

# Computational Capabilities of Recurrent NARX Neural Networks

Hava T. Siegelmann, Bill G. Horne, and C. Lee Giles, *Senior Member, IEEE*

**Abstract**—Recently, fully connected recurrent neural networks have been proven to be computationally rich—at least as powerful as Turing machines. This work focuses on another network which is popular in control applications and has been found to be very effective at learning a variety of problems. These networks are based upon Nonlinear AutoRegressive models with eXogenous Inputs (NARX models), and are therefore called *NARX networks*. As opposed to other recurrent networks, NARX networks have a limited feedback which comes only from the output neuron rather than from hidden states. They are formalized by

$$y(t) = \Psi(u(t - n_u), \dots, u(t - 1), u(t), y(t - n_y), \dots, y(t - 1))$$

where  $u(t)$  and  $y(t)$  represent input and output of the network at time  $t$ ,  $n_u$  and  $n_y$  are the input and output order, and the function  $\Psi$  is the mapping performed by a Multilayer Perceptron. We constructively prove that the NARX networks with a finite number of parameters are computationally as strong as fully connected recurrent networks and thus Turing machines. We conclude that in theory one can use the NARX models, rather than conventional recurrent networks without any computational loss even though their feedback is limited. Furthermore, these results raise the issue of what amount of feedback or recurrence is necessary for any network to be Turing equivalent and what restrictions on feedback limit computational power.

## I. INTRODUCTION

THE computational capabilities of recurrent neural networks have been studied for at least fifty years. Some of the earliest work in this area by McCulloch and Pitts showed that networks of neuron-like elements are capable of implementing some types of finite state machines (FSM's) [1]. Later, Minsky showed that any FSM could be mapped into such a network [2]. More recently, new results have been developed to improve the efficiency of this mapping [3]–[5], [30]. All of these results assume that the nonlinearity used in the network is a hard-limiting threshold function. However, when recurrent networks are used adaptively, continuous-valued, differentiable nonlinearities are almost always used. Thus, an interesting question is how the computational complexity changes for these types of functions. It has been recently shown that such networks are at least as powerful as Turing machines, and in some cases can have super-Turing capabilities [6]–[9]. The proof utilizes a construction that shows how

Manuscript received March 18, 1995; revised August 7, 1995.

H. T. Siegelmann is with the Faculty of Industrial Engineering and Management, Technion—The Israel Institute of Technology, Haifa 32000, Israel (e-mail: iehava@ie.technion.ac.il).

B. G. Horne is with the NEC Research Institute, Princeton, NJ 08540 USA (e-mail: horne@research.nj.nec.com).

C. L. Giles is with the NEC Research Institute, Princeton, NJ 08540 USA and UMIACS, University of Maryland, College Park, MD 20742 USA (e-mail: giles@research.nj.nec.com).

Publisher Item Identifier S 1083-4419(97)00800-5.

fully connected networks can simulate pushdown automata with two stacks, which are computationally equivalent to Turing machines. The stacks are encoded in two of the nodes of the network with the remaining nodes used to simulate the finite state control. There is an initial period during which the network reads the input, then the network performs the desired computation, and finally the output of the network is decoded.

An important class of discrete-time nonlinear systems is the *Nonlinear AutoRegressive with eXogenous Inputs* (NARX) model [10]

$$y(t) = f(u(t - n_u), \dots, u(t - 1), u(t), y(t - n_y), \dots, y(t - 1)) \quad (1)$$

where  $u(t)$  and  $y(t)$  represent input and output of the network at time  $t$ ,  $n_u$  and  $n_y$  are the input and output order, and the function  $f$  is a nonlinear function. When the function  $f$  can be approximated by a Multilayer Perceptron, the resulting system is called a *NARX network* [11], [12]. It has been demonstrated that this particular model is well suited for modeling nonlinear systems such as heat exchangers [11], waste water treatment plants [13], [14], catalytic reforming systems in a petroleum refinery [14], nonlinear oscillations associated with mutli-legged locomotion in biological systems [15], and various artificial nonlinear systems [11], [12], [16]. Furthermore, in a previously published paper we benchmarked NARX networks against nine other recurrent neural network architectures on problems including grammatical inference and nonlinear system identification [17], [18]. We found that NARX networks typically converge much faster and generalize better than these other networks. We have also shown that NARX networks perform better on problems involving *long-term dependencies* [19], [20].

Based on the mapping theorems of [21], [22], NARX networks should be capable of representing arbitrary systems expressible in the form of (1). However, using such an approach there is no bound to the number of nodes required to achieve a good approximation. Furthermore, it is not clear how such systems relate to conventional models of computation. In this paper we explore the computational capabilities of this network compared to those of the fully connected networks. We prove that NARX networks are computationally at least as strong as fully connected networks within a linear slowdown. This implies that NARX networks *with a finite number of nodes and taps* are at least as powerful as Turing machines, and thus are universal computation devices. This result is somewhat unexpected given the limited nature of feedback in these networks.

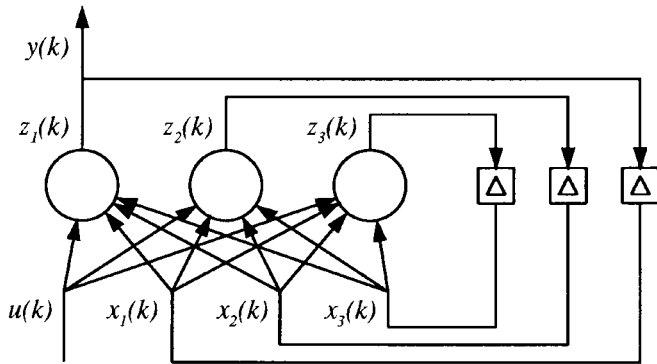


Fig. 1. A fully connected recurrent neural network.

