Bounds on the complexity of recurrent neural network implementations of finite state machines

Bill G. Horne[†] and Don R. Hush
Dept. of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM 87131

[†] Currently at:
NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Address for correspondance:

Bill G. Horne
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
(609) 951–2676
horne@research.nj.nec.com

Running title: "Recurrent neural networks and finite state machines"

Bounds on the complexity of recurrent neural network implementations of finite state machines

Abstract

In this paper the efficiency of recurrent neural network implementations of m–state finite state machines will be explored. Specifically, it will be shown that the node complexity for the unrestricted case can be bounded above by $O\left(\sqrt{m}\right)$. It will also be shown that the node complexity is $O\left(\sqrt{m \log m}\right)$ when the weights and thresholds are restricted to the set $\{-1, 1\}$, and $O\left(m\right)$ when the fan–in is restricted to two. Matching lower bounds will be provided for each of these upper bounds assuming that the state of the FSM can be encoded in a subset of the nodes of size $\lceil \log m \rceil$.

Keywords

Automata theory, circuit complexity, finite state machines, logic synthesis, neural networks, recurrent neural networks.

Nomenclature

Q	a finite set of states
Σ	the input alphabet
Δ	the output alphabet
δ	a transition function
λ	an output function
q_0	the initial state
$\bar{\phi}$	a function which is a composite of δ and λ
ϕ	an implementation of $\bar{\phi}$ as a boolean logic function
$m = \|Q\|$	number of states in an FSM
$N_{\mathcal{M}}$	the number of minimal finite state machines
x	the number of inputs of a static logic function
y	the number of outputs of a static logic function
L_n	the number of threshold logic functions
ν	number of vertices used by a partially specified logic function
L_n^ν	the number of partially specified threshold logic functions
k	number of nodes of the recurrent neural network
$\|W\|$	the maximum size of a set of allowable integer values weights
layer (i)	the layer containing the i^{th} node
Λ_i	the set of nodes connected to the inputs of node i.
log	logarithm base 2
ln	natural logarithm
e	the base of the natural logarithm
$O(\cdot)$	asymptotic upper bound
$\Omega(\cdot)$	asymptotic lower bound
$\Theta(\cdot)$	asymptotic tight bound

1 Introduction

Feedforward networks, such as the Multilayer Perceptron, have had a great deal of success, particularly for static problems such as classification. However, many problems of interest in signal processing, control, and computer science are inherently dynamic and thus are not appropriate problem domains for these static systems. In the past few years there has been considerable interest in recurrent neural networks, which are capable of modeling nonlinear dynamic systems. Of fundamental importance, and the topic of this paper, is understanding how efficiently these networks scale to large problems. Although there are many ways to measure efficiency, we shall be concerned with the *node complexity*, which, as its name implies, is a calculation of the required number of nodes. Node complexity is a useful measure of efficiency since the amount of resources required to implement or even simulate a recurrent neural network is typically related to the number of nodes in the network. Node complexity can also be related to the efficiency of learning algorithms for these networks and perhaps to their generalization ability.

In this paper, we explore the efficiency of recurrent neural network implementations of finite state machines (FSMs) when the nodes of the network use hard–limiting nonlinearities. Our analysis of node complexity is *worst case* since we will be interested in the ability of recurrent neural networks to implement arbitrary FSMs, and as a result, includes those FSMs that are most difficult for recurrent neural networks to implement. Although FSMs represent a limited class of systems, they are an important one. They are found in millions of products from simple traffic light controllers to the most sophisticated microprocessors. In fact, any computational device with a finite amount of memory can be viewed as an FSM. They are also an important theoretical tool that play a central role in both formal language theory and computability theory.

The relationship between recurrent neural networks and FSMs has been studied since the earliest days of neural networks (McCulloch and Pitts, 1943). However, it wasn't until the 1960s that it was shown that recurrent neural networks are capable of implementing arbitrary FSMs. The first result in this area was due to Minsky (Minsky, 1967), who showed that m–state FSMs can be implemented in a fully connected recurrent neural network. Although circuit complexity was not the focus of his investigation it turns out that his construction, which mapped the transitions of the FSM to the nodes in the network, yields $O(m)$ nodes. This construction was guaranteed to use weight values limited to the set $\{1, 2\}$.

