Abstract

Many research works deal with chaotic neural networks for various fields of application. Unfortunately, up to now these networks are usually claimed to be chaotic without any mathematical proof. The purpose of this paper is to establish, based on a rigorous theoretical framework, an equivalence between chaotic iterations according to Devaney and a particular class of neural networks. On the one hand we show how to build such a network, on the other hand we provide a method to check if a neural network is a chaotic one. Finally, the ability of classical feedforward multilayer perceptrons to learn sets of data obtained from a dynamical system is regarded. Various Boolean functions are iterated on finite states. Iterations of some of them are proven to be chaotic as it is defined by Devaney. In that context, important differences occur in the training process, establishing with various neural networks that chaotic behaviors are far more difficult to learn.

Neural Networks and Chaos: Construction, Evaluation of Chaotic Networks, and Prediction of Chaos with Multilayer Feedforward Networks

Jacques M. Bahi,Jean-François Couchot,Christophe Guyeux,Michel Salomon
Computer Science Laboratory (LIFC), University of Franche-Comté,
IUT de Belfort-Montbéliard, BP 527,
90016 Belfort Cedex, France

December 24, 2011

Chaotic neural networks have received a lot of attention due to the appealing properties of deterministic chaos (unpredictability, sensitivity, and so on). However, such networks are often claimed chaotic without any rigorous mathematical proof. Therefore, in this work a theoretical framework based on the Devaney's definition of chaos is introduced. Starting with a relationship between discrete iterations and Devaney's chaos, we firstly show how to build a recurrent neural network that is equivalent to a chaotic map and secondly a way to check whether an already available network is chaotic or not. We also study different topological properties of these truly chaotic neural networks. Finally, we show that the learning, with neural networks having a classical feedforward structure, of chaotic behaviors represented by data sets obtained from chaotic maps, is far more difficult than non chaotic behaviors.

1 Introduction

Several research works have proposed or used chaotic neural networks these last years. The complex dynamics of such networks lead to various potential application areas: associative memories [?] and digital security tools like hash functions [?], digital watermarking [?, ?], or cipher schemes [?]. In the former case, the background idea is to control chaotic dynamics in order to store patterns, with the key advantage of offering a large storage capacity. For the latter case, the use of chaotic dynamics is motivated by their unpredictability and random-like behaviors. Indeed, investigating new concepts is crucial for the computer security field, because new threats are constantly emerging. As an illustrative example, the former standard in hash functions, namely the SHA-1 algorithm, has been recently weakened after flaws were discovered.

Chaotic neural networks have been built with different approaches. In the context of associative memory, chaotic neurons like the nonlinear dynamic state neuron [?] frequently constitute the nodes of the network. These neurons have an inherent chaotic behavior, which is usually assessed through the computation of the Lyapunov exponent. An alternative approach is to consider a well-known neural network architecture: the MultiLayer Perceptron (MLP). These networks are suitable to model nonlinear relationships between data, due to their universal approximator capacity [?, ?]. Thus, this kind of networks can be trained to model a physical phenomenon known to be chaotic such as Chua's circuit [?]. Sometime a neural network, which is build by combining transfer functions and initial conditions that are both chaotic, is itself claimed to be chaotic [?].

What all of these chaotic neural networks have in common is that they are claimed to be chaotic despite a lack of any rigorous mathematical proof. The first contribution of this paper is to fill this gap, using a theoretical framework based on the Devaney's definition of chaos [?]. This mathematical theory of chaos provides both qualitative and quantitative tools to evaluate the complex behavior of a dynamical system: ergodicity, expansivity, and so on. More precisely, in this paper, which is an extension of a previous work [?], we establish the equivalence between chaotic iterations and a class of globally recurrent MLP. The second contribution is a study of the converse problem, indeed we investigate the ability of classical multilayer perceptrons to learn a particular family of discrete chaotic dynamical systems. This family is defined by a Boolean vector, an update function, and a sequence defining the component to update at each iteration. It has been previously established that such dynamical systems are chaotically iterated (as it is defined by Devaney) when the chosen function has a strongly connected iterations graph. In this document, we experiment several MLPs and try to learn some iterations of this kind. We show that non-chaotic iterations can be learned, whereas it is far more difficult for chaotic ones. That is to say, we have discovered at least one family of problems with a reasonable size, such that artificial neural networks should not be applied due to their inability to learn chaotic behaviors in this context.

The remainder of this research work is organized as follows. The next section is devoted to the basics of Devaney's chaos. Section 3 formally describes how to build a neural network that operates chaotically. Section 4 is devoted to the dual case of checking whether an existing neural network is chaotic or not. Topological properties of chaotic neural networks are discussed in Sect. 5. The Section 6.1 shows how to translate such iterations into an Artificial Neural Network (ANN), in order to evaluate the capability for this latter to learn chaotic behaviors. This ability is studied in Sect. 6.2, where various ANNs try to learn two sets of data: the first one is obtained by chaotic iterations while the second one results from a non-chaotic system. Prediction success rates are given and discussed for the two sets. The paper ends with a conclusion section where our contribution is summed up and intended future work is exposed.

2 Chaotic Iterations according to Devaney

In this section, the well-established notion of Devaney's mathematical chaos is firstly recalled. Preservation of the unpredictability of such dynamical system when implemented on a computer is obtained by using some discrete iterations called "asynchronous iterations", which are thus introduced. The result establishing the link between such iterations and Devaney's chaos is finally presented at the end of this section.

In what follows and for any function f, f^n means the composition $f \circ f \circ \ldots \circ f$ (n times) and an **iteration** of a **dynamical system** is the step that consists in updating the global state x^t at time t with respect to a function f s.t. $x^{t+1} = f(x^t)$.