We also provide some related results concerning NARX networks with nonlinear output functions. In particular, when hard-limiting nonlinearities are used, we show that NARX networks are only capable of implementing a subclass of FSM's called Finite Memory Machines (FMM's). However, we show that FSM's can be simulated by FMM's within a sublinear slowdown.

## II. RECURRENT NEURAL NETWORK MODELS

We consider two recurrent neural network models: 1) fully connected networks; and 2) NARX networks. We shall restrict our attention to single-input, single-output systems, which are sufficient for establishing the computational capabilities of the network. These results can easily be extended to the multi-variable case, by simply replacing scalars by vectors where appropriate and creating multiple tapped delay lines from the outputs of the network. Each tapped delay line would be constructed following the method used for a single output.

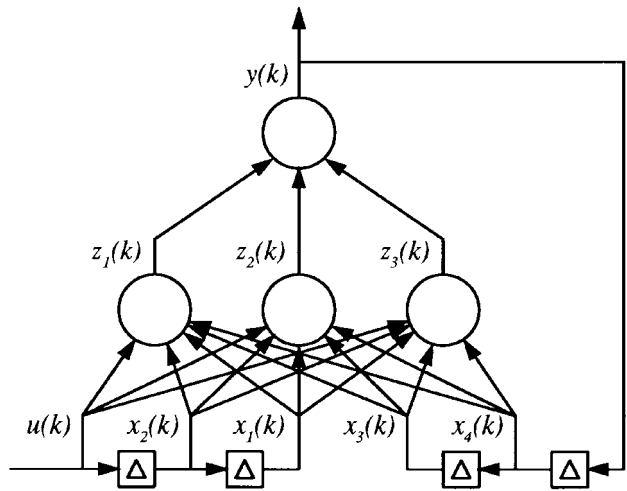
We shall adopt the notation that  $x$  corresponds to a state variable,  $u$  to an input variable,  $y$  to an output variable, and  $z$  to a node activation value. In each of these networks we shall let  $N$  correspond to the dimension of the state space. When necessary to distinguish between variables of the two networks, those associated with the NARX network will be marked with a tilde, e.g.,  $x_i(t)$  and  $\tilde{x}_i(t)$  will refer to the  $i$ th state variable in the fully connected and NARX networks, respectively.

The state variables of a recurrent network are defined to be the memory elements, i.e., the set of time delay operators. In a fully connected network there is a one-to-one correspondence between node activations and state variables of the network, since each node value is stored at every time step. Specifically, the value of the  $N$  state variables at the next time step are given by

$$x_i(t+1) = z_i(t).$$

Each node weights and sums the external inputs to the network and the states of the network. Specifically, the activation function for each node is defined by

$$z_i(t) = \sigma \left( \sum_{j=1}^N a_{i,j} x_j(t) + b_i u(t) + c_i \right) \quad (2)$$

Fig. 2. A NARX network with  $n_u = n_y = 2$  and  $\bar{H} = 3$ .

where  $a_{i,j}$ ,  $b_i$ , and  $c_i$  are fixed real valued weights, and  $\sigma$  is a nonlinear function which will be discussed below. The output is assigned arbitrarily to be the value of the first node in the network

$$y(t) = z_1(t).$$

The network is said to be fully connected because there is a weight between every pair of nodes. However, when weight  $a_{i,j} = 0$  there is effectively no connection between nodes  $i$  and  $j$ . Thus, a fully connected network is very general, and can be used to represent many different kinds of architectures, including those in which only a subset of the possible connections between nodes are used. Alternatively, one can think of fully connected networks as a single layer of nodes with complete feedback, as shown in Fig. 1.

A NARX network consists of a Multilayer Perceptron (MLP) which takes as input a window of past input and output values and computes the current output. Specifically, the operation of the network is defined by

$$\tilde{y}(t) = \Psi(\tilde{u}(t - n_u), \dots, \tilde{u}(t - 1), \tilde{u}(t), \tilde{y}(t - n_y), \dots, \tilde{y}(t - 1)) \quad (3)$$

where the function  $\Psi$  is the mapping performed by the MLP, as shown in Fig. 2.

The states of the NARX network correspond to a set of two tapped-delay lines. One consists of  $n_u$  taps on the input values, and the other consists of  $n_y$  taps on the output values. Specifically, the states are updated as

$$\tilde{x}_i(t+1) = \begin{cases} \tilde{u}(t) & i = n_u \\ \tilde{y}(t) & i = n_u + n_y \\ \tilde{x}_{i+1}(t) & 1 \leq i < n_u \text{ and } n_u < i < n_u + n_y \end{cases}$$

so that at time  $t$  the taps correspond to the values

$$\tilde{\mathbf{x}}(t) = [\tilde{u}(t - n_u) \quad \dots \quad \tilde{u}(t - 1) \quad \tilde{y}(t - n_y) \quad \dots \quad \tilde{y}(t - 1)].$$

The MLP consists of a set of nodes organized into two layers<sup>1</sup>. There are  $\tilde{H}$  nodes in the first layer which perform the function

$$\tilde{z}_i(t) = \sigma \left( \sum_{j=1}^{\tilde{N}} \tilde{a}_{i,j} \tilde{x}_j(t) + \tilde{b}_i \tilde{u}(t) + \tilde{c}_i \right) \quad i = 1, \dots, \tilde{H}.$$

The output layer consists of a single linear node

$$\tilde{y}(t) = \sum_{j=1}^{\tilde{H}} w_{i,j} \tilde{z}_j(t) + \theta_i.$$

A detailed picture of a NARX network with  $n_u = n_y = 2$  and  $\tilde{H} = 3$  is shown in Fig. 2.