Since a recurrent neural network with k hard–limiting nodes is capable of representing as many

as 2^k states, one might wonder if an m–state FSM could be implemented by a network with $\log m$ nodes. However, it was shown in (Alon et al., 1991) that the node complexity for a standard fully connected network, such as a Hopfield network (Hopfield, 1982), is $\Omega\left((m \log m)^{1/3}\right)$. They were also able to improve upon Minsky's result by providing a construction which is guaranteed to yield no more than $O\left(m^{3/4}\right)$ nodes. In this paper we improve this upper bound to $O(\sqrt{m})$.

In the same paper lower bounds on node complexity were investigated as the network was subject to restrictions on the possible range of weight values and the fan–in and fan–out of the nodes in the network. They limited their investigation to fully connected recurrent neural networks and discovered that the node complexity for the case where the weights are restricted to a finite size set is $\Omega\left(\sqrt{m \log m}\right)$. Alternatively, if the nodes in the network were restricted to have a constant fan–in then the node complexity becomes $\Omega(m)$. However, they left open the question of how tight these bounds are and if they apply to variations on the basic architecture.

In this paper, we provide upper bounds that match the lower bounds above. Specifically, we show that a node complexity of $O\left(\sqrt{m \log m}\right)$ can be achieved if the weights are restricted to the set $\{-1, 1\}$, and that the node complexity is $O(m)$ for the case when the fan–in of each node in the network is restricted to two.

Finally, we explore the possibility that implementing finite state machines in more complex models might yield a lower node complexity. Specifically, we explore the node complexity of a general recurrent neural network topology, that is capable of simulating a variety of popular recurrent neural network architectures. Except for the unrestricted case, we will show that the node complexity is no different for this architecture than for the fully connected case if the number of feedback variables is limited to $\lceil \log m \rceil$, i.e. if the state of the FSM is encoded optimally in a subset of the nodes. We leave it as an open question if a sparser encoding can lead to more efficient implementations.

All of the results reported by Minsky, Alon *et al.*, and ourselves involve networks with thresholding nonlinearities. Recently, some interesting work has been done to determine the node complexity for networks with either linear, sigmoidal or saturated activation functions. In (Siegelmann et al., 1992), a methodology was developed for estimating the node complexity required to recognize a regular language using a second order recurrent neural network with various types of activation functions. It has also been shown that there exist finite size recurrent neural networks that are Turing equivalent if the nonlinearity used is a saturated linear function and the weights are restricted to a small set of rational numbers (Siegelmann and Sontag, 1991; Siegelmann and Sontag, 1992). This work has recently been extended to a large class of "sigmoid–like" networks (Kilian and

Siegelmann, 1993). These results are significant since they show that basic convergence questions concerning these architectures is undecidable even for fixed–size networks. It has also been shown that super–Turing capabilities can be achieved in recurrent neural networks with continuous–valued nonlinearities if there are no restrictions on the weights (Siegelmann and Sontag, 1994).

2 Background

2.1 Finite State Machines

FSMs may be defined in several ways. In this paper we shall be concerned with Mealy machines, although our approach can easily be extended to other formulations, such a Moore machines, to yield equivalent results.

Definition 1 A *Mealy machine* is a sextuple $\mathcal{M} = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

Q is a finite set of *states*;

Σ is a finite set of symbols called the *input alphabet*;

Δ is a finite set of symbols called the *output alphabet*;

$\delta : Q \times \Sigma \longrightarrow Q$ is a *transition function*;

$\lambda : Q \times \Sigma \longrightarrow \Delta$ is an *output function*;

q_0 is the initial state.

□

Throughout this paper both the input and output alphabets will be binary (i.e. $\Sigma = \Delta = \{0, 1\}$). In general, the number of states, $m = |Q|$, may be arbitrary.

It will be convenient to combine the transition and output functions into a single function $\bar{\phi}$ of the form $\bar{\phi} : Q \times \Sigma \longrightarrow Q \times \Delta$. Since any element of Q can be encoded as a binary vector whose minimum length is $\lceil \log m \rceil$, the function $\bar{\phi}$ can be implemented by a boolean logic function of the form

$$\phi : \{0,1\}^{\lceil \log m \rceil + 1} \longrightarrow \{0,1\}^{\lceil \log m \rceil + 1}.$$

As a result, FSMs can be implemented as a block of combinatorial logic, which implements the function ϕ, coupled with a finite memory, which stores the state of the FSM.

