ELSEVIER

Discrete Applied Mathematics 86 (1998) 63–79

DISCRETE APPLIED MATHEMATICS

# Vapnik–Chervonenkis dimension of recurrent neural networks

Pascal Koiran[a,*,1], Eduardo D. Sontag[b,2]

[a] *Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon – CNRS, 46 allée d'Italie, 69364 Lyon Cedex 07, France*

[b] *Department of Mathematics, Rutgers University, New Brunswick, NJ 08903, USA*

Received 19 December 1996; received in revised form 15 July 1997; accepted 23 October 1997

**Abstract**

Most of the work on the Vapnik–Chervonenkis dimension of neural networks has been focused on feedforward networks. However, recurrent networks are also widely used in learning applications, in particular when time is a relevant parameter. This paper provides lower and upper bounds for the VC dimension of such networks. Several types of activation functions are discussed, including threshold, polynomial, piecewise-polynomial and sigmoidal functions. The bounds depend on two independent parameters: the number $w$ of weights in the network, and the length $k$ of the input sequence. In contrast, for feedforward networks, VC dimension bounds can be expressed as a function of $w$ only. An important difference between recurrent and feedforward nets is that a fixed recurrent net can receive inputs of arbitrary length. Therefore we are particularly interested in the case $k \gg w$. Ignoring multiplicative constants, the main results say roughly the following:

- For architectures with activation $\sigma =$ any fixed nonlinear polynomial, the VC dimension is $\approx wk$.
- For architectures with activation $\sigma =$ any fixed *piecewise* polynomial, the VC dimension is between $wk$ and $w^2k$.
- For architectures with activation $\sigma = \mathscr{H}$ (threshold nets), the VC dimension is between $w\log(k/w)$ and $\min\{wk\log wk, w^2 + w\log wk\}$.
- For the standard sigmoid $\sigma(x) = 1/(1+e^{-x})$, the VC dimension is between $wk$ and $w^4k^2$.

An earlier version of this paper has appeared in *Proc. 3rd European Workshop on Computational Learning Theory*, Lecture Notes in Computer Science vol. 1208, Springer, Berlin, 1997, pp. 223–237. © 1998 Elsevier Science B.V. All rights reserved.

* Corresponding author.

[1] This research was carried out in part while visiting DIMACS and the Rutgers Center for Systems and Control (SYCON) at Rutgers University.

[2] This research was supported in part by US Air Force Grant AFOSR-94-0293.

## 1. Introduction

In this paper we deal with questions that arise when training and testing data which have a time series structure. This is fairly a common situation in many applications. It arises in control problems, when the inputs to a regulator are time-dependent measurements of plant states, or in speech processing, where inputs are windowed Fourier coefficients and signal levels at each instant.

In PAC-theoretic terms, it is natural to take into account this additional structure through the use of hypotheses classes $\mathscr{F}$ which consist of dynamical systems. We will take the inputs (for training and testing) to be functions of time, and the hypotheses classes will be defined by means of dynamical recognizers, which allow one to exploit the information inherent in the correlations and dependencies that exist among the terms of the input sequence. (As a close analogy, Kalman filtering, which relies on linear dynamical systems for extracting information – filtering of noise – from a stream of data, is perhaps the most successful known example of an application of the idea of using dynamical systems as data processors.) Through a limitation of the memory and power (dynamic order, number of adjustable parameters) of the elements of $\mathscr{F}$, and analyzing behavior for longer and longer input sequences, one is able to focus on the properties that truly reflect the dependence of $f(u)$ on long-term time correlations in the input sequence $u$. This paradigm is inspired by the use of finite automata for the recognition of languages, and the use of recursive least squares techniques in statistical problems, but the interest here is in nonlinear, continuous-state, dynamical systems. In particular, we use recurrent (sometimes "feedback" or "dynamic") neural networks. In contrast to feedforward nets, which only contain static units, recurrent nets incorporate dynamic elements (delay lines), and their behavior is described by means of systems of difference equations. (It is also possible to study *continuous-time* nets, which are defined in terms of differential equations, but we restrict attention in this paper to the discrete time case. Similar results can be obtained in the continuous-time framework, however.)

Recurrent networks are among the models considered by Grossberg (see e.g. [8]) and his school during the last 20 or more years, and include the networks proposed by Hopfield (see e.g. [9]) for associative memory and optimization. Among other applications, they have been employed in the design of control laws, speech recognition and speaker identification, formal language inference, and sequence extrapolation for time series prediction. For references, see for instance the book [2], which emphasizes digital signal processing, [6] for formal language learning, and [12] for control. In addition, theoretical results about neural networks established their universality as models for systems approximation [16] as well as analog computing devices [13, 14].

As a simple example, consider the network in Fig. 1. The behavior of this net is described by the following system of difference equations:

$$x_1(t+1) = \sigma_1(2x_1(t) + x_2(t) - u_1(t) + u_2(t)), \tag{1a}$$

$$x_2(t+1) = \sigma_2(-x_2(t) + 3u_2(t)), \tag{1b}$$

$$y(t) = x_1(t), \tag{1c}$$

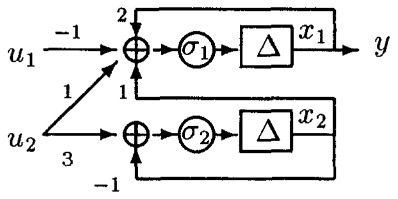

Fig. 1. Example of recurrent net ($\Delta$ indicates a unit delay).

where $\sigma_1$ and $\sigma_2$ are two scalar nonlinear functions $\mathbb{R} \to \mathbb{R}$ (called the "activation functions" of the net), the input to the system is given by the vector $u(t) = (u_1(t), u_2(t))$, and the output at time $t$ is the current value of $x_1$. (We usually write "$x^+(t)$" instead of "$x(t+1)$" and often omit arguments "$t$".)

One way to associate an input/output behavior to a dynamic net, given a specified set of initial conditions, is by looking at the last output that results after a finite-length input has been presented. For instance, in the above example, suppose that we fixed the initial state as $x_1(0) = x_2(0) = 0$. Then, given any sequence of inputs $u(t) = (u_1(t), u_2(t))$ defined for $t = 0, \ldots, T-1$, we may solve iteratively the system of difference equations (1) with $u$ substituted on the right-hand side. In this way we obtain a state trajectory $(x_1(t), x_2(t))$. We may then read-out the output $y(T) = x_1(T)$ and interpret it as the output of the network in response to the forcing function $u$.