*Definition 1:* A function  $\sigma$  is said to be a *bounded, one-side saturated (BOSS) function* if it satisfies the following conditions:

a)  $\sigma$  has a bounded range, i.e.,  $L \leq \sigma(x) \leq U, L \neq U$  for all  $x \in \mathbb{R}$ .

b)  $\sigma$  is left-side saturated<sup>2</sup>, i.e., there exists a finite value  $s$ , such that  $\sigma(x) = S$  for all  $x \leq s$ .

c)  $\sigma$  is nonconstant (i.e., there exist at least two values  $x_1$  and  $x_2$  such that  $\sigma(x_1) \neq \sigma(x_2)$ ).  $\square$

BOSS functions include many sigmoid-like functions; for example, *hard-limiting threshold functions*

$$\sigma(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \quad (4)$$

and the saturated linear function

$$\sigma(x) = \begin{cases} 0 & x \leq 0 \\ x & 0 < x < 1 \\ 1 & x \geq 1 \end{cases} \quad (5)$$

are both BOSS functions.

Although the sigmoid function,  $\sigma(x) = [1 + \exp(-x)]^{-1}$ , is not considered to be a BOSS function because it does not saturate, it can be slightly modified to be so. Specifically, a “one side saturated sigmoid,”

$$\sigma(x) = \begin{cases} 0 & x \leq c \\ \frac{1}{1 + e^{-x}} & x > c \end{cases}$$

is a BOSS function, where  $c \in \mathbb{R}$ .

### III. MAIN RESULT

In this section, we prove that NARX networks with BOSS functions are capable of simulating fully connected networks with only a linear slowdown. Because of the universality of some types of fully connected networks with a finite number of nodes, we conclude that the associated NARX networks are Turing universal as well.

*Theorem 1:* NARX networks with one hidden layer of nodes with BOSS activation functions and a linear output node can simulate fully connected recurrent networks with BOSS activation functions with a linear slowdown.  $\square$

<sup>1</sup>More layers could be used, but are not necessary for our purposes.

<sup>2</sup>Equivalently, the function can be defined to be saturated to the right, i.e.,  $\sigma(x) = S$  for all  $x \geq s$ , and we would obtain the same results.

*Proof:* To prove the theorem we show how to construct a NARX network  $\mathcal{N}$  that simulates a fully connected network  $\mathcal{F}$  with  $N$  nodes, each of which uses a BOSS activation function  $\sigma$ . The NARX network requires  $N + 1$  hidden layer nodes, a linear output node, an output shift register of order  $n_y = 2N$ , and no taps on the input. Without loss of generality we assume that the left saturation value of  $\sigma$  is  $S = 0$ . This restriction makes the proof somewhat simpler, but can be easily relaxed.

The simulation suffers a linear slowdown; specifically, if  $\mathcal{F}$  computes in time  $T$ , then the total computation time taken by  $\mathcal{N}$  is  $(N + 1)T$ . In particular, time  $t$  is simulated during time steps  $k = (N + 1)t + i, i = 1, \dots, N + 1$ . Because of the linear slowdown, the input to  $\mathcal{N}$  must be kept constant for each simulation period, i.e.,

$$\tilde{u}((N + 1)t + i) = u(t) \quad i = 1, \dots, N + 1.$$

For each  $1 \leq i \leq N, \mathcal{N}$  will simulate the value of exactly one of the nodes in  $\mathcal{F}$ . The additional time step will be used to encode a sequencing signal indicating which node should be simulated next. Specifically,

$$\tilde{z}_i((N + 1)t + j) = \begin{cases} z_i(t) & 1 \leq i = j \leq N \\ \sigma(\alpha) & i = j = N + 1 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

for  $1 \leq i \leq N + 1$ , where  $\sigma(\alpha)$  is related to the sequencing signal and will be discussed at length below.

The output taps of  $\mathcal{N}$  will be used to store the simulated states of  $\mathcal{F}$ ; no taps on the input are required, i.e.,  $n_u = 0$ . At any given time the tapped delay line must contain the complete set of values corresponding to all  $N$  nodes of  $\mathcal{F}$  at the previous simulated time step. To accomplish this, a tapped delay line of length  $n_y = 2N$  is sufficient. Specifically, at time  $(N + 1)t + i$ , the tapped delay line contains the values ( $i = 1, \dots, N + 1, j = 1, \dots, 2N$ )

$$\tilde{x}_j((N + 1)t + i) = \begin{cases} z_{i+j+1}(t-2) & 0 < i + j < N \\ \mu & i + j = N \\ z_{i+j-N}(t-1) & N < i + j < 2N + 1 \\ \mu & i + j = 2N + 1 \\ z_{i+j-2N-1}(t) & 2N + 1 < i + j \end{cases} \quad (7)$$

where  $\mu$  (the sequencing signal) is outside the range  $[L, U]$  (see Definition 1A); this constant will be discussed shortly. With this representation the taps will always contain all of the values of  $\mathcal{F}$  at time  $t-1$  immediately preceding the sequencing signal,  $\mu$ , to indicate where these variables are in the tap. The contents of the taps at various times are illustrated in Fig. 3.

