Computational capabilities of recurrent NARX neural networks

Hava T. Siegelmann
Department of Information Systems Engineering
Faculty of Industrial Engineering and Management
Technion (Israel Institute of Technology)
Haifa 32000, Israel
Email: iehava@ie.technion.ac.il

Bill G. Horne and C. Lee Giles*
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
Email: {horne,giles}@research.nj.nec.com

* Also with
UMIACS
University of Maryland
College Park, MD 20742

Technical Report
UMIACS–TR–95–12 and CS–TR–3408
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

Computational capabilities of recurrent NARX neural networks

Hava T. Siegelmann
Department of Information Systems Engineering
Faculty of Industrial Engineering and Management
Technion (Israel Institute of Technology)
Haifa 32000, Israel
Email: `iehava@ie.technion.ac.il`

Bill G. Horne and C. Lee Giles*
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
Email: {`horne,giles`}`@research.nj.nec.com`

* Also with
UMIACS
University of Maryland
College Park, MD 20742

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

$$y(t) = \Psi\left(u(t-n_u), \ldots, u(t-1), u(t), y(t-n_y), \ldots, y(t-1)\right),$$

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 Ψ 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.

1 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 (FSMs) [16]. Later Minsky showed that any FSM could be mapped into such a network [17]. More recently, new results have been developed to improve the efficiency of this mapping [1, 12, 24]. 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 [13, 21, 22, 23]. The proof utilizes a construction that shows how 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 [15]:

$$y(t) = f\left(u(t-n_u), \ldots, u(t-1), u(t), y(t-n_y), \ldots, y(t-1)\right), \tag{1}$$

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* [4, 18]. It has been demonstrated that this particular model is well suited for modeling nonlinear systems such as heat exchangers [4], waste water treatment plants [25, 26], catalytic reforming systems in a petroleum refinery [26], nonlinear oscillations associated with mutli–legged locomotion in biological systems [27], and various artificial nonlinear systems [4, 18, 20]. Furthermore, we have found that gradient descent learning is more effective in NARX networks than in recurrent neural network architectures that have "hidden states" on problems including grammatical inference and nonlinear system identification [9, 11]. Typically, the network converges much faster and generalizes better than other networks.

Based on the mapping theorems of [5, 8], NARX networks should be capable of representing arbitrary systems expressible in the form of equation (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.

We also provide some related results concerning NARX networks with non-linear output functions. In particular, when hard–limiting nonlinearities are used, we show that NARX networks are only capable of implementing a subclass of finite state machines (FSMs) called finite memory machines (FMMs). However, we show that FSMs can be simulated by FMMs within a sublinear slowdown.

2 Recurrent Neural Network Models

We consider two recurrent neural network models: fully–connected networks and NARX networks. We shall restrict our attention to single–input, single–output systems although our results can easily be extended to the multi–variable case.

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

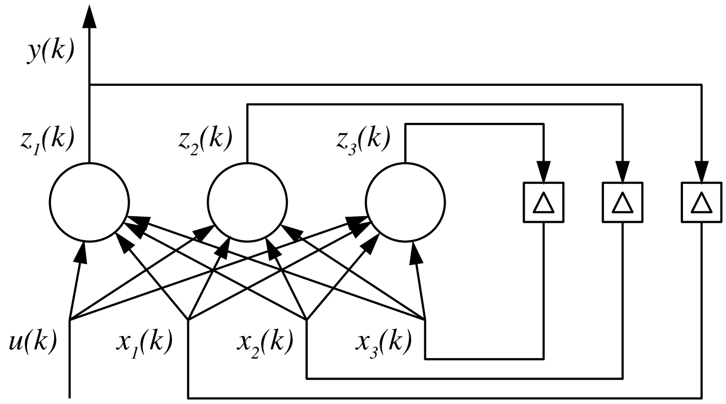

Figure 1: A fully connected recurrent neural network.