2.1 Devaney's chaotic dynamical systems

Various domains such as physics, biology, or economy, contain systems that exhibit a chaotic behavior, a well-known example is the weather. These systems are in particular highly sensitive to initial conditions, a concept usually presented as the butterfly effect: small variations in the initial conditions possibly lead to widely different behaviors. Theoretically speaking, a system is sensitive if for each point x in the iteration space, one can find a point in each neighborhood of x having a significantly different future evolution. Conversely, a system seeded with the same initial conditions always has the same evolution. In other words, chaotic systems have a deterministic behavior defined through a physical or mathematical model and a high sensitivity to the initial conditions. Besides mathematically this kind of unpredictability is also referred to as deterministic chaos. For example, many weather forecast models exist, but they give only suitable predictions for about a week, because they are initialized with conditions that reflect only a partial knowledge of the current weather. Even if the differences are initially small, they are amplified in the course of time, and thus make difficult a long-term prediction. In fact, in a chaotic system, an approximation of the current state is a quite useless indicator for predicting future states.

From mathematical point of view, deterministic chaos has been thoroughly studied these last decades, with different research works that have provide various definitions of chaos. Among these definitions, the one given by Devaney [**?**] is well-established. This definition consists of three conditions: topological transitivity, density of periodic points, and sensitive point dependence on initial conditions.

Topological transitivity is checked when, for any point, any neighborhood of its future evolution eventually overlap with any other given region. This property implies that a dynamical system cannot be broken into simpler subsystems. Intuitively, its complexity does not allow any simplification.

However, chaos needs some regularity to "counteracts" the effects of transitivity. In the Devaney's formulation, a dense set of periodic points is the element of regularity that a chaotic dynamical system has to exhibit. We recall that a point x is a **periodic point** for f of period $n \in \mathbb{N}^*$ if $f^n(x) = x$. Then, the map f is **regular** on the topological space $(\mathcal{X}, \tau)$ if the set of periodic points for f is dense in $\mathcal{X}$ (for any $x \in \mathcal{X}$, we can find at least one periodic point in any of its neighborhood). Thus, due to these two properties, two points close to each other can behave in a completely different manner, leading to unpredictability for the whole system.

Let us recall that f has **sensitive dependence on initial conditions** if there exists $\delta > 0$ such that, for any $x \in \mathcal{X}$ and any neighborhood V of x, there exist $y \in V$ and $n > 0$ such that $d(f^n(x), f^n(y)) > \delta$. The value δ is called the **constant of sensitivity** of f.

Finally, the dynamical system that iterates f is **chaotic according to Devaney** on $(\mathcal{X}, \tau)$ if f is regular, topologically transitive, and has sensitive dependence to its initial conditions. In what follows, iterations are said to be chaotic (according to Devaney) when the corresponding dynamical system is chaotic, as it is defined in the Devaney's formulation.

2.2 Asynchronous Iterations

Let us firstly discuss about the domain of iteration. As far as we know, no result rules that the chaotic behavior of a dynamical system that has been theoretically proven on $\mathbb{R}$ remains valid on the floating-point numbers, which is the implementation domain. Thus, to avoid loss of chaos this work presents an alternative, that is to iterate Boolean maps: results that are theoretically obtained in that domain are preserved in implementations.

Let us denote by $[\![a;b]\!]$ the following interval of integers: $\{a, a+1, \ldots, b\}$, where $a < b$. In this section, a *system* under consideration iteratively modifies a collection of n components. Each component $i \in [\![1;n]\!]$ takes its value x_i among the domain $\mathbb{B} = \{0,1\}$. A *configuration* of the system at discrete time t is the vector $x^t = (x_1^t, \ldots, x_n^t) \in \mathbb{B}^n$. The dynamics of the system is described according to a function $f : \mathbb{B}^n \to \mathbb{B}^n$ such that $f(x) = (f_1(x), \ldots, f_n(x))$.

Let be given a configuration x. In what follows $N(i,x) = (x_1, \ldots, \overline{x_i}, \ldots, x_n)$ is the configuration obtained by switching the $i-$th component of x ($\overline{x_i}$ is indeed the negation of x_i). Intuitively, x and $N(i,x)$ are neighbors. The discrete iterations of f are represented by the oriented *graph of iterations* $\Gamma(f)$. In such a graph, vertices are configurations of $\mathbb{B}^n$ and there is an arc labeled i from x to $N(i,x)$ if and only if $f_i(x)$ is $N(i,x)$.

In the sequel, the *strategy* $S = (S^t)^{t \in \mathbb{N}}$ is the sequence defining which component to update at time t and S^t denotes its $t-$th term. This iteration scheme that only modifies one element at each iteration is usually referred as *asynchronous iterations*. More precisely, we have for any i, $1 \le i \le n$,

$$\left\{ \begin{array}{l} x^0 \in \mathbb{B}^n \\ x_i^{t+1} = \left\{ \begin{array}{l} f_i(x^t) \text{ if } S^t = i \ , \\ x_i^t \text{ otherwise } . \end{array} \right. \end{array} \right. \quad (1)$$

Next section shows the link between asynchronous iterations and Devaney's chaos.

2.3 On the link between asynchronous iterations and Devaney's Chaos

In this subsection we recall the link we have established between asynchronous iterations and Devaney's chaos. The theoretical framework is fully described in [**?**].

