DEEP LEARNING FOR MUSICAL SCENARIO INFERENCE AND PREDICTION

Application to structured co-improvisation

THÉIS BAZIN

ENS Cachan, Université Paris Saclay

Under the supervision of

PHILIPPE and JÉRÔME

IRCAM, UMR STMS 9912
Paris 75004, France
Équipe Représentations Musicales

Master's degree
SAR/ATIAM
Faculty of Computer Science
Université Pierre-et-Marie Curie / IRCAM / Télécom ParisTech

March–July 2016

ABSTRACT

The field of *musical scenario inference* aims at developing systems and algorithms to *automatically extract abstract temporal scenarios in music*. We call *scenario* any underlying symbolic sequence that constitutes a higher-level abstraction of an original input sequence. Such an underlying sequence implicitly encodes the *temporal relations* between events in a musical piece by producing an ordered series of symbols. Musical works exhibit temporal dependencies at *multiple time-scales*, from *local* melodic events to *long-term* harmonic progressions.

Multiple systems have been introduced in order to capture short or long term dependencies between musical events. Nonetheless, existing systems fail at taking into account the interactions between these various time scales.

In this research project, we propose a method to tackle this issue and infer abstract scenarios through the use of *deep recurrent neural networks*. We introduce a system that is able to extract an abstract sequence of symbols from an input musical sequence, as well as perform predictions on the probable continuations of this sequence.

A theoretical application to the *co-improvisation problem* is introduced. Co-improvisation engines seek to *generate new sequences resembling* some example input sequence. A crucial aspect of such co-improvisation systems is the ability to introduce *anticipations*, so as to generate *transitions* between different parts. This requires knowledge of some underlying scenario to the generation. Existing systems that offer prediction capacities rely on a pre-defined abstract scenario. The architecture we propose would improve on this by replacing this pre-defined scenario with one *automatically inferred in real-time* by our scenario inference and prediction tool, incorporating *dynamically refined short-term predictions* over the future. Through dynamic training via *adversarial training*, this system can furthermore improve the accuracy of its predictions in *real-time*.

Keywords— musical scenario inference, machine learning, neural networks, deep learning, recurrent networks, co-improvisation, style modeling

ACKNOWLEDGEMENTS

First and foremost, many thanks to my advisors for tutoring me during this internship.

To Philippe, thank you for your fine insights into deep learning, which I'm sure will benefit me in the years to come – and for the nice cat T-Shirts whose pictures brought me a decent share of likes on Facebook ©.

To Jérôme, I am grateful for your continuous efforts in helping not lose focus of the musical task at hand and keep the concepts clear and precise. Thank you, too, for the occasional wise word on keeping a cool head in the (at times) stressful world of academia – and for the nice dance animation, though I'm not really sure if you are to be thanked on that one.

Thanks to the ATIAM pedagogical team for the large overview of the field of musical sciences you offered us during this semester.

A very special thank you to Cyrielle for allowing us to grab food from the delicious buffets so many times.

Thanks to my tutor at ENS Cachan, Hubert Comon, for his most helpful and reassuring availability.

Many thanks also to Jonathan Laurent for partially proofreading this report as well as very helpfully proofreading many of my life choices. I wish you all the best at CMU!

Thanks at last to my mother, without whom I would not be where am I now. (Or anywhere else, for the matter.)

CONTENTS

LIST OF FIGURES

ACRONYMS

LSTM Long Short-Term Memory

MLP Multi-Layer Perceptron

NLP Natural Language Processing

NN Neural Network

RNN Recurrent Neural Network

SGD Stochastic Gradient Descent

VAE Variational Autoencoder

Part I

INTRODUCTION

In this introduction, we first present the general problem of musical *underlying abstract sequence inference*. We then present an overview of this field of research, from formal methods to statistical approaches.

Finally, we focus on the techniques applied in this research project and propose an introduction to the key *machine learning* concepts at stake.

1 CONTEXT

Computational music analysis aims at offering ways of extracting *semantical* information from either symbolic or signal-level music.

Various approaches to this analysis exist [45], with varying goals. These range from the study of the instantaneous evolution of *concrete sound properties* such as timbre, e. g. via Fourier transforms or, more generally, audio features, to more *formal* approaches, focusing on the extraction of underlying *high-level representations* which represent an *abstraction* of the analyzed music.

In the context of the present research project, we focus on this second group of approaches and seek to provide *methods to extract abstractions from raw music*. We will therefore consider musical structure as being *any sequence of symbolic labels* which constitutes a higher-level representation of an input musical sequence: we call such a descriptive sequence a *scenario*. Thus, a given abstract scenario can be (though potentially not in a unique way) instantiated back into its original sequence. Examples of such musical scenarios include harmonic progressions (e. g. the Blues grid) or, at a more semantical level, functional analysis of chords (e. g. Schenkerian anaylsis [16]).

The extraction of abstract sequential representations can thus be seen first as a musicological tool, offering musical representations [45]. It can also be used to perform structured *predictions*, by proposing some potential continuation(s) to a given sequence based on the computed scenario. Furthermore, it can be used in conjunction with other tools to enhance them with knowledge automatically inferred from this analysis.

These approaches are closely related to those applied in the context of *natural language processing* (NLP) [12], which aims at extracting *meaning* from *syntactic textual information*. The computational advantages of both disciplines are the same: they help machines put some *meaning* and *sense* into syntactic data. Algorithms can then reason and make structured computations on those high-level abstractions.

Applications in NLP include for instance *sentiment analysis* (also called opinion mining), which aims at inferring the opinions (either positive or negative) associated with text fragments, e. g. for market analysis of user reviews on Amazon products or trend analysis on Twitter topics. NLP language models and algorithms also drive the speech recognition methods used in systems such as Apple's Siri or Google's Google Now, as discussed in the review by Henderson [23].

In effect, techniques applied in NLP can often be applied to the inference of musical scenarios, since both operate on abstract sequences of symbols. This is the case for instance for the tools (deep recurrent neural networks) we apply in our project, which were applied with success to text analysis [49].

Systems for the inference of underlying scenarios in music exist in various forms, some of which are aimed at capturing arbitrary long-term dependencies in music. Yet, they fail at properly structuring and disentangling the various time-scales at which a music piece can unfold.

This is the problem of *multiscale music analysis for scenario inference*, which we propose to tackle through the use of *deep recurrent artificial neural networks* (deep RNNs). These machine learning systems have proved successful at learning temporal pattern hierarchies at multiple scales in the context of NLP [49].

In this respect, we train deep recurrent neural networks (deep LSTMs) to perform *sequence prediction* and *symbolization* on musical sequences. Our system takes as input a sequence of *chroma* vectors and both:

- Predicts (one or more) probable subsequent chroma vectors,
- Symbolizes the chroma via *clustering*.

This indeed infers a representative *abstract sequence* from a concrete musical sequence, incorporating short-term lookahead into the future.

Then, applying this tool, we imagine the prototype for an architecture for co-improvisation. Co-improvisation seeks to learn some notion of sequential structure from a corpus of musical examples and synthesize new musical sequences *compatible* under some chosen criterion with the input corpus and can be used for instance in a live accompaniment context, generating some backing music for a human improviser.

In a co-improvisation context with a live human improviser, two key features of a good system arise:

- The ability to adapt *reactively* adapt to the musician's dynamic parameters, e. g. pitch, volume or note density,
- The ability to perform *anticipations* on the music played by the musician, which allows the systems to introduce smooth transitions, e. g. cadences or modulations, in synchronization with the human improviser.

Existing reactive co-improvisation architectures [3, 37] do not allow the introduction of additional temporal constraints, either short-term

or long-term: they operate at a *completely local scale*. Existing systems for guided co-improvisation with anticipations [39] require some predefined *scenario* that both the musician and the machine are expected to follow. Thus they cannot provide anticipations for an completely free improviser, for which no a priori temporal scenario has been defined.

The system we outline would thus bridge the gap in existing co-improvisation systems with an architecture able to both *reactively* adapt to a musician *and* provide *anticipations* without any prior knowledge about the music played by the musician, thanks to the notions of musical structure learned by our prediction systems on a vast corpus of musical works. Such a behaviour would be more in par with that of a real improviser, who continuously listens to the music played by his fellow improvisers to estimate the current direction of the improvisation and play accordingly.

We also envision an original way of further adapting this to a particular musician's style, using *adversarial networks*, a means of training networks to *generate* examples indistinguishable from a provided distribution of example data. A brief review of *generative networks* and *adversarial training* is proposed.

Note that these applications are still under work and remain somewhat prospective.

We first present a review of the field of computational musical scenario inference. We then introduce the tools applied specifically for this research project. To that end, an introduction to the key machine-learning concepts used is proposed.

After this introduction, we present the *chroma prediction network* we propose. This model allows to infer a sequence of symbols from a sequence of chroma vectors and predict (one or more) probable subsequent chroma vectors. Using it in real-time allows to analyze the music played by a live musician and anticipate his playing through prediction. We also present the techniques applied for the symbolization step of our proposed scenario inference system.

Finally we introduce the prototype for hybrid reactive/structured musical co-improvisation we envision, making direct use of the scenario inference technique we propose. We also present the theoretical style modeling framework we will be developing. To this end, a review of *adversarial training* is proposed.

2

DEEP LEARNING FOR MUSICAL SCENARIO INFERENCE

The field of *musical scenario inference* focuses on both the processing of *symbolic* or *signal* representations of music. It aims at extracting *abstract underlying temporal representation*(s) from given music pieces [45]. These abstract representations can take various forms, as will be seen in the different approaches presented below. Their binding trait is that these representations are expected to encode in a computational form some high-level temporal properties of the musical sequences considered.

2.1 EXISTING APPROACHES

A variety of approaches have been proposed for the inference of musical scenarios. These can be roughly divided into two main groups: approaches based on formal methods on one side and statistical models on the other side.

FORMAL METHODS The first group of approaches entails models and representations which require to some extent a formal model of music. As such, these might prove more capable of providing interpretable insights on the structure of the analyzed music.

Within this field, the work on grammatic structures in music by Lerdahl and Jackendoff [34] is significant. Their *Generative Theory of Tonal Music* develops a generative musical grammar incorporating elements of cognitive science. It aims at reproducing the way a listener unfolds and understands the temporal structure of a musical work.

A similar grammatic approach has been proposed for the automatic extraction of harmonic content, by classifying series of chords in a piece based on a structural and sequential hierarchy. This has been developed in the strongly typed functional programming language Haskell by De Haas et al. [14] and draws on works on context-free grammars for the modeling of Western tonal harmony by Rohrmeier [47].

Even though these methods provide a form of high-level semantics for music generation, they remain inherently limited by the fact that they require strong assumptions on the structure of music itself. Indeed, in the context of Rohrmeier's study for instance, constraining music to follow a generative grammar structure might be too limiting.

STATISTICAL MODELS The second group of approaches is defined by techniques based on statistical models. These are radically different from formal approaches in that they attempt to extract sequential evolutions by *analyzing musical examples* rather than by fixing *a priori* rules on the music. Statistical approaches thus trade the insights offered by strict formal methods for greater flexibility. By imposing less a priori assumptions on the structure of music, those methods are, therefore, more able to extract *unexpected* structure as opposed to methods which are *constructed* to extract musical scenarios that one *specifically* looks for.

Nowadays, musical scenario inference is also commonly used as an application example in general machine learning papers, even by authors not originally from the field of music informatics (e.g. the recent paper by Paiement, Bengio, and Eck [43]). Audio or symbolic music time-series are sequences of n-dimensional vectors, with a large quantity of data available in the form of MIDI or audio files.