We next show how to chose the dynamics of the hidden neurons. The sequencing signal is chosen in such a way that we can define a simple function  $f_\mu$  that is used to either “turn off” neurons or to yield a constant value, according to the values in the taps. Let  $\mu = U + \epsilon$  for some positive constant  $\epsilon$ . We define the affine function

$$f_\mu(x) = x - \mu. \quad (8)$$

Then,  $f_\mu(\mu) = 0$  and  $f_\mu(x) \leq -\epsilon$  for all  $x \in [L, U]$ . According to (6) and (7), node  $\tilde{z}_i((N + 1)t + j)$  may take on a nonzero value only when  $i = j$ , or equivalently when

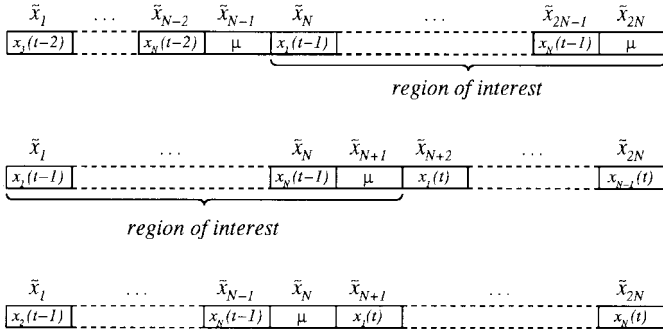


Fig. 3. The contents of the output tapped delay line of the NARX network at times  $(N+1)t+1$  when  $x_1(t)$  is to be simulated next (top),  $(N+1)t+N$  when  $x_N(t)$  is to be simulated next (middle), and  $(N+1)t+(N+1)$  when the sequencing signal  $\mu$  is to be generated next (bottom). After each time step, the contents of the taps move to the left, and the value of the output is stored in the right most tap.

$\tilde{x}_{2N-i+1} = \mu_i$ ; in this case, the values of  $z_i(t-1)$  are stored in the taps  $\tilde{x}_{N+m-i}$ ,  $m = 1, \dots, N$ . Thus, using (2) and (8), the  $i$ th node in the hidden layer of  $\mathcal{N}$  is updated as follows:

$$\tilde{z}_i(k+1) = \sigma \left( \left[ \sum_{m=1}^N a_{i,m} \tilde{x}_{N+m-i}(k) + b_i u(k) + c_i \right] + \beta_i [\tilde{x}_{2N-i+1}(k) - \mu] \right) \quad (9)$$

where the constant  $\beta_i$  is large enough to make the input to  $\sigma$  less than  $s$  when  $\tilde{x}_{2N-i+1}(k) \neq \mu$  so that the whole function is zero<sup>3</sup>.

There exists at least one fixed value  $\alpha$  such that  $\sigma(\alpha) \neq 0$ . The value  $\tilde{z}_{N+1}(k)$  will toggle between 0 and  $\sigma(\alpha)$ . Specifically,  $\tilde{z}_{N+1}(k)$  should equal  $\sigma(\alpha)$  only when  $\tilde{x}_N = \mu$ , otherwise it should equal zero. Thus, using (8), its update equation can be written

$$\tilde{z}_{N+1}(k) = \sigma(\beta_{N+1}[\tilde{x}_N(k) - \mu] + \alpha) \quad (10)$$

where once again,  $\beta_{N+1}$  is large enough to make the entire function zero when  $\tilde{x}_N(k) \neq \mu$ .

So far the construction ensures that (6) will hold. Next, the output node of  $\mathcal{N}$  is then simply the linear combination

$$\tilde{y}(k) = \frac{\mu}{\sigma(\alpha)} \tilde{z}_{N+1}(k) + \sum_{i=1}^N \tilde{z}_i(k) \quad (11)$$

so that the output of the network is equal to the value of the currently active hidden layer node, which in turn ensures that the feedback will be consistent with (7).

Finally, we consider the initial conditions of the network. The taps should be initially configured as follows:

$$\tilde{x}_j(0) = \begin{cases} * & 0 < j < N \\ x_{j-N+1}(0) & N \leq j < 2N \\ \mu & j = 2N \end{cases}$$

where  $*$  stands for any value in the range  $[L, U]$ . At the next time step the network will be ready to simulate  $z_1(1)$ . Q.E.D.

<sup>3</sup>We assume the value of the input is bounded.

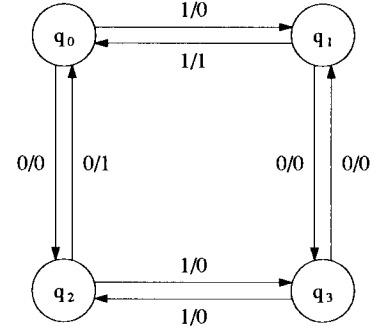


Fig. 4. The Dual Parity FSM.

It has been shown that fully connected networks with a fixed, finite number of saturated linear activation functions are universal computation devices [7], [8]. As a result it is possible to simulate a Turing machine with the NARX network such that the slowdown is constant regardless of problem size. Thus, we conclude that

*Corollary 1:* NARX networks with one hidden layer of nodes with saturated linear activation functions and linear output nodes are Turing equivalent.  $\square$

#### IV. RELATED RESULTS

In this section, we look at variants of the NARX networks, in which the output functions are not linear combiners but rather some kind of nonlinear activation function.

##### A. Hard-Limiters

If the nonlinearity is a hard-limiting function [see (4)] and the inputs are binary, then recurrent neural networks are only capable of implementing FSM's, and NARX networks are only capable of implementing a subset of FSM's called Finite Memory Machines (FMM's) [23], [24], which are defined to be an FSM whose input/output relationship can be described by the equation

$$y(t) = \phi(u(t-n_u), \dots, u(t-1), u(t), y(t-n_y), \dots, y(t-1))$$

where  $u(t)$  and  $y(t)$  assume boolean values, and  $\phi$  is a combinational logic function. Clearly this equation has the same form as (3), so when a hard-limiter is used for the nonlinearity of the output node, the function  $\Psi$  is a logic function, and it is clear that NARX networks are equivalent to FMM's.

