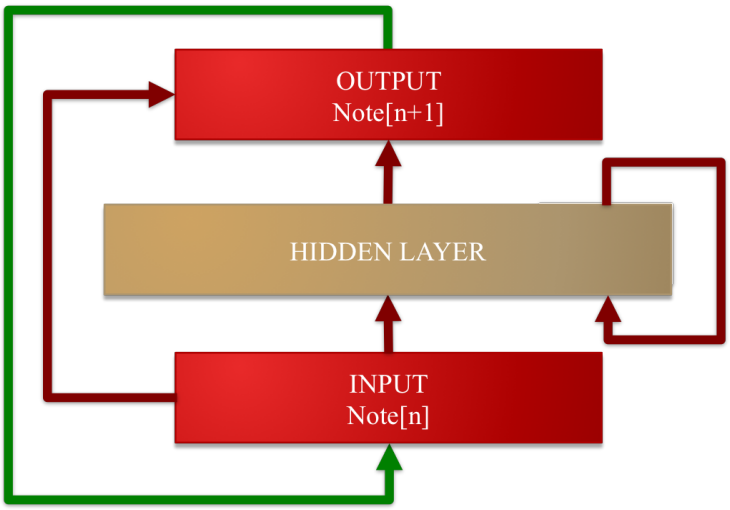


Figure 3.2 – INON Network architecture of the generative mode.

In the generative mode, the output of the network, the predicted upcoming note at the current time step, becomes the input of the next time step (see green arrow on Figure 3.2). Applying this architecture to a network that has correctly learned to predict every note based on sequence history would theoretically result in a network independent of any external information about the song that generates the learned musical sequence.

Data: song in LSTM representation
Result: Connection weights updated to predict the song
Network initialization;
while *Accuracy(T) < 1 (equation 3.3)* **do**
    T=T+1;
    for *each note n in the song* **do**
        Forward pass
        External input: $y^i_{input}[n] = note^i[n]$;
        Roll over: $y^i[n-1] = y^i[n]$;
        Update forget and input gates activations: equations 1.16-1.19;
        Update memory cell states: $s_{c^v_j}[n] = y_{\varphi_j}[n] s_{c^v_j}[n-1] + y^{in_j}[n] g(net_{c^v_j}[n])$;
        Update out gate activations: equations 1.22 and 1.23;
        Update memory cell activations: $y^{c^v_j}[n] = y^{out_j}[n] h(s_{c^v_j}[n])$;
        Update output unit activations: equations 1.25 and 1.26;
        Update memory cell, in and forget gates partial derivatives: Equations 1.27-1.29;
        Backward pass
        Update output errors: $e_k[n] = t^k[n] - y^k[n]$ with $t^k[n] = note^k[n+1]$;
        if $\chi(y^k[n]) = t^k[n]$ **then**
            Prediction n is correct
        end
        Update output unit δs: $\delta_k[n] = f'(net_k[n]) e_k[n]$;
        Update out gate and memory cell δs:
        $\delta_{out_j}[n] = f'(net_{out_j}[n]) \sum_{v=1}^{S_j} h(s_{c^v_j}[n]) \sum_k w_{kc^v_j} \delta_k[n]$ and
        $e_{s_{c^v_j}}[n] = y^{out_j}[n] h'(s_{c^v_j}[n]) \sum_k w_{kc^v_j} \delta_k[n]$;
        Compute and store local weight change $\Delta w_{ji}[n]$ for every connections: Equations 1.34-1.38;
    end
    if *Accuracy(T) < 1* **then**
        Update weights: $w_{ji}[T+1] = w_{ji}[T] + \sum_{n=0}^{N} \Delta w_{ji}[n]$;
    end
end

Algorithm 1: Learning a single musical sequence with the INON model

3.2 Monophonic Musical Sequence

As mentioned, the first musical sequence to be learned by LSTM networks is the monophonic melody of the *Frère Jacques* song. In this section, it is showed that the long short term memory network can learn the sequence using the predictive mode and how the accuracy of the network changes with respect to time.

3.2.1 Predictive Mode

Topology				**Learning rate**	**Accuracy threshold**
Input units	MCB	MCs	Output units		
7	4	7	7	0.01	100%

Table 3.1 – *Frère Jacques monophonic* Fixed parameters, if not mentioned, for the monophonic *Frère Jacques* experiments

The network is composed of 4 memory cell blocks with 7 memory cells each. Training is stopped when each of the output units are closer to the target state (error is less than 0.5) for one complete trial (presentation of all notes in the song). It corresponds to an accuracy of 100%. After training, the network is tested on the whole musical sequence and performances are evaluated in term of accuracy, mean square error over all output units and the trial number needed to learn the note transitions of the song. The monophonic version of the song is learned

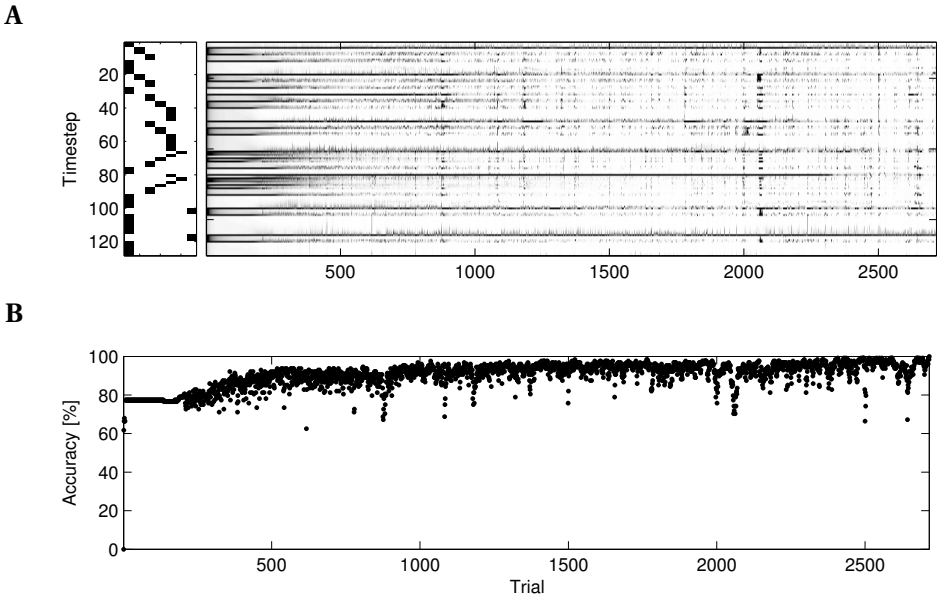


Figure 3.3 – *Frère Jacques monophonic* **A**. Left: musical sequence to be learned. Right: MSE at each predicted note over time during the training phase. Black indicates the maximum MSE. **B**. Accuracy at each trial.

by the INON predictive mode in the sense that all note transitions are correctly predicted at the end of the training phase (Figure 3.3). The network quickly learns to continuously reproduce the last note at the next time steps and then begin to learn the transitions between notes as seen on Figure 3.3 **A**. It is interesting to note hat during learning transitions compete with each other and a network that has learned the song could de-learn it if training is not stopped at the right time (Figure 3.4). Alternatively, a *cooling* scheme could be employed as the α-decay LSTM from Gers *et al.* [12].

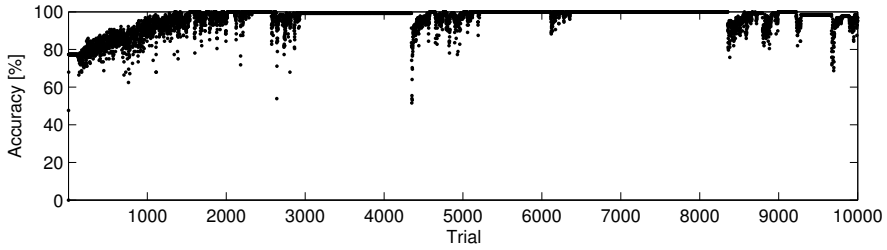


Figure 3.4 – *Frère Jacques monophonic* Accuracy at each trial for a retrain network without stopping.

3.2.2 Generative Mode

The generative mode applied to INON LSTM networks that have been trained on the monophonic version of *Frère Jacques* is able to completely and periodically reproduce the learned musical sequence. More detailed results for the generative mode are presented in the next section.

3.3 Polyphonic Musical Sequence

After showing that LSTM networks are able to learn the song melody using the INON predictive mode, it is inquired if the chosen music representation could also be used for a more general case with polyphonic and monophonic predictions. In this section, the effects of several network parameters on the training performances are studied. Finally, the INON generative mode is validated on the polyphonic version of *Frère Jacques*.

3.3.1 Predictive Mode

Topology				**Learning rate**	**Accuracy threshold**
Input units	MCB	MCs	Output units		
9	4	9	9	0.01	100%

Table 3.2 – *Frère Jacques polyphonic* Fixed parameters, if not mentioned, for the polyphonic *Frère Jacques* experiments

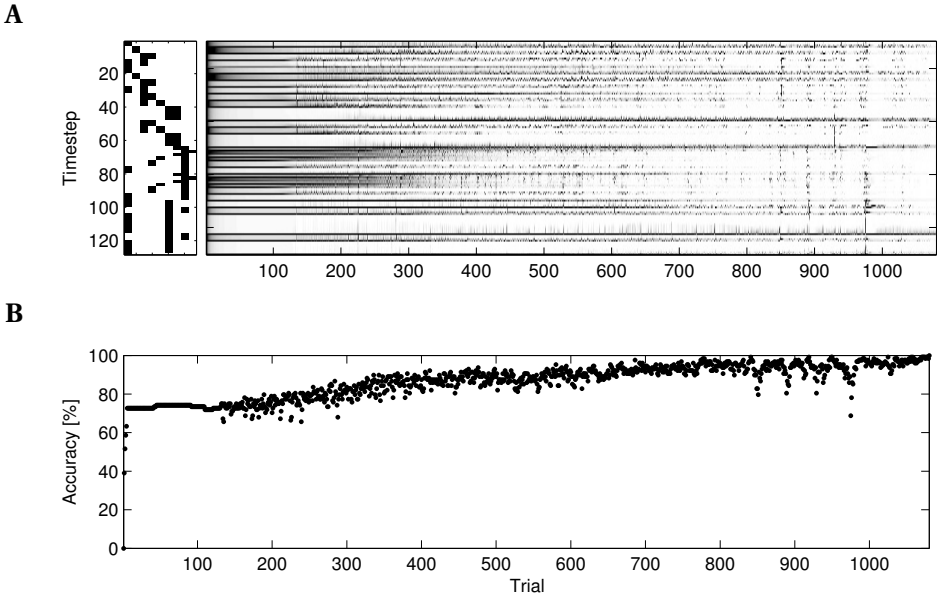


Figure 3.5 – *Frère Jacques polyphonic* **A**. Left: musical sequence to be learned. Right: MSE at each predicted note over time during the training phase. Black indicates the maximum MSE. **B**. Accuracy at each trial.

The Figure 3.3 shows that learning the polyphonic version of the song is possible within 1100 trials using the network parameters described in Table 3.2. The proposed music representation and note selection can therefore be used to learn polyphonic music with LSTM networks. It can be observed that the accuracy evolution over time is less noisy than for the monophonic version. This observation could arise from the fact that harmonies are effectively reducing the song Markovianity since transitions from given notes are more easily separable. When Eck

et al. used the harmony grid of blues songs as supplementary inputs to LSTM networks [26], they benefit from this observation.

3.3.2 Parameter Analysis

The network setup used so far has several free parameters that could be tuned to the task complexity (e.g. learning rate and topology). Moreover, some effect may increase the learning performance of the current model (e.g. noise or removing useless connections). Here, it is studied in several experiments the effect of different network parameters on the training performance in order to acquire knowledge on the behavior of LSTM networks with respects to these parameters. For each experiment, 10 training sessions are run where all parameters but the one that is studied are fixed. A training session is stopped when the accuracy over the last trial is 100% or the 10000th trial is reached. In the latter case, the sessions are considered null, otherwise completed. In each following Figures, Figure **A** shows the count of completed sessions from the 10 that are run and statistics (Figures **B** and **C**) are only applied on the training sessions that reached the stopping criterion (100% accuracy).

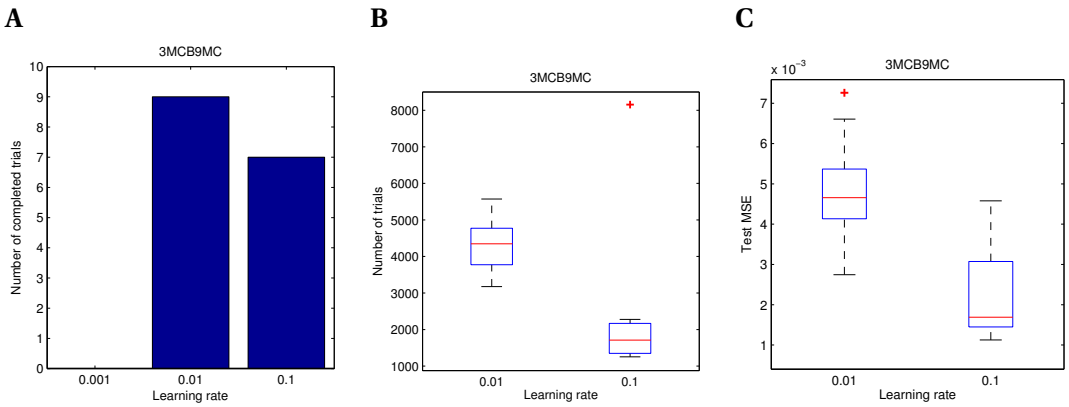


Figure 3.6 – *Frère Jacques polyphonic* Learning rate analysis. The subplot title '3MCB9MC' refers to a LSTM network composed of 3 MCBs each with 9 MCs. **A**. Number of training sessions that reached the stopping criterion before the 10'000th trial. **B**. Boxplots of the number of trials needed to reach the stopping criterion. **C**. Boxplots of the mean square error observed over the all sequence.

Learning rate The effect of three different learning rates on the training phase outcome is studied in this experiment. The learning rate is a very important parameter in order to find the global minimum. Indeed, a too high learning rate would results in high jumps and the network would struggle to find any minimum, even local, and a too small learning rate could trap the network in local minima. A learning rate of 0.001 is not able to reach the stopping criterion before the 10'000th trial and should thus not be used to learn this song (Figure 3.6 **A**). It is observed that increasing the learning rate beyond a certain value would make the network to have higher jumps and eventually not reach the convergence criterion. However,

using a slightly higher learning rate that the one that complete the most training sessions may decrease the training time (Figure 3.6 **B**) and eventually lead to a lower mean square error (Figure 3.6 **C**).

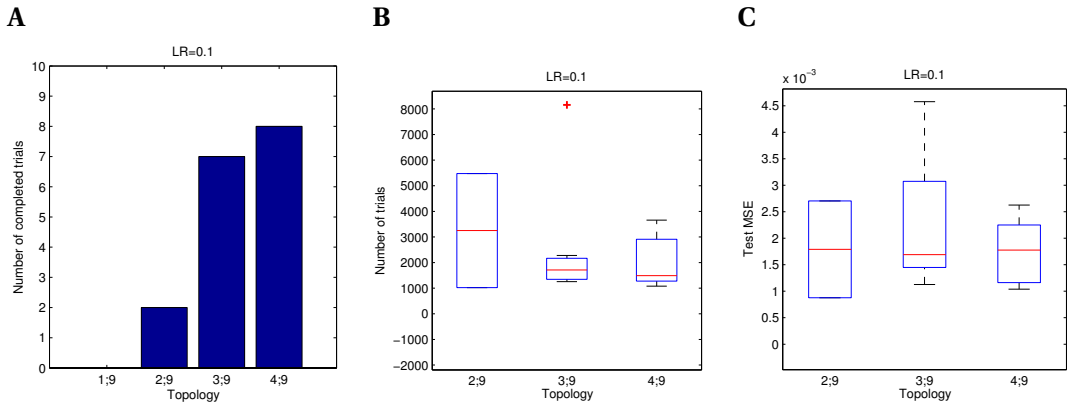


Figure 3.7 – *Frère Jacques polyphonic* Number of MCBs analysis. The x axis is the different topologies that are evaluated. 2;9 refers to a network composed of 2 MCBs each with 9 MCs. **A**. Number of training sessions that reached the stopping criterion before the 10'000th trial. **B**. Boxplots of the number of trials needed to reach the stopping criterion. **C**. Boxplots of the mean square error observed over the all sequence.

Number of MCBs The effect of the topology is important to study, to be able to evaluate the need of adapting the topology to more complex tasks. In this experiment I studied the number of MCBs for a fixed number of MCs. Results of the experiment are shown in Figure 3.7. There is a clear jump in the count of completed training sessions from the use of 2 to 3 MCBs (Figure 3.7**A**). Below a network constituted of 3 MCBs, almost all sessions are null and consequently these topology are not able learn the sequence. Thus, it should be preferred for this task a topology that include at least 3 MCBs. However, the number of MCBs does not impact on the number of trials needed (Figure 3.7**B**) nor on the final MSE (Figure 3.7**C**). The main conclusion from the MCBs number analysis is that, if the network is not able to learn a musical sequence, one should try to slowly increase the number of MCBs.