Amongst methods of underlying sequence uncovering using classical statistical approaches, *Hidden Markov Models* (HMMs) have been widely popular for their ability to automatically develop a structured (though necessarily finite) memory of their input data. One can mention the work of Paiement, Bengio, and Eck [43], in which they apply HMMs to the problems of chord instantiation (the choice of notes for a given chord) and melodic prediction. Another example is the study by Raphael and Stoddard [46] of the use of HMMs for functional harmonic analysis.

Moving on to more recent methods, a wide number of approaches – including the one presented in this report – leverage *deep learning* techniques. We present an overview of the key concepts in deep learning in Chapter 3. Briefly put, deep neural networks have the advantage, compared to other machine learning techniques, of being very *flexible* in terms of learning [5].

Indeed, they are built around *hierarchical successions of several linear and non-linear operations* and as such are capable of computing highly non-linear functions through *successive non-linear transformations* of their input data. In comparison, Support-Vector Machines rely on a single non-linear kernel and therefore have a more limited capacity in handling complex data.

As a first example of symbolic scenario inference systems leveraging deep learning, we mention the work of Soltau et al. [53]. They use standard neural networks with an ad-hod, hardcoded notion of time to perform temporal structure analysis on music for style inference.

Seemingly even more adapted to the analysis of musical time-series are *recurrent neural networks* (RNNs). These neural networks are built

specifically for the analysis of time-series and try to capture *temporal dependencies* in the sequences they analyze. (More details on these techniques are given in Chapter 4.)

Using recurrent neural networks has the advantage of involving very few assumptions on the analyzed music: the only assumption is that the music should embed *some form* of underlying temporal structure that can be witnessed through statistical regularities. Hence, a system that defines a mechanism for *memory* will be able to extract temporal knowledge and, then, perform inference and predictions on the music.

An example of such work is the paper by Boulanger-Lewandowski, Bengio, and Vincent [10], which makes use of the RNN-RBM model (a kind of *deep generative graphical model*) to tackle the task of *melodic prediction*: "given a sequence of symbolic music, predict the next music vector in the melody".

RNNs give promising results in capturing *long-term* dependencies in music. However they still do not address the issue of *multi-scale* evolutions, which is a crucial issue for the proper analysis of musical scenarios.

This research project is aimed at tackling this issue through the use of deep recurrent networks.

Lastly, promising current results blend statistical approaches with the formal methods described previously: the project MorpheuS[1] [24] aims at mixing traditional machine-learning (iterative optimization algorithms) with elements of formal harmony theory. This goes to show that the two groups of approaches described are not completely disconnected from one another but can benefit from one another.

The approach to scenario inference we propose thus falls within the scope of statistical methods. It makes use of deep recurrent neural networks, namely deep LSTMs.

1 Hybrid machine learning - optimization techniques to generate structured music through morphing and fusion.

3 MACHINE LEARNING

We now propose a selective review of the field of machine learning, by introducing its required general notions.

Then, we focus on the tools applied in our research project, namely *neural networks* and their more modern evolution, *deep neural networks*. First, we present the basic blocks of neural networks, artificial neurons. Then we show how standard neural networks by combining such neurons. Finally, in order to better understand the inner workings of neural networks, we present the example of *convolutional neural networks*, which compute *high-level features* over the data they are fed with.

3.1 GENERAL NOTIONS

In this preliminary section, we introduce the generic notions required for machine learning approaches, from the definition of the problems considered to the general gradient-based training techniques.

FUNCTION APPROXIMATION Most machine learning techniques can be formally seen as a means of *estimating a function*.

That is, given an unknown function $f : \mathcal{F} \to \mathcal{G}$ between two (often high-dimensional) spaces $\mathcal{F}$ (the input space) and $\mathcal{G}$ (the output space), find an estimate $\tilde{f}_\Theta$ of f dependent on a set of parameters Θ.

The parameters Θ can be chosen to be a multi-dimensional vector storing the values of each parameter in the machine learning system, for instance the positions of the centroids in the k-means method or the connexion weights in a neural network.

PROBLEM DEFINITION One often has a high-dimensional input space $\mathcal{F} = \mathbb{R}^n$ and seeks to obtain output values in either $\mathcal{G} = \mathbb{R}^m$ or $\mathcal{G} = \{g_1, ..., g_m\}$, a finite set of abstract *labels*.

Canonical examples for $\mathcal{G} = \mathbb{R}^m$ are *regression problems*, where one tries to approximate a vector function between two high-dimensional real spaces, that is, a *transformation* of the input data. A popular example of regression is *polynomial regression*, a very simple model which attempts to *fit* a polynomial to a set of values, thus allowing to describe them with a limited set of polynomial coefficients as well as perform interpolation on the values of the unknown function underlying the data.

Traditional examples of problems where the output space $\mathcal{G}$ is *finite* are *classification problems*: given some distribution of data, partition

the distribution into a set of *classes*. A popular classification problem is the problem of mapping handwritten digits to the integer they represent. This problem has been well studied within the machine learning community [32], as will be presented in Section 3.4.

METRIC AND ERROR To define and quantitatively evaluate the quality of the approximate function $\tilde{f}_\Theta$, one needs a metric on the output space, i.e. a function $d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$.

Given the metric d on $\mathcal{G}$ and a current value of the parameters Θ, the point-wise *error* on a single data point x in $\mathcal{F}$ is defined as

$$d(f(x), \tilde{f}_\Theta(x))$$

Note that, ideally, one would use a norm $\|\,.\,\|$ on the functional space $\mathcal{F} \to \mathcal{G}$ and try to minimize the quantity $\|f - \tilde{f}_\Theta\|$. This would ideally lead to a perfect estimation, with optimal parameters Θ^* such that $\|f - \tilde{f}_{\Theta^*}\| = 0$, i.e. $\tilde{f}_{\Theta^*} = f$. But performing the optimization on the functional space as a whole is *impossible*, since *one does not have knowledge of* f. This inherent limitation is the reason behind the introduction of the core principle of machine learning, which is defined as *training by examples*.

TRAINING BY EXAMPLES, ERROR MINIMIZATION Approximating the function f (*learning* it) is done by an iterative process of *error minimization* on a given set of *training examples*, for which the value of the input function f is known.

Formally, the training examples are provided as a set of N pairs $(x^i, f(x^i))_i$ with $i = 1 \dots N$ and $x^i \in \mathcal{F}$ for all i.

The *total error*, or *loss*, $\mathcal{L}$ over the dataset is then defined as

$$\mathcal{L}(\Theta) := \sum_{i=1}^{N} d\left(f(x^i), \tilde{f}_\Theta(x^i)\right)$$

The goal of the training process is therefore to minimize the loss $\mathcal{L}$, i.e. trying to find the optimal value Θ^* of the parameters such that

$$\Theta^* = \arg\min_\Theta(\mathcal{L}(\Theta))$$

GRADIENT DESCENT AND LEARNING RATE A standard way of performing this error minimization is through *gradient descent*.

Informally, gradient descent amounts to *blindly* looking for the *lowest* point in a mountains field: because of the absence of visibility, one can only make decisions on which path to follow based on *local information*. Then, gradient descent consists in repeatedly picking the *direction of steepest descent* as the *best direction available*. In that sense, gradient descent is a *greedy algorithm* [9].

The main issue with gradient descent is the fact that a mountains field can very well have numerous valleys, that is, *several local minima*.

Furthermore, based solely on local information, one is *unable to assess whether a given local minimum is a global minimum*. Local minima may thus hinder the algorithm from finding a global optimum.

In mathematical terms: provided that the error function $\mathcal{L}$ is differentiable with respect to the parameters Θ (providing, for any Θ the gradient $\nabla_\Theta \mathcal{L}(\Theta)$), looking for the optimum Θ^* can be done in an *iterative* fashion with elementary parameter updates of the form

$$\Theta_{t+1} \leftarrow \Theta_t - \alpha * \nabla_\Theta \mathcal{L}(\Theta_t)$$

In this formula, α, an external parameter (or *hyper-parameter*) of the training algorithm, is called the *learning rate*. If gradient descent is seen as a means of taking successive steps in the *direction of greatest decrease* of the loss function with respect to the parameters, then α controls the size of those steps.

This method does not necessarily yield the optimal Θ for error functions which have multiple local minima. However it has been shown to converge to a local minimum, provided that α be small enough [52]. Indeed, big values of α will often yield faster convergence but can also lead to a slower or even divergent training, by causing the error function to "jump" over the minima as the update steps are too large.

OPTIMIZED LEARNING RATE Advanced versions of gradient descent exist, which involve automatic adjustments to the learning-rate.

For instance, the method ADAGRAD [15] uses parameter-specific updates of the form

$$\Theta^k_{t+1} = \Theta^k_t - \frac{\alpha_0}{\sqrt{G_k + \epsilon}} \nabla \mathcal{L}(\Theta^k_t)$$

for each parameter Θ^k, where α_0 is the initial learning rate and G_k is proportional to the *square root of the sum of the squares of the previous gradients* applied to parameter Θ^k.

That is, a *progressively decreasing learning-rate* adapted to each of the parameters is used, allowing to:

1. Perform larger steps at the beginning of the optimization, when the distribution of parameters may lie far from any minimum of the error function, so as to quickly move around the parameter space and cross several local minimum areas,

2. Progressively lower (proportionally to the *norm of the cumulated gradient applied to each parameter*) the learning rate for each parameter individually. Parameters which have received strong updates are supposed to have left their initial random area and hopefully reached a zone near a minimum, thus the gradient updates become more refined to avoid "jumping" over this local minimum.

The ADADELTA method [56] further improves on ADAGRAD by allowing the learning rate for each parameter to periodically increase, thus avoiding the convergence of the learning rates towards 0 implied by ADAGRAD. This allows the parameters to keep varying, rather than receive infinitesimally small updates after a while.

STOCHASTIC GRADIENT DESCENT (SGD) Other forms of gradient descent have been proposed, based on the behaviour of the algorithm at the *first steps of optimization*, after a random initialization of the parameters. In this early optimization stage, the parameters may be in a rather "flat" zone, with no clear direction for the closest local minimum.

Performing gradient updates based on a gradient computed over the whole dataset may therefore have only a dim impact on the error value, specially when this dataset is large (datasets with size greater than one million examples are not uncommon [8, 26]).

One way of tackling this issue is SGD (*Stochastic Gradient Descent*) [9], a widely popular alternative to vanilla gradient descent. SGD operates on mini-batches – small sets training examples sampled at random from the training set – rather than on the full training set. The idea is that these small gradient updates are faster to compute and that quickly performing many updates using randomly chosen examples allows to "scan" the error space and hopefully get closer to a minimum. Once an area close to a minimum has been reached, performing standard gradient descent on the full dataset will become meaningful and one can thus switch back to standard gradient descent after a few steps of SGD.

3.2 NEURAL NETWORKS

In this section we begin to introduce neural networks, a specific type of machine learning systems which has proved very efficient in many tasks in the last few years.

Neural networks are defined as *connectionist* systems: they are built by connecting several identical computation modules, namely *neurons*. In this section, we first present the definition of a single neuron. In the next section, we then build a *perceptron*, the most basic kind of neural network, by *connecting several neurons*.

DEFINITION *Artificial neurons* (as shown in Figure 1), were defined by McCulloch and Pitts [35]. They are the building brick of neural networks.

A single artificial neuron is a function $\tilde{f} : \mathbb{R}^n \to \mathbb{R}$ defined as

$$\begin{cases} \tilde{f}(x) = \phi(y) \\ y = \langle x, w \rangle + b = b + \sum_{k=1}^{n} x_k . w_k \end{cases}$$

Hence, it is composed of:

1. A *transfer function*: a function mapping an input to a single value. Usually this function is a *dot-product*, a linear projection of the input vector on a vector w, which outputs a weighted sum of the inputs. For flexibility, a constant term, the *bias*, b is added to this transfer function (otherwise, neurons could not even approximate a non-zero constant).