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 the the ith 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

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

where $a_{i,j}$, b_i, and c_i are fixed real valued weights, and σ 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 Figure 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

$$\tilde{y}(t) = \Psi\left(\tilde{u}(t-n_u), \ldots, \tilde{u}(t-1), \tilde{u}(t), \tilde{y}(t-n_y), \ldots, \tilde{y}(t-1)\right) \ , \tag{3}$$

where the function Ψ is the mapping performed by the MLP, as shown in Figure 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

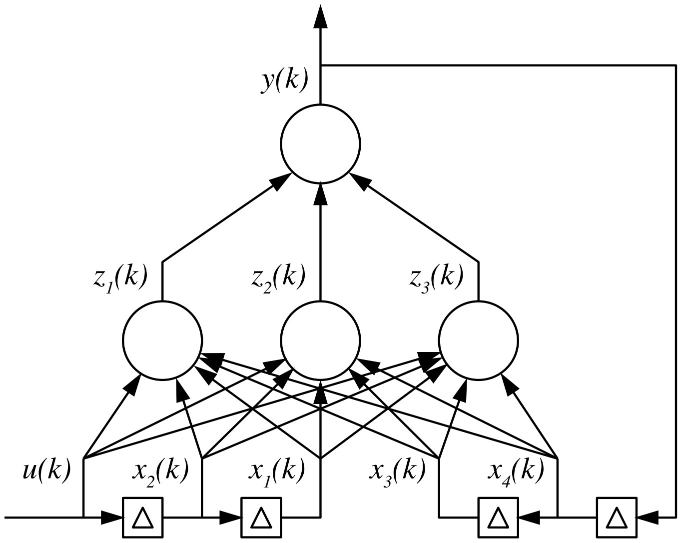

Figure 2: A NARX network with $n_u = n_y = 2$ and $\tilde{H} = 3$.

states are updated as

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

so that at time t the taps correspond to the values

$$\tilde{\boldsymbol{x}}(t) = \begin{bmatrix} \tilde{u}(t-n_u) & \dots & \tilde{u}(t-1) & \tilde{y}(t-n_y) & \dots & \tilde{y}(t-1) \end{bmatrix} .$$

The MLP consists of a set of nodes organized into two layers[2]. There are $\tilde{H}$ nodes in the first layer which perform the function

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

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 Figure 2.

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

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

ii. σ is left–side saturated[3], i.e. there exists a finite value s, such that $\sigma(x) = S$ for all $x \le s$.

[2] More layers could be used, but are not necessary for our purposes.

[3] Equivalently, the function can be defined to be saturated to the right, i.e. $\sigma(x) = S$ for all $x \ge s$, and we would obtain the same results.

iii. σ is non–constant.

$\square$

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

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

and the saturated linear function

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

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",

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

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

3 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. $\square$

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 σ. 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 σ is $S = 0$. This assumption makes the proof somewhat simpler, but can be easily generalized.

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 $k = (N + 1)t + i$, $i = 1, \ldots, N$. Because of the linear slowdown, the input to $\mathcal{N}$ must be kept constant for each simulation period, i.e.