We introduce the function F_f that is defined for any given application $f : \mathbb{B}^n \to \mathbb{B}^n$ by $F_f : [\![1;n]\!] \times \mathbb{B}^n \to \mathbb{B}^n$, s.t.

$$F_f(s,x)_j = \left\{ \begin{array}{l} f_j(x) \text{ if } j = s \ , \\ x_j \text{ otherwise } . \end{array} \right. \tag{2}$$

With such a notation, asynchronously obtained configurations are defined for times
$t = 0, 1, 2, \dots$ by:

$$\left\{ \begin{array}{l} x^0 \in \mathbb{B}^n \text{ and} \\ x^{t+1} = F_f(S^t, x^t) \ . \end{array} \right. \tag{3}$$

Finally, iterations defined in Eq. (3) can be described by the following system:

$$\left\{ \begin{array}{lcl} X^0 & = & ((S^t)^{t\in\mathbb{N}}, x^0) \in [\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n \\ X^{k+1} & = & G_f(X^k) \\ \multicolumn{3}{l}{\text{where } G_f\left(((S^t)^{t\in\mathbb{N}}, x)\right) = \left(\sigma((S^t)^{t\in\mathbb{N}}), F_f(S^0, x)\right) \ ,} \end{array} \right. \tag{4}$$

where σ is the so-called shift function that removes the first term of the strategy (*i.e.*, S^0). This definition allows to link asynchronous iterations with classical iterations of a dynamical system. Note that it can be extended by considering subsets for S^t.

To study topological properties of these iterations, we are then left to introduce a **distance** d between two points (S, x) and $(\check{S}, \check{x})$ in $\mathcal{X} = [\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$. Let $\Delta(x, y) = 0$ if $x = y$, and $\Delta(x, y) = 1$ else, be a distance on $\mathbb{B}$. The distance d is defined by

$$d((S,x);(\check{S},\check{x})) = d_e(x,\check{x}) + d_s(S,\check{S}) \ , \tag{5}$$

where

$$d_e(x,\check{x}) = \sum_{j=1}^{n} \Delta(x_j, \check{x}_j) \in [\![0;n]\!] \tag{6}$$

and

$$d_s(S,\check{S}) = \frac{9}{2n} \sum_{t=0}^{\infty} \frac{|S^t - \check{S}^t|}{10^{t+1}} \in [0;1] \ . \tag{7}$$

This distance is defined to reflect the following information. Firstly, the more two systems have different components, the larger the distance between them. Secondly, two systems with similar components and strategies, which have the same starting terms, must induce only a small distance. The proposed distance fulfills these requirements: on the one hand its floor value reflects the difference between the cells, on the other hand its fractional part measures the difference between the strategies.

The relation between $\Gamma(f)$ and G_f is obvious: there exists a path from x to x' in $\Gamma(f)$ if and only if there exists a strategy s such that iterations of G_f from the initial point (s,x) reach the configuration x'. Using this link, Guyeux [?] has proven that,

Theorem 1 *Let $f : \mathbb{B}^n \to \mathbb{B}^n$. Iterations of G_f are chaotic according to Devaney if and only if $\Gamma(f)$ is strongly connected.*

Checking if a graph is strongly connected is not difficult (by the Tarjan's algorithm for instance). Let be given a strategy S and a function f such that $\Gamma(f)$ is strongly connected. In that case, iterations of the function G_f as defined in Eq. (4) are chaotic according to Devaney.

Let us then define two functions f_0 and f_1 both in $\mathbb{B}^n \to \mathbb{B}^n$ that are used all along this paper. The former is the vectorial negation, *i.e.*, $f_0(x_1,\dots,x_n) = (\overline{x_1},\dots,\overline{x_n})$. The latter is $f_1(x_1,\dots,x_n) = (\overline{x_1}, x_1, x_2, \dots, x_{n-1})$. It is not hard to check that $\Gamma(f_0)$ and $\Gamma(f_1)$ are both strongly connected, then iterations of G_{f_0} and of G_{f_1} are chaotic according to Devaney.

With this material, we are now able to build a first chaotic neural network, as defined in the Devaney's formulation.

3 A chaotic neural network in the sense of Devaney

Let us build a multilayer perceptron neural network modeling $F_{f_0} : [\![1;n]\!] \times \mathbb{B}^n \to \mathbb{B}^n$ associated to the vectorial negation. More precisely, for all inputs $(s,x) \in [\![1;n]\!] \times \mathbb{B}^n$, the output layer will produce $F_{f_0}(s,x)$. It is then possible to link the output layer and the input one, in order to model the dependence between two successive iterations. As a result we obtain a global recurrent neural network that behaves as follows (see Fig. 1).

- The network is initialized with the input vector $\left(S^0, x^0\right) \in [\![1;n]\!] \times \mathbb{B}^n$ and computes the output vector $x^1 = F_{f_0}\left(S^0, x^0\right)$. This last vector is published as an output one of the chaotic neural network and is sent back to the input layer through the feedback links.

- When the network is activated at the t^{th} iteration, the state of the system $x^t \in \mathbb{B}^n$ received from the output layer and the initial term of the sequence $(S^t)^{t\in\mathbb{N}}$ (*i.e.*, $S^0 \in [\![1;n]\!]$) are used to compute the new output vector. This new vector, which represents the new state of the dynamical system, satisfies:
$$x^{t+1} = F_{f_0}(S^0, x^t) \in \mathbb{B}^n \ . \tag{8}$$

The behavior of the neural network is such that when the initial state is $x^0 \in \mathbb{B}^n$ and a sequence $(S^t)^{t\in\mathbb{N}}$ is given as outside input, then the sequence of successive published output vectors $\left(x^t\right)^{t\in\mathbb{N}^*}$ is exactly the one produced by the chaotic iterations formally described in Eq. (4). It means that mathematically