   The computed value y (in $\mathbb{R}$) is called the *pre-activation*.

2. An *activation function* ϕ: a non-linear function applied on the pre-activation y.

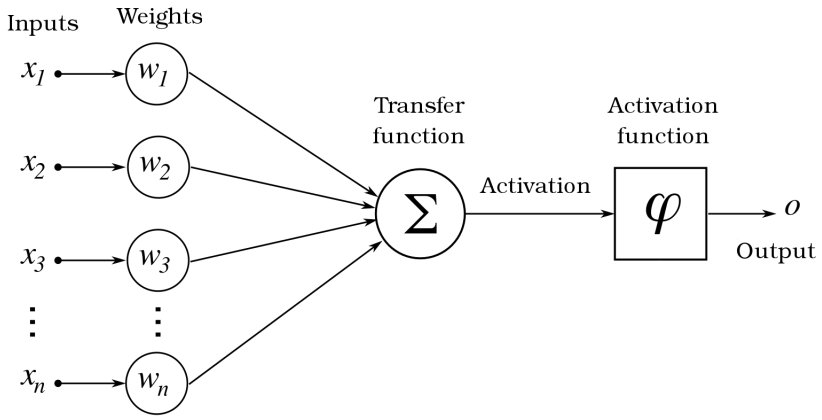

Figure 1: Definition of an artificial neuron

INTERPRETATION The algorithmic idea behind the transfer function is that the neuron is expected to take a *decision* and output a given value y *based on all of its inputs*. Therefore, a function is needed that *aggregates* all inputs into one meaningful value, on which to perform the decision. This is the main "trainable" operation of the neuron, indeed the *weights* w_k of the projection vector are the variable parameters Θ that are optimized during training.

After this information aggregation has been done, yielding the activation y, the activation function ϕ computes the actual value $o = \phi y$ returned by the neuron.

Commonly used non-linearities ϕ include: the hyperbolic tangent, the sigmoid functions or the Rectified Linear Units [18, 31].

DECISION PROBLEMS A large class of machine learning problems can be fruitfully seen as *binary decision* problems: "given an input x, does x satisfy some property P?". This will allow us to give some interpretations of the computations done by neurons.

In this context, the activation function is the function that performs the decision making. An example of such decision function ϕ is the *Heaviside function*, H, the most simple function which performs binary decisions on real-values. It is defined as

$$H : x \mapsto \begin{cases} 1, & x \geqslant 0 \\ 0, & \text{otherwise} \end{cases}$$

Example. An example of a *decision making* problem that can be naturally performed by a neuron is the problem of evaluating the position of point on the real plane $\mathbb{R}^2$, e.g. with respect to the line (an *affine hyperplane* of $\mathbb{R}^2$) L of points with y-coordinate 1, i.e. L defined as

$$L = \{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2 | \ x \in \mathbb{R}, y = 1 \}$$

Indeed, to decide if a point x with coordinates (x_1, x_2) lies above the line L, one can compute the projection $\langle x, w \rangle$ of x on the vector w *orthogonal* to L, that is, $w = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. This yields the value x_2. Then x lies above L if and only if $x_2 \geqslant 1$, or equivalently $x_2 - 1 \geqslant 0$, i.e. $H(x_2 - 1) = 1$.

We can then write $x_2 - 1 = \langle x, w \rangle - 1$, i.e. we set the bias b to -1.

Then one can perform this operation using an artificial neuron with weights $w = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, bias $b = -1$ and activation function H. This neuron outputs 1 for a given input point if and only if it lies above the affine vector L, otherwise it outputs 0.

GEOMETRICAL INTERPRETATION We have presented the interpretation of a neuron as a means of making decisions. A neuron can also fruitfully be interpreted *geometrically* as a *transformation of its input data*.

To begin with, note that networks use *generic activation functions*, not tailored specifically to the problem and the data at hand.

Given a particular problem to solve, then, the solution might be hard to evaluate on the *raw data* using only generic functions. The neuron must therefore *learn to transform its input data* (in effect "bending" the input space $\mathcal{F}$) in order to find a transformed space in which this decision making can be found as a linear separation, i. e. amounts to splitting the space with an hyperplane, as can be done with a neuron.

Indeed, a neuron first performs a projection of its input data on an hyperplane of the input space. The optimal *direction and affine position* of this hyperplane given the problem to solve is learned by the algorithm. After this projection has been computed, the subsequent non-linear function can be seen as a *deformation* of the hyperplane on which the data is projected. This allows to project on *more complex manifolds* than hyperplanes.

FEATURES INTERPRETATION A final, alternative interpretation of the behavior of a neuron is in terms of *feature detection*.

In the neuron's definition, the dot-product $y = \langle x, w \rangle$ of the input x with the internal vector w yields an estimation of the *alignment of those two vectors*. This follows the definition

$$\langle x, w \rangle = \|x\|.\|w\|.\cos(\theta)$$

with θ the *angle* between x and w, which is maximal when x and w are *aligned*, i.e. *equal up to rescaling*.

Thus a neuron operates as a *detector* for a given *feature* w: it outputs a maximal result when its input is aligned with w.

PERCEPTRON The single neuron is a function from a multi-dimensional vector to a single real. In order to *approximate multi-dimensional functions*, we introduce the *perceptron*.

A perceptron $g : \mathbb{R}^n \to \mathbb{R}^m$ is a collection of m individual neurons $g_1, \dots, g_m$, with weight vectors $w^1, \dots, w^m$ each of dimension n and non-linearity ϕ (identical for all neurons). All neurons g_i take the same vector x as input and each *outputs one dimension of the output vector*, i.e. for a given x in $\mathbb{R}^n$, $g(x) = \begin{pmatrix} g_1(x) \\ \vdots \\ g_m(x) \end{pmatrix}$.

Mathematically, a perceptron performs:

- A matrix-vector multiplication, Wx, where the matrix W is composed of the weight vectors for each individual neuron, in lines:

$$W = \begin{pmatrix} (w^1)^\top \\ \vdots \\ (w^m)^\top \end{pmatrix}$$

- Followed by a point-wise application of the non-linearity ϕ:

$$\phi(x) = \begin{pmatrix} \phi(x_1) \\ \vdots \\ \phi(x_n) \end{pmatrix}$$

Informally, perceptrons make use of different neurons, each trained to solve one simple task, to combine their outputs and solve a more complex task.

Example. Using three neurons, one can decide whether a point $\begin{pmatrix} a \\ b \end{pmatrix}$ of $\mathbb{R}^2$ lies in the first quadrant $Q = \{\begin{pmatrix} x \\ y \end{pmatrix} \mid x \geqslant 0 \text{ and } y \geqslant 0\}$.

In order to do so, one trains a neuron to detect if the point lies above the horizontal axis, and a second one to detect if it lies right of the vertical axis, which yields a new vector of $\mathbb{R}^2$.

Finally, one can use a third neuron on top of the outputs of those two neurons to check whether they both output the value 1. Note that in doing so, we have actually built a multilayer perceptron (presented in Section 3.3). It can be proved [36] that solving this toy problem with neural networks can only be done using *at least two layers*.

Another example is shown in Figure 2, which illustrates the idea of input space bending. The task is to predict if a point in $\mathbb{R}^2$ belongs to either the blue or the red part of the space. As illustrated, the input dataset cannot be split by a line.

After going through the first layer, composed of two neurons with sigmoid as non-linearity, the input space has been transformed as shown below. The two regions are now separable using a line and a single neuron above the outputs of this layer can indeed solve the problem.

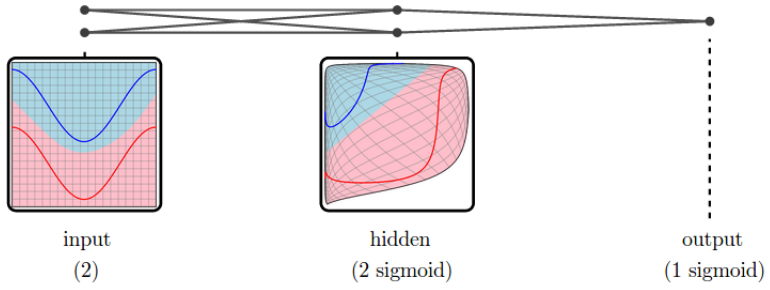

Figure 2: Transformation of the input space learned by a neural network (via [41])

DATA REPRESENTATION With the previously introduced interpretation of neurons in terms of feature detection, the perceptron can also be seen to detect a *set of features* in its inputs. In this perspective, a perceptron with m parallel units returns, for a given input, a vector of size m containing the activation of *each of the* m *feature detectors*.

Let us stress the fact here that the interest of neural networks specifically lies in the fact that these features (i.e. the weight matrix W) are

automatically learned. That is, they are chosen and adjusted to *compactly describe* and therefore optimally approximate the function f.

The output of a perceptron can therefore be interpreted as an *alternative representation* of the input data, in terms of *explanatory features*.

TRAINING The gradient-descent algorithm on single-layer perceptrons is rather trivial. If the non-linearity ϕ is differentiable, the function computed by the perceptron is the composition of a linear operation and a differentiable function, its derivative can therefore be computed using the chain-rule.

Following the notations used for a single neuron, we denote as y^i the quantity fed into the activation function for neuron g_i, i.e. $g_i(x) = \phi(y^i)$.

The gradient value at x in $\mathbb{R}^n$ for the weight w^i_j of the neuron g_i is then equal to

$$\begin{aligned}\frac{\partial g}{\partial w^i_j}(x) &= \frac{\partial \phi}{\partial y^i}(y^i) \cdot \frac{\partial y^i}{\partial w^i_j}(x) \\ &= \phi'(y^i) \cdot \frac{\partial \left(b^i + \sum_{k=1}^n x_k . w^i_k\right)}{\partial w^i_j}(x) \\ &= \phi'(y^i) \cdot x_j\end{aligned}$$

NOTE: PARAMETER INITIALIZATION In the absence of any preliminary knowledge of the data considered, the initial parameter distribution can only be initialized by being randomly drawn from some chosen distribution.

Standard distributions as the uniform or normal distributions for instance can be used. More advanced methods exist though, aimed specifically at special machine learning architectures, such as `kaiming` for neural networks using Rectified Linear Units [22].

3.3 DEEP NEURAL NETWORKS

PRINCIPLE OF COMPOSITIONALITY A motivation behind the use of deep architectures is the *principle of compositionality*. This principle states that real-world observations can be explained by the *composition of elementary elements* (e.g. the ocean is made of water drops, which are themselves made of water molecules, in turn made of atoms, down to the sub-atomic particle level).

Remark. Compositionality in tonal music

This principle appears to apply fittingly to Western tonal music, where a *polyphonic* music piece is composed of individual *voices*, themselves built in musical *phrases*, each made of consecutive *chords* and those chords are made of elementary *notes*.

This observation motivates the application of such compositional approaches to music. Note that this is the *sole assumption* made about the musical structure.

MULTI-LAYER PERCEPTRON A *Multi-Layer Perceptron* (MLP) is a network composed by a succession of perceptrons, each called a *layer* of processing.

The inputs of the perceptron at layer $n+1$ are the outputs of the perceptron at layer n. An example of MLP is displayed on Figure 3. In this Figure, the neurons j and i respectively at layers $(n-1)$ and n are connected with a weight W_{ji}^{n}.