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

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,

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

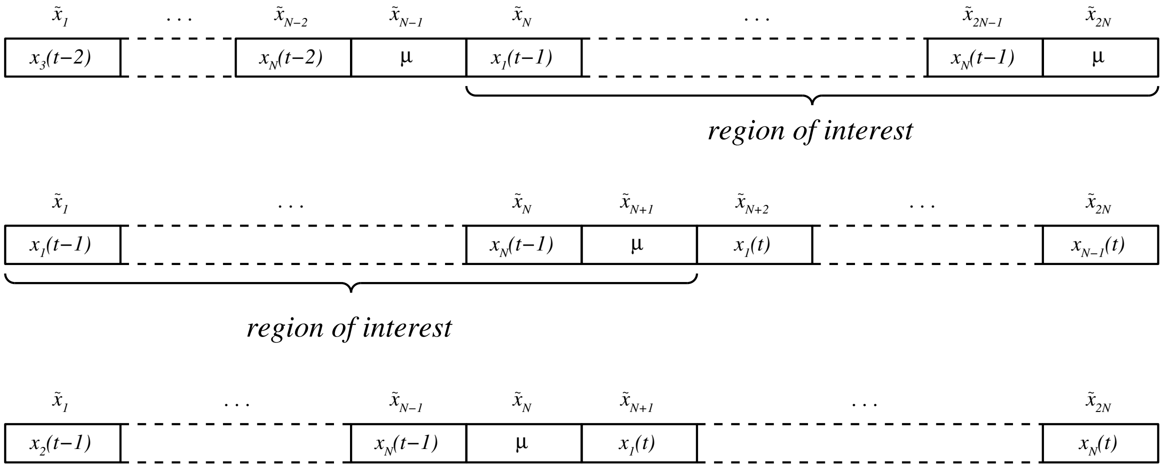

Figure 3: The contents of the output tapped delay line of the NARX network at times $(N+1)t+1$ when $x_1(t)$ is to be simulated next (top), $(N+1)t+N$ when $x_N(t)$ is to be simulated next (middle), and $(N+1)t+(N+1)$ when the sequencing signal μ is to be generated next (bottom). After each time step,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

for $1 \le i \le N+1$, 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 ($i = 1, \ldots, N+1,\ j = 1, \ldots, 2N$):

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

where μ (the sequencing signal) is outside the range $[L, U]$ (see Definition 1i); 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, μ, to indicate where these variables are in the tap. The contents of the taps at various times are illustrated in Figure 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_μ 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 ϵ. We define the affine function,

$$f_\mu(x) = x - \mu\ . \tag{8}$$

Then, $f_\mu(\mu) = 0$ and $f_\mu(x) \le -\epsilon$ for all $x \in [L, U]$. According to equations (6) and (7), node $\tilde{z}_i((N+1)t+j)$ may take on a non–zero value only when $i = j$, or equivalently when $\tilde{x}_{2N-i+1} = \mu$; in this case, the values of $z_j(t-1)$ are stored in the taps $\tilde{x}_{N+m-i}$, $m = 1, \ldots, N$. Thus, using equations (2) and (8), the ith node in the hidden layer of $\mathcal{N}$ is updated as follows.

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

where the constant β_i is large enough to make the input to σ less than s when $\tilde{x}_{2N-i+1}(k) \neq \mu$ so that the whole function is zero[4].

There exists at least one fixed value α 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 equation (8), its update equation can be written

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

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

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

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

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 equation (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.

It has been shown that fully connected networks with a fixed, finite number of saturated linear activation functions are universal computation devices [21, 22]. 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. □

4 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.

4.1 Hard–limiters

If the nonlinearity is a hard–limiting function (see equation (4)) and the inputs are binary, then recurrent neural networks are only capable of implementing Finite State Machines (FSMs), and NARX networks are only capable of implementing a subset of FSMs called *finite memory machines*

[4] We assume the value of the input is bounded.

(FMMs) [3, 14], which are defined to be an FSM whose input/output relationship can be described by the equation

$$y(t) = \phi\left(u(t-n_u), \ldots, u(t-1), u(t), y(t-n_y), \ldots, y(t-1)\right) ,$$

where $u(t)$ and $y(t)$ assume boolean values, and ϕ is a combinational logic function. Clearly this equation has the same form as equation (3), so when a hard–limiter is used for the nonlinearity of the output node, the function Ψ is a logic function, and it is clear that NARX networks are equivalent to FMMs.

Not all FSMs are FMMs. FMMs 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 Figure 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 Figure 5, is an FMM since for any input sequence of length two, the state of the FSMs can always be determined from knowledge of the past two inputs and the last output as illustrated in Table 1.

Intuitively, the reason why FMMs 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 FSMs 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 FSMs. 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. □

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 σ.

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. Figure 6 illustrates the tap contents at various times. The indexing scheme is similar to the one given in equation (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,

$$f(x_1, x_2) = x_1 + x_2 - 2 . \tag{12}$$

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

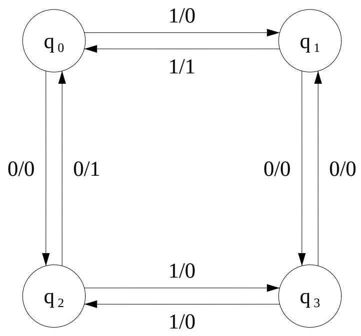

Figure 4: The Dual Parity FSM.