Not all FSM's are FMM's. FMM's have the property that the state of the machine can always be determined from a finite number of observations of the inputs and outputs of the system when the initial state is unknown. In other words, the states of an FMM are observable. For example, the Dual Parity FSM, shown in Fig. 4, is not finite memory since one can observe an infinite sequence of ones at the input and an infinite sequence of zeros at the output without being able to determine whether the FSM is in state  $q_2$  or  $q_3$ . In contrast, the FSM shown in Fig. 5, is an FMM since for any input sequence of length two, the state of the FSM's can always be determined from knowledge of the past two inputs and the last output as illustrated in Table I.

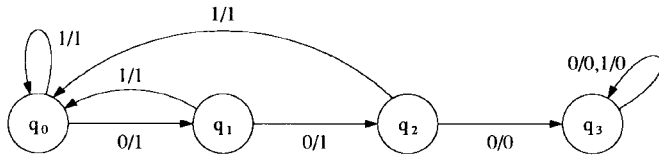


Fig. 5. A Finite Memory Machine (FFM).

Intuitively, the reason why FMM's are constrained is that there is a limited amount of information that can be represented by feeding back the outputs alone. If more information could be inserted into the feedback loop, then it should be possible to simulate arbitrary FSM's in structures like NARX networks. In fact, we next show that this is the case. We will prove that NARX networks with hard-limiting nonlinearities are capable of simulating fully connected networks with a slowdown proportional to the number of nodes. As a result, the NARX network will be able to simulate arbitrary FSM's. To do this, the network uses the extra time steps associated with the slowdown to insert information about the state of the FSM. We provide an upper bound for the amount of slowdown, which is a function of the number of states of the FSM.

**Theorem 2:** NARX networks with hard-limiting activation functions and one hidden layer of nodes can simulate fully connected networks with hard-limiting activation functions with a linear slowdown.  $\square$

*Proof:* By a construction similar Theorem 1, we show that a NARX network  $\mathcal{N}$ , consisting of a shift-register of length  $4N+1$ ,  $N+1$  BOSS hidden neurons, and a hard-limiter activation at the output level, can simulate a fully connected network  $\mathcal{F}$  with  $N$  nodes, each of which uses a hard-limiting activation function  $\sigma$ .

The simulation suffers a linear slowdown. Except here, if  $\mathcal{F}$  computes in time  $T$ , then the total computation time taken by  $\mathcal{N}$  is  $(2N+3)T$ . The extra computations are used to implement a null signal (chosen to be zero) between the simulation of each node, and the "end of sequence" signal (chosen as two consecutive 1s). By interleaving the simulation of the node values with zeros, the only way two consecutive ones can appear within the tap is if they correspond to the end of sequence signal.

The network will require a tapped delay line of length  $n_y = 4N+1$  on the output, but still no taps on the input. Fig. 6 illustrates the tap contents at various times. The indexing scheme is similar to the one given in (7), but because of the interleaved zeros, it is excessively cumbersome, and so we omit it for the sake of brevity. With this representation the taps will always contain all of the values of  $\mathcal{F}$  at time  $t-1$  preceding the sequencing signals, to indicate where these variables are in the tap.

We pursue a similar approach to define the dynamic equations of the neurons: we define a simple function  $f$  that "turns off" nodes or produces a constant value, depending on the contents of the taps. Specifically, define the affine function

$$f(x_1, x_2) = x_1 + x_2 - 2. \quad (12)$$

Then,  $f(1, 1) = 0$  and  $f(x, 0)$  and  $f(0, x)$  are both less than or equal to  $-1$  for all  $x \in \{0, 1\}$ .

TABLE I  
THE STATE OF THE MACHINE AS A FUNCTION OF THE  
PREVIOUS TWO INPUTS AND PREVIOUS OUTPUT

$y(t-1)$	$u(t-2)$	$u(t-1)$	state
0	0	0	$q_3$
0	0	1	$q_3$
0	1	0	$q_3$
0	1	1	$q_3$
1	0	0	$q_2$
1	0	1	$q_0$
1	1	0	$q_1$
1	1	1	$q_0$

The network will still have  $N+1$  hidden nodes, corresponding to the nodes of  $\mathcal{F}$ . Each node will correspond to the values

$$\tilde{z}_i((2N+3)t+j) = \begin{cases} z_i(t) & i = 2j-1, \quad 1 \leq j \leq N \\ \sigma(\alpha) & i = N+1, \\ & 2N+1 \leq j \leq 2N+2 \\ 0 & \text{otherwise.} \end{cases} \quad (13)$$

The  $i$ th node has a nonzero value when  $\tilde{x}_{4N-2i+1} = \tilde{x}_{4N-2i+2} = 1$ , and values of  $z_1, \dots, z_N$  correspond to tap values  $\tilde{x}_{2(N+m-i)-1}$ ,  $m = 1, \dots, N$ . So, using (2) and (12), the  $i$ th node in the hidden layer of  $\mathcal{N}$  is updated as follows:

$$\tilde{z}_i(k+1) = \sigma \left( \left[ \sum_{m=1}^N a_{i,m} \tilde{x}_{2(N+m-i)-1}(k) + b_i u(k) + c_i \right] + \beta_i [\tilde{x}_{4N-2i+1}(k) + \tilde{x}_{4N-2i+2}(k) - 2] \right)$$

where the constant  $\beta_i$  is large enough to make the whole function 0 if  $\tilde{x}_{4N-2i+1}(k)$  and  $\tilde{x}_{4N-2i+2}(k)$  are not equal to one.