## 2. Precise definitions

In order to be able to formally state our main results, we now provide precise definitions of recurrent nets. However, just as one does with Turing machines – or similar models of computation – in the design and analysis of algorithms we revert to a more informal approach, leaving implicit the precise specification of networks in the formalism introduced here.

By an $n$-dimensional, $m$-input, $p$-output initialized *recurrent net* we mean a 5-tuple

$$\Sigma = (A, B, C, x^0, \sigma) \tag{2}$$

consisting of three matrices $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{p \times n}$, a vector $x^0 \in \mathbb{R}^n$, and a diagonal mapping

$$\sigma : \mathbb{R}^n \to \mathbb{R}^n : \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \mapsto \begin{pmatrix} \sigma_1(x_1) \\ \vdots \\ \sigma_n(x_n) \end{pmatrix}, \tag{3}$$

where $\sigma_1, \ldots, \sigma_n$ are maps $\mathbb{R} \to \mathbb{R}$. The (discrete time) *system induced by the net* (2) is the set of $n$ coupled difference equations, plus measurement function:

$$x(t+1) = \sigma(Ax(t) + Bu(t)), \quad x(0) = x^0, \quad y(t) = Cx(t). \tag{4}$$

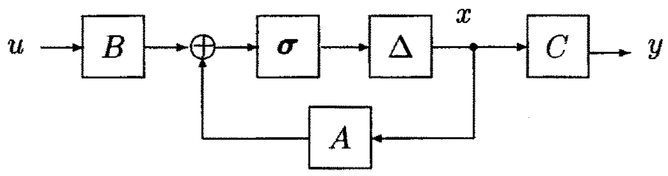

Fig. 2. Recurrent net ($\Delta$ indicates a unit delay).

One also writes (4) simply as $x^+ = \boldsymbol{\sigma}(Ax + Bu)$, $x(0) = x^0$, $y = Cx$. The component maps $\sigma_1, \ldots, \sigma_n$ of $\boldsymbol{\sigma}$ are the *activations of the net.* If it is the case that all the $\sigma_i$ are equal to a fixed function $\sigma$, we say that the net is *homogeneous* with activation $\sigma$ and write also $\vec{\sigma}^{(n)}$ instead of $\boldsymbol{\sigma}$. The spaces $\mathbb{R}^m$, $\mathbb{R}^n$, and $\mathbb{R}^p$ are called respectively the input, state, and output spaces of the net.

As an illustration, take again the diagram in Fig. 1. This is a pictorial representation of a two-dimensional, 2-input, single-output recurrent net defined by

$$A = \begin{pmatrix} 2 & 1 \\ 0 & -1 \end{pmatrix}, \quad B = \begin{pmatrix} -1 & 1 \\ 0 & 3 \end{pmatrix}, \quad C = (1 \quad 0), \quad \boldsymbol{\sigma} = \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix},$$

where the inital state $x^0$ was left unspecified. It induces the system (1).

In the present context, one interprets the vector equations for $x$ in (4) as representing the evolution of an ensemble of $n$ "neurons" (also called sometimes "units", or "gates") where each coordinate $x_i$ of $x$ is a real-valued variable which represents the internal state of the $i$th neuron, and each coordinate $u_i$, $i = 1, \ldots, m$ of $u$ is an external input signal. The vector $x^0$ lists the initial values of these states. The coefficients $A_{ij}, B_{ij}$ denote the weights, intensities, or "synaptic strengths," of the various connections. The coordinates of $y(t)$ represent the output of $p$ probes, or measurement devices, each of which averages the activation values of many neurons. Often $C$ is just a projection onto some coordinates, that is, the components of $y$ are simply a subset of the components of $x$.

The linear systems customarily studied in control theory (see e.g. the textbook [15]) are precisely the homogeneous recurrent nets with identity activation and $x^0 = 0$.

Fig. 2 gives a "block diagram" of (4).

To each initialized recurrent net $(A, B, C, x^0, \boldsymbol{\sigma})$ we associate a discrete time input/output behavior. Assume given a sequence $u = u(0), \ldots, u(k-1)$ of elements of the input space $\mathbb{R}^m$. One may iteratively solve the difference equation (4) starting with $x(0) = x^0$, thereby obtaining a sequence of state vectors $x(1), \ldots, x(k)$. In this manner, each initialized recurrent net induces a mapping, on inputs of fixed length $k$,

$$\lambda_\Sigma^k : (\mathbb{R}^m)^k \to \mathbb{R}^p : u \mapsto y(k) = Cx(k) \tag{5}$$

which assigns to the input $u$ the last output produced in response.

**Remark 1.** One may broaden the notion of recurrent net by allowing "biases" or "offsets", i.e. nonzero vectors $d \in \mathbb{R}^n$ and $e \in \mathbb{R}^p$ in the update and the measurement equations respectively. These equations would then take the more general form $x^+ = \boldsymbol{\sigma}(Ax+Bu+d)$, $y = Cx+e$. Despite the fact that biases are useful, and we employ them in proofs, we do not need to include such an extension in the formal definition. This is because the input/output behavior of any such net also arises as the input/output behavior of a net in the sense defined earlier (zero biases), with state space $\mathbb{R}^{n+1}$ and same activations. The simulation is achieved by means of the introduction of an additional variable $z$ whose value is constantly equal to a nonzero number $z_0$ in the range of one of the activations, say $\sigma$, in such a manner that the equations become $x^+ = \boldsymbol{\sigma}(Ax+zd'+Bu)$, $z^+ = \sigma(a_0 z)$, $y = Cx+ze'$, where $a_0$ is chosen so that $\sigma(a_0 z_0) = z_0$ and $d', e'$ are so that $z_0 d' = d$ and $z_0 e' = e$ (if the only activation is $\sigma \equiv 0$, there would be nothing to prove).

*In the sequel, we will always take the number of output components of our networks to be one.* This makes it possible to see their input/output behavior as classifiers. Moreover, $C$ will always be the vector $C = (1\ 0 \cdots 0)$. In other words, there is a designated output unit among the $n$ state units, and we assume without loss of generality that it is $x_1$. The network's output is taken to be the state of this unit at the end of a computation. If one is interested in averaging the state of several units, this effect can often be obtained with the above convention by adding an additional unit to the network, which will also serve as output unit.

## 2.1. Architectures

Roughly, by an “architecture” one means a choice of interconnection structure and of the activation functions $\sigma$ for each neuron, leaving weights and initial states unspecified, as parameters. One may also stipulate that the initial state, or just certain specific coordinates of it, should be zero (as with linear systems in control theory). Feedback networks with a fixed architecture provide parametric classes of dynamical systems. We formalize the notion of architecture by means of incidence matrices, employing binary matrices in order to specify the allowed interconnection patterns and initial states.