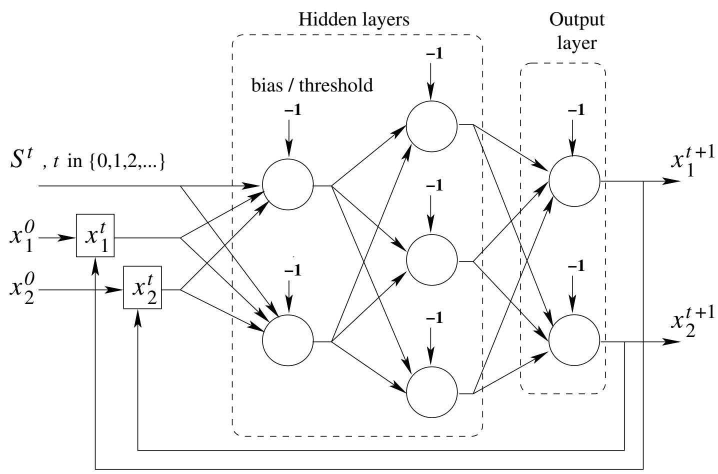

Figure 1: A perceptron equivalent to chaotic iterations

if we use similar input vectors they both generate the same successive outputs $(x^t)^{t\in\mathbb{N}^*}$, and therefore that they are equivalent reformulations of the iterations of G_{f_0} in $\mathcal{X}$. Finally, since the proposed neural network is built to model the behavior of G_{f_0}, whose iterations are chaotic according to the Devaney's definition of chaos, we can conclude that the network is also chaotic in this sense.

The previous construction scheme is not restricted to function f_0. It can be extended to any function f such that G_f is a chaotic map by training the network to model $F_f : [\![1;n]\!] \times \mathbb{B}^n \to \mathbb{B}^n$. Due to Theorem 1, we can find alternative functions f for f_0 through a simple check of their graph of iterations $\Gamma(f)$. For example, we can build another chaotic neural network by using f_1 instead of f_0.

4 Checking whether a neural network is chaotic or not

We focus now on the case where a neural network is already available, and for which we want to know if it is chaotic. Typically, in many research papers neural network are usually claimed to be chaotic without any convincing mathematical proof. We propose an approach to overcome this drawback for a particular category of multilayer perceptrons defined below, and for the Devaney's formulation of chaos. In spite of this restriction, we think that this approach can be extended to a large variety of neural networks.

We consider a multilayer perceptron of the following form: inputs are n binary digits and one integer value, while outputs are n bits. Moreover, each binary output is connected with a feedback connection to an input one.

- During initialization, the network is seeded with n bits denoted $\left(x_1^0, \dots, x_n^0\right)$ and an integer value S^0 that belongs to $[\![1;n]\!]$.
- At iteration t, the last output vector $(x_1^t, \dots, x_n^t)$ defines the n bits used to compute the new output one $\left(x_1^{t+1}, \dots, x_n^{t+1}\right)$. While the remaining input receives a new integer value $S^t \in [\![1;n]\!]$, which is provided by the outside world.

The topological behavior of these particular neural networks can be proven to be chaotic through the following process. Firstly, we denote by $F : [\![1;n]\!] \times \mathbb{B}^n \to \mathbb{B}^n$ the function that maps the value $(s, (x_1, \dots, x_n)) \in [\![1;n]\!] \times \mathbb{B}^n$ into the value $(y_1, \dots, y_n) \in \mathbb{B}^n$, where $(y_1, \dots, y_n)$ is the response of the neural network after the initialization of its input layer with $(s, (x_1, \dots, x_n))$. Secondly, we define $f : \mathbb{B}^n \to \mathbb{B}^n$ such that $f(x_1, x_2, \dots, x_n)$ is equal to

$$\left(F\left(1, (x_1, x_2, \dots, x_n)\right), \dots, F\left(n, (x_1, x_2, \dots, x_n)\right)\right) \quad . \tag{9}$$

Thus, for any j, $1 \le j \le n$, we have $f_j(x_1, x_2, \dots, x_n) = F(j, (x_1, x_2, \dots, x_n))$. If this recurrent neural network is seeded with $\left(x_1^0, \dots, x_n^0\right)$ and $S \in [\![1;n]\!]^{\mathbb{N}}$, it produces exactly the same output vectors than the chaotic iterations of F_f with initial condition $\left(S, (x_1^0, \dots, x_n^0)\right) \in [\![1;n]\!]^{\mathbb{N}} \times \mathbb{B}^n$. Theoretically speaking, such iterations of F_f are thus a formal model of these kind of recurrent neural networks. In the rest of this paper, we will call such multilayer perceptrons "CI-MLP(f)", which stands for "Chaotic Iterations based MultiLayer Perceptron".

Checking if CI-MLP(f) behaves chaotically according to Devaney's definition of chaos is simple: we need just to verify if the associated graph of iterations $\Gamma(f)$ is strongly connected or not. As an incidental consequence, we finally obtain an equivalence between chaotic iterations and CI-MLP(f). Therefore, we can obviously study such multilayer perceptrons with mathematical tools like topology to establish, for example, their convergence or, contrarily, their unpredictable behavior. An example of such a study is given in the next section.

5 Topological properties of chaotic neural networks

Let us first recall two fundamental definitions from the mathematical theory of chaos.

Definition 1 A function f is said to be **expansive** if $\exists \varepsilon > 0$, $\forall x \neq y$, $\exists n \in \mathbb{N}$ such that $d\left(f^n(x), f^n(y)\right) \geq \varepsilon$.

In other words, a small error on any initial condition is always amplified until ε, which denotes the constant of expansivity of f.