Number of MCs Figure 3.8 shows the result obtained for variable number of MCs in each MCB. The number of MCs should be as low as possible to minimize the CPU training time, since the more MCs the more connections and consequently mathematical operations. However, less than 9 MCs per MCB is not sufficient for the LSTM network to reliably learn the polyphonic *Frère Jacques* song (Figure 3.8**A**. Again for this topology experiment, the training sessions that were completed resulted in the same training performances (Figures 3.8**A** and **B**). Finally, if the network does not learn a musical sequences, modifying the number of MCs may result in a more reliable learning. Surprisingly, for the polyphonic and monophonic version of the song, the ideal number of MCs for a LSTM network composed of 4 MCB is exactly the

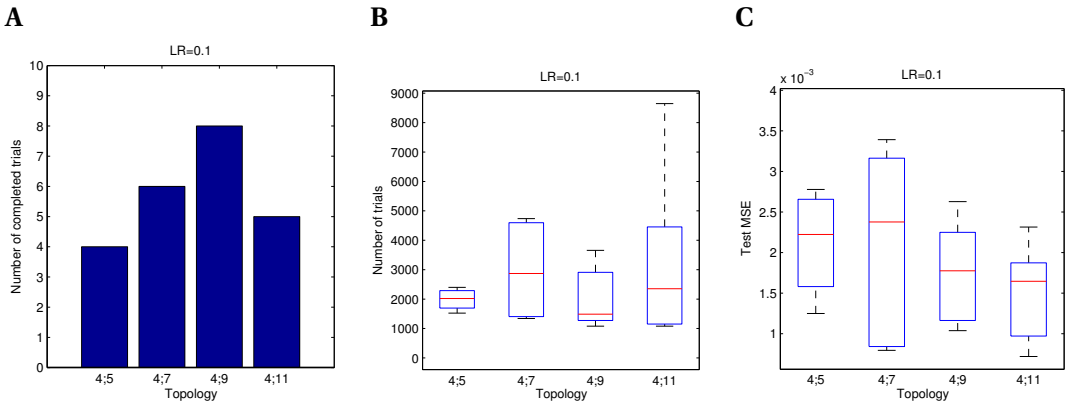


Figure 3.8 – *Frère Jacques polyphonic* Number of MCs per block analysis. **A**. Number of training sessions that reached the stopping criterion before the 10'000th trial. **B**. Boxplots of the number of trials needed to reach the stopping criterion. **C**. Boxplots of the mean square error observed over the all sequence.

length of the note vector in the network representation (the number of pitches present in the dataset). For more complex music learning tasks (Chapter 4), setting the number of MCs per block to the number of output units will be a starting point.

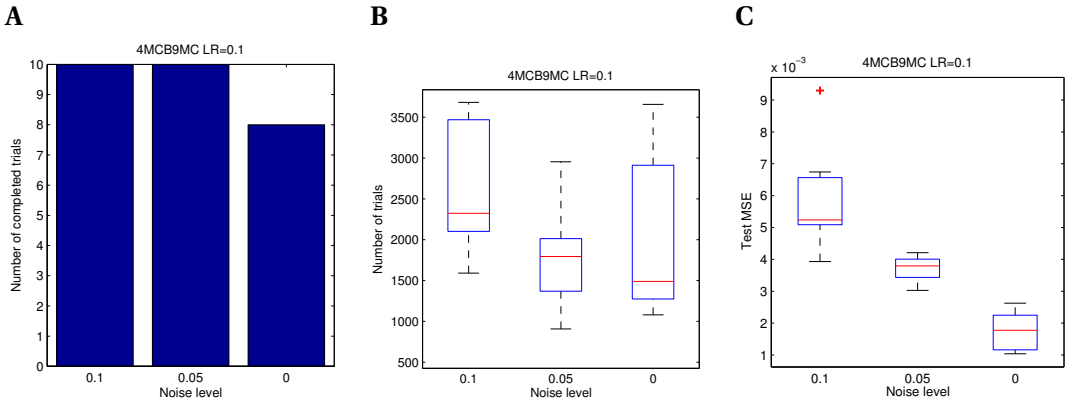


Figure 3.9 – *Frère Jacques polyphonic* Noise analysis. **A**. Number of training sessions that reached the stopping criterion before the 10'000th trial. **B**. Boxplots of the number of trials needed to reach the stopping criterion. **C**. Boxplots of the mean square error observed over the all sequence.

Noise I studied the effect of noise on the learning process, since it could provide an escape from local minima traps [27], it is hypothesized that noise could improve the training performances. Noise is introduced by building a collection of slightly modified song; a collection of 100 songs is built where each component of a note is added a uniformly distributed noise in the range [-σ σ]. The activation range of the output units consequently has to be modified so that it can accept non binary values in the range [0-σ 1+σ]. This is achieved by replacing the

activation function of output units f with f_{out} (equation 3.4). Noise seems to help the network to reach the stopping criterion (Figure 3.9**A**) but does not decrease the trials needed to reach it (Figure 3.9**B**). In addition, for higher noise levels, the distance between the output and the target note is increasing (Figure 3.9**C**) resulting in significantly decreasing accuracy on the test sequence ($\sigma = 0$). This effect comes from the stopping criterion. Indeed, training is stopped when the accuracy has reached 100%, however, the noise could be the reason why a given prediction is considered correct and when applied on the test sequence, this prediction is considered wrong. This analysis is however very interesting because it shows that the network output units do not saturate to their maximal and minimal values, but instead converge to the expected value of the target note (1 when active, 0 otherwise) even under a larger dynamic range of the output unit activation functions. This observation makes possible to use such modified LSTM networks on real valued targets instead of binary or integers, which will be important for the separation of time scale model in section 4.3.

$$f_{out}(x) = \frac{1+2\sigma}{1+exp(-x)} - \sigma \tag{3.4}$$

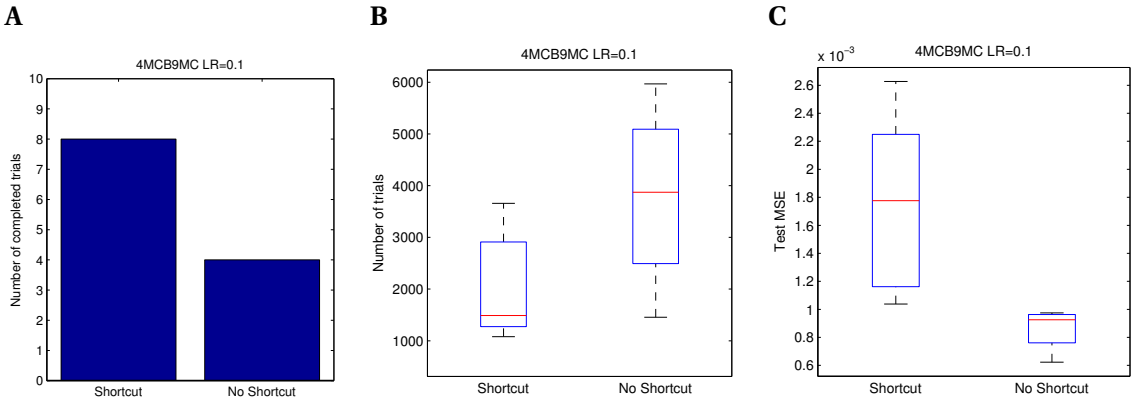


Figure 3.10 – *Frère Jacques polyphonic* Connection analysis. **A**. Number of training sessions that reached the stopping criterion before the 10'000th trial. **B**. Boxplots of the number of trials needed to reach the stopping criterion. **C**. Boxplots of the mean square error observed over the all sequence.

Input to output synapses In this experiment wether the impact of the connections form input to output units are essential or helpful in the musical sequence learning task. The results (Figure 3.10) show that removing these connections resulted in lowered testing MSE (Figure 3.10**C**) but at the significant cost of less completed training sessions (Figure 3.10**A**) and more trials per session to reach the stopping criterion (Figure 3.10**B**). I therefore choose to maintain these connections for the rest of the study.

3.3.3 Generative Mode

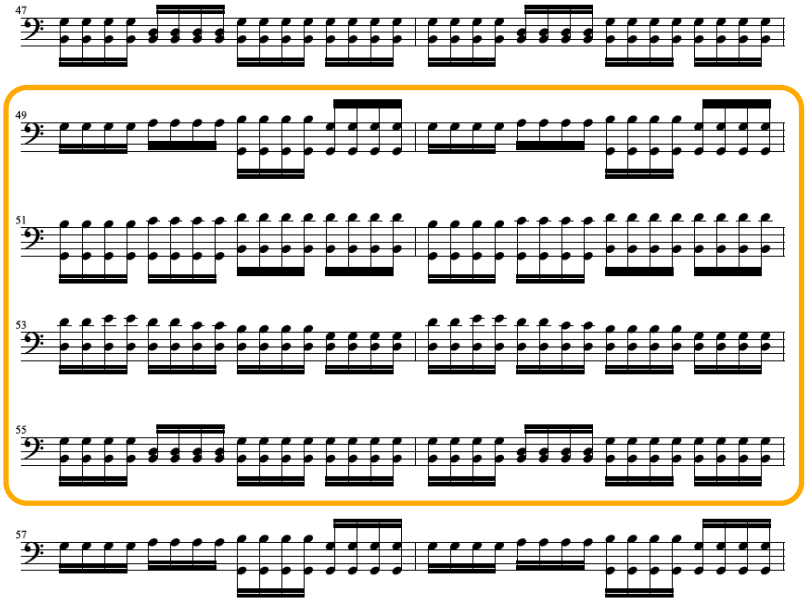


Figure 3.11 – *Frère Jacques polyphonic* Output of a trained network. The learned song *Frère Jacques* is reproduced without error for as long as the simulation is running. The yellow box highlights a full period of the song.

Once the network is able to correctly predict every transition in the song, I apply the generative mode architecture (see section 3.1.2). When the INON generative mode is applied, the network does not have any external information about the song and the input is simply the previous output note. Using this mode, trained INON networks achieve complete and periodic exact reproduction of the learned sequence. An extract of a network trained with the polyphonic version of *Frère Jacques* output back translated into musical notation (see section 2.2) is shown in Figure 3.11. The states and activations of the hidden layer units that resulted in this musical output are shown on Figures 3.12 and 3.13.

Interpretation of network mechanisms Figures 3.12 and 3.13 show the MC states and the gate activations across one full period of the learned sequence for the first two MCBs and last two MCBs respectively. The intrinsic frequency of the MC inner states and gate activations are decreasing across MCBs. It is observed that the MCs of the fourth block have their inner states oscillating with the same period as the learned musical sequence (Figure 3.13**E**). This MCB is thus encoding for the longer time structure of the song, while the first two MCB's MCs are encoding the fast time scale of the sequence (Figure 3.12**C** and **E**). This separation of time scale mechanism is the one deployed by LSTM networks to assess for non Markovianities in the song. LSTM networks are learning to recruit hidden units [28] in terms of the longer time scale MCs and, using this mechanism, learns to make the state space effectively Markovian. In addition, the first two blocks are resetting their states with their corresponding forget gates in order to

produce the needed outputs at the right times (Figure 3.12**B** and **D**), while the third and fourth blocks exhibit almost constant, never resetting, forget gate activations (Figure 3.13**B** and **D**). This is comforting the idea that the network mechanisms to learn non Markovian sequences are taking advantage of the multiple blocks in order to encode different time scales. From this analysis, the serial recruitment of MCs in each MCB during the initial training phase (see **Gradual increase of gate biases** in section 3.1) is effectively helping to train LSTM networks in this direction. Finally, it can be observed that for similar patterns in the song (e.g. the two first bars of *Frère Jacques*), the gate activation patterns for all MCBs are very similar. It is therefore, as expected, the MCs that carry most of the information that informs the network in which of the two repetitions of this pattern it actually is.

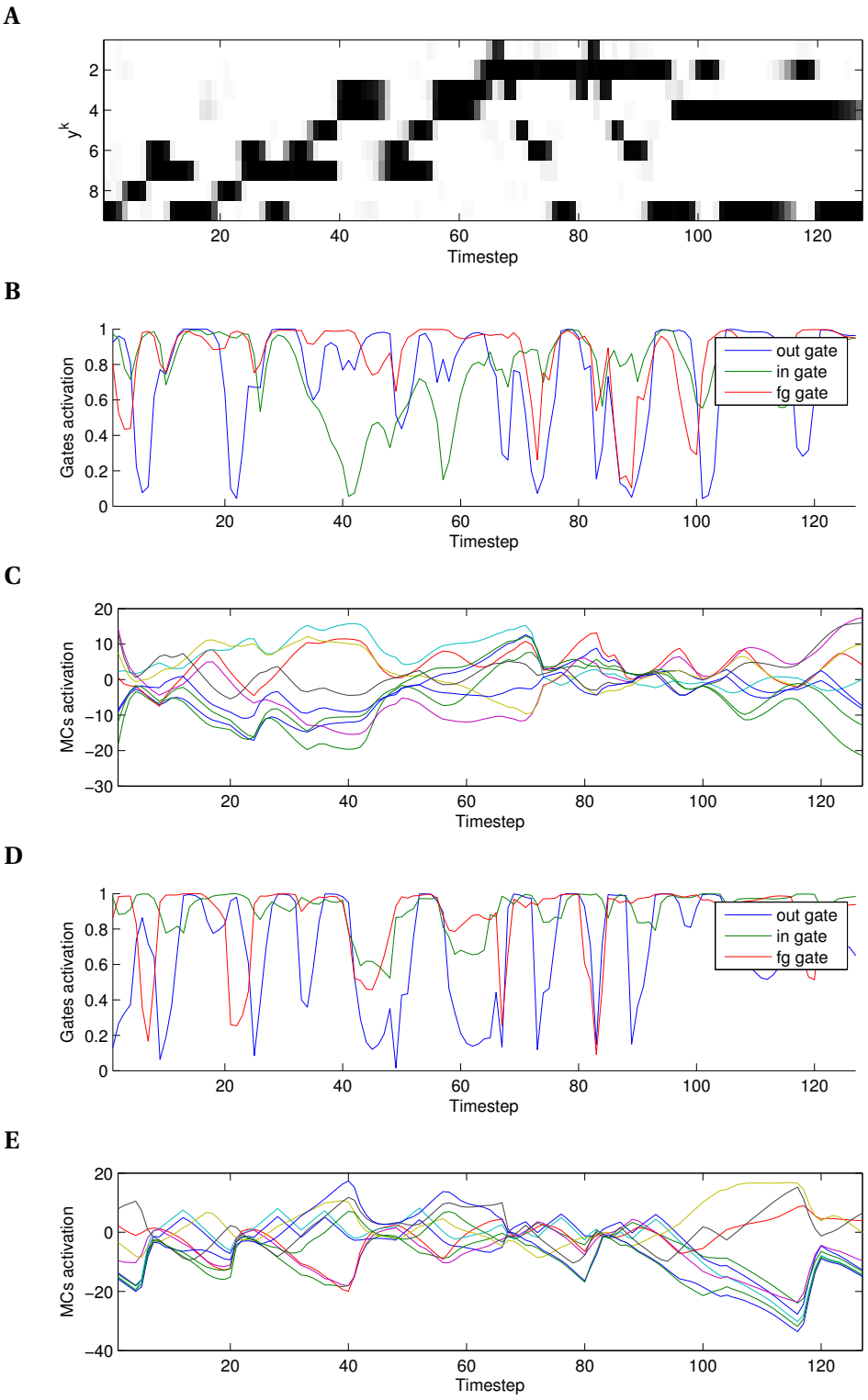


Figure 3.12 – *Frère Jacques polyphonic* **A** Network output for a full period of *Frère Jacques.* **B** Gates activation of the first memory cell block. **C** Memory cells activation of the first memory cell block. **D** Gates activation of the second memory cell block. **D** Memory cells activation of the second memory cell block.