By an $n$-dimensional $m$-input *recurrent architecture* we mean a quadruple

$$\mathscr{A} = (\alpha, \beta, \xi, \boldsymbol{\sigma}) \tag{6}$$

consisting of two matrices $\alpha \in \{0,1\}^{n\times n}$ and $\beta \in \{0,1\}^{n\times m}$, a vector $\xi \in \{0,1\}^n$, and a diagonal mapping $\boldsymbol{\sigma}$ as in Eq. (3). An *initialized recurrent net with architecture* $\mathscr{A}$ is an instantiation obtained by choosing values for the nonzero entries, that is, any initialized recurrent net $(A, B, C, x^0, \boldsymbol{\sigma}')$ such that $\boldsymbol{\sigma} = \boldsymbol{\sigma}'$, $C = (1\ 0 \cdots 0)$ and the entries of the matrices and vector satisfy $A_{ij} = 0$ whenever $\alpha_{ij} = 0$, $B_{ij} = 0$ whenever $\beta_{ij} = 0$, and $x_i^0 = 0$ whenever $\xi_i = 0$.

We say also here that the component maps $\sigma_1, \ldots, \sigma_n$ of $\boldsymbol{\sigma}$ are the activations of the net, which is homogeneous with activation $\sigma$ if all $\sigma_i$ are equal to a fixed function $\sigma$. The spaces $\mathbb{R}^m$ and $\mathbb{R}^n$ are, respectively, the input and state spaces of the architecture.

Suppose that the binary matrices $\alpha$, $\beta$ and the vector $\xi$ have exactly $\kappa, \lambda$ and $\nu$ nonzero entries, respectively; then we call the number $w := \kappa + \lambda + \nu$ the number of *parameters* or *weights* of $\mathscr{A}$, and call $\mathbb{R}^w$ the parameter or weight space. Arrange the indices of the nonzero entries in any fixed manner, for instance by listing their nonzero entries row by row, for $\alpha, \beta$ and $\xi$ in that order. These indices are in one-to-one correspondence with the coordinates of vectors in $\mathbb{R}^w$. In this manner, one may view the architecture $\mathscr{A}$ as representing a parameterized system $x^+ = \boldsymbol{\sigma}(\alpha x + \beta u)$, $x(0) = \xi$, $y = x_1$ where, by substituting the parameters $\rho \in \mathbb{R}^w$ into the nonzero entries of $(\alpha, \beta, \xi)$, every possible initialized recurrent net $\Sigma = \mathscr{A}(\rho)$ with architecture $\mathscr{A}$ results.

For example, if arbitrary initial states are allowed, the diagram in the right part of Fig. 1 is a recurrent net with architecture $\mathscr{A}$, where

$$\alpha = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \beta = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \xi = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \boldsymbol{\sigma} = \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix},$$

and the corresponding parameterized system (with parameter space $\mathbb{R}^8$) is

$$x_1(t+1) = \sigma_1(\rho_1 x_1(t) + \rho_2 x_2(t) + \rho_4 u_1(t) + \rho_5 u_2(t)), \tag{7a}$$

$$x_2(t+1) = \sigma_2(\rho_3 x_2(t) + \rho_6 u_2(t)), \tag{7b}$$

$$x_1(0) = x_1^0, \tag{7c}$$

$$x_2(0) = x_2^0, \tag{7d}$$

$$y(t) = x_1(t). \tag{7e}$$

Recalling the notations in Eq. (5), for each recurrent architecture $\mathscr{A}$ and each $k > 0$, we may introduce the set

$$\mathscr{F}_{\mathscr{A},k} := \{\lambda_\Sigma^k, \Sigma = \mathscr{A}(\rho), \rho \in \mathbb{R}\} \tag{8}$$

of mappings $(\mathbb{R}^m)^k \to \mathbb{R}$. Elements of this set are the input/output mappings induced on inputs of length $k$ by each possible initialized recurrent net with architecture $\mathscr{A}$.

## 2.2. VC dimension

*In all our results, we will take the number of input components (m) to be one, and, except in Theorem 7, we consider only homogeneous (all activations equal) architectures.* By $\sigma$-architecture, we mean an architecture where all activations are the same function $\sigma : \mathbb{R} \to \mathbb{R}$. The choice of $m = 1$ makes our lower bounds more interesting. It is fairly easy, though notationally somewhat more cumbersome, to extend the upper bounds to vector inputs. The same can be said about the homogeneity assumption, although it is the case that some *proofs* use nonhomogeneous nets in intermediate steps.

Given any $\mathscr{A}$ with $m=1$, and any $k>0$, we denote

$$\mathscr{B}_{\mathscr{A},k} := \{H \circ \lambda_{\Sigma}^{k}, \Sigma = \mathscr{A}(\rho), \rho \in \mathbb{R}^{w}, \xi \in \mathbb{R}^{n}\}, \tag{9}$$

where $H$ is the threshold function: $H(x)=0$ if $x \leqslant 0$ and $H(x)=1$ if $x>0$. This is a class of mappings $\mathbb{R}^k \to \{0,1\}$, and we write $\mathrm{VC}(\mathscr{A},k)$ to denote its VC dimension (we recall that the VC dimension of a family of binary-valued functions on a domain $X$ is the cardinality of the largest subset of $X$ that is shattered by $\mathscr{F}$, and that $A \subseteq X$ is said to be shattered if the restriction of $\mathscr{F}$ to $A$ is $\{0,1\}^A$). We refer to this quantity also as the "VC dimension of $\mathscr{A}$ when receiving inputs of length $k$".

We are particularly interested in understanding the behavior of $\mathrm{VC}(\mathscr{A},k)$ as $k \to \infty$, for various recurrent architectures, as well as the dependence of this quantity on the number of weights and the particular type of activation being used. In particular, we continue the work described in [5] (see also [18] for related work), which had obtained estimates of these quantities for architectures with identity activations.

By a *threshold* recurrent architecture we mean a homogeneous one with $\sigma = H$. As in [17], we say that $\sigma : \mathbb{R} \to \mathbb{R}$ is *sigmoidal*, or *a sigmoid*, if:
1. $\sigma$ is differentiable at some point $x_0$ where $\sigma'(x_0) \neq 0$.
2. $\lim_{x \to -\infty} \sigma(x) = 0$ and $\lim_{x \to +\infty} \sigma(x) = 1$. (the limits 0 and 1 can be replaced by any distinct numbers).