The node that implements the sequencing signal becomes activated either when  $\tilde{x}_{2N-2}(k) = \tilde{x}_{2N-1}(k) = 1$  or when  $\tilde{x}_{2N-1}(k) = \tilde{x}_{2N}(k) = 1$ , as illustrated in Fig. 6. Since the logic function  $f(a, b, c) = ab+bc$  is a threshold logic function, it follows that the sequencing signal can be implemented as a single node.

The output node of  $\mathcal{N}$  is then simply the function

$$\tilde{y}(k) = \sigma \left( \sum_{i=1}^{N+1} \tilde{z}_i(k) \right).$$

The interleaved zeros are implemented by default since no hidden layer nodes will be activated when the sequencing signal is in a position where the next value to be produced is an interleaved zero.

As in Theorem 1, the taps are initialized to values appropriate for simulating  $z_1(1)$  on the first time step. Q.E.D.

In [4], it was shown that any  $n$ -state FSM can be implemented by a four layer recurrent neural network<sup>4</sup> with  $O(\sqrt{n \log n})$  hard-limiting nodes. It is trivial to show that a fully connected recurrent neural network can simulate an  $L$ -layer recurrent network with a slowdown of  $L$ . Based on the fact that a NARX network with hard-limiting output nodes is only capable of implementing FMM's, we conclude that

<sup>4</sup>A multilayer recurrent network is like the network shown in Fig. 1, except that the single layer feedforward section is replaced by a MLP.

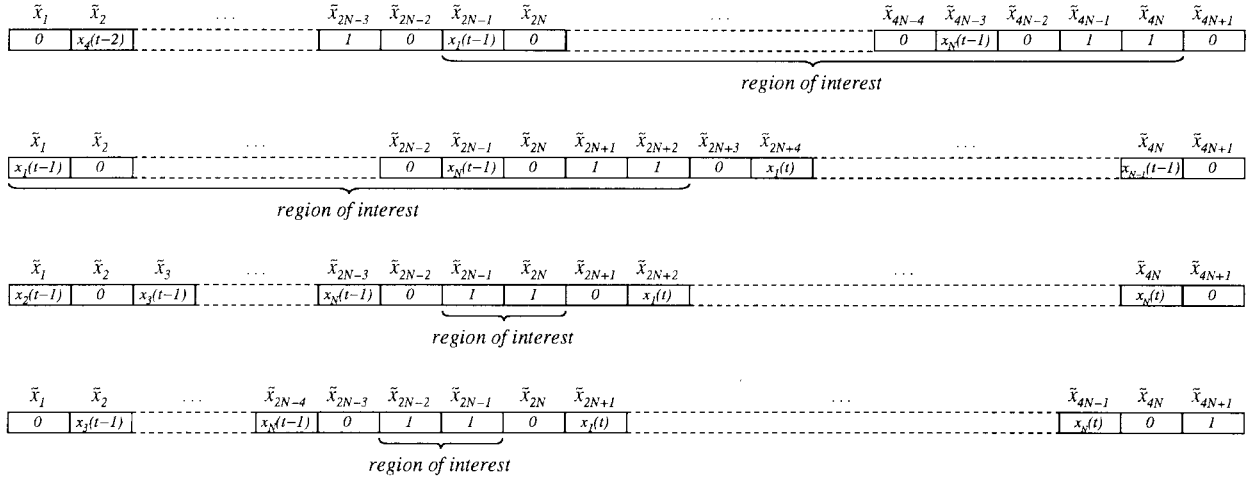


Fig. 6. The contents of the output tapped delay line of the NARX network at times  $(2N+3)t+1$  when  $x_1(t)$  is to be simulated next (top),  $(2N+3)t+2N-1$  when  $x_N(t)$  is to be simulated next (middle top),  $(2N+3)t+2N+1$  when the first timing signal is to be generated next (middle bottom), and  $(2N+3)t+2N+2$  when the second timing signal is to be generated next. After each time step, the contents of the taps move to the left, and the value of the output is stored in the last tap on the right.

*Corollary 2:* For every FSM  $\mathcal{M}$ , there exists an FMM which can simulate  $\mathcal{M}$  with  $O(\sqrt{n \log n})$  slowdown.

### B. Partially Affine Output Functions

Theorem 1 holds also when the output nonlinearity is *partially affine*. Denote an affine transformation by

$$A_{[a,b],[c,d]}(x) = \frac{(d-c)(x-a)}{b-a} + c$$

so that if  $x \in [a, b]$ , then  $A_{[a,b],[c,d]}(x) \in [c, d]$ . Then a nonlinearity is said to be partially affine if  $\sigma(x) = A_{[a,b],[c,d]}(x)$ , for  $x \in [a, b]$ . For example, the saturated linear function given in (5) is partially affine with  $a = c = 0$  and  $b = d = 1$ .

The modification of Theorem 1 is simply acquired by transforming the values of the hidden layer nodes, which are in  $[L, U+\epsilon] = [L, \mu]$  to the range  $[a, b]$ . These values are then passed through the partially affine region to produce values in the range  $[c, d]$ , which is fed back. This transformation can be undone by another affine transformation which converts values in  $[c, d]$  to  $[L, \mu]$ .