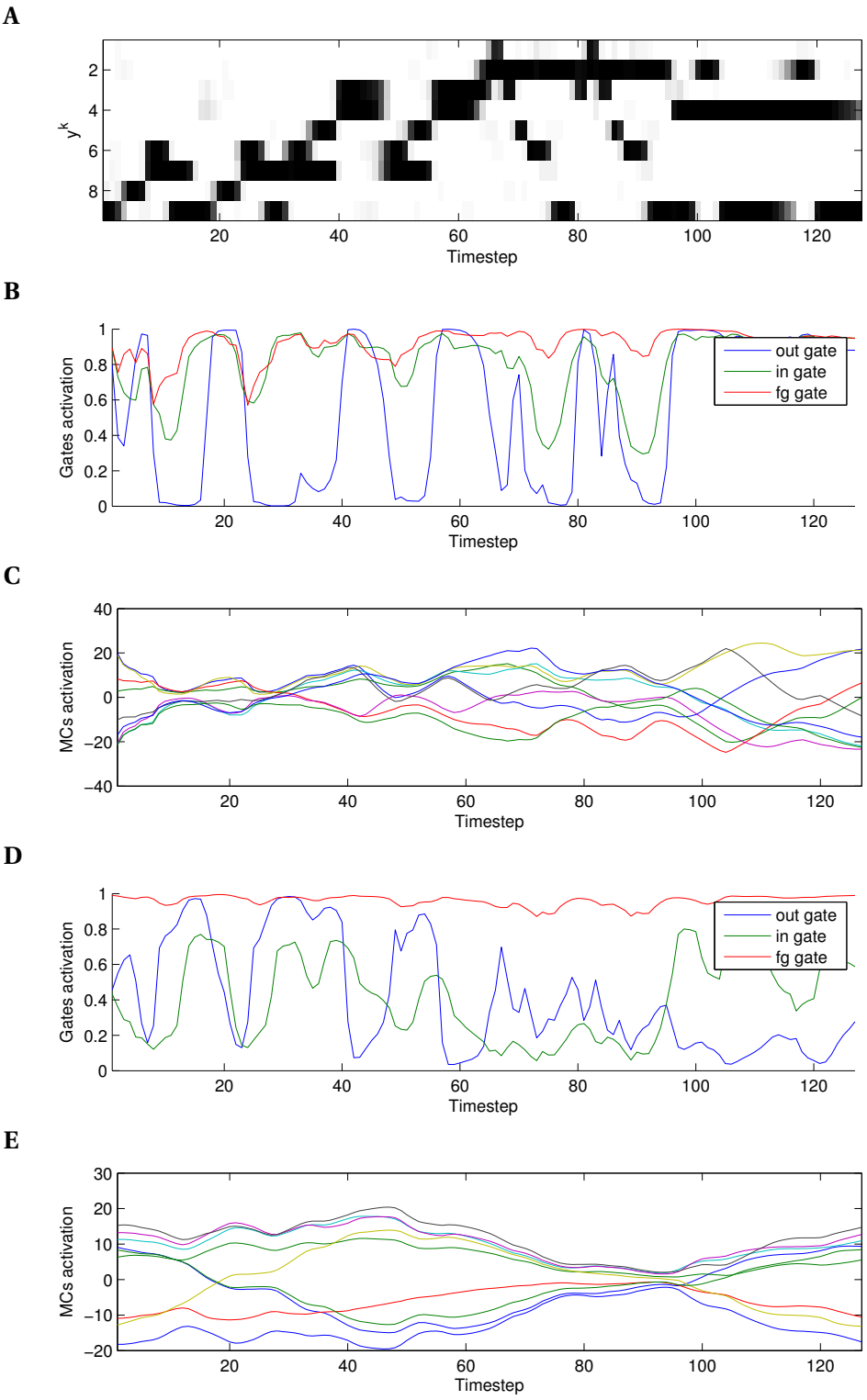


Figure 3.13 – *Frère Jacques polyphonic* **A** Network output for a full period of *Frère Jacques*. **B** Gates activation of the third memory cell block for a full period of *Frère Jacques*. **C** Memory cells activation of the third memory cell block. **D** Gates activation of the fourth memory cell block. **E** Memory cells activation of the fourth memory cell block.

4 Learning Multiple Musical Patterns – The Separation of Time Scales Model

I showed in the previous chapter that LSTM networks working with notes as input and output (INON model) are able to learn a single, yet non-Markovian, musical sequence. Indeed, the 128 sixteenth note transitions in the LSTM representation of *Frère Jacques* are predicted with an accuracy of 100%. Furthermore, self-fed trained INON networks using the generative mode architecture resulted in a periodic and exact autonomous reproduction of the learned musical sequence.

In this chapter, I extend the previous model to longer and more complex musical sequences. In particular, it is studied how extended INON LSTM networks performs on the J.S. Bach cello suites. It is first described a novel approach to learn musical sequences using LSTM networks, the separation of time scales model. In the second part, the results obtained on either a single prelude of the cello suite or three of them are discussed along with an analysis of the different free network parameters.

4.1 Separation of Time Scales

To learn more complex and longer musical sequences simply by using the architecture described in the previous chapter does not seem feasible. Indeed, LSTM networks fail to learn both the long time structure and the fast transitions using the previously described architecture (see Figure 4.1). Therefore, I will present here the separation of time scales model, which will be shown to provide a solution to this problem.

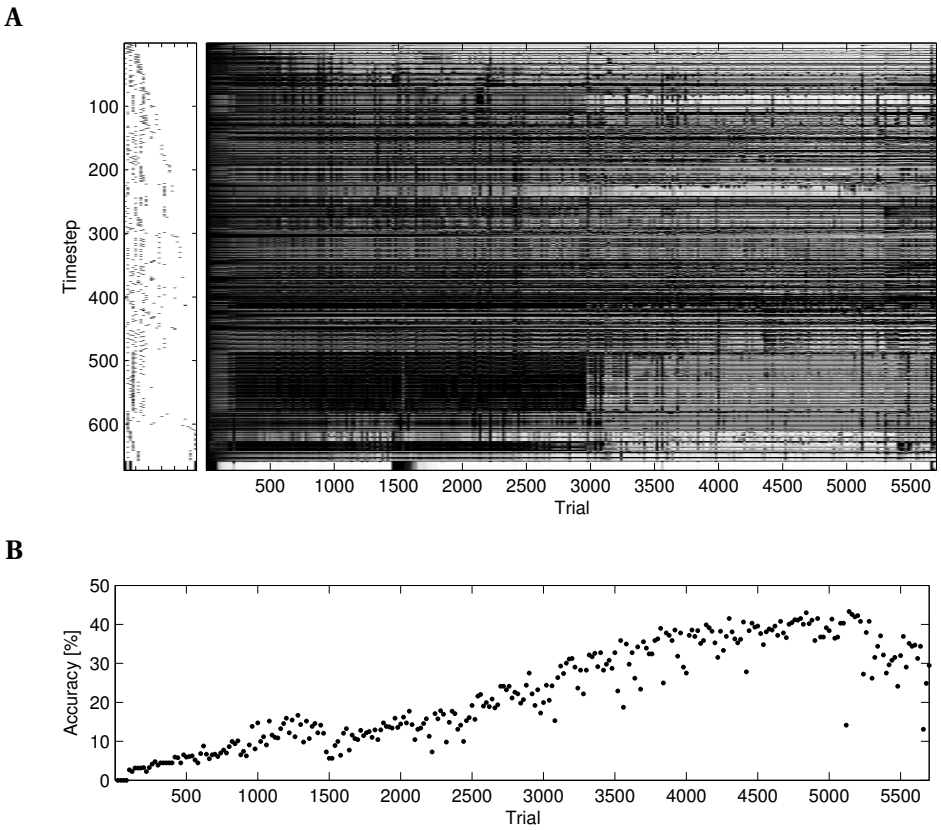


Figure 4.1 – *Prelude CS 1* Training phase results using the predictive mode of the INON model. **A**. Left: musical sequence to be learned. Right: MSE at each predicted note over time during the training phase. Black indicates the maximum MSE. **B**. Accuracy at each trial.

Thinking as Mozart retrieving the *Miserere*, we might suppose that he started by remembering the long time structure of the song and began to produce note combinations based on this longer structure. To mimic this, it is proposed to separate the time scales of the musical sequences to be learned into a slow and a fast component. One LSTM network would then be dedicated to each time scale and connections from the slow time scale network to the fast time scale network created. This idea arose by observing the internal mechanism of LSTM networks to solve non-Markovian sequence learning tasks. Indeed, as previously discussed (see **Interpretation of network mechanisms** in section 3.3.3) LSTM networks exhibit a time scale hierarchy across MCBs, which allows them to learn complex sequences. Applying

a supplementary separation of time scales is consequently coherent with LSTM network principles.

How the slow and the fast time scale networks interact is explained in the following section. It is then presented what are the slow and fast time scales applied to musical sequences (sections 4.1.2 and 4.1.3).

4.1.1 Separation of Time Scales Architecture

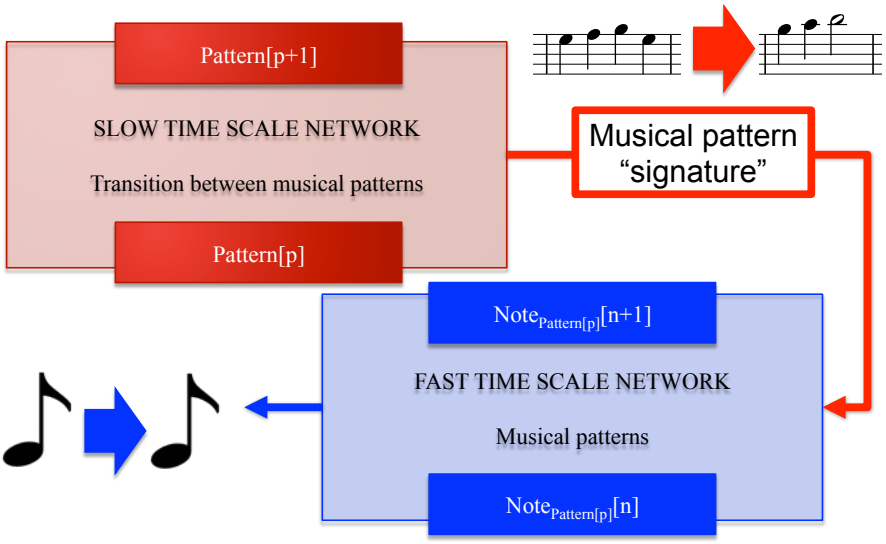


Figure 4.2 – Model architecture for the separation of time scales approach.

As schematized in Figure 4.2, the slow time scale LSTM network sends information about the long time structure of the song to the fast time scale network. The information about the long time structure of the song is encoded by the musical pattern *signatures*. The fast time scale LSTM network is trained to learn the transitions between notes of a given pattern, while the slow time scale LSTM network is trained to learn transitions between each patterns. In other word the slow time scale LSTM network biases the fast time scale LSTM network toward the the production of the musical pattern corresponding to the state of the slow time scale network.

In the following, the predictive and generative modes adapted to the separation of time scales architecture are detailed.

Learning – Predictive Mode

For the predictive model of the separation of time scales architecture, the fast and slow time scale LSTM networks are trained separately with their respective predictive mode as presented in section 4.2.1 for the fast time scale LSTM networks and 4.3.1 for the slow time scale LSTM networks.

Reproducing – Generative Mode

Self feedback independently each of the two trained networks provides a complete generative mode for autonomous music production based on the learned production rules of both time scales.

4.1.2 Fast Time Scale

The fast time scale is chosen to be the sequence of notes in musical patterns. These patterns are parts of the song to be learned or reproduced. One free parameters of the slow and fast time scale is the pattern length and it should be carefully chosen to maximize the efficiency of the model (see the analysis in Figures 4.8 and 4.10). The bar is the most natural subdivision in most classical music pieces and is therefore a natural candidate as pattern length. It is especially important in Bach music because of the information it carries about symbolism; Bach had the habit and the gift to be able to produce incredibly beautiful pieces of music while constraining itself to *hide* information that only appears after precise analysis about the song structure, which often involves bars counting. Examples are found in almost all its compositions. For these reasons, it has been chosen a pattern length of 1 or 2 bars, which corresponds to 16 and 32 sixteen notes respectively for musical pieces with time signature 4/4 as Bach prelude of the first cello suite.

4.1.3 Slow Time Scale

The slow time scale represents how musical patterns are ordered into a wholwmusical sequence. To represent a musical pattern I chose to apply a principal component analysis (PCA) on the repertoire of musical patterns (in the network representation) present in the sequence (or repertoire of sequences) to be learned. The intuition for the use of a PCA for musical pattern signatures is that similar musical patterns will be mapped to similar principal component (PC) values [29]. The principles of a PCA applied to musical patterns are that patterns are represented in new dimensions along which the variance between different musical pattern unique *identifiers* is maximized. PCA on the collection of musical patterns to be learned is then a tool to associate each pattern to points in these new dimensions or PCs. It is the first PC values that are taken to uniquely identify musical patterns from each others.

Furthermore, because patterns that are similar would have similar PCs values, the unique

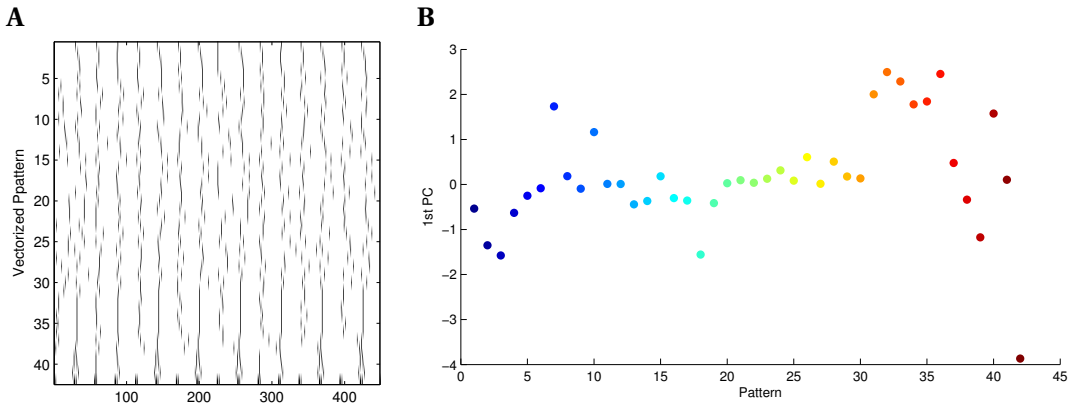


Figure 4.3 – *Prelude CS 1* **A** Data matrix for PCA on the prelude from the first cello suite of J.S. Bach. Each row is a pattern: 16 notes in a row. **B** Value of the first principal component for each of the 42 patterns. Pattern order is kept in the x-axis.

pattern identifiers given to the fast time scale network are called *external biases*. I will show later that the shared similarities between actual patterns and their associated PC values will help the fast time scale LSTM networks to learn the different musical patterns. Indeed, due to the choice of PCs as pattern signature, similar musical patterns would have similar bias activations, which will in turn help LSTM networks to retrieve the correct patterns. This phenomenon is discussed in section 4.2.3.

The data matrix (Figure 4.3**A**) is formed from the musical sequence that is to be learned. To construct the PCA data matrix, the sequence is separated in patterns of given and fixed length (here 16 notes), then these patterns are vectorized (each of the pattern notes in the network representation are linearized in one row of the data matrix). PCA is applied using the MATLAB standard PCA package on this matrix. PCs are sorted from higher to lower eigenvalues of the diagonalized covariance matrix. The projection of each of the 42 bars from Bach prelude of the first cello suite in the first PC space is shown on Figure 4.3**B**. Interestingly, it seems that there is an order in the time evolution of the PC value across patterns. This order most probably comes from the highly structured properties of music in general and even more of Bach music. This observation is promising to be able to learn the slow time scale production rules since the pattern *walk* in the PCs space is not random.

4.2 Learning Musical Patterns – Fast Time Scale

It is presented next the fast time scale network. This network produces the actual note of each pattern given the external biases. The network architecture and algorithm are presented first, then the results and effect of several network parameters are studied.

4.2.1 Network Architecture

The network architecture is similar to the one presented in section 3.1. However, I added external biases that carry the information about the long time structure. These biases are integrated in the network as additional input units and are fully connected with the hidden and output layer. The connections are plastic, hence the network should learn how to use these biases to produce the wanted musical patterns. The external biases corresponding to a pattern are sustained for as long as the pattern is being learned or predicted.

Starting note of patterns To be able to independently generate a complete musical sequence, the network should also predict the first note of each pattern. Indeed, if the predictive mode is able to predict every note transitions from the first note, in the generative mode the first note would also be needed as external information to reproduce the sequence, which is to be avoided for a completely autonomous. To achieve autonomous prediction of the first note of each musical pattern, it is added a special binary input unit $y_{input^{start}}$ that is active only for the first prediction of each pattern. When this input unit is activated, the note given to the network is a silence (all pitches non active) and the network task is then to predict the first note of the corresponding pattern.