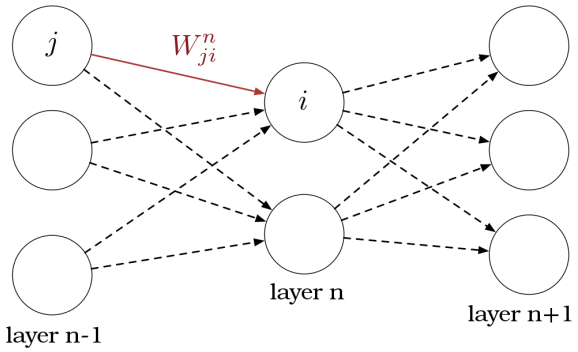

Figure 3: Three layers of a Multi-Layer Perceptron

Note that the non-linearity ϕ on the output of each layer becomes crucial in the context of MLPs. Indeed, if we only performed a linear operation at each layer, then the whole network would compute a succession of linear operations, i.e. it would be equivalent to computing a single linear operation.

In order to approximate more complex functions that linear ones, the non-linearities are therefore required.

MOTIVATIONS We have seen how to define a single perceptron layer.

Why would one want to use more layers? During the course of our running example of point location in $\mathbb{R}^2$, note that we introduced a second layer to be able to locate a point with respect to two distinct lines. One could wonder if this second layer was really necessary. Indeed, one could think solving this seemingly simple problem would be possible by using a single layer perceptron. It turns out that *this is not possible*, as was formally proved by Minsky and Papert [36].

In this paper, it is shown that adding more layers to a network does augment its representation power: there are classes of functions which cannot be approximated using a single layer perceptron, but can be approximated by adding more layers to the network. Further-

more, adding layers to a network can drastically (*exponentially*) *reduce the number of units* needed to solve a given problem.

TRAINING: BACKPROPAGATION The canonical gradient descent algorithm on MLPs is called *backpropagation* and was introduced by Rumelhart, Hinton, and Williams [48].

The back-propagation algorithm is initialized at the output layer of the network, where the network's error is computed. It then consists in estimating, for any given example, the *contribution* of each neuron to the global error on this example. The weights of each neuron are then adjusted so as to minimize this individual contribution.

If the non-linearity ϕ and the metric d are differentiable, the gradient of the error can be computed by application of the *chain-rule*.

The main advantage of backpropagation is that it requires a number of computations *linear* in the number of units of the network.

VANISHING GRADIENT AND LACK OF COMPUTATIONAL POWER An issue empirically occurs when training deep networks, called the *vanishing gradient* [17]. This relates to the fact that the gradients in a deep network tend to get exponentially smaller as one goes from the output layer to the input layer.

This can lead to a situation where the gradient of error on the first layers is so small that the updates have no quantitative effect on the parameters. In that case, the first layers of the network are not trained. This is a serious problem, since they are randomly initialized and, if they are not trained, their effect on the input data is simply to mangle it.

This is due to the fact that the recursive expression of the gradient obtained by the chain-rule is *multiplicative* (because the chain-rule itself is multiplicative). Add to this the fact that the most widely used non-linearities in the 90s and beginning of the 2000s were sigmoid and the hyperbolic tangent, which have derivative with norm bounded in $[0, 1]$. Using such non-linearities, at each layer during backpropagation, the initial gradient of error is multiplied by quantities *always less than* 1. In that case, the gradient shrinks exponentially along the layers.

Note that the problem of the *exploding* gradient also exists. It occurs when the quantities by which the initial gradient is multiplied are always greater than 1, leading to exploding weights in the first layers, with values too large to store.

Simultaneously, in the 90s and beginning of the 2000s, another issue for deep learning was the lack of computational power and the scarcity of data to analyze. Indeed, deep networks have a very high learning power, with a lot of parameters (weights) to train, thus they

require intense computations to train, as well a a high quantity of data to reach proper parameter estimations.

THE DEEP LEARNING REVIVAL These issues strongly held back the development of deep neural networks until about 10 years ago when Bengio et al. [7] introduced a new way of training these networks that got rid of the vanishing/exploding gradient issue.

This method, *greedy layer-wise pre-training*, consists in training the layers successively to *reconstruct* their input, i.e. the output of the previous layer. This proves very efficient in providing a good initialization of the weights of the network and completely removes the risk of vanishing gradients, since the gradients are never backpropagated on more than one layer.

In the meantime, computational power had gratly increased, partly thanks to the development of GPUs, computing units optimized for matrix computations. The availability of data also greatly expanded, with the development of the Internet providing a wealth of content uploaded by individuals. An example is the Flickr dataset [26], which contains one million of annotated pictures uploaded by individuals.

Altogether, this finally made it possible to train very deep networks and sparked a renewed interest for deep learning techniques.

TRAINING WITH RECTIFIED LINEAR UNITS Various other solutions to the vanishing gradient problem have since emerged, the newest and currently most popular of which proves empirically very efficient. It makes use of specific non-linearities: *Rectified Linear Units*, or ReLUs, as displayed in Figure 4.

ReLUs, defined as $x \mapsto \max(0, x)$, are not smooth around zero, which is theoretically a problem when computing the gradient.

In practice, implementations [13] ignore this non-linearity and simply trim the gradient to zero below zero.

Why ReLUs actually really work remains somewhat obscure, but the results are currently the best available [18].

3.4 CONVNETS AND HIGH-LEVEL REPRESENTATIONS

Note that although convolutional neural networks are not applied in the following, they constitute an insightful illustration of the capacity of deep neural networks to learn high-level features from their input data, hence their presentation in this review.

CONVOLUTIONAL NEURAL NETWORKS Convolutional neurons, introduced by LeCun et al. [33], are an extension of standard neurons in which the transfer function, which maps the input to the value sent

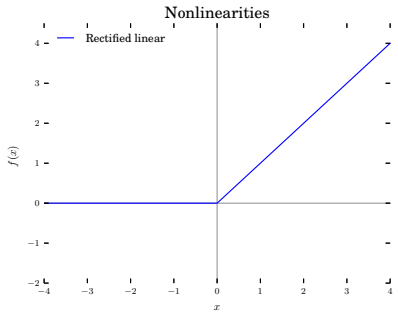

Figure 4: Rectified Linear Unit activation functions

into the non-linear activation function, is a *convolution* with a learned kernel.

These neurons search (through the convolution operation) for occurrences of their kernel in their input data and activate when this feature is detected. The size of the learned features is a parameter of the neuron.

Convnets are (usually deep) neural networks built using such convolutional neurons.

HIGHER-LEVEL REPRESENTATIONS Convnets rely strongly on the compositionality principle and compute successive higher-level representations of their inputs.

Indeed, if one perceptron computes a high-level representation of its input data, then, the perceptron at layer $(n+1)$ computes *a higher-level representation of the representation* computed at layer n. Now, given that a convolutional layer operates by detecting occurrences of some small-scale features within its input, then, in a Convnet, the convolutional layer $n+1$ operates by detecting *co-activations of the features detected by the previous layer*, that is, *features at a bigger scale*.

Convolutional features of size 3×3 are common for picture analysis, see for instance the network built in [51] to perform image classification

If the compositionality principle is respected, then such an approach makes sense. Indeed, the data can be decomposed into a structure made of blocks, themselves built by the simultaneous presence (co-activation) of several smaller blocks. Each convolutional layer then extracts a refinement of those structural blocks.

CLASSIFICATION EXAMPLE: MNIST To illustrate the idea of high-level representations extraction, consider the concrete case of *handwritten digit classification*, where one trains a system to associate pictures of handwritten digits with the actual digit they represent.

This has been extendedly studied using the MNIST dataset [32]. A sample from this database is presented in Figure 5.

In the context of pictures, the computed features will themselves be pictures, i.e. a picture of a digit is recognized by decomposing it

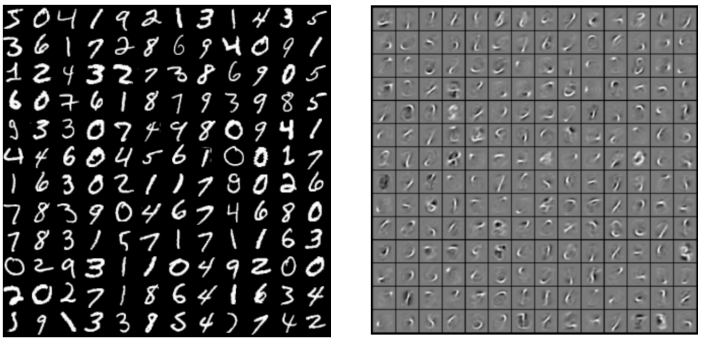

Figure 5: Sample (left) and features (right) from the MNIST dataset (via [1])

over a set of elementary pictures. The above layers in the networks operate on combinations of the features at their underlying layer.

Empirically, one observes that when training a convolutional networks on the MNIST dataset, the *high-level features* obtained resemble those presented in Figure 5. The first layer features are small *linear edge detectors* (the most simple of pen strokes). These basic features are then used on the above layers to detect more and more complex *pen strokes*, up to complete digit recognition. This indeed follows the compositionality principle.

A NOTE ON LEARNING POWER AND QUANTITY OF DATA The deeper a network and the more trainable parameters it has, the more flexible it will be and the more complex functions it will be able to approximate. But this has a cost, as stated before, since a network with many parameters will require many training examples to properly approximate any given function (because its parameters lie in a very high dimensional space). By showing the network many different examples, it is able to build a precise representation of the distribution of the input data manifold.

Hence the following informal principle:

> "*The more data, the better*"

Or, stated in another way, "data is the best *regularizer*", in the sense that plentiful data strongly *constrains* the network's parameters to progressively reach their proper distribution.

This principle guided the choice of a very large dataset for the work presented here.

4

RECURRENT NEURAL NETWORKS

Now that the general machine learning and deep learning notions have been introduced, we presents a quick overview of the field of recurrent neural networks: networks that deal with time series, ie. functions of the time.

We consider in the following that the inputs to the network are functions $t \mapsto x(t)$ of the *discrete* time t.

We start by introducing general recurrent neural networks. We then move on to presenting the main model used in our research project: LSTM *networks*. These networks are known to give state-of-the-art results on time series analysis.

ON LEARNING AND MEMORY The main motivation for the introduction of RNNs is the introduction of *memory* in learning. Indeed, in order to properly analyze sequences that vary over time which can follow arbitrarily complex structure and have some arbitrarily long temporal dependencies, one requires some notion of memory.

4.1 GENERIC RNN

RNNs are a way to deal with memory whilst alleviating the power of neural networks. They are almost identical to standard NNs, to the decisive difference that they have *recurrent* connections, which allow them to carry the output of a particular neron at further time steps. Hence, they operate *sequentially* on their input sequences and are fed at step t *both* with their *input data* $x(t)$ at time step t and the *network's output* $o(t-1)$ at the *previous* step. This allows for the transport of some information within the network over time.

One can convert an RNN to a standard NN. To do so, one *unrolls* the recurrent connections over time, by duplicating the layers at each time-step and explicitly writing the memory connections. This yields a standard NN. This process is illustrated in Figure 6.

For a network with non-linearity ϕ, the RNN learns two weight matrices W_{in} et W_{rec}. W_{in} operates on the inputs, and W_{rec} operates on the recurrent "feed-back" connexion (which feeds the network's output back into it).

Using these weights, the output at step t is computed according to the following equation:

$$o(t) = W_{in} \cdot x(t) + W_{rec} \cdot o(t-1)$$

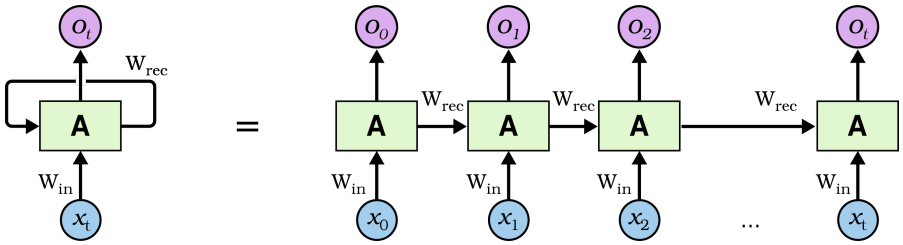

Figure 6: Unfolding an RNN through time (via [40])

BACKPROPAGATION THROUGH TIME This unrolling through time is actually at the basis of the training algorithm for recurrent networks: the *Backpropagation Through Time* (BPTT).