Specifically, the representation of the contents of the taps given in (7) is modified as follows:

$$\tilde{x}_j((N+1)t+i) = \begin{cases} A_{[L,\mu],[c,d]}(z_{i+j+1}(t-2)) & 0 < i+j < N \\ A_{[L,\mu],[c,d]}(\mu) & i+j = N \\ A_{[L,\mu],[c,d]}(z_{i+j-N}(t-1)) & N < i+j < 2N+1 \\ A_{[L,\mu],[c,d]}(\mu) & i+j = 2N+1 \\ A_{[L,\mu],[c,d]}(z_{i+j-2N-1}(t)) & 2N+1 < i+j \end{cases}$$

These values can be achieved by modifying the output node (11) to

$$\tilde{y}(k) = \sigma \left( -NA_{[L,\mu],[a,b]}(0) + \frac{\mu}{\sigma(\alpha)} A_{[L,\mu],[a,b]}(\tilde{z}_{N+1}(k)) + \sum_{i=1}^N A_{[L,\mu],[a,b]}(\tilde{z}_i(k)) \right).$$

Although only one hidden layer node is active, the affine transformation will, in general, convert zero node values to some nonzero value. The term  $NA_{[L,\mu],[a,b]}(0)$  compensates for this bias.

The hidden layer nodes are then modifications of (9)

$$\tilde{z}_i(k+1) = \sigma \left( \sum_{j=1}^N a_{i,j} A_{[c,d],[L,\mu]}(\tilde{x}_{N+j-i}(k)) + b_i u(k) + c_i + \beta_i [A_{[c,d],[L,\mu]}(\tilde{x}_{2N-i+1}(k)) - \mu] \right)$$

for  $i = 1 \dots N$ , and (10)

$$\tilde{z}_{N+1}(k) = \sigma(\beta_{N+1} [A_{[c,d],[L,\mu]}(\tilde{x}_N(k)) - \mu] + \alpha).$$

## V. CONCLUSION

Recent results suggest that gradient descent learning is more effective in NARX networks than in recurrent neural network architectures that have “hidden states” [18]. We have also shown that NARX networks perform better on problems involving *long-term dependencies* [20]. We have shown that NARX networks are capable of simulating fully connected networks within a linear slowdown, and as a result are universal dynamical systems. This theorem is somewhat surprising since the nature of feedback in this type of network is so limited, i.e., only output neuron feedback.

What does the Turing equivalence of neural networks imply? It implies that these networks are capable of representing solutions to just about any classical computational problem we want to apply them. Thus, we conclude that in theory one may use NARX networks in place of fully recurrent nets without loosing any computational power.

On the other hand, Turing equivalence implies that the space of possible solutions is extremely large. Thus, it may be prohibitively difficult to search with gradient descent learning algorithms. So far, experience indicates that it is difficult to

learn even small FSM's from example strings in either of these types of networks (unless the FSM has little logic in its implementation [25]). Often, a solution is found that classifies the training set perfectly, but the network in fact learns a chaotic system which cannot necessarily be equated with any finite state machine [26].

We also showed some related results that NARX networks with neurons with hard-limiting nonlinearities are only capable of implementing a subclass of finite state machines called finite memory machines. But, if a sublinear slowdown is allowed, then such networks can implement arbitrary finite state machines.

Our results open several questions for future research. What is the simplest feedback or recurrence necessary for any network to be Turing universal? What do these results imply about the computational power of recurrent networks with local recurrence [27]–[29]? And finally, can the efficiency of the simulation described in this paper be improved upon?

#### ACKNOWLEDGMENT

The authors would like to thank P. Tiño for many helpful comments.