2.1.1 Minimal Finite State Machines

For any given $\mathcal{M}$, there exist an infinite collection of FSMs that are equivalent to $\mathcal{M}$ that will exhibit the same input/output behavior. An FSM that has the minimum number of states for a given input/output behavior is said to be *minimal*. Note that the minimal FSM for a given behavior is unique up to a relabelling of the states (Hopcroft and Ullman, 1979).

The number, $N_{\mathcal{M}}$, of different minimal FSMs with m states will be used to determine lower bounds on the number of gates required to implement an arbitrary FSM in a recurrent neural network. It has been shown that (Alon et al., 1991)

$$\frac{(2m)^m\,(2^m-2)}{m} \leq N_{\mathcal{M}}.$$

However, it will be easier to work with the simpler bound

$$(2m)^m \leq N_{\mathcal{M}}, \tag{1}$$

which can be used without affecting the final results.

2.2 Recurrent Neural Networks

As pointed out in (Nerrand et al., 1993), many of the most popular discrete–time recurrent neural network models can be implemented as a feedforward network, such as a multilayer perceptron (MLP), whose outputs are fed back recurrently through a set of unit time delays, as illustrated in Figure 1. The most direct implementation of this model is the architecture proposed in (Robinson and Fallside, 1988). However, many recurrent neural network architectures can be cast into this framework. For example, fully connected networks (Hopfield, 1982; Williams and Zipser, 1989) fit this model when the the feedforward network is simply a single layer of nodes. Even models which appear very different (Back and Tsoi, 1991; Elman, 1990; Jordan and Rumelhart, 1992; Narendra and Parthasarathy, 1990) can be cast into this framework. Our results will not be applicable to networks with high order terms (Giles et al., 1990; Watrous and Kuhn, 1992).

All of the bounds described in this paper will be derived for multilayer architectures. However, an L–layer network can always be implemented as a single–layer, fully–connected model, such as the Hopfield model, if one is willing to allow an L time step *slowdown* for each single time step of the system the network is simulating (Alon et al., 1991; Siegelmann and Sontag, 1991). The basic idea

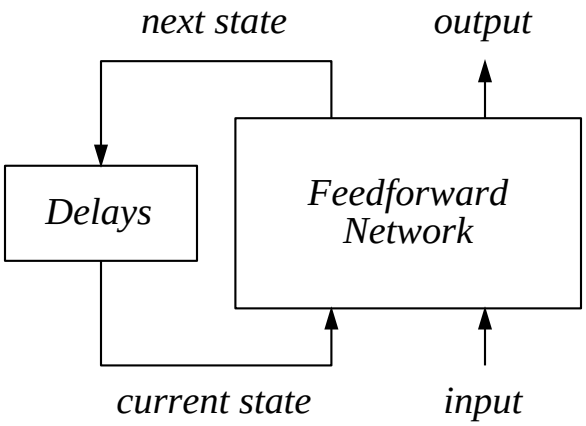

Figure 1: A multilayer recurrent neural network.

involves adding a set of nodes which function as a "counter" for the network, so that the timing of the output and state transitions are implemented appropriately.

The fundamental processing unit in the neural network models we wish to consider is the *perceptron*, which is a biased, linearly weighted sum of its inputs followed by a hard–limiting nonlinearity whose output is zero if its input is negative and one otherwise. The *fan–in* of the perceptron is defined to be the number of non–zero weights. When the values of inputs to the nodes are binary (as they are in this paper), the perceptron is often referred to as a *threshold logic unit* (TLU).