In particular, the *standard sigmoid* is $\sigma(x) = 1/(1+\mathrm{e}^{-x})$. By $\sigma$-gate, or $\sigma$-unit, we mean a gate with activation $\sigma$. These two special cases are worth recording: by linear (respectively, threshold) unit we mean a unit with activation a linear or threshold function.

## 2.3. Statements of main results

For each $\mathscr{A}$ and $k$, by "unfolding" the iterations, one may also see the class $\mathscr{B}_{\mathscr{A},k}$ as a class of classifiers representable by feedforward neural nets (with $k$ "hidden layers"). This trivial fact allows one to easily obtain estimates, based on those bounds which were developed (cf. [1, 4, 7, 10]) for the feedforward case.

**Theorem 1.** *For recurrent architectures, with w weights receiving inputs of length k:*
1. *The VC dimension of threshold recurrent architectures is* $\mathrm{O}(kw \log kw)$.
2. *If* $\sigma : \mathbb{R} \to \mathbb{R}$ *is a fixed piecewise-polynomial function, the VC dimension of recurrent architectures with activation* $\sigma$ *is* $\mathrm{O}(kw^2)$.
3. *The VC dimension of recurrent architectures with activation the standard sigmoid is* $\mathrm{O}(k^2w^4)$.

The bounds would seem to be too conservative, since they completely disregard the fact that the weights in the different layers of the "unfolded" net are actually the same. The surprising aspect of the results to be stated next (and of the results in [5]) is that we obtain lower bounds which do not look much different. We first state two more upper bounds. The first one is interesting because for fixed $w$, it shows a $\log k$ dependence, rather than the $k \log k$ obtained by unfolding.

**Theorem 2.** *The VC dimension of an n-dimensional threshold recurrent architecture, with w weights and receiving inputs of length k, is* $O(wn + w \log kw)$.

**Theorem 3.** *Let* $\sigma : \mathbb{R} \to \mathbb{R}$ *be a fixed polynomial function. The VC dimension of recurrent architectures with activation* $\sigma$, *with w weights and receiving inputs of length k, is* $O(kw)$. *Moreover, if* $\sigma$ *is linear this bound can be improved to* $O(w \log k)$.

For a corresponding lower bound in the linear case, see [5]. We now turn to other lower bounds.

**Theorem 4.** *The VC dimension of threshold recurrent architectures, with w weights and receiving inputs of length* $k = \Omega(w)$, *is* $\Omega(w \log(k/w))$.

Here and throughout the paper, the $\Omega$ symbol is to be interpreted as follows: "there exist universal constants $c_1, c_2, c_3 > 0$ such that for every $w \geqslant c_1$ and every $k \geqslant c_2 w$, there exists a threshold recurrent architecture with $w$ weights which has VC dimension at least $c_3 w \log(k/w)$ for inputs of length $k$".

**Theorem 5.** *Let* $\sigma$ *be an arbitrary sigmoid. The VC dimension of recurrent architectures with activation* $\sigma$, *with w weights and receiving inputs of length k, is* $\Omega(wk)$.

It is possible to generalize Theorem 5 to even more arbitrary gate functions:

**Theorem 6.** *Let* $\sigma$ *be a function which is twice continuously differentiable in an open interval containing some point* $x_0$ *where* $\sigma''(x_0) \neq 0$. *The VC dimension of recurrent architectures with activation* $\sigma$, *with w weights and receiving inputs of length k, is* $\Omega(wk)$.

This is an intermediate technical result, but it seems of interest in its own right:

**Theorem 7.** *The VC dimension of recurrent architectures with threshold and linear activations, with w weights and receiving inputs of length k, is* $\Omega(wk)$.

It is interesting to contrast the situation with the one that holds for feedforward nets. For the latter, it holds, in general terms, that linear activations provide VC dimension proportional to $w$, threshold activations give VC dimension proportional to $w \log(w)$, and piecewise polynomial activations result in VC dimension proportional to $w^2$.

Proofs and a few additional results can be found in Sections 3 and 4.

## 3. Threshold networks

### 3.1. Lower bounds

**Lemma 1.** *Given two integers $m, L > 0$ such that $L$ is a power of 2, let $k = mL$ and consider the following family $\mathscr{F}$ of boolean functions on $\{0,1\}^k$: the functions in $\mathscr{F}$ are indexed by $m$ parameters $t_0, \ldots, t_{m-1} \in \{0, \ldots, L-1\}$. The corresponding function maps an input $u = (u(k-1), \ldots, u(0)) \in \{0,1\}^k$ to*

$$f_{t_0,\ldots,t_{m-1}}(u) = \bigvee_{j=0}^{m-1} u(jL + t_j),$$

*i.e., we select one input in each interval of the form $[jL, (j+1)L - 1]$ and take the logical OR of these boolean values.*

*The VC dimension of $\mathscr{F}$ is exactly $m \log L$.*

**Proof.** Since each parameter $t_j$ can take $L$ distinct values, there are at most $L^m = 2^{m \log L}$ functions in $\mathscr{F}$. Hence the VC dimension of $\mathscr{F}$ is at most $m \log L$. In order to show that this upper bound is tight, we will construct a set $S$ of $s = m \log L$ inputs $u_0, \ldots, u_{s-1}$ such that each labeling $(\varepsilon_0, \ldots, \varepsilon_{s-1}) \in \{0,1\}^s$ of $S$ can be obtained in the following way: let $t_j$ be the integer with binary digits (from the low-order bit to the high-order bit) $\varepsilon_{j \log L}, \varepsilon_{j \log L + 1}, \ldots, \varepsilon_{(j+1) \log L - 1}$. Then $f_{t_0,\ldots,t_{m-1}}(u_i) = \varepsilon_i$.

Input $u_i$ is defined as follows: write $i = q \log L + r$, with $q \in \{0, \ldots, m-1\}$ and $r \in \{0, \ldots, \log L - 1\}$. Then $u_i(t) = 0$ for every $t \notin [qL, (q+1)L - 1]$. For $t \in [qL, (q+1)L - 1]$, one can write $t = qL + r'$ where $r' \in \{0, \ldots, L-1\}$. We set $u_i(t) = 1$ if the bit of weight $2^r$ of $r'$ is equal to 1; otherwise, $u_i(t) = 0$.