Learning – Predictive Mode

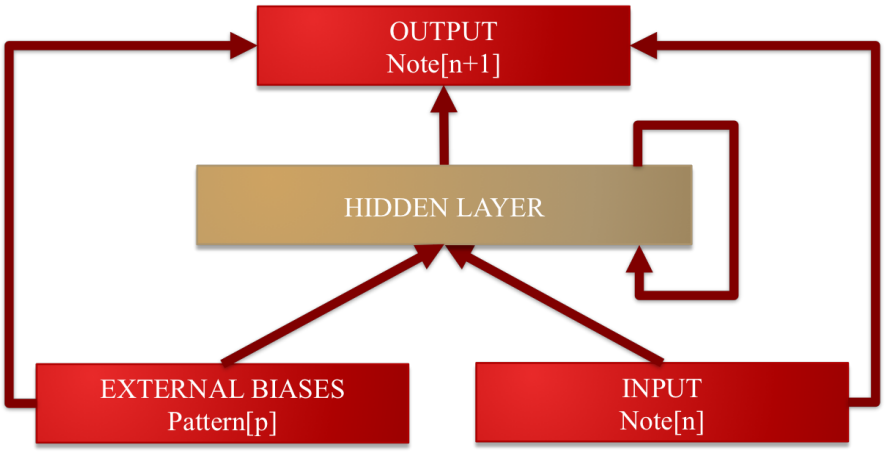


Figure 4.4 – Architecture of the predictive mode for fast time scale LSTM networks.

The predictive mode algorithm (Algorithm 2) is applied on the dataset to be learned with a

network architecture as presented in Figure 4.4. The training consists in finding the connection weights that are able to predict the upcoming note of every pattern. The network is trained until a threshold level of accuracy is reached. As a preprocessing step, it is needed to compute the external biases of the dataset. The main parameter of this step is the pattern length. Afterwards, the network is trained on the dataset patterns with the computed external biases. The accuracy is the proportion of correctly predicted notes across all patterns with respect to the total amount of predictions.

Data: Patterns and external biases in LSTM representation
Result: Connection weights updated to predict each pattern
Network initialization;
while *Accuracy* $< \theta$ **do**
  for *each pattern p in the song* **do**
    Reset all activations and MC states;
    for *each note n in pattern p* **do**
      if *first note in pattern* **then**
        External input: $y^i_{input^{note}}[0] = 0$, $y^i_{input^{bias}} = PC_p[i]$, $y_{input^{start}} = 1$;
      else
        External input: $y^i_{input^{note}}[n] = note^i_p[n]$, $y^i_{input^{bias}} = PC_p[i]$, $y_{input^{start}} = 0$;
      end
      Forward pass
      Update unit activations and partial derivatives
      Backward pass
      Compute local weight changes to decrease the output error
    end
  end
  if *Accuracy* $< \theta$ **then**
    Update weights;
  end
end

Algorithm 2: Learning multiple musical pattern – fast time scale

Reproducing – Generative Mode

If the fast time scale network is able to correctly predict a collection of patterns given their biases after training, when self-fed with its own output, it should theoretically be able to reproduce the pattern. It is also inquired what happens to the network when it is given non learned external biases in chapter 5.

Effect of a prediction error In the predictive mode, the note to note transitions are learned and the song to be learned is known by the network preventing any consequence to a wrong prediction. However, in the generative mode, only the external biases and their transitions are

known and a mistake in a prediction would output a wrong note as in the predictive mode, however, this error is then propagated along the remaining predictions of the pattern. This is the reason why a high accuracy in the network predictive performances is needed to be able to reproduce the learned patterns. However, if the focus is music composition, a lower accuracy may be wanted in order to let the network make mistakes that could eventually create new musical patterns [26].

4.2.2 Predictive Mode

Topology				
Input units	External biases	MCB	MCs	Output units
28	25(87% data explained)	5	10	28
PCA				
Pattern length	Pattern number			
16 notes	42			
Training				
Learning rate	Accuracy threshold			
0.01	90%			

Table 4.1 – *Prelude CS 1* Fixed parameters, if not mentioned, for the prelude of the first cello suite experiments

Topology				
Input units	External biases	MCB	MCs	Output units
37	70(82% data explained)	5	10	37
PCA				
Pattern length	Pattern number			
16 notes	197			
Training				
Learning rate	Accuracy threshold			
0.01	90%			

Table 4.2 – *Preludes CS 1,3 and 4* Fixed parameters, if not mentioned, for the three preludes of the first, third and fourth cello suites experiments

Datasets For the following results, either the single prelude from the first Bach cello suite (*Prelude CS 1*) or three preludes from Bach first, third and fourth cello suits (*Preludes CS 1,3 and 4*), are taken as dataset to be learned by the networks. The dataset used to generate the data is specified in each figure. For each dataset, different constant parameters are applied as shown in Tables 4.1 and 4.2. I present results from both datasets in order to observe the networks behavior when the dataset is increased, while still having a smaller dataset that allows for quicker computations.

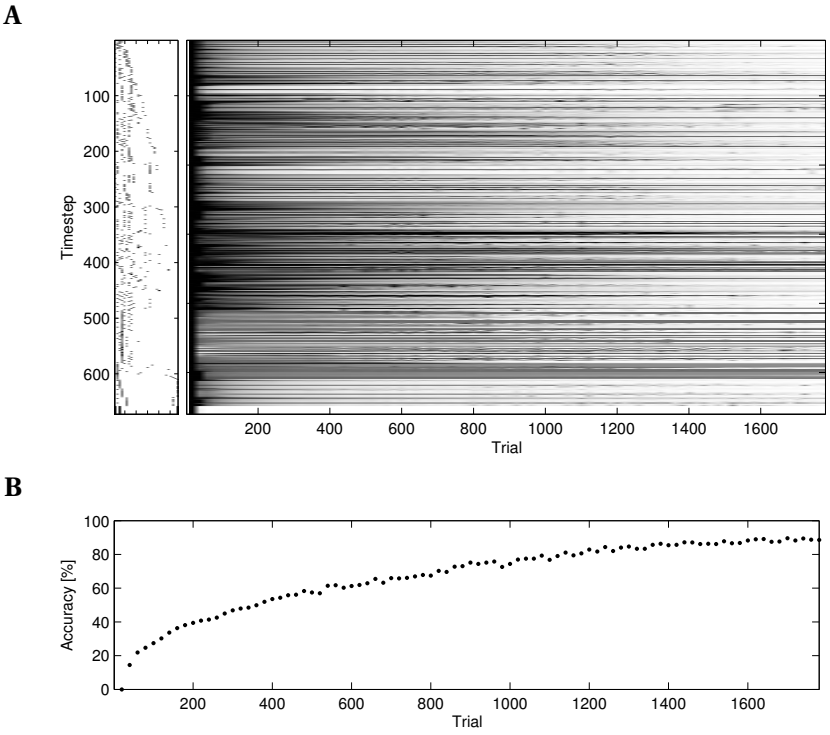


Figure 4.5 – *Prelude CS 1* **A**. Left: musical sequence to be learned. Right: MSE at each predicted note over time during the training phase. Black indicates the maximum MSE. **B**. Accuracy at each trial.

Learning patterns from Bach prelude of the first cello suite The predictive mode applied on the prelude of the first cello suite is able to learn 90% of the patterns in a number of trials of mean 1877 ± 132 (n=10). A typical evolution if the error along training is showed in Figure 4.5. Using the separation of time scale and principal components as external biases, the LSTM is able to learn a much longer and complex musical sequence than the *Frère Jacques* song. In addition, the accuracy evolution during training is much less stochastic. However, with this topology (10 MCs in each of the 5 blocks) the accuracy reaches a plateau at 94.9% accuracy as seen on Figure 4.6.

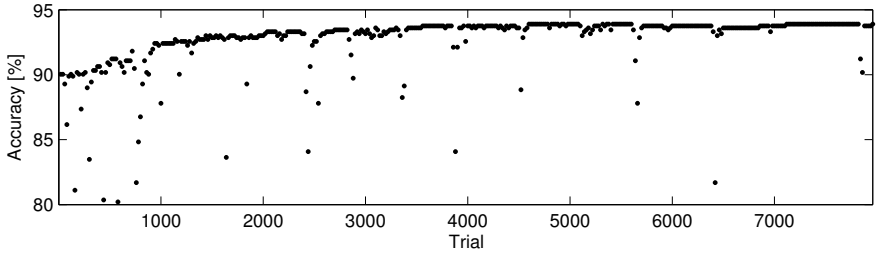


Figure 4.6 – *Prelude CS 1* Accuracy evolution on a retrained network.

Learning patterns from three Bach preludes The predictive mode applied on the three preludes is able to learn 90% of the patterns in a number of trials of mean 2246 ± 214 (n=5). The training takes more than four times more time to see all patterns when two songs are added (42 patterns for prelude CS 1 versus 197 patterns for the three preludes). However the number of trials needed to reach 90 % accuracy is only slightly bigger. In addition, the number of principal components needed for learn 197 patterns are also not linearly dependent to the number of patterns in the dataset (25 principal components for prelude CS 1 versus 70 principal components for the three preludes). One can deduce that this model is a powerful tool to learn long musical sequences that can come from different songs. I should mention that one limiting factor is computation time, which is in this case around two days on the LCN servers.

4.2.3 Parameter analysis

In the following section, it is first validated the chosen architecture for the fast time scale network, especially the external biases. It is then studied the network capacity along with the effect on learning of some of the network parameters such as the topology, the pattern length and the number of external biases.

Validation of the separation of time scales model As shown in the beginning of this chapter (Figure 4.1), when a INON LSTM network is applied on the *Prelude CS 1* dataset, it fails to learn the sequence. The accuracy is only reaching 40% after 5000 trials. This observation validates the separation of time scales model presented here, since using the latter model LSTM networks are abel to learn the same dataset (Figure 4.5).

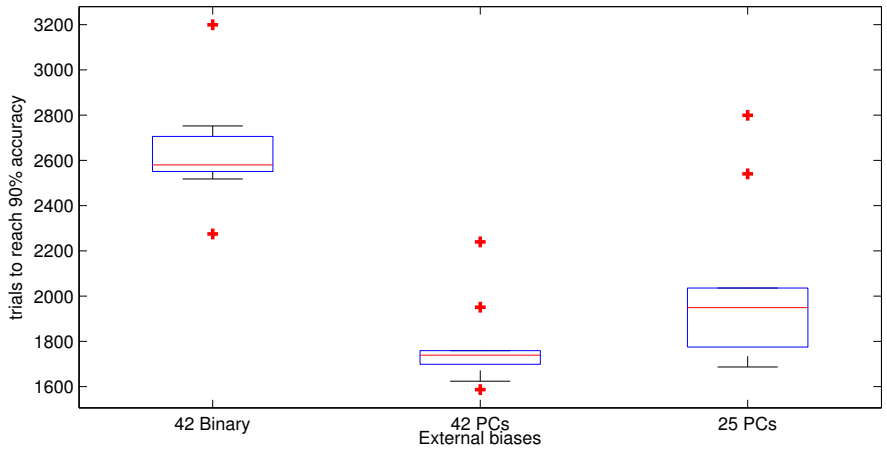


Figure 4.7 – **Prelude CS 1** Boxplot for N=10 training sessions for different external biases.

Validation of principal components as external biases It is inquired if the use of principal components as external biases is an efficient way of biasing the network toward a pattern. To

do so, I compared the number of trials needed to learn the sequence when binary biases are used as external biases versus principal components (Figure 4.7). To implement binary biases, I created one additional input unit for each of the 42 patterns instead of the bias units used so far. When a pattern is being trained, the corresponding input units has an activation of 1, 0 otherwise. I compared the training performance of the 42 binary biases with 42 PC external biases and 25 PC bias units, which I later show to be the ideal number of external biases for this dataset. Using binary biases, the fast time scale LSTM network is also able to learn the musical sequence as there is one unique input units dedicated to each pattern. However, the number of trials needed to learn the sequence is much higher than when principal components are used. As the binary biases does not account for similarities in patterns, it is an expected outcome. This observation validates the use of principal components as external biases for the fast time scale network as it is more efficient than binary biases. In addition, learning with ANNs transitions between pattern signatures (the slow time scale) encoded as a collection of binary biases is not an option and strongly lacks generalization. Indeed, on one hand, the number of binary bias units will be as high as the number of musical patterns to learn and on the other hand, the transitions between each patterns is a serial activation of each and every external binary bias units, which is clearly not a task to solve for ANNs.

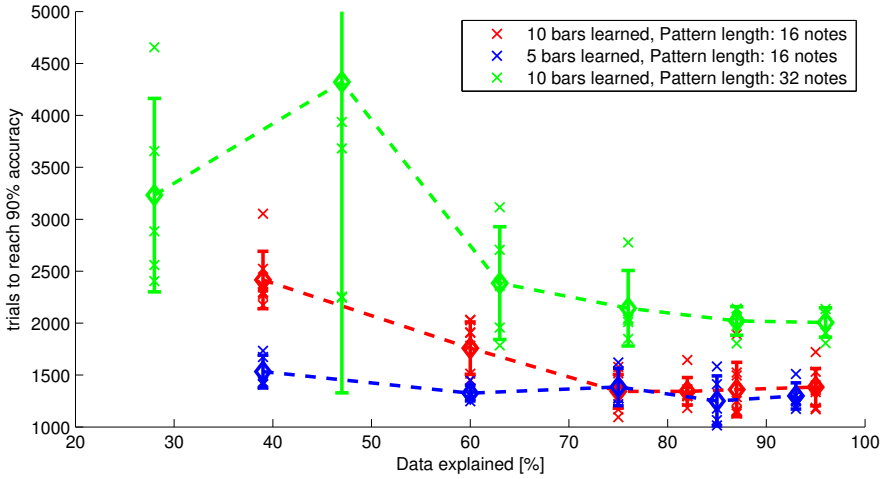


Figure 4.8 – **Prelude CS 1** Trials needed to reach 90% of accuracy on 5 or 10 first bars of the prelude from the first cello suite of J.S. Bach in function of the external biases length and the pattern length. The x axis shows how much of the data is explained by the principal components used as external biases.

Number of principal components as external biases It is studied the number of principal components that are required to efficiently learn the songs. In addition, a double pattern length of 32 notes is inquired for the *Prelude CS 1* dataset. The results of this analysis are showed in Figure 4.8 and 4.9 respectively for the *Prelude CS 1* and *Preludes CS 1, 3 and 4* datasets. The pattern length that corresponds to one bar in a time signature of 4/4 (16 notes) is easier to learn by the network than a pattern length of two bars This is the reason why 16 notes

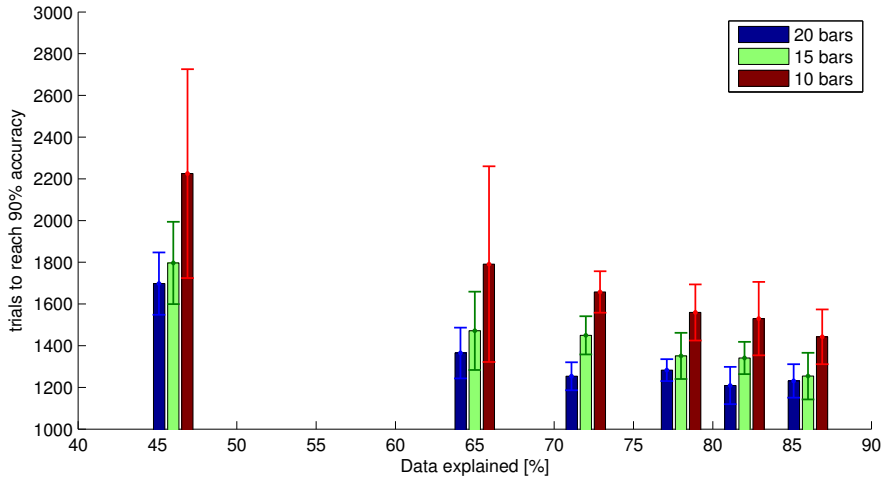


Figure 4.9 – **Preludes CS 1,3 and 4** Trials needed to reach 90% of accuracy on 10, 15 and 20 randomly chosen bars of the preludes from the first, third and fourth cello suites of J.S. Bach in function of the external biases length (N=10). The x axis is how much of the data are explained by the principal components used as external biases. The pattern length is 16 notes in the network representation.

long pattern is chosen as the standard in this report. The trials needed to learn the song(s) are decreasing when more principal components are added to the external biases. After 70% of data explained by the principal components, a plateau is reached. For both dataset, it is chosen a tradeoff between a minimal number of external biases and a minimal training time. This tradeoff is chosen to be the number of principal components needed to explain 85% of the original data. Counterintuitively, it is observed for the bigger dataset only that the number of training trials is decreasing with increasing number of patterns learned (see Figure 4.9). However, a small amount of pattern are learned to do this analysis. Therefore, this effect is further studied in the capacity analysis but it can already be concluded that it is easier for the fast time scale networks to learn patterns when it is trained on a certain amount of external biases values.