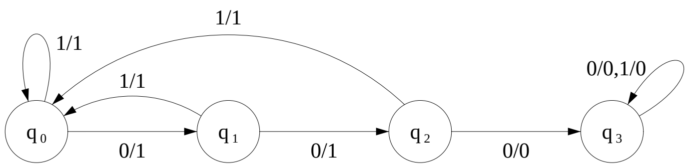

Figure 5: A finite memory machine.

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

Table 1: The state of the machine as a function of the previous two inputs and previous output.

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

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

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

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

where the constant β_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 Figure 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 [12] it was shown that any n–state FSM can be implemented by a four layer recurrent neural network[5] with $O\left(\sqrt{n \log n}\right)$ 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 FMMs, we conclude that

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

4.2 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 equation (5) is partially affine with $a = c = 0$ and $b = d = 1$.

[5] A multilayer recurrent network is like the network shown in Figure 1, except that the single layer feedforward section is replaced by a MLP.

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 equation (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 equation (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 equation (9)

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

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

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

5 Conclusion

Recent results suggest that gradient descent learning is more effective in NARX networks than in recurrent neural network architectures that have "hidden states" [11]. We have shown here that these 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 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 FSMs from example strings in either of these types of networks (unless the FSM has little logic in its implementation [10]). 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 [19].

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 two questions for a future research. First, what is the simplest feedback or recurrence necessary for any network to be Turing universal, and second, what do these results imply about the computational power of recurrent networks with local recurrence [2, 6, 7].

Acknowledgements

We would like to thank Peter Tiňo and Hanna Siegelmann for many helpful comments.

References

[1] N. Alon, A.K. Dewdney, and T.J. Ott. Efficient simulation of finite automata by neural nets. *Journal of the Association of Computing Machinery*, 38(2):495–514, 1991.

[2] A.D. Back and A.C. Tsoi. FIR and IIR synapses, a new neural network architecture for time series modeling. *Neural Computation*, 3(3):375–385, 1991.

[3] T.L. Booth. *Sequential machines and automata theory.* Wiley, New York, NY, 1967.

[4] S. Chen, S.A. Billings, and P.M. Grant. Non–linear system identification using neural networks. *International Journal of Control*, 51(6):1191–1214, 1990.

[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[6] B. de Vries and J.C. Principe. The gamma model — A new neural model for temporal processing. *Neural Networks*, 5:565–576, 1992.

[7] P. Frasconi, M. Gori, and G. Soda. Local feedback multilayered networks. *Neural Computation*, 4:120–130, 1992.

[8] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.

[9] C.L. Giles and B.G. Horne. Representation and learning in recurrent neural network architectures. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 128–134, 1994.

[10] C.L. Giles, B.G. Horne, and T. Lin. Learning a class of large finite state machines with a recurrent neural network. Technical Report UMIACS–TR–94–94 and CS–TR–3328, Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland, 1994.

[11] B.G. Horne and C.L. Giles. An experimental comparison of recurrent neural networks. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*. MIT Press, 1995. To appear.

[12] B.G. Horne and D.R. Hush. Bounds on the complexity of recurrent neural network implementations of finite state machines. In J.D. Cowen, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 359–366. Morgan Kaufmann, 1994.

[13] J. Kilian and H.T. Siegelmann. On the power of sigmoid neural networks. In *Proceedings of the Sixth ACM Workshop on Computational Learning Theory*, pages 137–143. ACM Press, 1993.

[14] Z. Kohavi. *Switching and finite automata theory.* McGraw–Hill, New York, NY, 2nd edition, 1978.

[15] I.J. Leontaritis and S.A. Billings. Input–output parametric models for non–linear systems: Part I: deterministic non–linear systems. *International Journal of Control*, 41(2):303–328, 1985.

[16] W.S. McCulloch and W.H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[17] M.L. Minsky. *Computation: Finite and infinite machines.* Prentice–Hall, Englewood Cliffs, 1967. Chapter: Neural Networks. Automata made up of parts.

[18] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1:4–27, March 1990.

[19] J.B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7(2/3):227–252, 1991.

[20] S.-Z. Qin, H.-T. Su, and T.J. McAvoy. Comparison of four neural net learning methods for dynamic system identification. *IEEE Transactions on Neural Networks*, 3(1):122–130, 1992.