A count of the number of different threshold logic functions will be needed to develop lower bounds on the node complexity required to implement an arbitrary logic function. Although there are a total of 2^{2^n} logic functions of n variables, they are not all linearly separable, and hence not implementable by a TLU. In fact, linearly separable logic functions are only a vanishingly small percentage of all logic functions for large n. The exact number of threshold logic functions of n–variables, denoted L_n, has only been determined for $n \leq 8$ (Muroga, 1971). For $n > 8$ only the following bounds are known (Muroga, 1971; Winder, 1970):

$$4 \cdot 2^{(n^2-n)/2} \leq L_n \leq \frac{2^{n^2+1}}{n!}. \tag{2}$$

These bounds have been generalized to *partially specified logic functions*, which are logic functions whose values are only defined over ν vertices of the unit hypercube. It has been shown that the

number of partially specified threshold logic functions, denoted L_n^ν, is (Winder, 1963; Winder, 1970)[1]

$$4 \cdot \nu^{(\log \nu - 1)/2} \leq L_n^\nu \leq \frac{2\nu^n}{n!}. \tag{3}$$

where the right hand side is valid only if $\nu \geq 3n + 2$. For $\nu = 2^n$, this expression simplifies to the bounds in equation (2).

Individual perceptrons are rather limited, but when perceptrons are connected together in a feedforward network, such as an MLP, they can implement many complex functions including arbitrary logic functions. In an MLP, each node belongs to a *layer* which is defined recursively as follows. If node i has weights connecting to the outputs of nodes $j \in \Lambda_i$ then

$$layer\,(i) = \max_{j \in \Lambda_i} layer\,(j) + 1.$$

Thus, nodes cannot receive inputs from nodes in the same layer (lateral connections) or from nodes in higher layers (feedback connections). The inputs to the network are considered to be in layer zero.

There are several ways to choose which nodes will be the outputs of the network. Clearly, the nodes in the last layer must be used as the outputs of the network, otherwise there would be no point in implementing them. However, nodes at intermediate layers may also be used as outputs.

The most common feedforward architecture is one in which nodes in layer l have weights that connect to the outputs of nodes in layer $l - 1$ only. In contrast, a network is said to be *lower triangular* if the l^{th} node is the only node in layer l and receives input from all nodes in previous layers (including the input layer). Figure 2 illustrates a lower triangular network with four nodes. A lower triangular network of k threshold logic elements is the most general topology possible for a feedforward network since all other feedforward networks can be viewed as a special case of this network with the appropriate weights set equal to zero. This is true for both single output and multi–output networks.

In (Alon et al., 1991), lower bounds on node complexity were explored for single–layer, fully–connected architectures. In this paper, we explore lower bounds for lower triangular architectures. This allows us to accomplish two things: first, we can explore the possibility that implementing finite state machines in more complex recurrent neural network models might yield a lower node

[1] Here and throughout the rest of the paper all logarithms are base 2 except for the natural logarithm which will be denoted "ln".

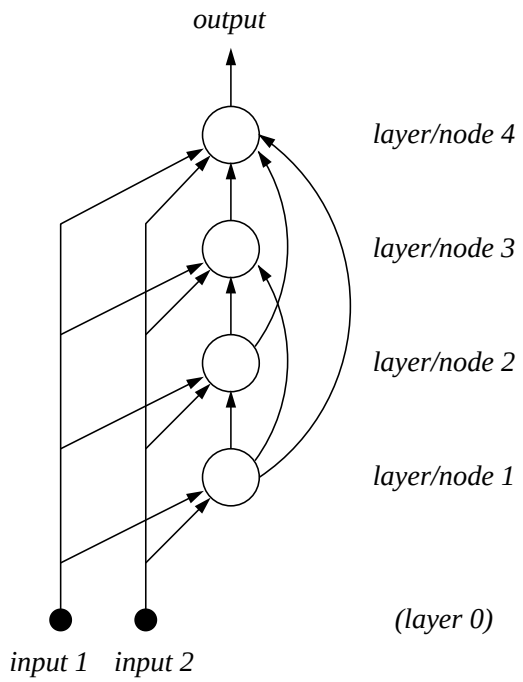

Figure 2: A four node lower triangular network.

complexity; second, we can account for the worst–case slowdown, since simulating a lower triangular network with a fully–connected network involves a slowdown that is equal to the number of nodes in the network.

3 The unrestricted case

In Sections 4 and 5 we shall consider imposing restrictions on the range of weights and the fan–in of the nodes in the network. In this section, we examine the most general case in which no such restrictions are imposed. This case explores the inherent power of recurrent neural networks, and is also important because it serves as a baseline from which one can evaluate the significance of imposing various restrictions on the node complexity.