#### REFERENCES

- [1] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.
- [2] M. Minsky, *Computation: Finite and Infinite Machines*. Englewood Cliffs, NJ: Prentice-Hall, 1967, ch. 3.
- [3] N. Alon, A. Dewdney, and T. Ott, "Efficient simulation of finite automata by neural nets," *J. Assoc. Comput. Mach.*, vol. 38, no. 2, pp. 495–514, 1991.
- [4] B. Horne and D. Hush, "Bounds on the complexity of recurrent neural network implementations of finite state machines," *Neural Networks*, vol. 9, no. 2, pp. 243–252, 1996.
- [5] H. Siegelmann, E. Sontag, and C. Giles, "The complexity of language recognition by neural networks," in *Algorithms, Software, Architecture (Proc. IFIP 12th World Computer Congr.)*, J. van Leeuwen, Ed. North-Holland: Amsterdam, 1992; pp. 329–335.
- [6] J. Kilian and H. Siegelmann, "On the power of sigmoid neural networks," in *Proc. 6th ACM Workshop on Computational Learning Theory*. New York: ACM, 1993, pp. 137–143.
- [7] H. Siegelmann and E. Sontag, "Turing computability with neural nets," *Appl. Math. Lett.*, vol. 4, no. 6, pp. 77–80, 1991.
- [8] ———, "On the computational power of neural networks," *J. Comput. Syst. Sci.*, vol. 50, no. 1, pp. 132–150, 1995.
- [9] ———, "Analog computation via neural networks," *Theor. Comput. Sci.*, vol. 131, pp. 331–360, 1994.
- [10] I. Leontaritis and S. Billings, "Input-output parametric models for nonlinear systems: Part I: deterministic nonlinear systems," *Int. J. Control*, vol. 41, no. 2, pp. 303–328, 1985.
- [11] S. Chen, S. Billings, and P. Grant, "Non-linear system identification using neural networks," *Int. J. Control*, vol. 51, no. 6, pp. 1191–1214, 1990.
- [12] K. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, Mar. 1990.
- [13] H.-T. Su and T. McAvoy, "Identification of chemical processes using recurrent networks," in *Proc. American Controls Conf.*, vol. 3, 1991, pp. 2314–2319.
- [14] H.-T. Su, T. McAvoy, and P. Werbos, "Long-term predictions of chemical processes using recurrent neural networks: A parallel training approach," *Industrial Engineering and Chemical Research*, vol. 31, pp. 1338–1352, 1992.
- [15] S. Venkataraman, "On encoding nonlinear oscillations in neural networks for locomotion," in *Proc. 8th Yale Workshop Adaptive and Learning Systems*, 1994, pp. 14–20.
- [16] S.-Z. Qin, H.-T. Su, and T. McAvoy, "Comparison of four neural net learning methods for dynamic system identification," *IEEE Trans. Neural Networks*, vol. 3, no. 1, pp. 122–130, 1992.
- [17] C. Giles and B. Horne, "Representation and learning in recurrent neural network architectures," in *Proc. 8th Yale Workshop Adaptive and Learning Systems*, 1994, pp. 128–134.
- [18] B. Horne and C. Giles, "An experimental comparison of recurrent neural networks," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. Touretzky, and T. Leen, Eds. Cambridge, MA: MIT Press, 1995, pp. 697–704.
- [19] Y. Benigo, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.
- [20] T. Lin, B. Horne, P. Tiño, and C. Giles, "Learning long-term dependencies is not as difficult with NARX recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 7, no. 6, 1996.
- [21] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [22] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, no. 3, pp. 183–192, 1989.
- [23] T. Booth, "Sequential machines and automata theory," New York, NY: Wiley, 1967.
- [24] Z. Kohavi, *Switching and Finite Automata Theory*, 2nd ed. New York: McGraw-Hill, 1978.
- [25] C. Giles, B. Horne, and T. Lin, "Learning a class of large finite state machines with a recurrent neural network," *Neural Networks*, vol. 8, no. 9, p. 1359, 1996.
- [26] J. Pollack, "The induction of dynamical recognizers," *Mach. Learn.*, vol. 7, no. 2/3, pp. 227–252, 1991.
- [27] A. Back and A. Tsoi, "FIR and IIR synapses, a new neural network architecture for time series modeling," *Neural Computation*, vol. 3, no. 3, pp. 375–385, 1991.
- [28] B. de Vries and J. Principe, "The gamma model—A new neural model for temporal processing," *Neural Networks*, vol. 5, pp. 565–576, 1992.
- [29] P. Frasconi, M. Gori, and G. Soda, "Local feedback multilayered networks," *Neural Computation*, vol. 4, pp. 120–130, 1992.
- [30] C. W. Omlin and C. L. Giles, "Constructing deterministic finite-state automata in recurrent neural networks," *J. Assoc. Comput. Mach.*, 1997.



**Hava T. Siegelmann** received the B.A. degree (summa cum laude) from the Technion, Haifa, Israel, in 1988, the M.Sc. degree from the Hebrew University, Jerusalem, Israel, in 1992, and the Ph.D. degree from Rutgers University, New Brunswick, NJ, (where she was on a Fellowship of Excellence) in 1993—all in computer science.

Currently, she is ALON Fellow and an Assistant Professor on the faculty of Industrial Engineering and Management at the Technion. She specializes in biologically motivated information processing systems—including neural networks and evolutionary algorithms—as well as in alternative models of computation (such as analog, distributed, and stochastic dynamics) that are relevant to natural systems. She has published in a variety of prestigious journals, including *Science*, *Theoretical Computer Science*, and the *Journal of Computer and Systems Science*, and has given numerous invited talks throughout her career.



**Bill G. Horne** received the B.S.E.E. degree from the University of Delaware, Newark, in 1986, and the M.S.E.E. and Ph.D. degrees in electrical engineering from the University of New Mexico, Albuquerque, in 1988 and 1993, respectively. The topic of his dissertation was the circuit complexity of neural networks.

Currently, he is Chief Technical Officer, Make-Waves, Inc., Watchung, NJ. His research interests involve neural networks, pattern recognition, and dynamical systems.



**C. Lee Giles** (S'80–M'80–SM'95) received the M.S. degree from the University of Michigan, Ann Arbor, and the Ph.D. degree in optical sciences from the University of Arizona, Tucson.

He is a Senior Research Scientist at NEC Research Institute, Princeton, NJ, and an Adjunct Associate Professor at the Institute for Advanced Computer Studies at the University of Maryland, College Park. Recently, he taught a graduate course on “Neural Networks” in the Electrical Engineering Department at Princeton University. His research interests are in neural networks and machine learning and their applications, and in optical computing and processing. He has published over 90 journal and conference papers and book chapters in these areas. Previously, he was a Program Manager at the Air Force Office of Scientific Research, Washington, DC, where he initiated and managed research programs in Neural Networks and in Optical Computing; a Research Scientist at the Naval Research Laboratory, Washington, D.C.; and an Assistant Professor of Electrical and Computer Engineering at Clarkson University, Potsdam, NY. Before graduate school, he was a Research Engineer at Ford Motor Scientific Research Laboratory.

Dr. Giles plays an active professional role in the neural networks and optical computing communities. He was one of the founding members of the Governors Board of the International Neural Network Society. He serves on many related conference program committees and has helped organize many related meetings and workshops. He has been an advisor and reviewer to government and university programs in both neural networks and in optical computing and processing. He has served or is currently serving on the editorial boards of IEEE TRANSACTIONS ON NEURAL NETWORKS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, *Journal of Parallel and Distributed Computing*, *Neural Networks*, *Neural Computation*, *Optical Computing and Processing*, *Applied Optics*, and Academic Press. In 1994, he co-edited a special issue for IEEE TRANSACTIONS ON NEURAL NETWORKS on “Dynamic Recurrent Neural Networks.” He is also a member of ACM, AAAI, INNS, and the OSA.