[21] H.T. Siegelmann and E.D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[22] H.T. Siegelmann and E.D. Sontag. On the computational power of neural networks. In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory*, pages 440–449. ACM Press, 1992. A full version will appear in the Journal of Computer and System Science.

[23] H.T. Siegelmann and E.D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131:331–360, 1994. Previous version appeared in Proceedings of the Second Israel Symposium on Theory of Computing and Systems, Natanya Israel, June, 1993. pp. 98-107.

[24] H.T. Siegelmann, E.D. Sontag, and C.L. Giles. The complexity of language recognition by neural networks. In J. van Leeuwen, editor, *Algorithms, Software, Architecture (Proceedings of IFIP 12th World Computer Congress)*, pages 329–335, Amsterdam, 1992. North–Holland.

[25] H.-T. Su and T.J. McAvoy. Identification of chemical processes using recurrent networks. In *Proceedings of the American Controls Conference*, volume 3, pages 2314–2319, 1991.

[26] H.-T. Su, T.J. McAvoy, and P. Werbos. Long–term predictions of chemical processes using recurrent neural networks: A parallel training approach. *Industrial Engineering and Chemical Research*, 31:1338–1352, 1992.

[27] S.T. Venkataraman. On encoding nonlinear oscillations in neural networks for locomotion. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 14–20, 1994.

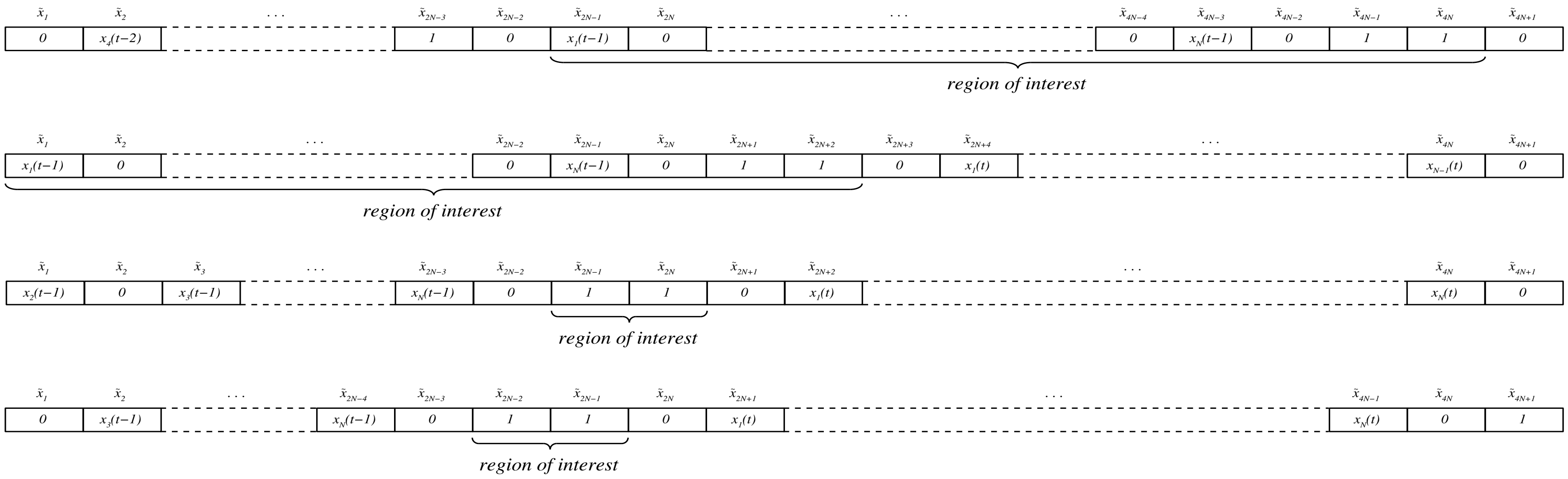

Figure 6: The contents of the output tapped delay line of the NARX network at times $(2N+3)t+1$ when $x_1(t)$ is to be simulated next (top), $(2N+3)t+2N-1$ when $x_N(t)$ is to be simulated next (middle top), $(2N+3)t+2N+1$ when the first timing signal is to be generated next (middle bottom), and $(2N+3)t+2N+2$ 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.