Capacity analysis The capacity of the networks, in term of the number of trials needed to learn the sequences is showed in Figures 4.10 and 4.11 for the *Prelude CS1* and *Preludes CS 1,3 and 4* datasets respectively. In Figure 4.10, it is added the same analysis with 2 bars long patterns (15 first PCs used as external biases). As previously observed, longer patterns are harder to learn and the training trials highly increases as the number of patterns to be learned increases. On the contrary, the number of trials needed to learn 90% of the 1 bar length patterns does not increase much as the number of patterns to learn increase, validating again the choice of this size of patterns. The same observation is made when the dataset is increased from 42 to 197 patterns (Figure 4.11). For each run of this experiment, the 197 patterns are learned in a random but constant order. The initial negative bump may come from this randomized order or that the network need to explore multiple regions of the PCs space in

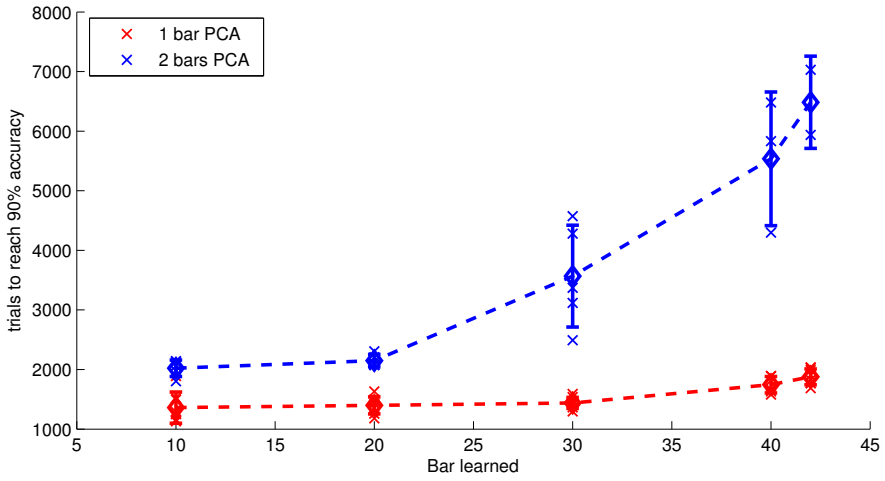


Figure 4.10 – *Prelude CS 1* Capacity analysis. Training trials in function of the number of patterns learned and the pattern length. 15 principal components are used as external biases for the 32 notes long patterns data versus 25 for the 16 notes long patterns.

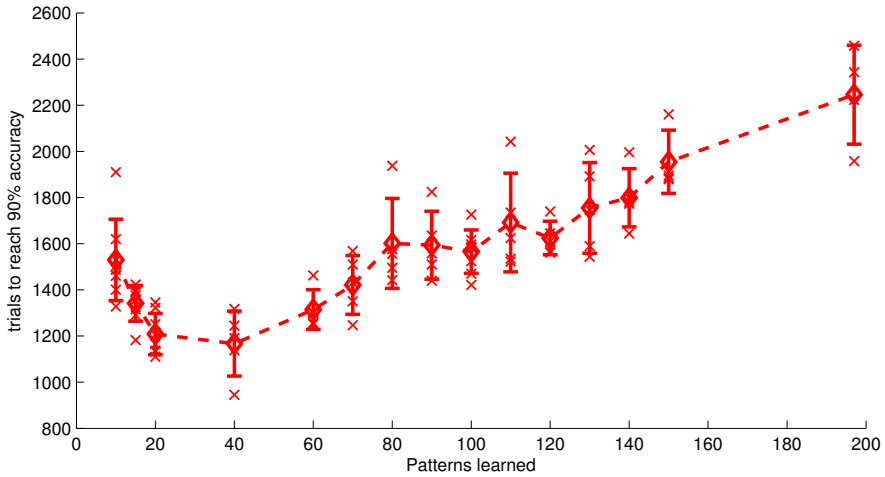


Figure 4.11 – *Preludes CS 1,3 and 4* Capacity analysis. Training trials in function of the number of pattern learned. Pattern length is 16 notes. Number of external biases is 70.

order to take a real advantage of the separation of time scales model, or both. However, the main conclusion from this capacity analysis is that the proposed separation of time scale algorithm combined with LSTM networks is a powerful tool to learn quickly very complex and long musical sequences.

Topology analysis How the topology influences learning was studied in section 3.3.2 for the single musical sequence learning LSTM algorithm. We observed that the hidden layer topology is a crucial parameters to allow for convergence to a network that has learn the sequence. In the Figure 4.12, it is seen that increasing by a factor 2 the MCBs' number does not change at all

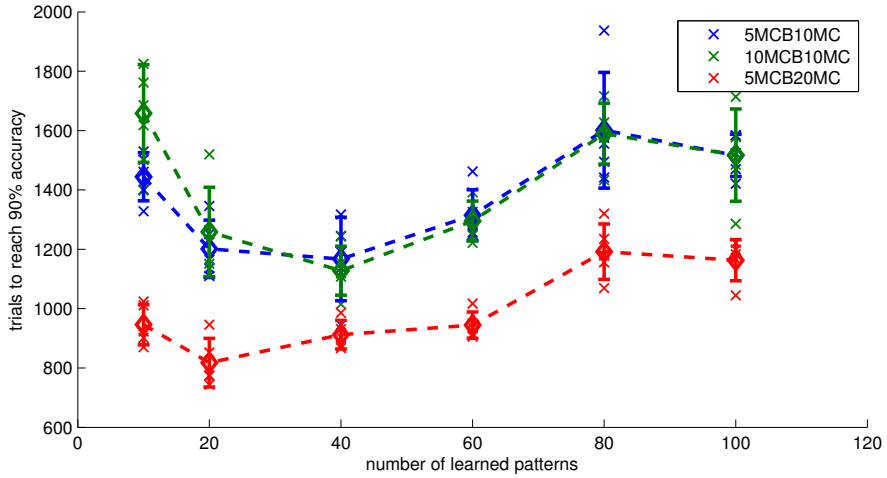


Figure 4.12 – *Preludes CS 1,3 and 4* Number of training trials for increasing number of pattern learned in function of the hidden layer topology. Pattern length is 16 notes. Number of external biases is 25.

the training trials needed to learn 90% of the patterns but increasing by the same factor the number of MCs per block does. However, the number of connections so much increased that the training time is way more than for the 5MCB10MC topology. The main conclusion is that the it should be found a compromise between the task complexity and the topology and that the number of MCBs does not need to be high to be able to learn complicated music.

4.2.4 Generative Mode

The generative results for the *Prelude CS 1* dataset comes from the more accurate network; a network composed of 20 MCs instead of 10 per block (Table 4.1). This fast time scale network is able to reproduce the learnt musical sequences independently from them (Figure 4.13) with an accuracy of 96.6%. Listening to the generative mode output is pleasant, even to the neurons of a master student, and Bach music is easily recognizable.

Network units receptive field to external biases Each units connected to the external biases that act as unique identifiers for musical patterns are modulated by the biases in a specific way that allows the fast time scale network to reproduce almost exactly all learned patterns. The receptive field (RF) of each network units to the external biases is computed as the net contribution of external biases to each unit's input (see equation 4.1). The Figure 4.14 shows this effect on selected network units (RFs of all units are presented in the Appendix). It can be observed that the two selected MCs have net activations due to the external biases that are dependent on the region where the patterns are projected in the principal components. In addition, those two MCs have opposed receptive fields; when patterns that are in a principal component region are positively contributing to the net activations of one unit, the same

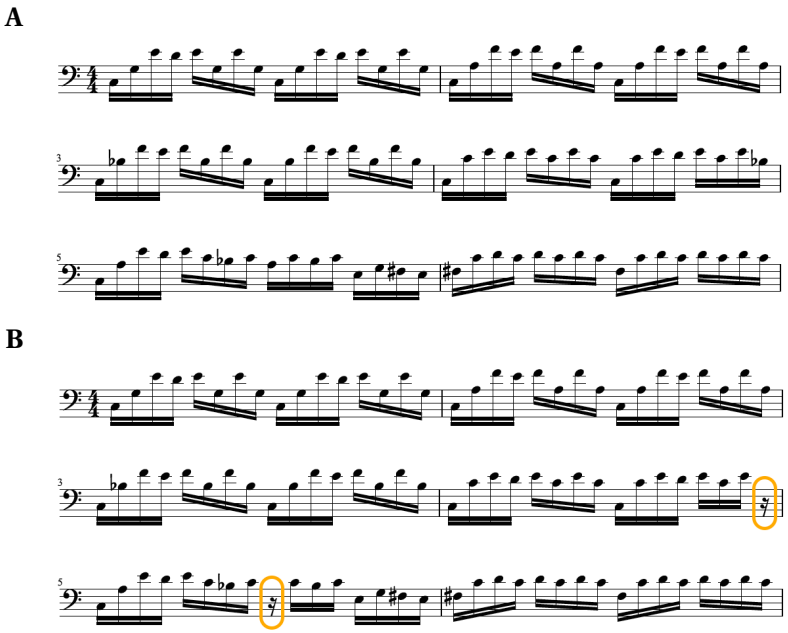
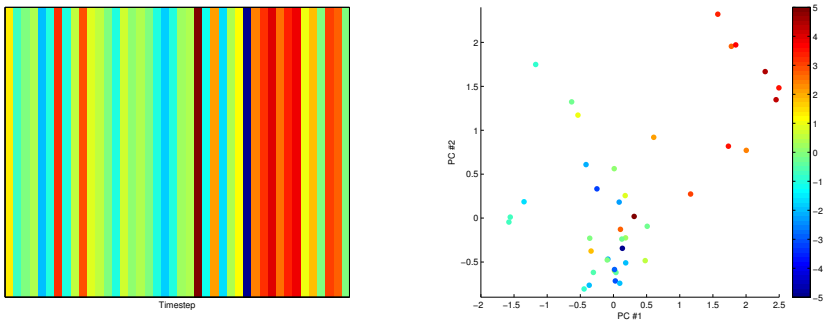


Figure 4.13 – *Prelude CS 1* Network output in musical notation for the first 6 patterns. **A** Target sequence **B** Generative mode result for a trained network (5MCB20MC) with a training accuracy of 96.6%. Error with respect to the original dataset are highlighted.

region tends to result in negative contribution for the other unit. This *specialization* of the unit is here clearly dependent on the similarities between patterns rather than the place they are displayed in the sequence as no order is present in the left figures. It should be pointed out that this behavior does not appear on every units but is an effect observed in every trained networks and is thought to be one of the mechanism that the fast time scale long short term memory networks are using to be able to learn the complex and long musical sequences that are fed to them. The similarities with the associative memory of Hopfield networks [11] should be highlighted. Indeed, trained fast time scales LSTM networks are using the compressed memory of patterns in the form of similarity-dependent musical patterns identifiers to recall the pattern they should produce.

$$RF^j[P] = \sum_p w_{pj} y^p_{input^{bias}} \tag{4.1}$$

A



B

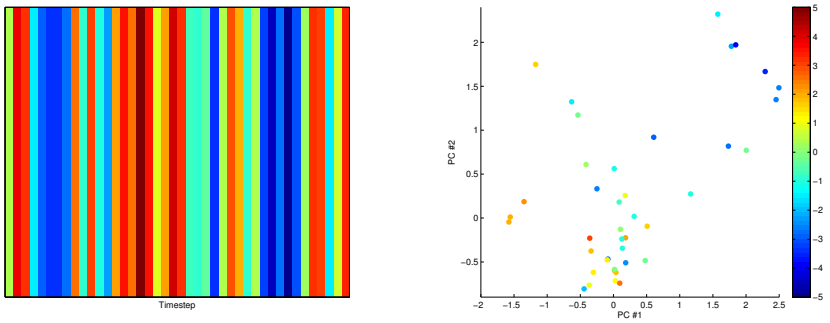


Figure 4.14 – *Prelude CS 1* Unit net input after training due to the external biases of each of the 42 patterns in temporal order (left) or in the first two principal components space (right). **A** 12th MC of the 5th block. **B** 15th MC of the 5th block.

4.3 Learning Pattern Transitions – Slow Time Scale

The role of the slow time scale network is to produce the pattern signatures encoded in principal components. The LSTM should be modified in order to learn transition between patterns that are real valued number. It is presented in this section how the LSTM network is modified to achieve this goal and the results obtained on the prelude of the first cello suite from J.S. Bach.

4.3.1 Network Architecture

The architecture is similar with the one used to learn a single musical sequence with the difference that input and output are now principal component values. The number of input and output units is consequently the number of principal components used to define a pattern in the fast time scale network. In order to obtained real valued number as output, the non linear function f in the equation that computes the output unit activations (Equation 1.26) is transformed to the f_{out} equation below similarly to the equation used to introduce noise in the network (see section 3.3.2). In this equation σ is the maximal real valued number found in the principal components (in absolute value) across all patterns. This transformation allows the output units to take any value in the principal components range.

$$f_{out}(x) = \frac{1+2\sigma}{1+exp(-x)} - \sigma \tag{4.2}$$

Learning – Predictive Mode

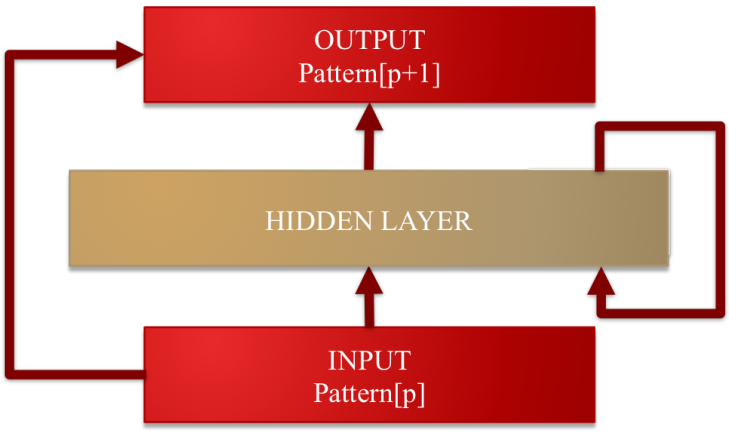


Figure 4.15 – Network architecture of the predictive mode for slow time scale LSTM networks.

The predictive mode algorithm is the same as Algorithm 1 with the change of output units

squashing function and the way a prediction is considered correct: if all output unit activations are close enough to their target values, the prediction is correct. Close enough is defined to be an error of less than 0.1% of the total range (2σ). The network is trained until the accuracy has reached 100% and the MSE over all patterns is less than 1e-12.

Reproducing – Generative Mode

The generative mode principle is to feedback the output of a trained network to the next time step input. The major issue of this mode for this application is that errors will accumulate across the sequence and only a network that is very precise could reproduce the pattern signatures up to the last one as presented in the generative mode section below. As a consequence, the stopping criterion of the predictive mode is adjusted: training with the predictive mode for a later application to the generative mode is stopped when all output units have an error of less than 1e-9.

4.3.2 Predictive Mode

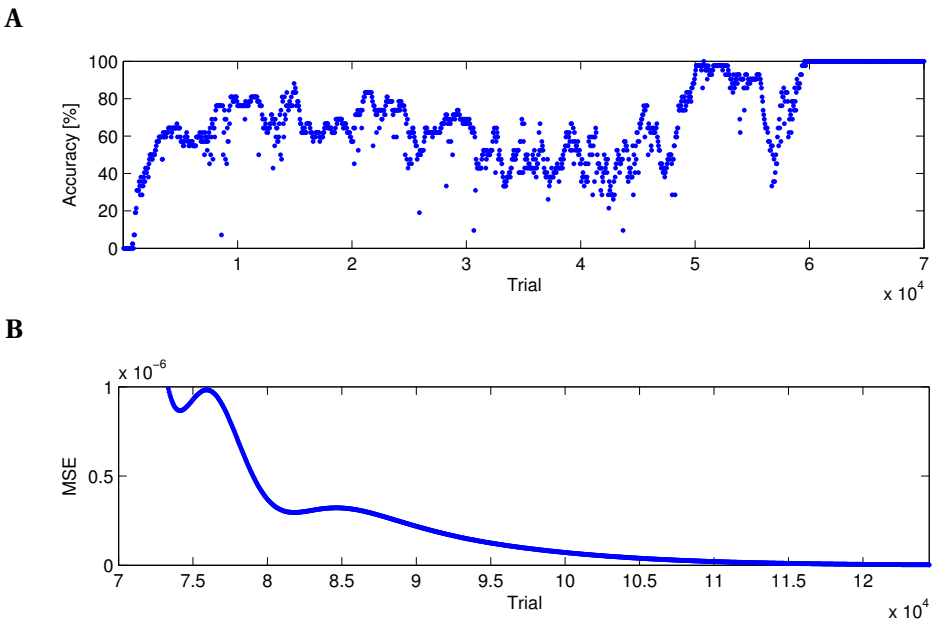


Figure 4.16 – *Prelude CS 1* **A**. Accuracy evolution during first part of training. **B**. Mean square error evolution during training end.

A network topology of 3 MCBs containing each 25 MCs is used to predict the transition in pattern unique identifiers, the 25 first principal components, of the prelude from Bach first cello suite. The predictive mode learns to predict every pattern transition with a very low error after 60'000 trials with a learning rate of 1e-3 (Figure 4.16). The accuracy is an arbitrary criterion that illustrates the network behavior during training; the network tries multiple local minima, where only a part of the pattern signature are *correct*, and is able to get out of them

until it finds the global minimum where output unit activations across the sequence are all very close to their target values. The mean square error is decreasing further after the accuracy has reached 100% implying that the network is further approaching its target values.