Definition 2 A discrete dynamical system is said to be **topologically mixing** if and only if, for any pair of disjoint open sets U,$V \neq \emptyset$, we can find some $n_0 \in \mathbb{N}$ such that for any n, $n \geq n_0$, we have $f^n(U) \cap V \neq \emptyset$.

Topologically mixing means that the dynamical system evolves in time such that any given region of its topological space might overlap with any other region.

It has been proven in Ref. [**?**] that chaotic iterations are expansive and topologically mixing when f is the vectorial negation f_0. Consequently, these properties are inherited by the CI-MLP(f_0) recurrent neural network previously presented, which induce a greater unpredictability. Any difference on the initial value of the input layer is in particular magnified up to be equal to the expansivity constant.

Let us then focus on the consequences for a neural network to be chaotic according to Devaney's definition. Intuitively, the topological transitivity property implies indecomposability, which is formally defined as follows:

Definition 3 A dynamical system $(\mathcal{X}, f)$ is **not decomposable** if it is not the union of two closed sets $A, B \subset \mathcal{X}$ such that $f(A) \subset A, f(B) \subset B$.

Hence, reducing the set of outputs generated by CI-MLP(f), in order to simplify its complexity, is impossible if $\Gamma(f)$ is strongly connected. Moreover, under this hypothesis CI-MLPs(f) are strongly transitive:

Definition 4 A dynamical system $(\mathcal{X}, f)$ is **strongly transitive** if $\forall x, y \in \mathcal{X}$, $\forall r > 0$, $\exists z \in \mathcal{X}$, $d(z, x) \ \leq \ r \Rightarrow \exists n \in \mathbb{N}^*$, $f^n(z) = y$.

According to this definition, for all pairs of points (x, y) in the phase space, a point z can be found in the neighborhood of x such that one of its iterates $f^n(z)$ is y. Indeed, this result has been established during the proof of the transitivity presented in Ref. [**?**]. Among other things, the strong transitivity leads to the fact that without the knowledge of the initial input layer, all outputs are possible. Additionally, no point of the output space can be discarded when studying CI-MLPs: this space is intrinsically complicated and it cannot be decomposed or simplified.

Furthermore, these recurrent neural networks exhibit the instability property:

Definition 5 A dynamical system $(\mathcal{X}, f)$ is **unstable** if for all $x \in \mathcal{X}$, the orbit $\gamma_x : n \in \mathbb{N} \longmapsto f^n(x)$ is unstable, that means: $\exists \varepsilon > 0$, $\forall \delta > 0$, $\exists y \in \mathcal{X}$, $\exists n \in \mathbb{N}$, such that $d(x, y) < \delta$ and $d\left(\gamma_x(n), \gamma_y(n)\right) \geq \varepsilon$.

This property, which is implied by the sensitive point dependence on initial conditions, leads to the fact that in all neighborhoods of any point x, there are points that can be apart by ε in the future through iterations of the CI-MLP(f). Thus, we can claim that the behavior of these MLPs is unstable when $\Gamma(f)$ is strongly connected.

Let us now consider a compact metric space (M, d) and $f : M \to M$ a continuous map. For each natural number n, a new metric d_n is defined on M by

$$d_n(x, y) = \max\{d(f^i(x), f^i(y)) : 0 \leq i < n\} \ . \tag{10}$$

Given any $\varepsilon > 0$ and $n \geqslant 1$, two points of M are ε-close with respect to this metric if their first n iterates are ε-close.

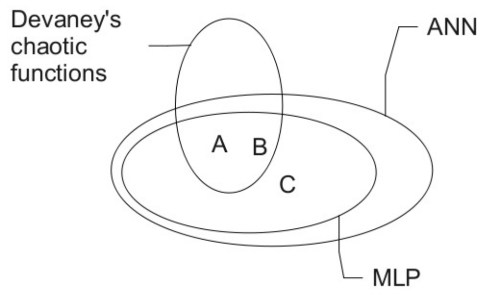

Figure 2: Summary of addressed neural networks and chaos problems

This metric allows one to distinguish in a neighborhood of an orbit the points that move away from each other during the iteration from the points that travel together. A subset E of M is said to be (n, ε)-separated if each pair of distinct points of E is at least ε apart in the metric d_n. Denote by $H(n, \varepsilon)$ the maximum cardinality of an (n, ε)-separated set,

Definition 6 The **topological entropy** of the map f is defined by (see e.g., Ref. [**?**] or Ref. [**?**])

$$h(f) = \lim_{\varepsilon \to 0} \left(\limsup_{n \to \infty} \frac{1}{n} \log H(n, \varepsilon) \right) .$$

Then we have the following result [**?**],

Theorem 2 *$(\mathcal{X}, d)$ is compact and the topological entropy of $(\mathcal{X}, G_{f_0})$ is infinite.*

Figure 2 is a summary of addressed neural networks and chaos problems. In Section 3 we have explained how to construct a truly chaotic neural networks, A for instance. Section 4 has shown how to check whether a given MLP A or C is chaotic or not in the sense of Devaney, and how to study its topological behavior. The last thing to investigate, when comparing neural networks and Devaney's chaos, is to determine whether an artificial neural network C is able to learn or predict some chaotic behaviors of B, as it is defined in the Devaney's formulation (when they are not specifically constructed for this purpose). This statement is studied in the next section.