3.1 Upper Bound

In order to derive an upper bound on the node complexity of recurrent neural network implementations of FSMs we shall utilize the following lemma, due to Lupanov (Lupanov, 1972; Lupanov, 1973).

Lemma 1 (Lupanov, 1972) Arbitrary boolean logic functions of the form $f : \{0,1\}^x \longrightarrow \{0,1\}^y$ in which $m = o(2^x)$, can be implemented in a four layer network of perceptrons with a node complexity

of $O\left(\sqrt{\frac{y2^x}{x-\log y}}\right)$. □

The proof of the following theorem is simply an application of Lemma 1.

Theorem 1 A four layer recurrent neural network can implement FSMs having m states with a node complexity of $O(\sqrt{m})$. □

Proof: An m–state FSMs can be implemented in a recurrent neural network in which the multilayer network performs a mapping of the form,

$$\phi : \{0,1\}^{\lceil \log m \rceil + 1} \longrightarrow \{0,1\}^{\lceil \log m \rceil + 1} .$$

Thus, the problem reduces to determining the node complexity of implementing such a logic function. Using $x = y = \lceil \log m \rceil + 1$, so that

$$\log m + 1 \leq x = y \leq \log m + 2 ,$$

then applying Lemma 1 gives a four layer network with an upper bound on the node complexity of

$$O\left(\sqrt{\frac{(\log m + 2)2^{\log m + 2}}{\log m + 1 - \log(\log m + 2)}}\right) = O\left(\sqrt{\frac{(\log m)2^{\log m}}{\log m}}\right) = O(\sqrt{m}) .$$

Q.E.D.

Corollary 1 A fully connected recurrent neural network can implement FSMs having m states with a node complexity of $O(\sqrt{m})$ and a four step slowdown. □

3.2 Lower Bound

Theorem 2 Multilayer recurrent neural networks can implement FSMs having m states with a node complexity of $\Omega(\sqrt{m})$ if the number of unit time delays is $\lceil \log m \rceil$. □

Proof: In order to prove the theorem we derive an expression for the maximum number of functions that a k–node recurrent neural network can compute and compare that against the minimum number of finite state machines. Then we solve for k in terms of the number of states of the FSM.

The total number of possible distinct configurations of a lower triangular feedforward network with k nodes is bounded by the maximum number of logic functions that the first node can compute, multiplied by the maximum number of logic functions that the second node can compute and so on.

For convenience let n equal the number of variables in the 0^{th} layer, which is given by the size of the state vector plus an additional variable corresponding to the input, i.e. let

$$n = \lceil \log m \rceil + 1. \tag{4}$$

Then, the number of inputs to the i^{th} node (numbered from 0 to $k-1$) in a lower triangular network is $n+i$. Thus, by equation (3), this node can compute at most $\frac{2\nu^{(n+i)}}{(n+i)!}$ logic functions. Consequently, the maximum number of configurations of a lower triangular network with k nodes is given by

$$\prod_{i=0}^{k-1} \frac{2\nu^{(n+i)}}{(n+i)!}. \tag{5}$$

Here, ν is at most 2^n since this is the maximum number of different inputs to the network and, as a result, is the maximum number of different inputs that nodes at higher layers may encounter even though their fan–in is greater than n. As discussed in Section 2.2, this equation only holds if $\nu \geq 3(n+i)+2$, so for $\nu = 2^n$, $k \leq 2^n/3$ which will be consistent with our result.

The total number of logic functions that such a network can compute depends on which of the nodes are chosen as the outputs of the network. Recall that if the network is to implement the mapping ϕ corresponding to an FSM, then the total number of outputs and state variables will be equivalent to the value of n in equation (4). Even for some fixed assignment of weights, various nodes in the network could be used as these outputs, and each could potentially compute a different logic function. Clearly, the last node must be an output or state variable, otherwise there would be no point in using it. Of the remaining $k-1$ nodes, $n-1$ must be chosen as outputs or state variables. (It will soon become clear $k \gg n$, so that there are more than $n-1$ nodes to choose from.) There are exactly $\binom{k-1}{n-1}$ such choices. Once the n output nodes have been chosen, it must be determined which node computes the first output, which computes the second output, and so on. There are exactly $n!$ such assignments. Therefore, the maximum number of logic functions that a lower triangular network can compute is

$$n!\binom{k-1}{n-1}\prod_{i=0}^{k-1}\frac{2^{n(n+i)+1}}{(n+i)!} = \frac{n(k-1)!}{(k-n)!}\prod_{i=0}^{k-1}\frac{2^{n(n+i)+1}}{(n+i)!}.$$