It simply consists in two steps:

1. First, the RNN is unfolded for some fixed amount ρ of time steps, yielding a standard NN,
2. Then, standard backpropagation is applied to the obtained NN, and the weights of the RNN are updated according to their time-position.

LONG-TERM MEMORY The main issue with vanilla RNN is their inability to handle long-term dependencies, that is, they cannot transport information over a long time period, as was formally investigated by Bengio, Simard, and Frasconi [6].

This is a problem when dealing with musical scenarios, for instance in the case of *da capos*. Those require knowledge of the very beginning of a potentially long sequence, right at its end.

The following section introduces a widely popular solution to this issue: Long Short-Term Memory networks.

4.2 LONG SHORT-TERM MEMORY

LONG SHORT-TERM MEMORY UNIT *Long Short-Term Memory* units (LSTM), introduced by Hochreiter and Schmidhuber [25], are a special kind of recurrent units aimed specifically at transporting information over long periods.

Informally, their originality and success comes from the fact that they are based on *multiplicative* operations, which allow them to *modulate* (scale) their input data with some *previously stored memory*.

An illustration of an LSTM unit is presented in Figure 7. Multiplicative interactions are depicted as black squares between connexions.

MEMORY CELL AND GATES More precisely, LSTM units hold a *memory cell*, which is expected to learn and store the internal memory of

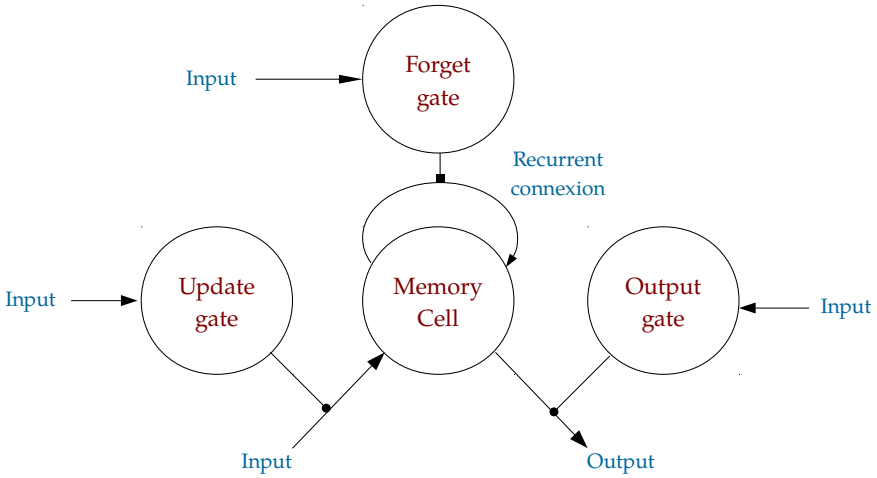

Figure 7: Long Short-Term Memory unit

the unit, and three *gates*, which *control* how the memory cell's content is being updated and used.

This "control" is done via a multiplicative operation, the point-wise multiplication of vectors.

The three gates, standard neural networks receiving input from the current input $x(t)$ to the network and the previous output $o(t-1)$, are:

1. A *forget* gate, which controls which part of the memory to *forget*. For instance, the network can "decide" , based on the input $x(t)$ and the previous output o_{t-1}, to "forget" all dimensions in the memory vector but the last, by modulating the recurrent connexion with the vector $\begin{pmatrix} 0 & \dots & 0 & 1 \end{pmatrix}^\top$.
2. An *update* gate, which controls which part of the units input to *store into the memory cell*. Similarly, the unit can decide to select the value at the first dimension in the current input, by multiplying it with the vector $\begin{pmatrix} 1 & 0 & \dots & 0 \end{pmatrix}^\top$.
3. An *output* gate, which controls which part of the memory to *output at the current time-step*.

At each time t, the content of the memory cell is then updated by combining – through a simple sum of the two vectors – the values stemming from the update gate and from the forget gate,

Optimizing the weights in those gates through training, the LSTM can be taught to *properly manage its memory* for various operations.

Through these mechanisms, it can transport information over long periods, by storing some values into the memory cell and simply

passing it on until some event in the input triggers the output of the memory.

Similarly to the interpretation of convolutional neurons as feature detectors, LSTM units can be interpreted as detectors for *arbitrarily long temporal patterns* within sequences. The patterns learned (the "features") are the sequences which maximally trigger the unit's output.

LSTM LAYER Akin to perceptrons, an LSTM *layer* is built by assembling together independent LSTMs.

Therefore, such layers learn to detect different patterns within their sequences.

LSTMs have had tremendous success in many applications within the time-series analysis field, including text [44] or video [54].

The work presented in this project mainly relies on those networks.

DEEP LSTM NETWORKS *Deep* LSTM networks are a natural extension to LSTM layers: they are networks composed of successive LSTM layers.

The expected effect is the same as deep convolutional networks, which learned features of *increasing scale* at each layer, by learning features *over* the features computed by their underlying layer.

Here, layer $n+1$ detects temporal patterns over the output of layer n, which in turn detected patterns over the outputs of layer $n-1$. Thus, layer $n+1$ can be seen to detect larger-scale patterns over the inputs of layer $n-1$

Thus, deep LSTM network can be expected to operate at varying scales and uncover complex temporal dependencies in the sequences they analyze.

In practice, deep LSTMs give state-of-the-art results on temporal analysis of time series, outperforming standard deep neural networks, e.g. in speech recognition [50].

Part II

PREDICTION AND ABSTRACT SCENARIO INFERENCE

In this second part, we present our approach to the *scenario inference* and *prediction* problem.

Prediction is performed through temporal analysis using recurrent neural networks. Due to the high representation power of these networks and their large number of parameters, we searched for a large dataset on which to train them. This lead us to the choice of the Million Song Dataset, which provides one million chromagrams.

A meta-optimization loop is implemented to devise an appropriate architecture for this task, given the large dataset at hand.

The abstraction step is done using clustering algorithms on the available chromas, effectively turning a sequence of chromas into a sequence of abstract labels.

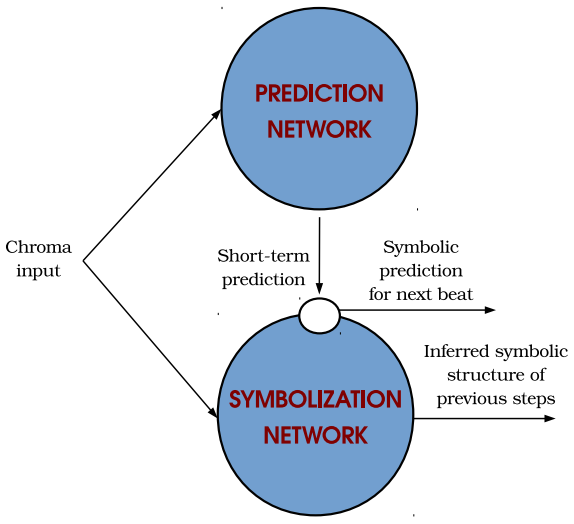

Figure 8: Proposed prediction and symbolization architecure

5 CHROMA PREDICTION

In this chapter, we present the prediction model we use for our musical scenario inference task. The goal here is to be able to predict subsequent steps in a musical sequence, allowing to extract some scenario *with lookahead* from musical sequences.

We propose in this first experimentation to train a model at a *beat-by-beat* rate to predict the one next chroma based on a sequence of chromas.

Remark. It should be noted prior to all discussion that, due to the large scale of the networks considered, training is costly and takes time. Thus, although the complete data importation/processing and training pipeline has been implemented during the course of the internship, quantitative results for this section are not yet available, since the training is still taking place at the time being. Updated results should be added as soon as available, including comparisons between different network architectures.

5.1 DATA

Given the representation power of LSTMs and their large number of parameters, a large dataset is required.

We use the MILLION SONG DATASET [8], which is one of the largest datasets available for symbolic music. It provides *chromagrams*, that is, sequences of *chroma vectors*, for one million of "popular" music pieces (see cited article for a description of how the dataset was built).

Remark. Chroma vectors are 12-dimensional arrays (of $[0,1]^{12}$) describing, for a given musical frame, the *relative importance* of each pitch class (from C to B♭) within this frame.

We therefore settled on building a prediction system for chroma vectors.

Given the data available (*event-based* chromagrams), one can reconstruct chromagrams at any chosen time-rate. For our preliminary experiments, we settled on *beat-by-beat* analysis of the chromagrams.

5.1.1 *Dataset split*

From the 1 million elements of the dataset, we extract $\approx 10\%$ to build a *test set*.

More precisely, the last 2 data folders of the dataset, i.e. folders `/data/Y/` *and* `/data/Z/`.

This test set is used to compare different network architectures. Examples from this subset are *not used during the training*, so as to perform the testing on examples *completely unknown* to the networks.

5.2 MODEL

Motivated by their strong results on time-series analysis and taking into account the expected complexity of the musical prediction problem – with e.g. a high quantity of different styles to learn –, we settled on the use of LSTM networks for the prediction task.

Considering that temporal evolutions within music can be found at varying time-scales, we furthermore use *deep* LSTM networks so as to take into account temporal dependencies at several time-scales.

The networks are then trained to perform the following task: "Given a sequence of chromas $x(0) \dots x(t_0 - 1)$ of duration t_0, predict the next chroma vector $x(t_0)$".

If we write $o(t_0) = f(x(0) \dots x(t_0 - 1))$ for the network's prediction at time t_0, the network's error at time t_0 is then equal to $d(o(t_0), x(t_0))$

Using our system, one can perform long-term predictions via the following recursive formula:

$$o(t_0 + 1) = f(x(1) \dots x(t_0 - 1) o(t_0))$$

That is, one successively predicts the next time-step, conditioned on the previous predictions.

Necessarily, the quality of the predictions will decrease with the length of the successive predictions performed.

Remark. Note that these networks are not by any means restricted to work on chromagrams. The choice of working with chroma vectors was mostly motivated by the availability of data.

5.2.1 *Slicing the examples*

LSTMs are heavy structures and operate slowly on long sequences, which involve *very large matrix multiplications* with the LSTM's internal weight matrices. Therefore, they are generally [50] trained using SGD on mini-batches of short sequences, obtained by *slicing* the sequences from the dataset.

The length t_0 of the sequences with which to train the network is a hyper-parameter of the network.

5.2.2 *Temporal horizon of prediction*

Alternatively to the model presented above, which predicts the single next chroma vector for an input sequence, one could train the network to predict the k subsequent chroma vectors conditioned on the input sequence.

The effect of this choice on the quality of the prediction is compared in the hyper-parameter optimization loop, by comparing the results for networks trained with different prediction horizons.

5.3 METRICS ON CHROMAS

In order to compute errors and evaluate the quality of the predictions, one must choose a metric on the space of chromas.

Note, first, that comparing different metrics between one another *in abstracto* does not make sense.

Nevertheless, if one fixes a metric of reference (e.g. the L_2-norm), one can compare the effect of the different metrics on the training. Indeed, one can train a given network using different metrics for the error, then compute the total error on the test set using the L_2-norm and compare this error between all the networks.

Furthermore, although various metrics exist, *interpreting* the errors on the outputs of a chroma prediction system is hard. It comes down to the difficulty of relating the error from a metric on the chroma space $[0,1]^{12}$ with actual human perception: completely mistaking a fifth for a third actually has far less perceptual impact than adding a bit of the diminished second to a chord major chord.