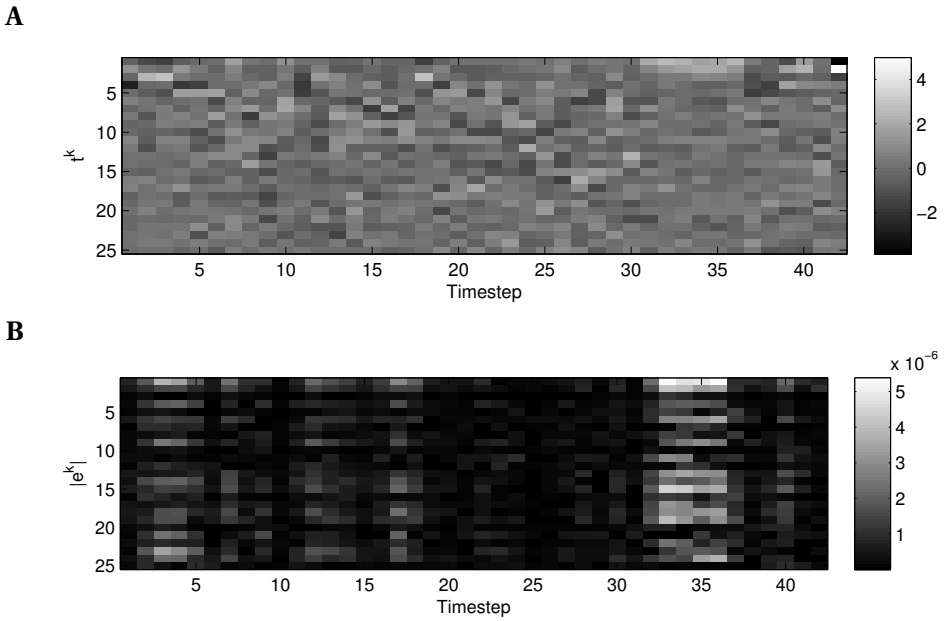


Figure 4.17 – *Prelude CS 1* **A** Target output, the pattern signatures across the sequence. **B** Absolute error between target value and output observed at each output units for a trained network.

Testing the network after training shows that it can predict every coordinates of pattern signatures from the previous pattern with a maximal error of 5e-6 (Figure 4.17).

4.3.3 Generative Mode

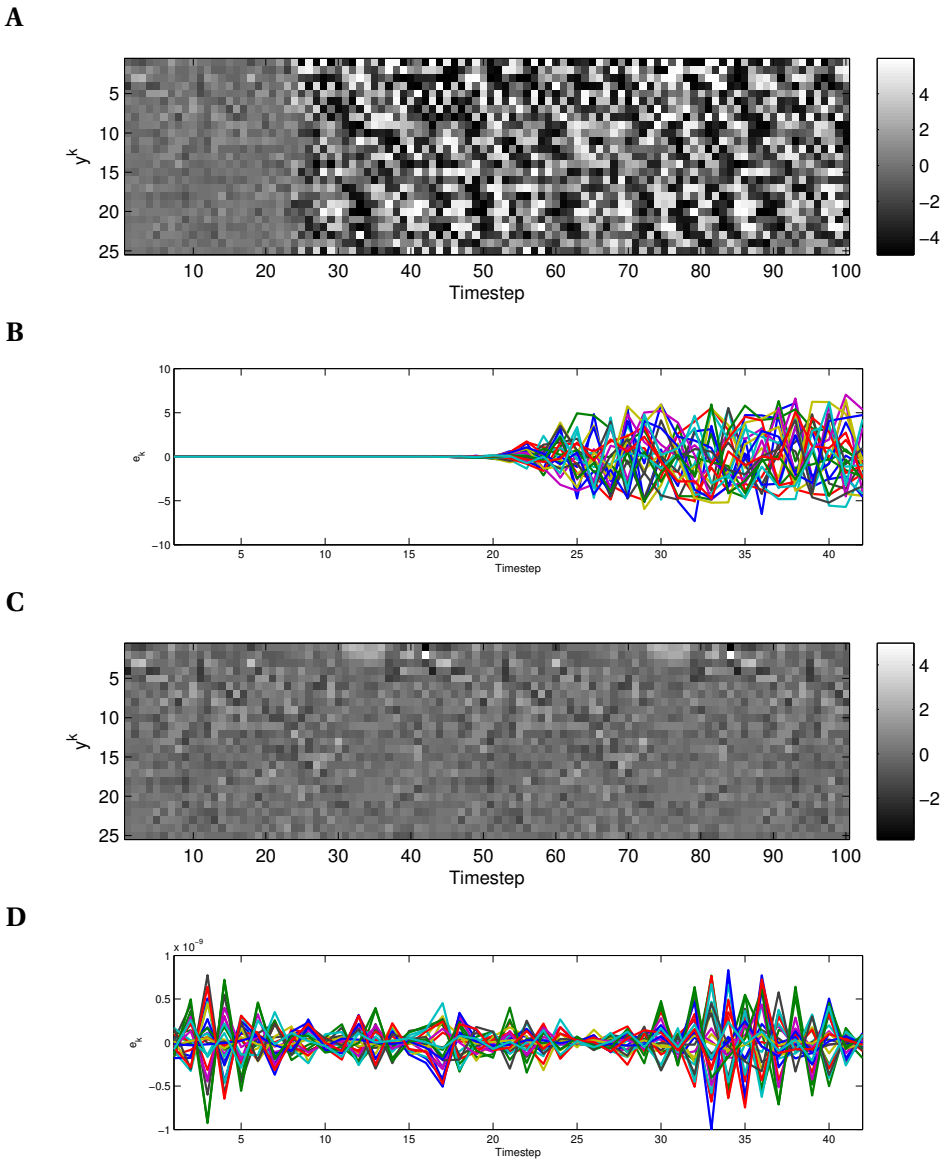


Figure 4.18 – *Prelude CS 1* **A** Network output using the generative mode for a network trained with the predictive mode stopping criteria. **B** Difference for one complete period between the generated sequence and the training target sequence for a network trained with the predictive mode stopping criteria. **C** Network output using the generative mode for a network trained with the generative mode stopping criteria. **D** Difference for one complete period between the generated sequence and the training target sequence for a network trained with the generative mode stopping criteria.

Errors accumulation The errors made at each timestep are propagated to the next predictions in the generative mode. The Figure 4.18 shows the error accumulation for two trained

network. The first one is trained until the predictive mode stopping criterion is reached. One can observed that after the 20th prediction, the difference between the generative mode output and the *true* pattern signature transitions explode due to error accumulation. The stopping criterion consequently needs to be refined. As it has been seen that the accuracy of the network could be further increased (mean square error is still decreasing after the predictive mode stopping criterion is reached), the refined stopping criterion of the predictive mode for the application to the generative mode is set to a slow time scale LSTM network that predicts every patterns with an error of less than 1e-9. Applying this criterion resulted in a network that can reproduce entirely all patterns (Figure 4.18**C** and **D**). Slow time scales LSTM networks are consequently not robust to noise but can be overfitted to the learned data in order to learn the slow time scale transitions in the form of transitions between musical pattern PC values.

Starting seed In order to correctly reproduce the slow time scale sequence with the real valued LSTM and without any starting seed, the cell states should be initialized to the state in which they are at the end of training. A starting seed that bring the cell states to their values at training end is another option. However, it is needed to give the learned sequence to the network for several full periods of the slow time scale sequence to reach the cell states that allows for perfect recovers of the song. The goal being to have a network that could reproduce a learned sequence independent from the sequence itself, this procedure is to be avoided. In the fast time scale network, this problem is solved by introducing a sequence start tag.

Combining slow and fast time scale networks – Full generative mode As expected by the independent analyses presented in the previous sections, when the output of trained slow time scale networks operating in the generative mode is fed to trained fast time scale network salso operating using the generative mode, the *Prelude CS 1* dataset is reproduced with an accuracy of up to 92% using the best performing networks for each time scale.

5 Creation of New Musical Patterns – Music Composition

I explained how long short term memory networks combined with the separation of time scale algorithm are able to learn complex and long musical sequences. After the training phase, the fast time scale network is able to predict up to 96% of the musical sequences when given proper biases toward the patterns in the form of principal components.

In this chapter, I study the effect on trained fast time scale network outputs when unlearned external biases are applied. This will ultimately lead to the creation of new musical patterns

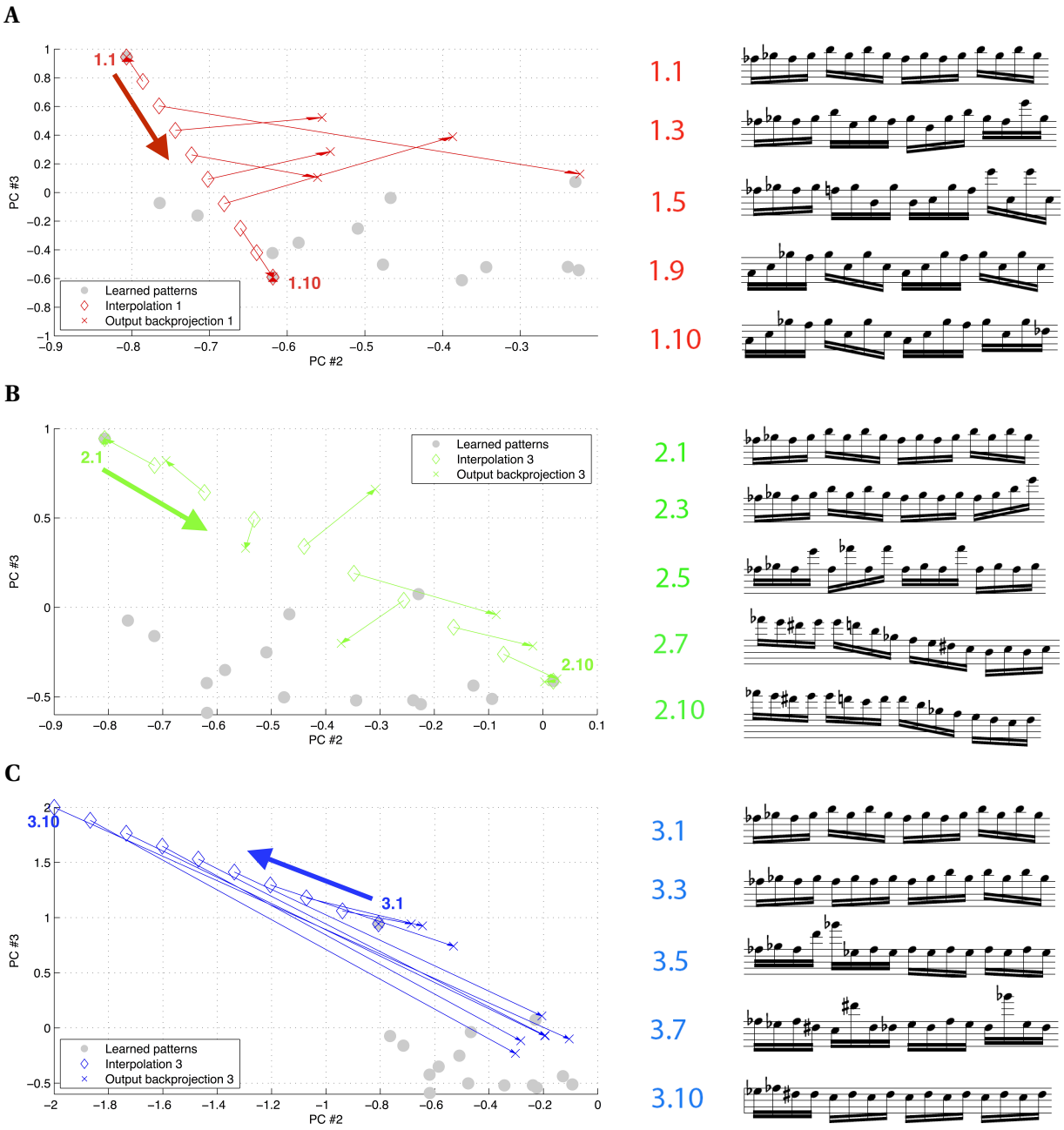


Figure 5.1 – *Prelude CS 1* Right: representation in the principal components 2 and 3 of the learned patterns, the interpolation coordinates and the resulted patterns. Left: Network output in musical notation for a selection of interpolation coordinates. **A** Interpolation 1 results, 10 steps from pattern 13 to pattern 16, which are close in the first principal component dimension. **B** Interpolation 2 results, 10 steps from pattern 13 to pattern 24, which are far in the first principal component dimension. **C** Interpolation 3 results, 10 steps from pattern 13 to a created pattern that stand in a region where no pattern stands in the first 3 principal component dimensions.

Pattern interpolation — exploring new regions in the principal components space The Figure 5.1 shows the results obtained when linear interpolations between selected patterns of the *Prelude CS 1* dataset are fed to the fast time scale network as external biases. Linear interpolation is made on all 25 principal components. All interpolations start from the 13th pattern. The interpolation 1 goes to a close pattern in the first principal components (pattern 16), interpolation 2 goes to a much less similar pattern (pattern 24) and the last interpolation goes toward a non existing pattern in a region where no trained patterns exists in the first 3 components. The interpolations consist of 10 steps (1.1 to 1.10, 2.1 to 2.10 and 3.1 to 3.10 for interpolation 1, 2 and 3 respectively) with the first and last interpolations being learned patterns for interpolations 1 and 2, while only the first interpolation (3.1) is a learned pattern for interpolation 3. The output musical pattern of the network is then projected back onto the PCs to visualize it. It is very interesting to see that pattern signatures in-between learned patterns resulted in the creation of new patterns that were not learned instead of collapsing onto already known points in the PC space, i.e. previously learnt patterns. More interestingly, these patterns sound as they could be part of the dataset, meaning, they could have been produced by Bach. In addition, the interpolation 3, which explore a region completely out of any receptive fields, produced pattern that sound less and less as Bach music as the coordinates increase away from any region with learnt patterns. These patterns, when back projected in the principal component dimensions, however, stand in region where patterns are. They are thus similar to other patterns but the production rules that governed their production are not in the Bach style.

Note selection for unlearned pattern signatures Note selection is changed with *a priori* knowledge about the dataset. Indeed, it has been observed that the trained networks are highly overfitting the data and that the same mechanisms that allows learning of such complex and long sequences are a withdrawal to pattern composition; a non learned external biases close to a learned one, would result in a slight change in the network unit receptive fields that could impact strongly on the output unit activations. For example, the interpolation closer to the learned patterns (1.2, 1.8 and 1.9) produced outputs that are similar to the closer learned pattern. However, the output unit activations are lowered due to the receptive field change. The contrary is observed for pattern signatures being far from any learned patterns, signatures that are out of any unit receptive fields, resulted in very high output unit activations for all output units. To asses this problem, the note selection for the interpolation experiments takes *a priori* knowledge about the dataset that consists in knowing that the majority of the patterns are monophonic and only the most active output unit is transformed as an activated pitch. Applying this note selection prevents composition of polyphonic new patterns, however, it is hypothesized that if the dataset included more polyphonic patterns, then receptive fields for polyphonic patterns would be present and the standard note selection could be used to produce new patterns.