To see that for any labeling $\varepsilon_0, \ldots, \varepsilon_{s-1}$ one has $f_{t_0,\ldots,t_{m-1}}(u_i) = \varepsilon_i$ (with $t_0, \ldots, t_{m-1}$ as defined above), note that by construction of $u_i$, $u_i(jL + t_j) = 0$ for all $j \neq q$. Hence $f_{t_0,\ldots,t_{m-1}}(u_i) = u_i(qL + t_q)$. Again by construction of $u_i$, this is just the bit of weight $2^r$ of $t_q$. However, by construction of the $t_j$'s, this bit is nothing but $\varepsilon_{q \log L + r} = \varepsilon_i$. Hence, we get the correct output. □

**Remark.** An explicit description of the inputs constructed in the above proof is as follows. Consider the $L$ by $\log L$ matrix

$$V = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

which is obtained by listing all binary row vectors of size $\log L$ in the reverse of their natural order (i.e., the leftmost bit is the least significant bit). Let $U$ be the direct sum matrix $V \oplus V \oplus \cdots \oplus V$ (this is the block-diagonal matrix with $m$ copies of $V$ on the diagonal). The input set $S$ is the same as the set of columns of this matrix. We may also describe the functions in $\mathscr{F}$ in this manner. Let $e_i$, $i=0,\dots,L-1$ be the unit row vector $(0,\dots,0,1,0,\dots,0)$ (with a "1" in the $(i+1)$st position) and consider the row vector $E_{t_0,\dots,t_{m-1}}=(e_{t_0},\dots,e_{t_{m-1}})$. Then, $f_{t_0,\dots,t_{m-1}}(u_i)$ is the product of $E_{t_0,\dots,t_{m-1}}$ and the $i$th column of $U$.

**Proof of Theorem 4.** We first assume that $w$ is of the form $8m+2$, for some $m \geqslant 1$. We also assume that $L=\kappa/m$ is a power of 2, where $\kappa=k-2$, and that $L \geqslant 2$. The (straightforward) generalization to arbitrary values of $w$ and $k$ is explained at the end of the proof.

We shall construct an architecture $\mathcal{N}$ of $w$ weights which implements the family $\mathscr{F}$ of Lemma 1 for inputs of length $\kappa$. It will become clear that in the recurrent network implementation, we need two additional inputs at time $\kappa$ and $\kappa+1$. This explains why $k=\kappa+2$. According to Lemma 1, our shattered set $S'$ will be of size $m\log L=[(w-2)/8]\cdot\log[8(k-2)/(w-2)]$. This is indeed $\Omega(w\log(k/w))$.

$S'$ is defined as follows. For each input $u$ in the shattered set $S$ of Lemma 1 there is an input $u' \in S'$ satisfying $u'(t)=2t+u(t)$ for $t=0,\dots,\kappa-1$. There are two additional "dummy" inputs $u'(\kappa)=0$ and $u'(\kappa+1)=0$ (the values $(0,0)$ can be replaced by an arbitrary pair of real numbers).

Let us now describe network $\mathcal{N}$. We need $m$ subnetworks to perform the tests "$u(jL+t_j)=1$?" for $j=0,\dots,m-1$. By construction of $S'$, this question has a positive answer if and only if there exists $t\in\{0,\dots,\kappa-1\}$ such that $u'(t)=2(jL+t_j)+1$. The outcome $E_j$ of this test can be computed by a simple network of 3 threshold units and 7 weights:

$$E_j=H[H(u'-\theta_j+0.5)+H(\theta_j+0.5-u')-1.5],$$

where $\theta_j=2(jL+t_j)+1$. The network has one additional threshold gate $o$ which serves as output unit. It keeps computing the OR of the $E_j$'s and of the previous output (to make sure that if some $E_j$ is equal to 1 at some time, the output remains 1 ever after). This can be implemented with $m+2$ weights as follows: $o^+=H(o+\sum_{j=1}^{m}E_j-0.5)$. Therefore, $\mathcal{N}$ has $7m+(m+2)=w$ weights.

Note that the last two inputs $u'(\kappa)$ and $u'(\kappa+1)$ are "wasted", i.e., they do not influence the final output $o(k)$. All gates are initialized to 0. This guarantees that the outputs at $t=1$ and $t=2$ are both 0. These outputs are "bogus" in the sense that they occur before even the first input $u'(0)$ is processed. If one of these bogus outputs was equal to 1 then the final output would be 1, no matter what the input sequence is (and we certainly do not want that to happen).

The generalization to arbitrary values of $w$ and $k$ is as follows: let $m=\lfloor(w-2)/8\rfloor$ and $L$ the largest power of 2 which is not larger than $(k-2)/m$ (we can assume that $L\geqslant 2$). The construction above yields a network of $w'=8m+2$ weights with VC dimension $m\log L$. This is still $\Omega(w\log(k/w))$, albeit with a slightly smaller constant. One can obtain a network of exactly $w$ weights by adding to the present construction $w'-w$ "dummy" units which are completely disconnected from the rest of the network (for instance, each dummy unit might be of the form $x^+=H(x)$). $\square$

### 3.2. Upper bounds

**Proof of Theorem 1** (*Threshold case*). By unfolding, the recurrent architecture can be simulated by a (depth $k$) feedforward threshold architecture with $kw$ weights. By this, we mean that any recurrent network obtained as an instantiation of the recurrent architecture can be simulated by a feedforward network obtained as an instantiation of the feedforward architecture (note that the threshold values in the first layer of that feedforward net depend on the weights of the recurrent network as well as on its initial state $x(0) \in \mathbb{R}^n$). The result then follows from the Baum-Haussler bound [1]. □

**Proof of Theorem 2.** Let $S = \{u_1, \ldots, u_s\}$ be a set of $s$ inputs. We will bound the number of distinct transition functions of the architecture for inputs in $S$. The transition function is of the form

$$\phi : (x,u) \mapsto \boldsymbol{H}^{(n)}(Ax + Bu),$$

where the network state $x$ is in $\{0,1\}^n$ and the input $u$ in $\mathbb{R}$. Since we are considering only inputs from $S$, $u$ can take any of the (at most) $ks$ values $u_i(t)$ ($i = 1, \ldots, s$; $t = 0, \ldots, k-1$). Hence the domain $D$ of $\phi$ has at most $|\{0,1\}^n| ks = 2^n ks$ elements. Let $T_i$ be the threshold function computed by gate number $i$. If this gate has $w_i$ incoming weights, then $T_i$ can induce at most $2|D|^{w_i}$ distinct functions on $D$ by, e.g., Sauer's lemma (see for instance [3]). Hence, there are at most $\prod_{i=1}^n 2|D|^{w_i} = 2^n |D|^{w-v}$ distinct transition functions, where $v$ is the number of entries equal to 1 in $\xi$ (in other words, $v$ is the number of "unspecified" coordinates of initial states; by definition, the total number of parameters is $w = \sum_{i=1}^n w_i + v$). If two settings of the architecture's parameters give rise to the same transition function and the initial states are the same, the functions induced on $S$ will be identical. Therefore if $S$ is to be shattered, $2^s \leqslant 2^n (2^n ks)^{w-v} \times 2^v \leqslant 2^n (2^n ks)^w$. This implies that $s \leqslant n(w+1) + w \log k + w \log s$, hence $s/2 \leqslant n(w+1) + w \log k$ or $s/2 \leqslant w \log s$. In both cases, $s = \mathrm{O}(wn + w \log kw)$. □

We do not know if a $\mathrm{O}(w \log kw)$ bound applies for all values of $k, w \geqslant 2$. It is clear from the proof of this theorem that the "extra" term $wn$ comes from the $2^n$ bound on the number of network states. One may be able to give better bounds for networks with a smaller number of "accessible" states.