With this in mind, the go-to metric on real-valued spaces is the L_2-norm, a completely *agnostic* metric (it makes no assumptions on the data). It is therefore not really semantical.

Other metrics exist, which may be more meaningfully interpretable.

A popular example is the Kullback-Leibler divergence [30], defined as

$$D_{KL}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

The KL-divergence estimates the *similarity* between two (here discrete) data distributions P and Q (in our prediction context, P would be the *prediction targets* for the chosen dataset and Q the actual *outputs* of the network).

One could also envision the use of a metric incorporating music-analysis notions, such as the *Tonnetz-distance* presented by Harte, Sandler, and Gasser [21]. This distance embeds some elementary notions of harmony theory, only taking into account the relations between tonics, thirds and fifths.

Another possible approach to a quantitative means of evaluating the quality of predictions – though rather extreme –, is to discard altogether the volume information and use a *binary* accuracy criterion, such as the Hit/Miss accuracy measure Acc presented by Bay, Ehmann, and Downie [4]:

THE ACC MEASURE Acc operates on binary activation vectors. If y is the *target vector* of binary activations and x the binary *prediction vector* (x is expected to predict the activations in y), we use the usual

definitions of *true positives*, *false positives*, *true negatives* and *false negatives*, as presented in [4].

For instance, a false positive (on dimension i) is an x such that $x_i = 1$ whereas $y_i = 0$.

We write TP for the number of true positives in the prediction x for target y, FP for the number of false positives and FN for the number of false negatives.

Using this, Acc is defined as

$$\mathrm{Acc}(x, y) = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Acc ranks *how well* the vector x predicts the activations of the target.

We can apply Acc to vectors of $[0, 1]^n$ by first thresholding them by some chosen value ϵ in $]0, 1[$. For chromas, a pitch class is considered "active" if its value is greater than ϵ.

We will use this measure to perform some (partly) interpretable quantitative evaluations of the trained networks.

5.4 IMPLEMENTATION AND HYPER-PARAMETERS OPTIMIZATION

The training is performed within the Torch [13] framework.

A pipeline to import the data was implemented. This implementation takes into account the issues associated with the large quantity of data in the dataset, which all cannot be loaded onto memory at once. We thus implemented a *sliding window* over the dataset. This sliding successively loads examples, slices them into small sequences of duration t_0 to feed the network with and aggregates them into a single matrix for faster computations.

Furthermore, the number of parameters to choose from when using LSTMs is large. It includes amongst others:

- The *number of* LSTM *layers* of the network,
- The *number of* LSTM *units* per layer,
- The *slicing duration* t_0 to use,
- The *duration* k of the predictions to perform,
- The *metric* d to use on the chroma space for training,
- The *initialization function* to apply on the network's weights,
- The eventual *non-linearities* to apply on the output of each LSTM layer.

Therefore, finding the best architecture for the task at hand is hard.

To perform this parameter adaptation, a hyper-optimization loop was implemented. This loops starts by *generating* several network architectures by *sampling* the value of each hyper-parameter from a provided distribution (in practice a *Gaussian kernel*). The networks are then *trained* with some iterations of SGD using the dataset provided. They are finally *ranked* by their total error value on the test subset.

At each iteration of this loop, the kernel for each hyper-parameter is adapted to try and focus on the best hyper-parameter values.

5.5 RESULTS

The training loop is still in progress. Results will be added when available.

6 CHROMA SYMBOLIZATION

The symbolization step takes a *chromagram* as input and outputs a sequence of *abstract* labels.

For our application, we implement a rather simple approach: we train a *clustering* algorithm on the MILLION SONG DATASET, which then allows to convert any chroma vector into an abstract class label.

6.1 CLUSTERING

The *clustering* problem is a traditional problem in machine learning.

Consider some input data $\mathcal{D} = \{x_i\}_{i=1\ldots N}$. A clustering algorithm with k classes is expected to split the dataset into k classes, i.e. return a partition $\mathcal{D}_1, \ldots, \mathcal{D}_k$ of $\mathcal{D}$ into k subsets.

Note that the value k is in some cases a hyper-parameter of the algorithm, to be fixed by the user, and in other cases it is devised by the algorithm itself.

ALGORITHMS Popular clustering algorithms include [27]:

- Hierarchical clustering, based on building a *taxonomic tree* of the data points. This taxonomic tree is based on a distance on the dataset.

  The advantage of hierarchical clusterings is that one can extract clusterings with arbitrary numbers of classes between 1 and N from a hierarchical clustering, by choosing at which number of classes to stop refining the tree. Their disadvantage is their quadratic construction time.

- k-means, with k fixed, attempts to build k classes $\mathcal{D}_1, \ldots, \mathcal{D}_k$ with *centroids* $\mu_1, \ldots, \mu_k$ via a stochastic process minimizing the quantity: $\sum_{i=1}^{k} \sum_{x \in \mathcal{D}_i} \|x - \mu_i\|^2$, the *within-cluster sum of squared distances*.

  Good clusters are therefore clusters with a small radius. A k-means clustering can be computed in time *linear* in the size of the dataset.

An example of k-means using 3 clusters on a dataset sampled from a Gaussian distribution is presented in Figure 9.

We use k-means in our application, because it is a simple to implement, popular model with *efficient implementations* [28].

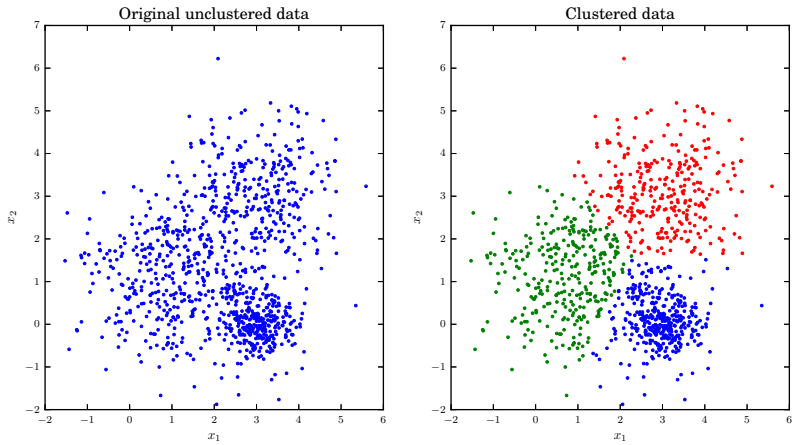

Figure 9: Clusters computed by k-means, $k = 3$

6.2 EVALUATION

Clusterings with different numbers of clusters will give different results. The "quality" of a clustering in out context is again hard to evaluate *in abstracto*: what makes a clustering good is somewhat subjective.

Whether a clustering is "good" or "bad" could nonetheless be assessed in the context of the co-improvisation application we present below. Indeed, when connected with a co-improvisation system, a "good" clustering will produce abstract sequences which will in turn provide "rich" improvisation possibilities, with satisfying musical results.

In this case, the quality of clusterings should therefore be evaluated in conjunction with a musician, who could judge and rank the co-improvisations generated using different clustering parameters.

Part III

CO-IMPROVISATION AND STYLE-ADAPTATION

In this last part, we propose a theoretical architecture for *reactive structured co-improvisation* and *style adaptation* using the scenario inference and prediction tool presented before.

Existing co-improvisation systems cannot *simultaneously* reactively adapt to a musician's style *and* anticipate on his playing. We propose to connect the symbolic outputs of our network to a pre-existing co-improvisation engine which operates on sequences of abstract labels. Using this system in real-time with a human improviser, we could both infer a scenario to his playing and anticipate on his next steps, allowing for smooth, synchronized transitions.

We furthermore propose a framework for *style adaptation*, that is, adapting our model's outputs to the style of a particular musician. This can be done by setting up an *adversarial network* [19]. To this effect, we present a quick introduction to the theory of adversarial networks.

Theoretical results show that, under some assumptions, this architecture leads to a good approximation of the musician's style, i. e. the system learns to better predict the musician's playing.

7

A STRUCTURED CO-IMPROVISATION ARCHITECTURE

7.1 ON CO-IMPROVISATION SYSTEMS

Co-improvisation is the general problem of synthesizing new, original sequences based on the analysis of a set of example sequences.

In the music informatics field, co-improvisation refers to systems which analyze some symbolic corpus of musical sequences and extract some *low-level abstract sequential structure* from this corpus. These systems first compute a symbolic abstraction of the provided corpus using the chosen criterion, then analyze it to find some temporal patterns and construct a *memory* which embeds these patterns. Co-improvisation then consists in synthesizing new sequences using this constructed structural memory.

These systems can also operate on concrete data, such as audio, via a symbolization process, as presented in Chapter 6, given a chosen symbolization step.

These systems can be used in real-time in conjunction with a live musician. For instance, one can generate some live harmonic accompaniment for a musician when the chosen criterion is harmonic similarity.

Existing approaches vary in how much they constrain the sequence generation and roughly fall into three general classes.

FREE GENERATION The first group of systems is devoted entirely to *free*, *unconstrained* generation. These systems first build their structured memory from the corpus, then freely navigate it to output some new sequences. Here, the different approaches differ in the *computational model* used for the construction of the memory.

Existing systems include the Continuator, by Pachet [42], which is based on Markov chains, and OMax [3], based on Factor Oracles [2], automata aimed at extracting repeated sub-sequences within a symbolic sequence.

REACTIVE SYNTHESIS The second group of approaches are *reactive systems* which extend the previous approaches by allowing to guide the synthesis by a stream of inputs. At each generation step, the next output is chosen based on a provided compatibility criterion. For instance, if the inputs are MIDI notes from a live musician, one can perform an harmonic analysis and constrain the model to generate some harmonically compatible accompaniment.

Example of such systems include SoMax – an extension to OMax which guides the run on the memory via a local optimization of the compatibility criterion. Another example is VirtualBand, by Moreira,

Roy, and Pachet [37]. VirtualBand which is built around a dictionary of solo instrumental recordings by different musicians on different instruments and in different styles. A solo musician can then construct his own band by selecting different instruments, then choosing a style, then playing: the backing music is generated using the recorded corpus and adapts (via real-time music information retrieval tools) to different parameters of the human musician, e. g. picth, volume and density of notes.

SCENARIO-BASED GENERATION Finally, other, recent approaches constrain the very musical scenario followed by the generation process. This scenario can be described in the form of a symbolic sequence, which the human improviser is also expected to follow. This sequence is then used by the co-improvisation engine to sequentially guide the generation, thus enforcing the underlying structure of the generated music.

One of the key advantages of this approach is the ability to use future information to make better generation choices. Indeed, by performing lookaheads into the future steps of the scenario, the co-improvisation engine can perform some *anticipations* on what the human improviser is going to do. In the case of a harmonic scenario (e. g. a chord progression), this allows to introduce cadences or modulations in synchronization with the musician.

An example of scenario-based system is Improtek by Nika, Chemillier, and Assayag [38]. Using pattern-matching algorithms on symbolic sequences, the symbolized inputs from the musician are compared with the scenario to follow it in real-time. Improtek is also partly reactive in that it can dynamically rewrite the scenario using external information [39], adapting the generated outputs in real-time to this evolving scenario. The scenario can be modified for instance via parameters controlled in real-time by an external operator or using a set of pre-coded formal rewriting rules.

Nonetheless, a basic scenario still has to be provided prior to the performance.

We propose a theoretical architecture that allows to bridge the apparent gap between the fully reactive methods and the scenario-based approaches, by providing a hybrid dynamically inferred-and-refined scenario. In the concept of this prototype, the scenario sent to the external scenario-based co-improvisation engine (for instance Improtek) is *inferred in real-time* from the music played by the human improviser and incorporates *dynamically updated short-term anticipations*.