6 Suitability of Feedforward Neural Networks for Predicting Chaotic and Non-chaotic Behaviors

In the context of computer science different topic areas have an interest in chaos, as for steganographic techniques [**?**, **?**]. Steganography consists in embedding a secret message within an ordinary one, while the secret extraction takes place once at destination. The reverse (*i.e.*, automatically detecting the presence of hidden messages inside media) is called steganalysis. Among the deployed strategies inside detectors, there are support vectors machines [**?**], neural networks [**?**, **?**], and Markov chains [**?**]. Most of these detectors give quite good results and are rather competitive when facing steganographic tools. However, to the best of our knowledge none of the considered information hiding schemes fulfills the Devaney definition of chaos [**?**]. Indeed, one can wonder whether detectors continue to give good results when facing truly chaotic schemes. More generally, there remains the open problem of deciding whether artificial intelligence is suitable for predicting topological chaotic behaviors.

6.1 Representing Chaotic Iterations for Neural Networks

The problem of deciding whether classical feedforward ANNs are suitable to approximate topological chaotic iterations may then be reduced to evaluate such neural networks on iterations of functions with Strongly Connected Component (SCC) graph of iterations. To compare with non-chaotic iterations, the experiments detailed in the following sections are carried out using both kinds of function (chaotic and non-chaotic). Let us emphasize on the difference between this kind of neural networks and the Chaotic Iterations based multilayer peceptron.

We are then left to compute two disjoint function sets that contain either functions with topological chaos properties or not, depending on the strong connectivity of their iterations graph. This can be achieved for instance by removing a set of edges from the iteration graph $\Gamma(f_0)$ of the vectorial negation function f_0. One can deduce whether a function verifies the topological chaos property or not by checking the strong connectivity of the resulting graph of iterations.

For instance let us consider the functions f and g from $\mathbb{B}^4$ to $\mathbb{B}^4$ respectively defined by the following lists:

$$[0, 0, 2, 3, 13, 13, 6, 3, 8, 9, 10, 11, 8, 13, 14, 15]$$

$$\text{and } [11, 14, 13, 14, 11, 10, 1, 8, 7, 6, 5, 4, 3, 2, 1, 0] \enspace .$$

In other words, the image of 0011 by g is 1110: it is obtained as the binary value of the fourth element in the second list (namely 14). It is not hard to verify that $\Gamma(f)$ is not SCC (*e.g.*, $f(1111)$ is 1111) whereas $\Gamma(g)$ is. The remaining of this section shows how to translate iterations of such functions into a model amenable to be learned by an ANN. Formally, input and output vectors are pairs $((S^t)^{t\in\mathbb{N}}, x)$ and $\left(\sigma((S^t)^{t\in\mathbb{N}}), F_f(S^0, x)\right)$ as defined in Eq. (4).

Firstly, let us focus on how to memorize configurations. Two distinct translations are proposed. In the first case, we take one input in $\mathbb{B}$ per component; in the second case, configurations are memorized as natural numbers. A coarse attempt to memorize configuration as natural number could consist in labeling each configuration with its translation into decimal numeral system. However, such a representation induces too many changes between a configuration labeled by a power of two and its direct previous configuration: for instance, 16 (10000) and 15 (01111) are close in a decimal ordering, but their Hamming distance is 5. This is why Gray codes [**?**] have been preferred.

Secondly, let us detail how to deal with strategies. Obviously, it is not possible to translate in a finite way an infinite strategy, even if both $(S^t)^{t\in\mathbb{N}}$ and $\sigma((S^t)^{t\in\mathbb{N}})$ belong to $\{1,\ldots,n\}^{\mathbb{N}}$. Input strategies are then reduced to have a length of size $l \in [\![2,k]\!]$, where k is a parameter of the evaluation. Notice that l is greater than or equal to 2 since we do not want the shift σ function to return an empty strategy. Strategies are memorized as natural numbers expressed in base $n+1$. At each iteration, either none or one component is modified (among the n components) leading to a radix with $n+1$ entries. Finally, we give an other input, namely $m \in [\![1,l-1]\!]$, which is the number of successive iterations that are applied starting from x. Outputs are translated with the same rules.

To address the complexity issue of the problem, let us compute the size of the data set an ANN has to deal with. Each input vector of an input-output pair is composed of a configuration x, an excerpt S of the strategy to iterate of size $l \in [\![2,k]\!]$, and a number $m \in [\![1,l-1]\!]$ of iterations that are executed.

Firstly, there are 2^n configurations x, with n^l strategies of size l for each of them. Secondly, for a given configuration there are $\omega = 1\times n^2 + 2\times n^3 + \ldots + (k-1)\times n^k$ ways of writing the pair (m,S). Furthermore, it is not hard to establish that

$$(n-1)\times\omega = (k-1)\times n^{k+1} - \sum_{i=2}^{k} n^i$$

then

$$\omega = \frac{(k-1)\times n^{k+1}}{n-1} - \frac{n^{k+1}-n^2}{(n-1)^2} \ .$$

And then, finally, the number of input-output pairs for our ANNs is

$$2^n \times \left(\frac{(k-1)\times n^{k+1}}{n-1} - \frac{n^{k+1}-n^2}{(n-1)^2}\right) \ .$$

For instance, for 4 binary components and a strategy of at most 3 terms we obtain 2304 input-output pairs.

6.2 Experiments

To study if chaotic iterations can be predicted, we choose to train the multilayer perceptron. As stated before, this kind of network is in particular well-known for its universal approximation property [**?**, **?**]. Furthermore, MLPs have been already considered for chaotic time series prediction. For example, in [**?**] the authors have shown that a feedforward MLP with two hidden layers, and trained with Bayesian Regulation back-propagation, can learn successfully the dynamics of Chua's circuit.

In these experiments we consider MLPs having one hidden layer of sigmoidal neurons and output neurons with a linear activation function. They are trained using the Limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-newton algorithm in combination with the Wolfe linear search. The training process is performed until a maximum number of epochs is reached. To prevent overfitting and to estimate the generalization performance we use holdout validation by splitting the data set into learning, validation, and test subsets. These subsets are obtained through random selection such that their respective size represents 65%, 10%, and 25% of the whole data set.