**Theorem 8.** *The VC dimension of a recurrent architecture of $n$ threshold units and $w$ weights receiving boolean inputs of length $k$ is $\mathrm{O}(wn + w \log w)$. (Note that this bound is independent of $k$.)*

**Proof.** This follows from the proof of Theorem 2. The domain of the transition function $\phi$ has only $2^{n+1}$ elements since $u \in \{0,1\}$. Hence one can set $|D| = 2^{n+1}$ in the proof of that theorem. □

The same result applies to architectures taking their inputs in any fixed finite set. This is in sharp contrast with the case of feedforward architectures, where maximum VC dimension $\Omega(w \log w)$ can be achieved with boolean inputs.

## 4. Sigmoidal networks

### 4.1. Upper bounds

**Proof of Theorem 1** (*Piecewise-polynomial case*). It takes $O(w)$ arithmetic operations to update the network's state after a new input component is received. Hence, the whole computation requires $O(kw)$ operations for inputs of length $k$. The architecture has $w+n \leqslant 2w$ programmable parameters, where $n$ is the number of units in the network. Hence by [7] (Theorem 2.3) its VC dimension is $O(w \times kw)$. $\square$

Interestingly, one can give a better upper bound for polynomial activation functions than for piecewise-polynomial activation functions. The linear case is included in [5].

**Proof of Theorem 3.** We denote by $W$ the vector listing all weights in the two systems matrices $\alpha$ and $\beta$, so that the parameter vector $\rho$ can be partitioned as $(W, \rho_0)$, where $\rho_0$ lists the weights in $\xi$. Let $P : \mathbb{R}^{w-v+1+n} \to \mathbb{R}^n$ be the function mapping $W$, the input $u \in \mathbb{R}$, and the network's current state $x \in \mathbb{R}^n$ to the next state $x^+ \in \mathbb{R}^n$. For instance, the network's state after reading $u(0)$ and $u(1)$ is $P(W, u(1), P(W, u(0), x(0)))$. If $\sigma$ is a degree-$d$ polynomial then each component of $P$ is a polynomial of degree $2d$. (this twofold increase is due to multiplications between weight and input or state variables; the degree in the weight variables is only $d$.) After the whole input $u \in \mathbb{R}^k$ has been read, the state of any unit in the network (and in particular the state of the output unit) can be expressed as a polynomial $P_k$ in $u \in \mathbb{R}^k$, $W \in \mathbb{R}^{w-v}$ and the parameters $\rho_0$ for the nonzero coordinates of $x(0) \in \mathbb{R}^n$. The degree of $P_k$ in the programmable parameters is at most $D_k = 2d^k + \sum_{j=1}^{k-1} d^j$. This follows from $D_1 = 2d$, and $D_{k+1} = d(D_k + 1)$. Here "$D_k + 1$" accounts for multiplication between weight and state variables, and multiplication by $d$ accounts for the application of $\sigma$. By [7] (Theorem 2.2) the VC dimension is bounded by $2w \log(8eD_k)$. (Note that the degree in the input variables does not appear in this bound.) The theorem follows from the obvious observations: $D_k = k+1$ for $d = 1$ and $D_k < 2d^{k+1}$ for $d \geqslant 2$. $\square$

**Proof of Theorem 1** (*Standard-sigmoidal case*). By unfolding, the recurrent architecture can be simulated by a feedforward net with $kn$ nodes, where $n$ is the number of nodes in the original architecture, and the same number $w$ of programmable parameters. By [10] there is a $O((kn)^2 w^2)$ upper bound on the VC dimension of that architecture. This is $O(k^2 w^4)$ as claimed.

*Note*: One can argue that the feedforward architecture has $kw$ weights, but many of those weights are "shared" and there are only $w + n \leqslant 2w$ programmable parameters.

The result in [10] explicitly allows such weight-sharing arrangements (see condition *e* in Section 4.1 of their paper). □

## 4.2. Lower bounds

Theorem 6 shows that the $O(kw)$ upper bound of Theorem 3 is tight (for non-linear polynomials). In fact, the matching $\Omega(kw)$ lower bound applies to a much wider class of functions than just polynomials. Let us consider first the simpler case of sigmoidal functions.

**Proof of Theorem 5.** This follows from Theorem 7 and the fact on any finite set of inputs, linear and threshold gates can be simulated by gates with activation $\sigma$. □

**Proof of Theorem 7.** We can assume that $\kappa = k - 2 \geqslant 1$. We also assume that $w$ is of the form $14v + 2$. As in Theorem 4, the generalization to other values of $w$ is straightforward. We first define the shattered set $S$: a sequence $u \in \mathbb{R}^\kappa$ is in $S$ if it has exactly one non-zero component, and that component is in $\{1,\dots,v\}$ (obviously, $|S| = \kappa v$). Next we define a family $\mathscr{F}$ of functions which shatters $S$. The functions in this family are indexed by $v$ parameters $w_1,\dots,w_v \in [0,1]$. Each parameter is assumed to have a finite $\kappa$-bit binary expansion $0.w_{i1}\dots w_{i\kappa}$. Given an input $u \in S$ with $i = u(j)$ as non-zero component, the corresponding output simply is $f_{w_1,\dots,w_v}(u) := w_{ij}$ (i.e., we select bit number $j$ of $w_i$). It is clear that $\mathscr{F}$ shatters $S$: any function $f : S \to \{0,1\}$ can be implemented by setting $w_{ij} = f(ie_j)$ ($e_j$ denotes the element of $S$ with a 1 in the $j$th position).

In a recurrent network implementation of this, the set $S'$ of shattered sequences is obtained by adding two “dummy” inputs $u(\kappa) = u(k-2) = 0$ and $u(\kappa+1) = u(k-1) = 0$ at the end of a sequence $(u(0),\dots,u(\kappa-1)) \in S$, as in the proof of Theorem 4.