If the network is to implement all possible logic functions of the form $\phi : \{0,1\}^n \longrightarrow \{0,1\}^n$, then this number of configurations must necessarily be greater than the minimum number of distinctly different FSMs, which according to equation (1) is $(2m)^m$. Since $n \leq \log m + 2$, or equivalently,

$m \geq 2^{n-2}$, the minimal number of finite state machines can be expressed in terms of n as,

$$2^{(n-1)2^{n-2}} \leq N_{\mathcal{M}}.$$

Thus, the value of k we seek may be found by manipulating the inequality

$$2^{(n-1)2^{n-2}} \leq \frac{n(k-1)!}{(k-n)!} \prod_{i=0}^{k-1} \frac{2^{n(n+i)+1}}{(n+i)!}.$$

Taking the logarithm of both sides and using the inequality in equation (A.1) of the appendix yields

$$\begin{aligned} (n-1)2^{n-2} &\leq \log n + k \log k - \frac{k-1}{\ln 2} - \log[(k-n)!] \\ &+ \sum_{i=0}^{k-1} \left[1 + n(n+i) - (n+i)\log(n+i) + \frac{n+i-1}{\ln 2} \right]. \end{aligned}$$

Collecting terms within the summation, simplifying, and eliminating some negative terms (assuming $k > n$) gives

$$(n-1)2^{n-2} \leq \log n + k \log k + k \left(1 + n^2 + \frac{n-1}{\ln 2} \right) + \frac{k(k-1)}{2} \left(n + \frac{1}{\ln 2} \right). \tag{6}$$

Assuming $k \geq n \geq 2$, then it can easily be shown that there exists a constant, c, such that

$$n2^n \leq ck^2 n.$$

So that

$$k = \Omega\left(\sqrt{2^n}\right).$$

Since $n = \lceil \log m \rceil + 1 > \log m$

$$k = \Omega\left(\sqrt{m}\right).$$

Q.E.D.

4 Restriction on weights and thresholds

All threshold logic functions can be implemented with perceptrons whose weight and threshold values are integers. It is well known that there are threshold logic functions of n variables that require a perceptron with weights whose maximum magnitude is $\Omega(2^n)$ and $O(n^{n/2})$ (Muroga, 1971). This implies that if a perceptron is to be implemented digitally, the number of bits required to represent each weight and threshold, in the worst case, will be a superlinear function of the fan–in. This is generally undesirable; it would be far better to require only a logarithmic number of bits per weight, or even better, a constant number of bits per weight.

There are two kinds of restrictions on the weights and thresholds of interest. First, it is desirable to limit both the weights and thresholds to integers of bounded magnitude. Preferably, these values should be limited so that a representation can be used that requires only a constant number of bits. For example, if they are limited to values from the set $\{-1, 1\}$, then only one bit is required for each weight. A weaker restriction is to limit the weights and thresholds to take on values from a finite size set but to leave the maximum magnitude of any weight unconstrained. For example, it might be desirable for some reason to limit them to the set $\{1, 2, 4, \ldots, 2^{32}\}$. Of course, the first limitation is a special case of the second.

4.1 Upper Bound

In order to derive the node complexity of recurrent neural network implementation of FSMs, we shall utilize the following lemma, proved recently in (Horne and Hush, 1994). The proof uses the same ideas used to bound the node complexity for alternating AND–OR circuits (Parberry, 1994; Red'kin, 1972).