Several neural networks are trained for both iterations coding schemes. In both cases iterations have the following layout: configurations of four components and strategies with at most three terms. Thus, for the first coding scheme a data set pair is composed of 6 inputs and 5 outputs, while for the second one it is respectively 3 inputs and 2 outputs. As noticed at the end of the previous section, this leads to data sets that consist of 2304 pairs. The networks differ in the size of the hidden layer and the maximum number of training epochs. We remember that to evaluate the ability of neural networks to predict a chaotic behavior for each coding scheme, the trainings of two data sets, one of them describing chaotic iterations, are compared.

Thereafter we give, for the different learning setups and data sets, the mean prediction success rate obtained for each output. Such a rate represents the percentage of input-output pairs belonging to the test subset for which the corresponding output value was correctly predicted. These values are computed considering 10 trainings with random subsets construction, weights and biases initialization. Firstly, neural networks having 10 and 25 hidden neurons are trained, with a maximum number of epochs that takes its value in $\{125, 250, 500\}$ (see Tables 1 and 2). Secondly, we refine the second coding scheme by splitting the output vector such that each output is learned by a specific neural network (Table 3). In this last case, we increase the size of the hidden layer up to 40 neurons and we consider larger number of epochs.

Table 1 presents the rates obtained for the first coding scheme. For the chaotic data, it can be seen that as expected configuration prediction becomes better when the number of hidden neurons and maximum epochs increases: an improvement by a factor two is observed (from 36.10% for 10 neurons and 125 epochs to 70.97% for 25 neurons and 500 epochs). We also notice that the learning of outputs (2) and (3) is more difficult. Conversely, for the non-chaotic case the simplest training setup is enough to predict configurations. For all these feedforward network topologies and all outputs the obtained results for the non-chaotic case outperform the chaotic ones. Finally, the rates for the strategies show that the different feedforward networks are unable to learn them.

For the second coding scheme (*i.e.*, with Gray Codes) Table 2 shows that any network learns about five times more non-chaotic configurations than chaotic ones. As in the previous scheme, the strategies cannot be predicted. Figures 3 and 4 present the predictions given by two feedforward multilayer perceptrons that were respectively trained to learn chaotic and non-chaotic data, using the second coding scheme. Each figure shows for each sample of the test subset (577 samples, representing 25% of the 2304 samples) the configuration that should have been predicted and the one given by the multilayer perceptron.

Table 1: Prediction success rates for configurations expressed as boolean vectors.

Networks topology: 6 inputs, 5 outputs, and one hidden layer				
Hidden neurons		10 neurons		
Epochs		125	250	500
Chaotic	Output (1)	90.92%	91.75%	91.82%
	Output (2)	69.32%	78.46%	82.15%
	Output (3)	68.47%	78.49%	82.22%
	Output (4)	91.53%	92.37%	93.4%
	Config.	36.10%	51.35%	56.85%
	Strategy (5)	1.91%	3.38%	2.43%
Non-chaotic	Output (1)	97.64%	98.10%	98.20%
	Output (2)	95.15%	95.39%	95.46%
	Output (3)	100%	100%	100%
	Output (4)	97.47%	97.90%	97.99%
	Config.	90.52%	91.59%	91.73%
	Strategy (5)	3.41%	3.40%	3.47%
Hidden neurons		25 neurons		
Epochs		125	250	500
Chaotic	Output (1)	91.65%	92.69%	93.93%
	Output (2)	72.06%	88.46%	90.5%
	Output (3)	79.19%	89.83%	91.59%
	Output (4)	91.61%	92.34%	93.47%
	Config.	48.82%	67.80%	70.97%
	Strategy (5)	2.62%	3.43%	3.78%
Non-chaotic	Output (1)	97.87%	97.99%	98.03%
	Output (2)	95.46%	95.84%	96.75%
	Output (3)	100%	100%	100%
	Output (4)	97.77%	97.82%	98.06%
	Config.	91.36%	91.99%	93.03%
	Strategy (5)	3.37%	3.44%	3.29%

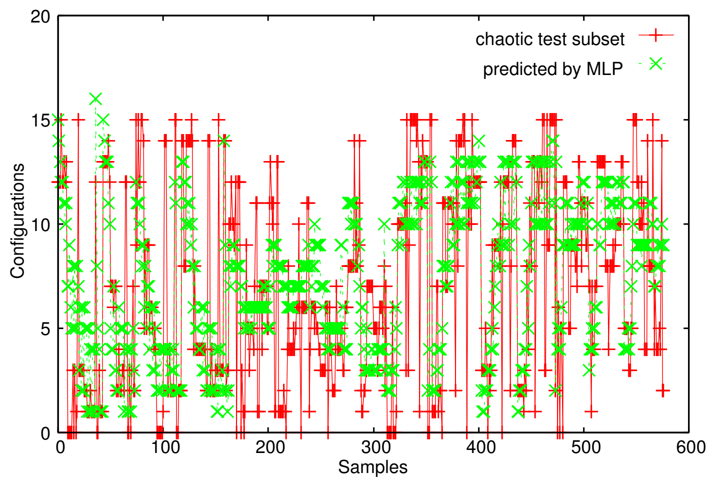

Figure 3: Second coding scheme - Predictions obtained for a chaotic test subset.

It can be seen that for the chaotic data the predictions are far away from the expected configurations. Obviously, the better predictions for the non-chaotic data reflect their regularity.