7.2 STRUCTURED CO-IMPROVISATION WITH INFERRED SHORT-TERM SCENARIO

This architecture makes crucial use of the scenario inference system presented in the previous section.

We fix here a *rate* at which our networks operate (this allows to operate on a *discrete* time-scale), for instance the scale of the beat, with a fixed tempo.

In the proposed architecture, the *short-term* scenario of the music played by the live musician is computed in real-time by analyzing its inputs, computing the associated chroma vector sequence and symbolizing it using the previously trained clustering. This short-term scenario is passed on to the co-improvisation tool, Improtek in our case.

Furthermore, the prediction network allows to generate a real-time probable scenario for the next time step(s), which indeed makes it possible to perform short-term anticipation on the music played by the musician (i. e. use at time t the inferred information for time $t+1, t+2\ldots$) and introduce smooth transitions in the music generated by the co-improvisation tool, for instance through cadences or modulations.

At each tick of the clock (defined by the chosen rate), a new short-term scenario is therefore passed on to Improtek, dependent on the current inputs to the system, the new predictions as well as the past predictions.

The whole architecture is displayed in Figure 10.

7.3 SOFTWARE ARCHITECTURE

In order to implement this architecture, an OSC (Open Sound Control [55]) network is set up.

The input stream (audio or MIDI) is run through Max/MSP which performs a beat-by-beat chroma analysis.

The prediction block functions as a server which receives the chroma vectors from Max/MSP, runs them through the prediction neural network and sends the predicted chroma vectors to the symbolization neural network via OSC.

The symbolization (clustering) engine receives both the input chromas and the chromas for the predicted continuation. It concatenates them and finally turns this sequence of chromas into an abstract sequence of cluster labels, the partly inferred, partly predicted real-time scenario.

This scenario is then sent to Improtek via OSC, which can generate an output sequence following it.

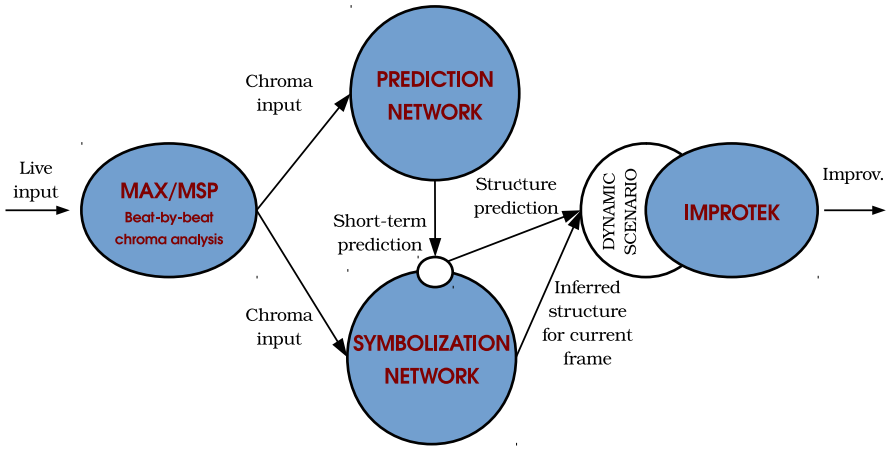

Figure 10: Proposed co-improvisation architecture

This process is *dynamic*: the short-term scenario used by Improtek is re-updated in real-time by comparing the predictions done with the actual outcomes. If the predictions were wrong, the sequence currently generated by Improtek is updated to follow the new scenario.

8

STYLE-ADAPTATION

In this last section, we propose the prototype of an architecture for *style-adaptation* (or style *modeling*). In this setup, the extension to the co-improvisation engine we presented in the previous section is further trained, in real-time, on the human improviser's inputs. The goal here is to better adapt the network's predictions to the music played by the musician, hopefully leading to a better scenario and more semantical synchronization between the human and the machine.

Two approaches to this are proposed, the first of which is rather trivial. The second relies on *adversarial training* [19], a recently introduced training framework for *generative models*. To this end, we present a review of generative networks and adversarial training. Adversarial training allows to build models which efficiently synthesize data resembling examples from a given dataset. Here, we want to build a network which generates predictions which closely resemble the playing of the live musician whose style we want to mimic.

In both cases, at each discrete time step t, the network's prediction $o(t)$ is compared with the musician's actual output $x(t)$ and we try to further minimize the prediction error.

Note that the gains implied by these approaches can be evaluated by monitoring the evolution of the mean prediction accuracy whilst performing this further training. But it should even more crucially be assessed during improvisation sessions with a human musician.

8.1 NAIVE FINETUNING

Finetuning refers to using a small task-specific dataset to further train a network already trained on a large corpus of examples.

This allows to effectively "finetune" the networks weights to better solve the task at hand.

The easy way of performing style-adaptation on a given musician is thus to finetune the prediction network using the the new data produced in real-time by the musician as new training examples. Thus, we continuously *perform gradient descent* on the *musician's outputs*: at each instant t, a new example pair is available where the training sequence is $x(t-t_0)x(t-t_0+1)\dots x(t-1)$ and the target is $x(t)$.

8.2 ADVERSARIAL TRAINING

The second approach is more advanced and relies on so called *adversarial training*.

The context here is that of *generative* networks: neural networks trained to generate new data resembling the data they were given as example. In a probabilistic approach, these networks try to *approximate the density distribution of their input data* in order to *sample from it* afterwards. Adversarial training is a means to further refine the outputs of such networks, hopefully making them *indistinguishable* from real example data.

We first present the Variational Autoencoder, a popular example of generative model, then we introduce adversarial training.

8.2.1 *Variational autoencoders*

Variational auto-encoders (VAEs) [29] are a popular type of generative networks.

These networks are built as the succession of two deep networks:

- An *encoder* E, which computes a vector $E(x)$ of high-level features based on the input data x. E can for instance be a standard MLP,
- A *decoder* D, which is trained to *reconstruct* the input data from the code computed by the encoder, i.e. trained with the target criterion $\forall x, D(E(x)) = x$.

Such a network is displayed in Figure 11.

The specificity of variational autoencoders is that they consider the outputs of the encoder as a *random variable* which follows a *chosen distribution*, dependent on some trainable parameters. For instance, the introductory article puts a *Gaussian prior* on the outputs of the encoder.

The parameters of the distribution are then trained so as to obtain a proper autoencoder. Indeed, by using a *specific training criterion* (obtained via the so-called *variational bound*), the networks can be trained in such a way that: for ϵ small enough compared to $E(x)$, $G(E(x) + \epsilon)$ constructs a data point which *resembles* the original data (according to the metric on the output space $\mathcal{G}$). In effect, one can generate new, meaningful data samples by small perturbations of the code generated by the variational auto-encoder.

VAEs were applied with success to the problem of image generation, for instance within the DRAW model developed at Google by Gregor et al. [20].

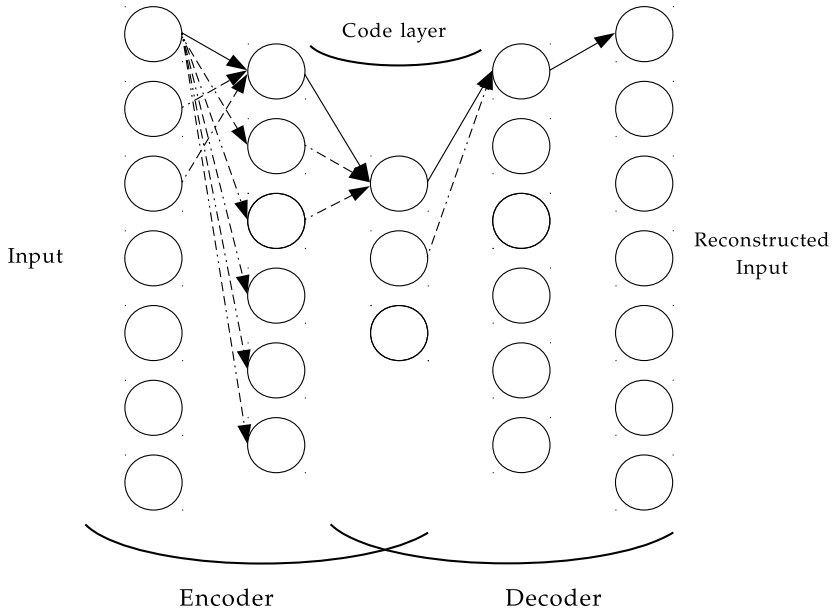

Figure 11: Autoencoder

8.2.2 *Adversarial networks*

Adversarial networks were introduced by Goodfellow et al. [19] as an original way of improving the capacities of a *generative* network to simulate a given process.

In this context, two networks are considered:

1. The *generative* network G, which is trained to simulate a data distribution $\mathcal{D}$ with $\mathcal{D} \subset \mathcal{G}$.

2. A *classifier* C, trained to recognize data in $\mathcal{D}$. For an input x in $\mathcal{G}$, C is expected to return 1 if $x \in \mathcal{D}$ and 0 otherwise.

After an initial phase of isolated training of those two networks, so as to have their weights properly initialized, they are further trained by connecting them and training them with the following criterion:

- An example x is either chosen from the provided dataset or synthesized by G,

- D tries to devise if x is a real datapoint from the provided dataset or if it was synthesized,

- If D is right, G receives a penalty, i. e. its parameters are updated so as to better synthesize,

- If D is wrong, it receives a penalty, i. e. its parameters are updated so as to better discriminate.

Formally, D and G play a *zero-sum* game. Using this setup, it can be theoretically shown [19] that the density estimated by G progressively converges towards the original data distribution. In other words, G learns to generate examples indistinguishable from the original data.

8.2.3 *Style adaptation via adversarial training*

Now that we have defined adversarial networks, we can see the style-adaptation problem as a generative problem in adversarial context: the prediction network indeed tries to simulate the playing of the live musician.

To set up the architecture, we replace the LSTMs used previously by their variational counterparts, as introduced by Bowman et al. [11].

Then, we use a classifier trained to discriminate between chromas generated by the prediction system and chromas stemming from a real musician. A possible choice for this classifier would be a Convnet as they have proved very powerful at classification (e. g. on MNIST [33]).

We can then use adversarial training to further adjust the prediction network to the musician's style.

8.3 EVALUATION

Three variants of our co-improvisation architecture should be compared:

1. The vanilla architecture, were the prediction network is not further trained during activity,
2. The "naive finetuning" architecture, in which traditional back-propagation/gradient descent-based is performed in real-time using the new examples continuously provided by the live musician,
3. The adversarial architecture, using a variational recurrent network.

The go-to means of evaluating the effect of these training techniques is via simple standard test error monitoring. Even better in the case of an improvisation tool will be to have a musician try out all variants and give feedback on each of them.

CONCLUSION

CONCLUSION

We have presented a machine learning approach to *musical scenario inference*, using *deep recurrent neural networks*. *Deep* LSTMs analyze time-series at multiple time-scales and allow to perform predictions taking these different time-scales into account. A network training pipeline was set up, using the large MILLION SONG DATASET. The computations are still running at the time being, with an hyper-parameter optimization loop devising an appropriate architecture for the considered problem.

By using *clustering methods*, we can furthermore turn any given sequence of chroma vectors into a sequence of abstract labels, thus extracting a notion of underlying higher-level structure from these chromagrams.

The presented tool could be used in conjunction with a so called *co-improvisation* system, a system built to synthesize sequences based on a corpus of examples, following some notion of temporal structure learned on this corpus. Our tool could provide a co-improvisation engine with a real-time, dynamic *scenario*, inferred for instance from the music played by a live musician.