Lemma 2 Arbitrary boolean logic functions of the form $f : \{0,1\}^x \longrightarrow \{0,1\}^y$ can be implemented in a four layer network of perceptrons whose weights and thresholds are restricted to the set $\{-1, 1\}$ with a node complexity of

$$\Theta\left(\sqrt{y2^x}\right).$$

□

Theorem 3 A four layer recurrent neural network that has nodes whose weights and thresholds are restricted to the set $\{-1, 1\}$ can implement an FSM having m states with a node complexity of

$O\left(\sqrt{m \log m}\right)$. □

Proof: Since m–state FSMs can be implemented in a recurrent neural network in which the multilayer network performs a mapping of the form,

$$\phi : \{0,1\}^{\lceil \log m \rceil + 1} \longrightarrow \{0,1\}^{\lceil \log m \rceil + 1} .$$

Thus, the problem reduces to determining the node complexity of implementing such a logic function. Using $x = y = \lceil \log m \rceil + 1$, so that

$$x = y \leq \log m + 2 ,$$

then applying Lemma 2 gives a four layer network with an upper bound on the node complexity of

$$O\left(\sqrt{(\log m + 2) 2^{\log m + 2}}\right) = O\left(\sqrt{m \log m}\right) .$$

Q.E.D.

Corollary 2 A fully connected recurrent neural network that has nodes whose weights and thresholds are restricted to the set $\{-1, 1\}$ can implement an FSM having m states with a node complexity of $O\left(\sqrt{m \log m}\right)$ and a four step slowdown. □

4.2 Lower Bound

Theorem 4 Multilayer recurrent neural networks that have nodes whose weights and thresholds are restricted to a set of size $|W|$ can implement FSMs having m states with a node complexity of $\Omega\left(\sqrt{\frac{m \log m}{\log |W|}}\right)$ if the number of unit time delays is $\lceil \log m \rceil$. □

Proof: The proof is similar to the proof of Theorem 2 which gave a lower bound for the node complexity required in an arbitrary network of threshold logic units. Here, the maximum number of functions that the i^{th} node can compute is bounded above by $|W|^{n+i+1}$. Thus, the maximum number of possible distinct configurations that a lower triangular feedforward network of TLUs whose weights and thresholds are restricted to a set of size $|W|$ is at most

$$\prod_{i=0}^{k-1} |W|^{n+i+1}.$$

By counting the number of ways to choose nodes for the output of the network, the maximum number of logic functions implementable by a such a network is

$$\frac{n(k-1)!}{(k-n)!}\prod_{i=0}^{k-1}|W|^{n+i+1}.$$

For such a network to implement an arbitrary FSM, the number of configurations must be greater than the minimum number of distinctly different FSMs. Thus the value of k we seek must satisfy

$$2^{(n-1)2^{n-2}} \leq \frac{n(k-1)!}{(k-n)!}\prod_{i=0}^{k-1}|W|^{n+i+1}.$$

Taking the logarithm of both sides gives

$$(n-1)2^{n-2} \leq \log n + \log(k-1)! - \log(k-n)! + \sum_{i=0}^{k-1}(n+i+1)\log|W|.$$

Eliminating negative terms (assuming $k > n$), using the bounds in equation (A.1) to simplify the right hand side, and simplifying the summation yields

$$(n-1)2^{n-2} \leq \log n + k\log k - \frac{k-1}{\ln 2} + \left(nk + \frac{k(k-1)}{2} + k\right)\log|W|. \tag{7}$$

Assuming $k \geq n \geq 2$, then it can easily be shown that there exists a constant, c, such that

$$n2^n \leq ck^2\log|W|.$$

So,

$$k = \Omega\left(\sqrt{\frac{n2^n}{\log|W|}}\right).$$

Since $n = \lceil\log m\rceil + 1 > \log m$ then

$$k = \Omega\left(\sqrt{\frac{m\log m}{\log|W|}}\right).$$

Q.E.D.

Clearly, for $W = \{-1, 1\}$ this lower bound matches the upper bound in Theorem 3.

5 Restriction on fan–in

A limit on the fan–in of a perceptron is another important practical restriction. In the networks discussed so far each node has an unlimited fan–in. In fact, in the constructions described above, many nodes receive inputs from a polynomial number of nodes (in terms of m) in a previous layer. In practice it is not possible to build devices that have such a large connectivity. Restricting the fan–in to 2 is the most severe restriction, and will be of primary interest in this paper.

5.1 Upper Bound

Once again, in order to derive the node complexity of recurrent neural network implementation of FSMs, we shall utilize the following lemma, proved in (Horne and Hush, 1994).