The parameters $w_1,\dots,w_v$ are stored in the initial states of units $x_1,\dots,x_v$. As the computation proceeds, these units will store shifted versions of the parameters. The leading bits of $x_1,\dots,x_v$ are stored in $v$ other units $y_1,\dots,y_v$. The initial state of $x_i$ is $w_i/2$; all other units are initialized to 0. (Note that this implies in particular that at $t = 0$, $y_i$ indeed stores the leading bit of $x_i$.) New values of $x_i$ and $y_i$ can be computed at each time step by the following 5-weight system:

$$x_i^+ = 2x_i - y_i,$$
$$y_i^+ = H(4x_i - 2y_i - 1).$$

The network should output 1 if the current input $u$ is equal to $i \neq 0$, and $y_i = 1$. This can be checked by computing $E_i = H[H(u - i + 0.5) + H(i + 0.5 - u) + y_i - 2.5]$ (this requires $3v$ threshold gates and $8v$ weights). There is one additional threshold gate $o$ which serves as output unit. It keeps computing the OR of the $E_i$'s and of the previous output. This can be implemented with $v + 2$ weights as in the proof of Theorem 4: $o^+ = H(o + \sum_{i=1}^{v} E_i - 0.5)$.

The architecture described above has $5v + 8v + (v + 2) = 14v + 2 = w$ weights. The output $f_{w_1,\dots,w_v}(u(0),\dots,u(\kappa - 1))$ is carried by the output unit at time $(\kappa - 1) + 3 = k$. Note that the last two inputs $u(\kappa)$ and $u(\kappa + 1)$ are "wasted", i.e., they do not influence the final output $o(k)$. Note also that the outputs at $t = 1$ and $t = 2$ are both 0 as needed. □

Theorem 6 generalizes Theorem 5 to even more arbitrary activations. For this we need some of the machinery of [11]. In particular, we need to allow networks with multiplication and division gates. These gates have fan-in two and number of weights also two (even though there is no natural numerical parameter associated to the gate; we need to assign weights to multiplication and division gates to account for the numerical parameters that will occur when simulating these gates by $\sigma$-gates). The output of a multiplication gate is defined as the product of its two inputs. The output of a division gate is defined as the quotient of its two inputs, assuming that the second input is nonzero. An input to a circuit is said to be *valid* if it does not cause a division by zero at any division gate. We will only work with sets of valid inputs (so the domain of the function computed by such a generalized network is a subset of $\mathbb{R}^m$ and shattering is only defined for subsets of this domain).

We will use *feedforward architectures* as building blocks in our recurrent architectures. The necessary background is standard and can be found, for instance, in [11]. To be self-contained, we recall that the units of feedforward architectures are grouped into *layers*. We use the same type of units as in recurrent architectures (in particular, multiplication and division gates are allowed, as mentioned earlier in this section). The inputs to the architecture are fed to units in the first layer. For $i > 1$, units in layer $i$ receive their inputs from layer $i - 1$. The last layer is made of a single gate: the output gate. The function computed by a gate is defined by a straightforward induction on its depth in the architecture. The function computed by the architecture is the function computed by the output gate.

Readers familiar with feedforward nets will notice that we do not allow connections between non-adjacent layers. For synchronization reasons, such connections are to be avoided in recurrent nets. One can always convert a non-layered feedforward architecture into a layered one by introducing delays (identity gates).

The following two lemmas from [11] are needed (the first one is well-known and easy to prove).

**Lemma 2.** *Let $\phi : [0,1] \to [0,1]$ be the logistic map $\phi(x) = 4x(1 - x)$. For every $n \geq 1$ and every $\varepsilon \in \{0,1\}^n$ there exists $x_1 \in [0,1]$ such that the sequence $(x_k)_{1 \leq k \leq n}$ defined by $x_{k+1} = \phi(x_k)$ for $k = 1,\dots,n - 1$ satisfies the following property: $0 \leq x_k < 1/2$ if $\varepsilon_k = 0$ and $1/2 < x_k \leq 1$ if $\varepsilon_k = 1$.*

The next result is essentially Lemma 1 from [11].

**Lemma 3.** *For every $n \geq 0$, there is a feedforward architecture $\mathcal{A}$ with inputs $(x, W_0, \dots, W_n)$ in $\mathbb{R}^{n+2}$ such that the following property holds: for every $\varepsilon > 0$ there* *exists a choice of the weights of $\mathscr{A}$ such that the function $f_\varepsilon$ implemented by the network satisfies* $\lim_{\varepsilon\to 0} f_\varepsilon(i, W_0, \dots, W_n) = W_i$ *for* $i = 0, \dots, n$.

*This architecture is made of linear, multiplication and division gates. It has $\Theta(n)$ weights and depth $\Theta(\log n)$.*

**Lemma 4.** *The VC dimension of recurrent architectures of linear, multiplication and division gates with w weights receiving inputs of length $k = \Omega(\log w)$ is $\Omega(wk)$.*

**Proof.** It is similar to that of Theorem 7. In particular, the shattered set $S \subseteq \mathbb{R}^k$ is the same and the class $\mathscr{F}$ of functions shattering $S$ is indexed in the same way. Hence, we will just sketch the main differences with the linear-threshold case in the implementation of $\mathscr{F}$ on a recurrent network. The bit-extracting device in Theorem 7 can be replaced by the following system:

$$x_i^+ = 4x_i(1 - x_i). \tag{10}$$

A value of $x_i$ smaller than 1/2 should be understood as encoding the binary digit 0 ("reject") and a value larger than 1/2 the digit 1 ("accept"). By Lemma 2, any (finite) binary sequence can be produced by (10) with a suitable choice of $x_i(0)$. This system can be implemented by a subnetwork of two linear gates (computing $1 - x_i$ and $4x_i$) and one product gate. It produces an output at every other time step. Therefore we can only feed an input to the network at every other time step, too (the gaps in the input sequence can be filled by arbitrary, meaningless values).

The output of (10) should be selected if the current input $u$ is equal to $i \neq 0$. By Lemma 3, this can be done (approximately) as follows:

$$E = f_\varepsilon(u, 0, x_1, \dots, x_v). \tag{11}$$

Note that the subnetwork implementing this function has depth $\Theta(\log w)$, whence the condition $k = \Omega(\log w)$ in the statement of the lemma (we need to add $\Theta(\log w)$ dummy inputs at the end of the input sequence). Since the network has to work only on a finite set of inputs, the construction will be correct if $\varepsilon$ is small enough (this can be justified as in [11]).