Thanks to this and without any prior knowledge of the music played by the live human improviser, the co-improvisation tool, e.g. Improtek, will be able to make predictions on the live musician's playing and introduce anticipation and transitions in the generated accompaniment.

Finally, this architecture could be further optimized using real-time training to perform *style adaptation* on a given musician. Two solutions are envisioned, a rather naive one and a more complex one, which makes use of *adversarial training*.

The results presented in this report are still prospective. The computations associated with LSTMs are heavy and the training of the networks is not done yet. There is therefore still work to do on gathering the quantitative results of the hyper-optimization loop and analyzing those results.

Once this will have been done, the co-improvisation architecture should be set-up and the style-adaptation schemes implemented. *In situ* testing with a musician will then be required to asses the usability and the advantages of the methods proposed.

BIBLIOGRAPHY

[1] D. Akagi. "A Primer on Deep Learning." blog post. 2013. URL: https://www.datarobot.com/blog/a-primer-on-deep-learning/.

[2] C. Allauzen, M. Crochemore, and M. Raffinot. "Factor oracle : a new structure for pattern matching." In: *26th Seminar on Current Trends in Theory and Practice of Informatics (SOFSEM'99)*. Ed. by P. Jan, T. Gerard, and B. Miroslav. Vol. 1725. LNCS. Milovy, Czech Republic, Czech Republic: Springer-Verlag, Nov. 1999, pp. 291–306. URL: https://hal-upec-upem.archives-ouvertes.fr/hal-00619846.

[3] G. Assayag et al. "OMAX Brothers: A Dynamic Topology of Agents for Improvisation Learning." In: *ACM Multimedia Workshop on Audio and Music Computing for Multimedia*. Santa Barbara, United States: Santa Barbara, 2006. URL: https://hal.inria.fr/hal-00839075.

[4] M. Bay, A. F. Ehmann, and J. S. Downie. "Evaluation of Multiple-Fo Estimation and Tracking Systems." In: *Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, Kobe International Conference Center, Kobe, Japan, October 26-30, 2009*. 2009, pp. 315–320. URL: http://ismir2009.ismir.net/proceedings/PS2-21.pdf.

[5] Y. Bengio, A. C. Courville, and P. Vincent. "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives." In: *CoRR* abs/1206.5538 (2012). URL: http://arxiv.org/abs/1206.5538.

[6] Y. Bengio, P. Simard, and P. Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult." In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. URL: http://www.iro.umontreal.ca/~lisa/pointeurs/ieeetrnn94.pdf.

[7] Y. Bengio et al. "Greedy layer-wise training of deep networks." In: *In NIPS*. MIT Press, 2007.

[8] T. Bertin-Mahieux et al. "The Million Song Dataset." In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*. 2011.

[9] L. Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent." In: *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Ed. by Y. Lechevallier and G. Saporta. Heidelberg: Physica-Verlag HD,

2010, pp. 177–186. URL: http://dx.doi.org/10.1007/978-3-7908-2604-3_16.

[10] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription." In: *ArXiv e-prints* (June 2012). arXiv: 1206.6392 [cs.LG].

[11] S. R. Bowman et al. "Generating Sentences from a Continuous Space." In: *CoRR* abs/1511.06349 (2015). URL: http://arxiv.org/abs/1511.06349.

[12] E. Cambria and B. White. "Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]." In: *IEEE Comp. Int. Mag.* 9.2 (2014), pp. 48–57. URL: http://dx.doi.org/10.1109/MCI.2014.2307227.

[13] R. Collobert, S. Bengio, and J. Marithoz. *Torch: A Modular Machine Learning Software Library*. 2002.

[14] W. B. De Haas et al. "Automatic functional harmonic analysis." In: *Computer Music Journal* 37.4 (2013), pp. 37–53.

[15] J. Duchi, E. Hazan, and Y. Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. Tech. rep. UCB/EECS-2010-24. EECS Department, University of California, Berkeley, Mar. 2010. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html.

[16] A. Forte and S. Gilbert. *Introduction to Schenkerian Analysis*. Norton, 1982. URL: https://books.google.fr/books?id=IL99ygAACAAJ.

[17] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*. 2010.

[18] X. Glorot, A. Bordes, and Y. Bengio. "Deep Sparse Rectifier Neural Networks." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*. Ed. by G. J. Gordon and D. B. Dunson. Vol. 15. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011, pp. 315–323. URL: http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf.

[19] I. J. Goodfellow et al. "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.

[20] K. Gregor et al. "DRAW: A Recurrent Neural Network For Image Generation." In: *CoRR* abs/1502.04623 (2015). URL: http://arxiv.org/abs/1502.04623.

[21] C. Harte, M. Sandler, and M. Gasser. "Detecting Harmonic Change in Musical Audio." In: *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia*. AMCMM '06. Santa Barbara, California, USA: ACM, 2006, pp. 21–26. URL: http://doi.acm.org/10.1145/1178723.1178727.

[22] K. He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *CoRR* abs/1502.01852 (2015). URL: http://arxiv.org/abs/1502.01852.

[23] M. Henderson. "Machine Learning for Dialog State Tracking: A Review." In: *Proceedings of The First International Workshop on Machine Learning in Spoken Language Processing*. 2015.

[24] D. Herremans and S. Kenneth. "Composing first species counterpoint with a variable neighbourhood search algorithm." In: *Journal of Mathematics and the Arts* 6.4 (2012), pp. 169–189. eprint: http://dx.doi.org/10.1080/17513472.2012.738554. URL: http://dx.doi.org/10.1080/17513472.2012.738554.

[25] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory." In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[26] M. J. Huiskes, B. Thomee, and M. S. Lew. "New Trends and Ideas in Visual Concept Detection: The MIR Flickr Retrieval Evaluation Initiative." In: *Proceedings of the International Conference on Multimedia Information Retrieval*. MIR '10. Philadelphia, Pennsylvania, USA: ACM, 2010, pp. 527–536. URL: http://doi.acm.org/10.1145/1743384.1743475.

[27] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data Clustering: A Review." In: *ACM Comput. Surv.* 31.3 (Sept. 1999), pp. 264–323. URL: http://doi.acm.org/10.1145/331499.331504.

[28] T. Kanungo et al. "An Efficient k-Means Clustering Algorithm: Analysis and Implementation." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 24.7 (July 2002), pp. 881–892. URL: http://dx.doi.org/10.1109/TPAMI.2002.1017616.

[29] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes." In: *CoRR* abs/1312.6114 (2013). URL: http://arxiv.org/abs/1312.6114.

[30] S. Kullback and R. A. Leibler. "On Information and Sufficiency." In: *Ann. Math. Statist.* 22.1 (Mar. 1951), pp. 79–86. URL: http://dx.doi.org/10.1214/aoms/1177729694.

[31] Y. LeCun et al. "Effiicient BackProp." In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998, pp. 9–50. URL: http://dl.acm.org/citation.cfm?id=645754.668382.

[32] Y. Lecun et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[33] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition." In: *Neural Comput.* 1.4 (Dec. 1989), pp. 541–551. URL: http://dx.doi.org/10.1162/neco.1989.1.4.541.

[34] F. Lerdahl and R. Jackendoff. *A generative theory of tonal music*. Cambridge. MA: The MIT Press, 1983.

[35] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. URL: http://dx.doi.org/10.1007/BF02478259.

[36] M. L. Minsky and S. A. Papert. *Perceptrons: Expanded Edition*. Cambridge, MA, USA: MIT Press, 1988.

[37] J. Moreira, P. Roy, and F. Pachet. "Virtualband: Interacting with Stylistically Consistent Agents." In: *ISMIR*. Ed. by A. de Souza Britto Jr., F. Gouyon, and S. Dixon. 2013, pp. 341–346. URL: http://dblp.uni-trier.de/db/conf/ismir/ismir2013.html#MoreiraRP13.

[38] J. Nika, M. Chemillier, and G. Assayag. "ImproteK: Introducing Scenarios into Human-Computer Music Improvisation." In: *ACM Computers in Entertainment, Special issue on Musical Metacreation* (2016). (To appear).

[39] J. Nika et al. "Guided improvisation as dynamic calls to an offline model." In: *Sound and Music Computing (SMC)*. Maynooth, Ireland, July 2015. URL: https://hal.archives-ouvertes.fr/hal-01184642.

[40] C. Olah. *Understanding LSTM Networks*. blog post. Aug. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[41] C. Olah. *Visualizing Representations: Deep Learning and Human Beings*. blog post. Jan. 2015. URL: https://colah.github.io/posts/2015-01-Visualizing-Representations/.

[42] F. Pachet. "The Continuator: Musical Interaction with Style." In: *Proceedings of the 2002 International Computer Music Conference, ICMC 2002, Gothenburg, Sweden, September 16-21, 2002*. 2002. URL: http://hdl.handle.net/2027/spo.bbp2372.2002.044.

[43] J.-F. Paiement, S. Bengio, and D. Eck. "Probabilistic models for melodic prediction." In: *Artificial Intelligence* 173.14 (2009), pp. 1266–1274. URL: http://www.sciencedirect.com/science/article/pii/S0004370209000654.

[44] H. Palangi et al. "Deep Sentence Embedding Using the Long Short Term Memory Network: Analysis and Application to Information Retrieval." In: *CoRR* abs/1502.06922 (2015). URL: http://arxiv.org/abs/1502.06922.

[45] J. Paulus, M. Müller, and A. Klapuri. "Audio-based music structure analysis." In: *in Proc. of the Int. Society for Music Information Retrieval Conference*. 2010.

[46] C. Raphael and J. Stoddard. "Functional Harmonic Analysis Using Probabilistic Models." In: *Computer Music Journal* 28.3 (2004), pp. 45–52. URL: http://www.jstor.org/stable/3681508.

[47] M. Rohrmeier. "A Generative Grammar Approach to Diatonic Harmonic Structure." In: *In Anagnostopoulou Georgaki, Kouroupetroglou, editor, Proceedings of the 4th Sound and Music Computing Conference*. 2007, pp. 97–100.

[48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Neurocomputing: Foundations of Research." In: ed. by J. A. Anderson and E. Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. URL: http://dl.acm.org/citation.cfm?id=65669.104451.

[49] T. N. Sainath et al. "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks." In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*. 2015, pp. 4580–4584. URL: http://dx.doi.org/10.1109/ICASSP.2015.7178838.

[50] H. Sak, A. W. Senior, and F. Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." In: *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*. 2014, pp. 338–342. URL: http://www.isca-speech.org/archive/interspeech_2014/i14_0338.html.

[51] P. Sermanet et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks." In: *CoRR* abs/1312.6229 (2013). URL: http://arxiv.org/abs/1312.6229.

[52] J. A. Snyman. *Practical mathematical optimization : an introduction to basic optimization theory and classical and new gradient-based algorithms*. Applied optimization. New York: Springer, 2005. URL: http://opac.inria.fr/record=b1132592.

[53] H. Soltau et al. "Recognition of music types." In: *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98, Seattle, Washington, USA, May 12-15, 1998*. 1998, pp. 1137–1140. URL: http://dx.doi.org/10.1109/ICASSP.1998.675470.

[54] N. Srivastava, E. Mansimov, and R. Salakhutdinov. "Unsupervised Learning of Video Representations using LSTMs." In: *CoRR* abs/1502.04681 (2015). URL: http://arxiv.org/abs/1502.04681.

[55] *Open Sound Control: State of the Art 2003*. OpenSound Control. Montreal, 2003, pp. 153–159. URL: http://cnmat.berkeley.edu/publications/open_sound_control_state_art_2003.

[56] M. D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method." In: *CoRR* abs/1212.5701 (2012). URL: http://arxiv.org/abs/1212.5701.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede.

https://bitbucket.org/amiede/classicthesis/

Final Version as of July 29, 2016 (`classicthesis` version 1.0).