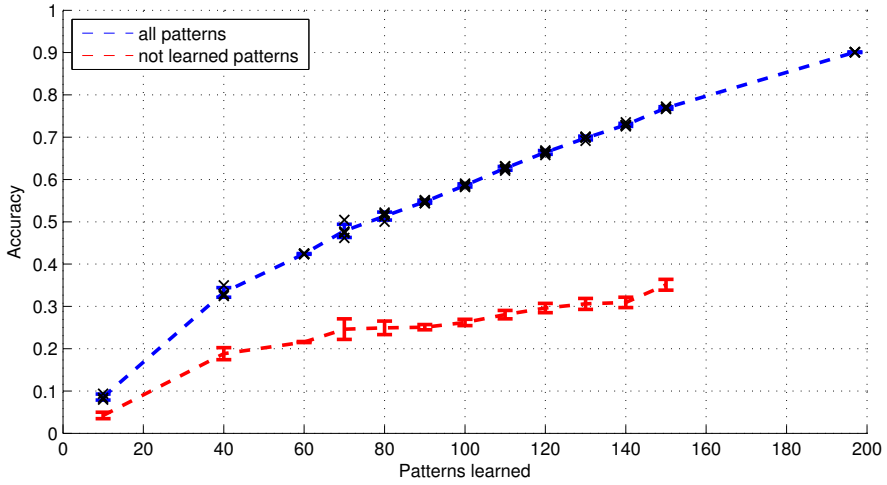
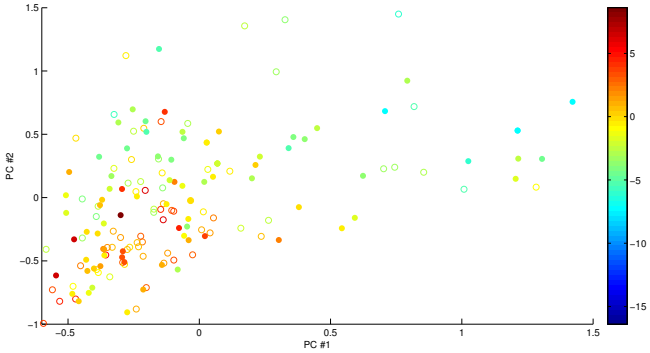


Figure 5.2 – *Preludes CS 1,3 and 4*

Generalization to non-learned patterns – Inferring patterns A network that has learned half of randomly selected patterns of the *Preludes CS 1,3 and 4* dataset is tested on the all dataset to observe its behavior with respect to the non learned patterns. The non learned patterns are correctly predicted for up to 35% (Figure 5.2). A network that has learned to predict correctly 90% of half of the patterns is able to predict correctly more than 25% of the remaining patterns. This observation can be explained by the similarity between patterns and the receptive fields of the learned patterns. A non learned pattern that stands in-between learned patterns and is part of multiple receptive field would activate the unit with respect to the weighted contribution of all receptive fields it is standing in and thus be predicted using a mix of the close patterns production rules, which in turn are the right production rules for up to 35% of the non learned pattern (Figure 5.3 shows the receptive field of two selected memory cells, all unit receptive fields are showed in the Appendix).

A



B

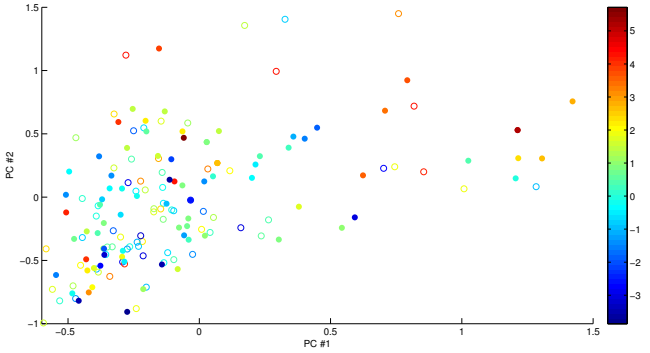


Figure 5.3 – *Preludes CS 1,3 and 4* Memory cell net activation due to the external biases after training until 90% of 100 randomly selected patterns. The empty circles correspond to non learned patterns while the filled circles correspond to learned pattern. **A** Memory cell number 8 of the third block. **B** Memory cell number 2 of the fifth block.

Conclusion

The present study demonstrated how long short-term memory (LSTM) artificial neural networks (ANNs) could be adapted to the specific task of learning and production of musical sequences. I presented two models for musical sequence learning with LSTM networks: the input-note-output-note (INON) and separation of time scales models. The first consists in applying the learning algorithm of LSTM networks introduced by Gers *et al.* [12] to predict each upcoming note from the previous one (note transition) given a history of notes presented to the network in a temporally sequential manner. The note history is reflected in the activation of the hidden units of trained LSTM networks. This model is referred as the INON model as both input (the current note) and output (the upcoming note for a trained network) are notes in the network representation. The separation of time scales model has been introduced, since the INON model failed to learn longer and more complicated musical sequences, e.g. Bach cello suites. Its basic principle relies on the separation of music time scales into slow and fast components with one LSTM network trained on the slow time scale transitions and another trained on the fast time scale transitions. The training of the separation of time scales model then consists of the fast time scale LSTM network learning the note-to-note transitions for a given musical sub-pattern belonging to a longer sequence, while the slow time scale LSTM network learns the transitions between musical pattern, linking them together into the whole sequence. The slow time scale network is working on top of the fast time scale network and informs the latter of which musical pattern it is currently learning or producing. The separation of time scales model has been shown to be very efficient at learning complex and very long musical sequences, here the cello suites from J.S. Bach. In addition, using the generative mode on trained fast and slow time scale LSTM networks, I was able to reproduce with very high accuracy Bach's musical sequences.

In the following, I will summarize the key findings of the current work on musical sequences learning and (re)production with LSTM ANNs and discuss open questions for several points.

Why LSTM networks? LSTM recurrent neural networks (RNNs) are capable of solving complex sequence learning tasks that were considered hard or even unlearnable by traditional RNNs [12]. Hochreiter's analysis [18] showed that conventional RNNs are suffering from exponential grow or vanishing error signal through time. Hochreiter and Schmidhuber solved this problem by creating RNNs that satisfied the condition for a constant flow of the error signal, the LSTM networks [5]. This central feature of LSTM networks allows them to bridge arbitrary time intervals between input events and their target output signals [5]. Since musical sequences have long time structures, LSTM ANNs were chosen to be applied on the specific task of musical sequence learning.

Music representation in LSTM networks To be able to apply LSTM networks to the task of musical sequences learning, I presented a way to represent music in these networks and to extract back notes from the network output. The music representation in LSTM ANNs is chosen to resemble the input from the lowest auditory neurons in the cochlea (tonotopic representation of the hair cells). From the harmony point of view, each pitch present in the musical sequence that is learned is associated with one single input and output unit of the ANN. From the temporal point of view, each musical sequence is discretized with sixteenth note steps. The network therefore sees a musical sequence as a sequence of activations of sixteenth notes. Interestingly, I showed that LSTM networks were not only able to learn monophonic note-to-note transitions but also polyphonic sequences, which are simply represented by having multiple input/output units active at the same time step in the chosen music representation. This music representation carries most of the information a music sheet has. However, no difference can be made from a sustained note from repeated ones. I expect that this information could not be discarded for faithful (re)production of some music styles and that it is an inherent property of music composition. Therefore, I suggest to include the information about note duration for future works.

Training phase The training phase for all LSTM network models consists in sequentially adapting the connection weights using the LSTM networks learning rules [12]. I chose to train LSTM networks from all models to learn music by predicting every time steps of a sequence from the previous one. The target outputs of the INON and fast time scale LSTM networks are notes in the network representation, while the target outputs of the slow time scale LSTM networks are the unique identifiers of musical patterns. The training phase of INON LSTM networks therefore consists in training the networks to output each upcoming note from the previous one, which is given as input. The training phase of the fast time scale LSTM networks is relatively similar to the one of INON networks. However, in addition to a note as input to the network, the unique identifiers of the current musical pattern to be learned are added. The training phase of the slow time scale networks is, for its part, completely different. Indeed, the training phase consists in training slow time scale LSTM networks to learn the transitions between the unique identifiers of each musical pattern, instead of note transitions. What is common to training phases of all models is that these training schemes are applied for all steps

in the sequence to be learned until the accuracy is maximized or reached a threshold. Several different training parameters could be changed for different application. For example, I expect that for an application focused on music composition, the accuracy of trained networks should not go further a certain level to prevent the overfitting effects observed in the current work, which was focused on accurate learning of musical sequences. Differential learning rate could also be applied in order to modulate the impact on learning from different time steps. Finally, one major limiting factor being the computation time of the training phase, I expect that meaningful pre initialization of weights could be efficient to stabilize and improve the learning speed as it has been shown by Corrêa *et al.* [30].

Transforming trained LSTM networks for autonomous music production If the output at each time step of (trained) INON and fast time scales LSTM networks is a note in the network representation, this note can be fed back to the network as the input of the next time step. In the current work, I referred to this self-feedback loop as the generative mode. If trained INON and fast time scale LSTM networks can accurately predict every note-to-note transitions of a musical sequences, applying the generative mode resulted in exact reproduction of the learned musical sequence. The same was applied to very accurate slow time scale networks and resulted in the reproduction of every musical pattern identifiers in the correct order. Combining the slow and fast time scale LSTM networks in the generative mode reproduced correctly more than 90% of Bach's prelude of the first cello suite, which is composed of 672 notes in the network representation.

Time scale hierarchy in LSTM networks The mechanisms deployed by LSTM networks to learn non-Markovian musical sequences were shown to be based on the time scale hierarchy across memory cell blocks (MCBs) where each MCB is working on its own intrinsic frequency. By this mechanisms, memory cells (MCs) from slower MCBs are encoding most of the information related to the long time structure of the sequence, while MCs from fast MCBs are mainly dedicated to notes production. Indeed, forget gates from fast MCBs are often resetting their MC states to be able to forward the right states to output units so that the latter produce the correct notes, while the forget gates from slow MCBs are typically never resetting and constant in order to inform fast MCs of the long time structure of the sequence. This mechanism is very similar from the one proposed for the separation of time scales model and was actually the inspiration for the creation of this model. Furthermore, I showed that the initialization of LSTM networks proposed by Hochreiter and Schmidhuber [5] and extended to forget gates by Gers *et al.* [12] prepared a favorable environment to the time scale hierarchy across MCBs. Indeed, by gradual increase of forget gate biases and decrease of input and output gate biases, it is forced the serial recruitment of MCs across MCBs during training, therefore externally helping LSTM networks to separate time scales and consequently to learn non-Markovian sequences. Bidirectional LSTM networks have been shown to significantly increase effectiveness of speech recognition tasks [6]. Because the task of phoneme classification highly depends on contextual information as music is, I expect that bidirectional LSTM networks

could also show a significant improvement in the performance for musical sequence learning tasks. Furthermore, introduction of *peepholes* connections from the inner cell states of MCs toward gates allow them to access the state of their MCs even when the output gate is closed and has been shown to increase performance of precise timing prediction [31]. As music is build on precise timing of notes and output of notes at given times may depends on internal states rather than only the seen output, I suggest that the current work could strongly benefit from such peephole connections. Finally, I also expect that multilayer LSTM networks could solve more complex tasks by adding supplementary control/memory mechanisms as reflected by the recent work on protein secondary structure prediction of Sonderby *et al.* where multi-layer bidirectional LSTM networks performed significantly better than other state of the art classifiers [15].

Separation of time scales model The separation of time scales model was shown to be able to learn very complex and long musical sequences extracted from Bach cello suites. It has been constructed so that its generalization capacity is maximized. The time scale of music is separated in two components. Slow time scale LSTM networks inferred the slow time scale structure of the musical sequences to be learned to fast time scale LSTM networks, which role is to produce the note-to-note transitions knowing the slow time scale. To represent the slow time scale of musical sequences it has been chosen to associate a musical pattern (one bar of the sequence to be learned) to principal components (PCs) from a PCA applied on the whole collection of musical patterns that are wanted to be learned. Slow time scale LSTM ANNs were able to predict and autonomously reproduce all pattern transitions in the form of real valued PCs from Bach's prelude of the first cello suite. The idea of the separation of time scales model is to forward the pattern unique identifiers from slow time scale networks to fast time scale LSTM networks so that the latter knows which pattern to produce. Training fast time scale networks to predict every note-to-note transition from the previous note and the pattern unique identifiers from slow time scale networks was shown to be fast and very efficient. Furthermore, the forward connections from the slow to the fast time scale network have been associated with recalling of musical pattern from a compressed memory of it. I should mention that although the slow time scale network learned the transitions between each pattern identifiers, the moment fast time scale networks access the next compressed representation of musical patterns is artificially set. Indeed, the next musical pattern identifiers are fed to trained fast time scale networks after they have produce as many note-to-note transitions as there are in one full pattern (16 notes for 1 bar length patterns). Therefore, I externally chose how long the fast time scale LSTM networks should recall the compressed memory of the pattern they are playing. For a fully autonomous model, I suggest to add to the learning phase of either time scale networks the information about when to switch to the next musical pattern.

How fast time scale LSTM networks learn multiple musical patterns For the special case of Bach's prelude of the first cello suite, I demonstrated that the first 25 PCs taken as external biases to fast time scale LSTM networks, are enough to be able to retrieve more than 95% of the original musical sequence composed of 42 musical patterns. Therefore, the separation of time scales model, in addition to solve the problem of complex and long musical sequence learning, apply a dimensionality reduction on the representation of patterns. I showed that each network unit connected to the external biases of trained fast time scale networks developed a receptive field (RF) to the pattern identifiers in the PC spaces. The networks are then taking advantage of these RFs to accurately predict each note transition of every learned musical pattern. This mechanism is similar to the associative memory of Hopfield networks [11] and I sowed that fast time scale LSTM networks effectively learn how to recall each musical pattern from their compressed identifiers, akin to feedforward inputs from brain memory regions. I suggest that other, even non-linear, compressed representation of musical patterns or high level features of music could be used and would yield similar results.

Slow time scale LSTM networks are not robust to noise I demonstrated that slow time scale LSTM networks should predict very precisely (i.e. error has to be less than 1e-12) every PC values of a pattern to be able to autonomously reproduce the pattern transitions. Indeed, on the contrary from the generative modes of INON and fast time scale networks, the generative mode of slow time scale networks does not filter the outputs. Therefore, the errors made by generative slow time scale LSTM networks are accumulated across time and only very accurate, consequently overfitted, slow time scale networks does not suffer from this error accumulation. To address this problem, I suggest to train slow time scale LSTM networks on noisy PCs or use other models, more robust to noise, as winnerless competition networks [32]. Finally, I propose to test other time scale separation methods such has using deep neural networks to encode and decode a musical sequence or pattern. Indeed, deep neural networks were recently found to be efficient to uniquely identify a sequence of word in order to translate it from French to English by using a simplified version of LSTM networks to produce the sentence in either languages [33].

Capacity of fast time scale LSTM networks The fast time scale networks capacity experiments were done by gradual increase of the number of musical patterns to be learned. For each of the capacity experiment, learning was stopped when the network accuracy was over 0.9. For all presented network topologies, fast time scale LSTM networks were always able to reach the stopping criterion. Impressively, the number of trials needed to complete the training sessions was only slowly increasing with the number of learned musical patterns. Furthermore, when the dataset was increased from 42 to 197 patterns, the number of training trials was not significantly different (p-value=0.0747). However, it was needed to increase the number of external biases from 25 to 70 respectively. These observations indicate that the fast time scale model is a robust and very efficient model to learn multiple and complex musical patterns and that aside from the training time, which was around 2 days on the LCN servers to

learn all 197 patterns of the second dataset, the capacity of fast time scale networks was not shown to be limited. Further experiments with bigger datasets or different stopping criteria should be done to better observe the capacity of the network.

Exploring the receptive fields of trained fast time scale LSTM networks In the last chapter, I presented how trained fast time scale LSTM networks respond to previously unseen external biases. Untrained external biases with values comprised in the RFs of fast time scale network units were shown to also activate the network units and that this activation was similar to a mix of activations due to the closest learned external biases. However, the responses of the network units to these unseen external biases were lowered for external biases close to learned ones and boosted for external biases far from the RF centroids. This affected highly the network output and a new filter for note selection from the output unit values of fast time scale LSTM networks was implemented using *a priori* knowledge about the musical patterns. Using this novel note selection filter, I showed that unseen external biases with values in close range from trained external biases resulted in the production of the exact same musical pattern as the one associated with the closer trained external biases. More importantly, external biases comprised in a region where multiple RFs are overlapping resulted in the production of new musical patterns in the same style as the learned musical sequence. Finally, when fast time scales LSTM networks were trained on half of the musical patterns, using the new filter for note selection, I demonstrated that the network was able to correctly reproduce more than 25% of the patterns from the other half thanks to the RF mechanism.

A Complementary Figures

Here, I present the receptive field to the external biases of all network units.

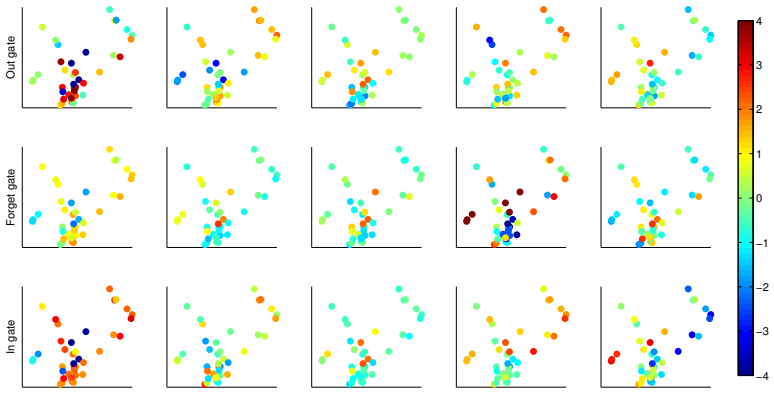


Figure A.1 – *Prelude CS 1* Gate net activations in response to external biases for a trained network with topology 5MCB20MC (96.6% accuracy).