Finally, the output unit accumulates the values of $E$, starting from the initial state $o = 0$. Note that these accumulated values are all (approximately) zero, except at most one of them. This is because any input in the shattered set $S$ of Theorem 7 has only one non-zero component. And whenever the current input component $u$ is zero, the function $f_\varepsilon$ in (11) selects the first number in the sequence $(0, x_1, \dots, x_v)$, that is, 0.

In order to implement this on a recurrent network, we have to introduce a delay since meaningful values of $E$ come only at every other time step. Therefore, one would like to write

$$o^+ = \mathrm{Id}(o) + E, \tag{12}$$

where the identity function Id is implemented by a linear gate. An input would be rejected if the output at time $k$ is smaller than 1/2; it would accepted if the output is larger than 1/2. The only problem with this construction is that the output unit might accumulate non-zero values of $E$ which occur even before the first input can be processed. In the proof of Theorem 7 we have checked "by hand" that this problem does not occur. Here we prefer to use instead a special-purpose device: we replace (12) by

$$o^+ = \mathrm{Id}(o) + s_0 E, \tag{13}$$

where $s_0$ is designed to output 0 for the first few $T = \mathrm{O}(\log v)$ time steps, and 1 thereafter. This can be done with the following system of $T+1$ units: $s_i^+ = s_{i+1}$ for $i = 0, \ldots, T-1$ and $s_T^+ = s_T$. These units are initialized as follows: $s_i(0) = 0$ for $i = 0, \ldots, T-1$, and $s_T(0) = 1$. □

**Theorem 9.** *The VC dimension of recurrent architectures of linear and multiplication gates with $w$ weights receiving inputs of length $k = \Omega(\log w)$ is $\Omega(wk)$.*

**Proof.** The theorem follows from Lemma 4 and the (simple) simulation of networks with linear, multiplication and division gates by networks with linear and multiplication gates only [11]. This simulation applies to feedforward as well as to recurrent networks. Note that since the length of the longest path in the network increases by a constant factor, it is necessary to pad the input sequence with $\mathrm{O}(\log w)$ dummy inputs. This changes only the implied constants in the $\Omega$ symbols.

As in [11], this result makes it possible to prove good VC dimension lower bounds for a wide class of transfer functions. The most important case is Theorem 6, which we can now prove.

**Proof of Theorem 6.** Linear and multiplication gates can be simulated by $\sigma$-gates as in [11]. The input sequence must be padded by a small number of dummy inputs as in the proof of Theorem 9. □

## 5. Final remarks

We have left several questions unanswered:

(1) For piecewise-polynomial functions, can one close the gap between the $\mathrm{O}(kw^2)$ upper bound and the $\Omega(kw)$ lower bound?

(2) This gap is even bigger for the standard sigmoid: $\mathrm{O}(k^2w^4)$ versus $\Omega(kw)$. A tight bound is probably too much to ask for since even for feedforward architectures there is a gap: $\mathrm{O}(w^4)$ versus $\Omega(w^2)$. A less ambitious goal would be to replace the $k^2$ factor in the upper bound by $k$.

(3) For threshold architectures, we have a tight $\Theta(w \log k)$ bound for $k \gg w$. However, this tight bound applies only when $k$ is exponentially larger than $w$. It would be interesting to have a tight bound when $k$ is polynomial in $w$.

## Acknowledgements

We thank an anonymous referee for his very careful reading of the manuscript and many useful comments.

## References

[1] E.B. Baum, D. Haussler, What size net gives valid generalization?, Neural Comput. 1 (1989) 151–160.
[2] Y. Bengio, Neural Networks for Speech and Sequence Recognition, Thompson Computer Press, Boston, 1996.
[3] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth, Learnability and the Vapnik–Chervonenkis Dimension, J. ACM 36 (1989) 929–965.
[4] T.M. Cover, Capacity problems for linear machines, in: L. Kanal (Ed.), Pattern Recognition, Thompson Book Co., 1968, pp. 283–289.
[5] B. Dasgupta, E.D. Sontag, Sample complexity for learning recurrent perceptron mappings, IEEE Trans. Inform. Theory, 42 (1996) 1479–1487. Summary in: D.S. Touretzky, M.C. Moser, M.E. Hasselmo (Eds.), Advances in Neural Information Processing Systems 8 (NIPS95), MIT Press, Cambridge, MA, 1996, pp. 204–210.
[6] C.L. Giles, G.Z. Sun, H.H. Chen, Y.C. Lee, D. Chen, Higher order recurrent networks and grammatical inference, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing Systems 2, Morgan Kaufmann, San Mateo, CA, 1990.
[7] P. Goldberg, M. Jerrum, Bounding the Vapnik–Chervonenkis dimension of concept classes parameterized by real numbers, Machine Learning 18 (1995) 131–148.
[8] S. Grossberg, The Adaptive Brain, 2 vols, Elsevier, Amsterdam, 1987.
[9] J.J. Hopfield, Neural networks and physical systems with emergent computational abilities, Proc. Natl. Acad. Sci. USA 79 (1982) 2554–2558.
[10] M. Karpinski, A. Macintyre, Polynomial bounds for VC dimension of sigmoidal and general Pfaffian neural networks, J. Comput. System Sci. 54 (1997) 169–176.
[11] P. Koiran, E.D. Sontag, Neural networks with quadratic VC dimension, J. Comput. System Sci. 54 (1997) 190–198.
[12] M.M. Polycarpou, P.A. Ioannou, Neural networks and on-line approximators for adaptive control, in: Proc. 7th Yale Workshop on Adaptive and Learning Systems, Yale University, 1992, pp. 793–798.
[13] H. Siegelmann, E.D. Sontag, On the computational power of neural nets, J. Comput. System Sci. 50 (1995) 132–150.
[14] H. Siegelmann, E.D. Sontag, Analog computation, neural networks, and circuits, Theoret. Comput. Sci. 131 (1994) 331–360.
[15] E.D. Sontag, Mathematical Control Theory: Deterministic Finite Dimensional Systems, Springer, New York, 1990.
[16] E.D. Sontag, Neural nets as systems models and controllers, in: Proc. 7th Yale Workshop on Adaptive and Learning Systems, Yale University, 1992, pp. 73–79.
[17] E.D. Sontag, Feedforward nets for interpolation and classification, J. Comput. System Sci. 45 (1992) 20–48.
[18] A.M. Zador, B.A. Pearlmutter, VC dimension of an integrate-and-fire neuron model, Neural Comput. 8 (1996) 611–624.