Let us now compare the two coding schemes. Firstly, the second scheme disturbs the learning process. In fact in this scheme the configuration is always expressed as a natural number, whereas in the first one the number of inputs follows the increase of the Boolean vectors coding configurations. In this latter case, the coding gives a finer information on configuration evolution.

Unfortunately, in practical applications the number of components is usually unknown. Hence, the first coding scheme cannot be used systematically. Therefore, we provide a refinement of the second scheme: each output is learned by a different ANN. Table 3 presents the results for this approach. In any case, whatever the considered feedforward network topologies, the maximum epoch number, and the kind of iterations, the configuration success rate is slightly improved. Moreover, the strategies predictions rates reach almost 12%, whereas in Table 2 they never exceed 1.5%. Despite of this improvement, a long term prediction of chaotic iterations still appear to be an open issue.

7 Conclusion

In this paper, we have established an equivalence between chaotic iterations, according to the Devaney's definition of chaos, and a class of multilayer perceptron neural networks. Firstly, we have described how to build a neural network that can be trained to learn a given chaotic map function. Secondly, we found a condition that allow to check whether the iterations induced by a function are chaotic or not, and thus if a chaotic map is obtained. Thanks to this condition

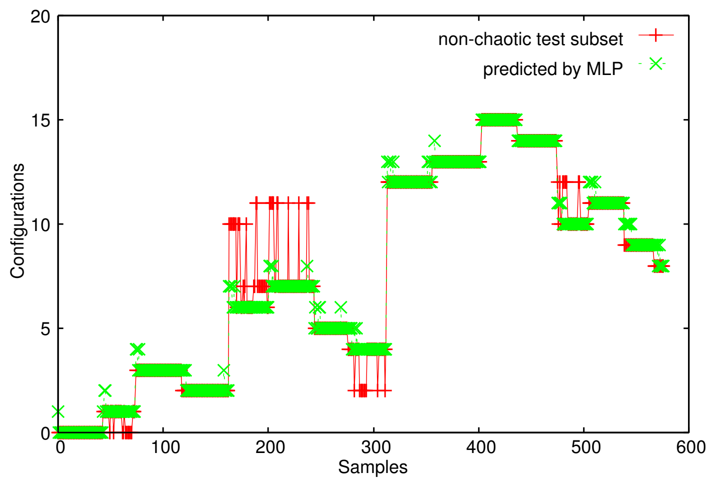

Figure 4: Second coding scheme - Predictions obtained for a non-chaotic test subset.

our approach is not limited to a particular function. In the dual case, we show that checking if a neural network is chaotic consists in verifying a property on an associated graph, called the graph of iterations. These results are valid for recurrent neural networks with a particular architecture. However, we believe that a similar work can be done for other neural network architectures. Finally, we have discovered at least one family of problems with a reasonable size, such that artificial neural networks should not be applied in the presence of chaos, due to their inability to learn chaotic behaviors in this context. Such a consideration is not reduced to a theoretical detail: this family of discrete iterations is concretely implemented in a new steganographic method [**?**]. As steganographic detectors embed tools like neural networks to distinguish between original and stego contents, our studies tend to prove that such detectors might be unable to tackle with chaos-based information hiding schemes.

In future work we intend to enlarge the comparison between the learning of truly chaotic and non-chaotic behaviors. Other computational intelligence tools such as support vector machines will be investigated too, to discover which tools are the most relevant when facing a truly chaotic phenomenon. A comparison between learning rate success and prediction quality will be realized. Concrete consequences in biology, physics, and computer science security fields will then be stated.

Table 2: Prediction success rates for configurations expressed with Gray code

Networks topology: 3 inputs, 2 outputs, and one hidden layer				
	Hidden neurons	10 neurons		
	Epochs	125	250	500
Chaotic	Config. (1)	13.29%	13.55%	13.08%
	Strategy (2)	0.50%	0.52%	1.32%
Non-Chaotic	Config. (1)	77.12%	74.00%	72.60%
	Strategy (2)	0.42%	0.80%	1.16%
	Hidden neurons	25 neurons		
	Epochs	125	250	500
Chaotic	Config. (1)	12.27%	13.15%	13.05%
	Strategy (2)	0.71%	0.66%	0.88%
Non-Chaotic	Config. (1)	73.60%	74.70%	75.89%
	Strategy (2)	0.64%	0.97%	1.23%

Table 3: Prediction success rates for split outputs.

Networks topology: 3 inputs, 1 output, and one hidden layer			
Epochs	125	250	500
Chaotic	Output = Configuration		
10 neurons	12.39%	14.06%	14.32%
25 neurons	13.00%	14.28%	14.58%
40 neurons	11.58%	13.47%	14.23%
Non chaotic	Output = Configuration		
10 neurons	76.01%	74.04%	78.16%
25 neurons	76.60%	72.13%	75.96%
40 neurons	76.34%	75.63%	77.50%
Chaotic/non chaotic	Output = Strategy		
10 neurons	0.76%	0.97%	1.21%
25 neurons	1.09%	0.73%	1.79%
40 neurons	0.90%	1.02%	2.15%

Epochs	1000	2500	5000
Chaotic	Output = Configuration		
10 neurons	14.51%	15.22%	15.22%
25 neurons	16.95%	17.57%	18.46%
40 neurons	17.73%	20.75%	22.62%
Non chaotic	Output = Configuration		
10 neurons	78.98%	80.02%	79.97%
25 neurons	79.19%	81.59%	81.53%
40 neurons	79.64%	81.37%	81.37%
Chaotic/non chaotic	Output = Strategy		
10 neurons	3.47%	9.98%	11.66%
25 neurons	3.92%	8.63%	10.09%
40 neurons	3.29%	7.19%	7.18%