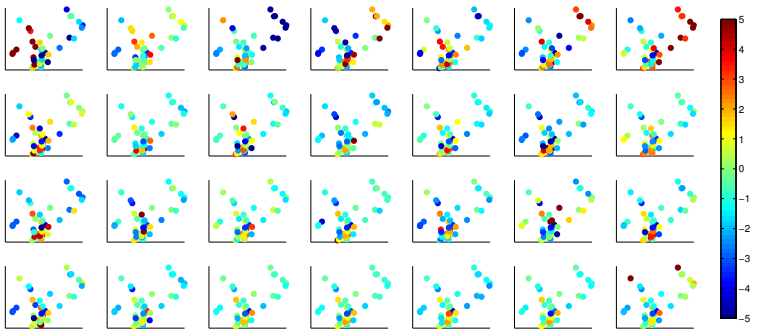


Figure A.2 – *Prelude CS 1* Output unit net activations in response to external biases for a trained network with topology 5MCB20MC (96.6% accuracy).

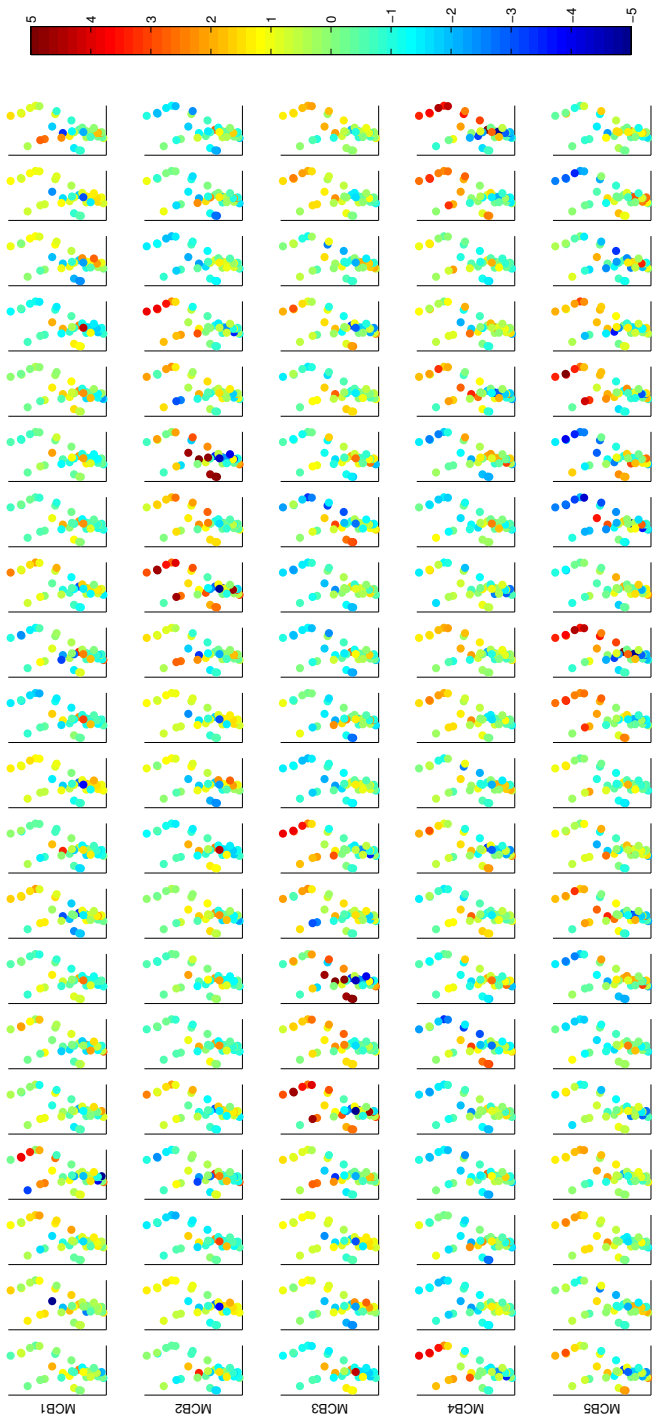


Figure A.3 – *Prelude CS 1* Memory cell net activations in response to external biases for a trained network with topology 5MCB20MC (96.6% accuracy).

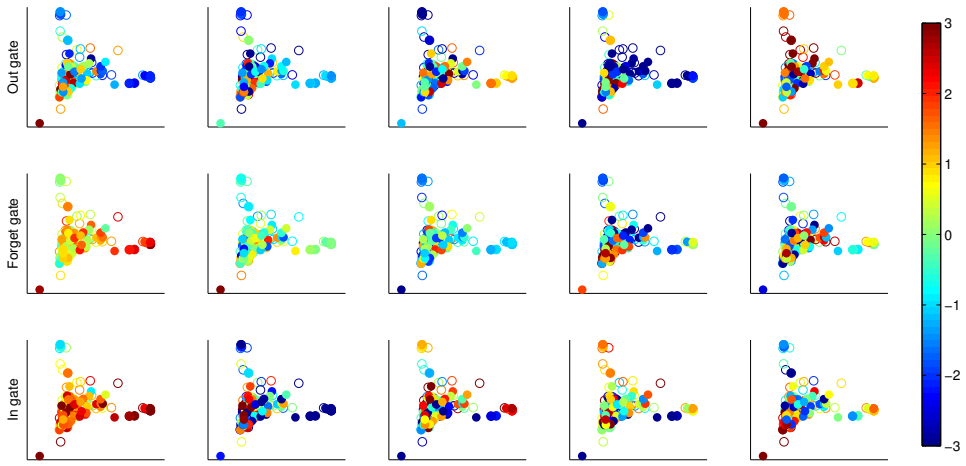


Figure A.4 – *Preludes CS 1,3 and 4* Gate net activations in response to external biases for a trained network on 100 randomly chosen patterns (90% accuracy). Filled circles represent learned patterns, empty circles represent non learned patterns.

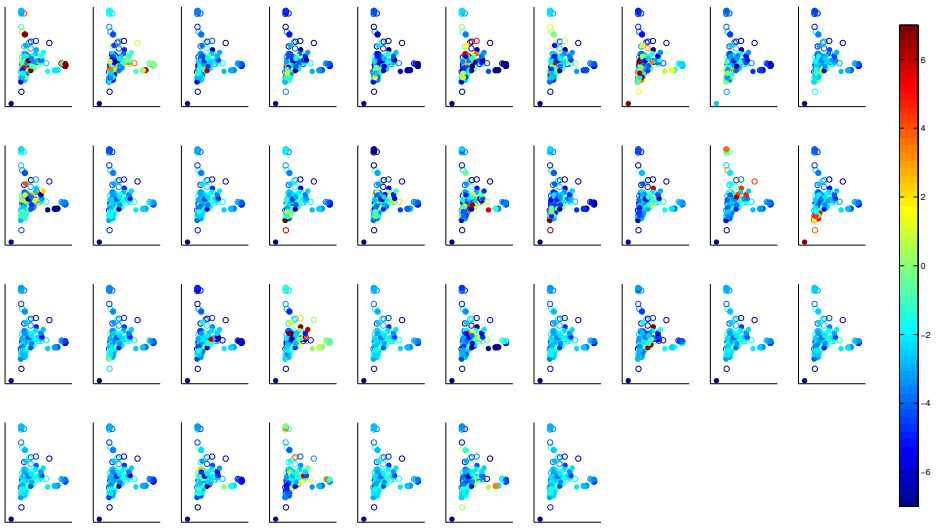


Figure A.5 – *Preludes CS 1,3 and 4* Output unit net activations in response to external biases for a trained network on 100 randomly chosen patterns (90% accuracy). Filled circles represent learned patterns, empty circles represent non learned patterns.

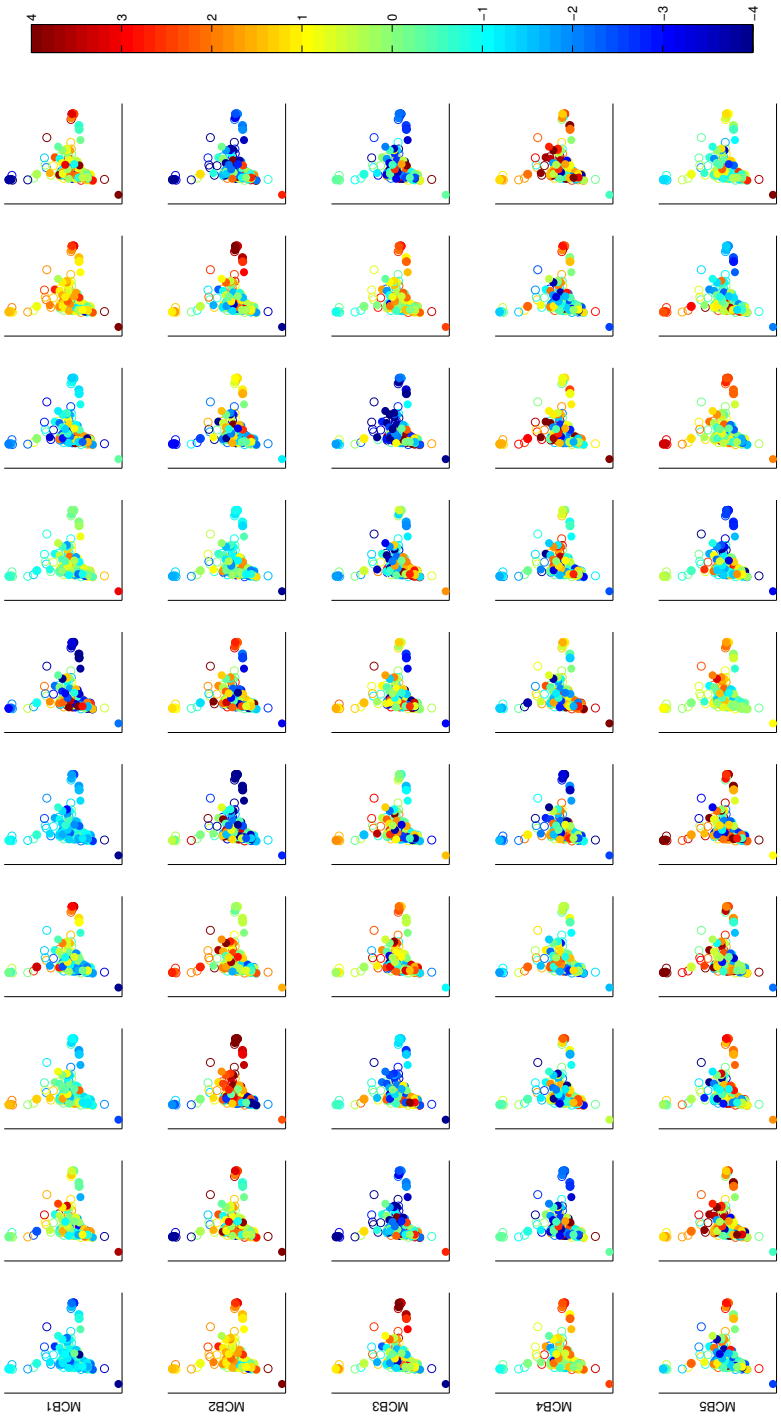


Figure A.6 – *Preludes CS 1,3 and 4* Memory cell net activations in response to external biases for a trained network on 100 randomly chosen patterns (90% accuracy). Filled circles represent learned patterns, empty circles represent non learned patterns.

Bibliography

[1] Pierre Boulez, Jean-Pierre Changeux, and Philippe Manoury. *Les Neurones enchantés: Le cerveau et la musique*. Odile Jacob, 2014.

[2] Douglas R Hofstadter. *Godel, Escher, Bach*. Penguin, 2000.

[3] Luigi F Menabrea and AA Lovelace. Sketch of the analytical engine invented by charles babbage, esq., by lf menabrea, of turin, officer of the military engineers. *Translated and with notes by AA L. Taylor's Scientific Memoirs*, 3:666–731, 1843.

[4] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Back-propagation: Theory, architectures and applications*, pages 433–486, 1995.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[7] Alex Graves and Juergen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 545–552, 2009.

[8] Stefan J Kiebel, Jean Daunizeau, and Karl J Friston. A hierarchy of time-scales and the brain. *PLoS computational biology*, 4(11):e1000209, 2008.

[9] Steven Greenberg and Brian ED Kingsbury. The modulation spectrogram: In pursuit of an invariant representation of speech. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 3, pages 1647–1650. IEEE, 1997.

[10] Dale Purves, George J Augustine, David Fitzpatrick, William C Hall, Anthony-Samuel LaMantia, James O McNamara, and Leonard E White. Neuroscience. *Sunderland, MA: Sinauer Associates*, 3, 2001.

[11] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[12] FA Gers, J Schmidhuber, and F Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Comput.*, 12(10):2451–2471, 2000.

[13] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868, 2009.

[14] Felix A Gers and Jürgen Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *Neural Networks, IEEE Transactions on*, 12(6):1333–1340, 2001.

[15] Søren Kaae Sønderby and Ole Winther. Protein secondary structure prediction with long short term memory networks. *arXiv preprint arXiv:1412.7828*, 2014.

[16] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 1988.

[17] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[18] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, 1991.

[19] Alex Graves. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[20] RB Dannenberg. Music representation issues, techniques, and systems. *Comput. Music J.*, 17(3):20–30, 1993.

[21] PM Todd. A Connectionist Approach To Algorithmic Composition. *Comput. Music J.*, 13(4):27–43, 1989.

[22] Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, pages 747–756. IEEE, 2002.

[23] Judy A Franklin. Computational models for learning pitch and duration using lstm recurrent neural networks. In *Proceedings of the Eighth International Conference on Music Perception and Cognition.(Society for Music Perception and Cognition, Evanston, IL, 2004)*, 2004.

[24] Tomasz Oliwa and Markus Wagner. Composing music with neural networks and probabilistic finite-state machines. *Appl. Evol. Comput.*, pages 2–7, 2008.

[25] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[26] Douglas Eck and J Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Ist. Dalle Molle Di Stud. Sull Intelligenza . . .* , 2002.

[27] Sergio Albeverio, Jianfeng Feng, and Minping Qian. Role of noises in neural networks. *Physical Review E*, 52(6):6593, 1995.

[28] Johanni Brea, Walter Senn, and Jean-Pascal Pfister. Sequence learning with hidden units in spiking neural networks. In *Advances in neural information processing systems*, pages 1422–1430, 2011.

[29] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2005.

[30] Débora C Corrêa, A Levada, and José Hiroki Saito. Stabilizing and improving the learning speed of 2-layered lstm network. In *Computational Science and Engineering, 2008. CSE'08. 11th IEEE International Conference on*, pages 293–300. IEEE, 2008.

[31] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *The Journal of Machine Learning Research*, 3:115–143, 2003.

[32] Valentin S Afraimovich, Mikhail I Rabinovich, and Pablo Varona. Heteroclinic contours in neural ensembles and the winnerless competition principle. *International Journal of Bifurcation and Chaos*, 14(04):1195–1208, 2004.

[33] Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares Holger Schwenk Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Colombo Florian

Birthdate: 20.03.1990

florian.colombo@epfl.ch
florian.colombo@gmail.com
Phone: +41 79 454 30 31

Av. Riant-mont 8
CH-1004 Lausanne

Formation

Neural Networks
Programing
Neurosciences

Master in Life Sciences. Neurosciences Orientation. 2012-2015
Minor in Computational Neurosciences
Lausanne Federal Institute of Technology (EPFL)

Signal Processing
Data Analysis and Classification

Bachelor in Life Sciences 2009-2012
Lausanne Federal Institute of Technology (EPFL)

Experience and Research Activities

Music encoding
Long sequence learning

Music Learning with Long Short Term Memory Networks 2014
Master Thesis. *Laboratory of Computational Neurosciences, EPFL*
Supervisor: A. Seeholzer. Professor: W. Gerstner

C++ Programing
Neural Networks

Implementation of the Long Short Term Memory Recurrent Neural Network for Sequences Learning 2014
Semester Project. *Laboratory of Computational Neurosciences, EPFL*
Supervisor: A. Seeholzer. Professor: W. Gerstner

Dynamical systems
Team work

Improvement of an Ultrasonic Generator for Lithotripsy 2013
Electro Medical Systems, Nyon

Virtual reality
Cognitive Neurosciences

Full Body Illusion and Proprioceptive Drift Measure with Virtual Reality. Implementation of a 3D Live Representation of Self using The Microsoft Kinnect 2012
Bachelor Thesis. *Laboratory of Cognitive Neurosciences, EPFL*
Supervisor: B. Herbelin. Professor: O. Blanke

Teaching
Supervising

Teaching Assistant at EPFL
C++ Programing course given by J.Sam 2011-2012
Biochemistry course given by C. Vandevyver 2013
Analysis II course given by F. Margot 2014
C++ Programing course given by M-O Gewaltig and J. Rougemont 2014

Extra Academic Activities

EPFL/UNIL Student Chamber Orchestra (SChO) 2013-…
Funder and president of the association

Musical Activity as a Cellist 1997-…
Certificate of musical studies (AVCEM)
Many Chamber Music, Orchestra and Solo Representations

Computer Science

C++, Matlab, Labview, OpenGL