Lemma 3 Arbitrary boolean logic functions of the form $f : \{0,1\}^x \longrightarrow \{0,1\}^y$ can be implemented in a $O(x)$–layer network of perceptrons restricted to fan–in 2 with a node complexity of

$$\Theta\left(\frac{y2^x}{x+\log y}\right).$$

□

Theorem 5 A $O(\log m)$–layer recurrent neural network that has nodes whose fan–in is restricted to two can implement FSMs having m states with a node complexity of $O(m)$ □

Proof: Since m–state FSMs can be implemented in a recurrent neural network in which the multi-layer network performs a mapping of the form,

$$\phi : \{0,1\}^{\lceil \log m \rceil + 1} \longrightarrow \{0,1\}^{\lceil \log m \rceil + 1} .$$

Thus, the problem reduces to determining the node complexity of implementing such a logic function. Using $x = y = \lceil \log m \rceil + 1$, so that

$$\log m + 1 \leq x = y \leq \log m + 2 ,$$

then applying Lemma 3 gives a $O(\log m)$ layer network with upper bound on the node complexity of

$$O\left(\frac{(\log m + 2)2^{\log m+2}}{\log m + 1 + \log(\log m + 1)}\right) = O\left(\frac{(\log m)2^{\log m}}{\log m}\right) = O(m) \ .$$

Q.E.D.

Corollary 3 A fully connected recurrent neural network that has nodes whose fan–in is restricted to two can implement FSMs having m states with a node complexity of $O(m)$ and a $O(\log m)$ step slowdown. □

5.2 Lower Bound

Theorem 6 Multilayer recurrent neural networks that have nodes whose fan–in is restricted to two can implement FSMs having m states with a node complexity of $\Omega(m)$ if the number of unit time delays is $\lceil \log m \rceil$. □

Proof: Once again the proof is similar to Theorem 2, which gave a lower bound for the node complexity required in an arbitrary network of threshold logic units. Here, the total number of possible distinct configurations that a lower triangular feedforward network of threshold logic units whose fan–in is restricted to 2 is at most

$$\prod_{i=0}^{k-1} \binom{n+i}{2} 14,$$

where the term $\binom{n+i}{2}$ is used since a node in the i^{th} layer has $n+i$ possible inputs from which two are chosen. The constant 14 represents the fourteen possible threshold logic functions of two variables. (The functions XOR and XNOR are not linearly separable.) By counting the number of ways to choose nodes for the output of the network, the total number of logic functions implementable by a such a network is

$$\frac{n(k-1)!}{(k-n)!} \prod_{i=0}^{k-1} \binom{n+i}{2} 14.$$

For such a network to implement an arbitrary FSM, the number of configurations must be greater than the minimum number of distinctly different FSMs. Thus the value of k we seek must satisfy

$$2^{(n-1)2^{n-2}} \leq \frac{n(k-1)!}{(k-n)!} \prod_{i=0}^{k-1} \binom{n+i}{2} 14.$$

Taking the logarithm of both sides and using the fact that $\binom{n+i}{2} < \frac{(n+i)^2}{2}$, gives

$$(n-1)2^{n-2} \leq \log n + \log(k-1)! - \log(k-n)! + \sum_{i=0}^{k-1} \left[\log 7 + 2\log(n+i)\right].$$

Terms of the form $\log x!$ can be simplified using the bounds developed in equation (A.1) of the appendix, the index of summation can be changed, and negative terms can be eliminated to give

$$(n-1)2^{n-2} \leq \log n + k\log k + k\log 7 + 2\sum_{i=n}^{n+k-1} \log i,$$

The inequality can be preserved by adding more terms to the last summation on the right; specifically, by making the lower index $i = 0$. Then the bounds in equations (A.1) and (A.2) of the appendix can be used and more negative terms can be eliminated to yield

$$(n-1)2^{n-2} \leq \log n + k\log k + k\log 7 + 2(n+k)\log n + 2(n+k)\log k. \tag{8}$$

Assuming $k \geq n \geq 2$, then it can easily be shown that there exists a constant, c, such that

$$n2^n \leq ck\log k$$

Using the inequality in equation (A.3) of the appendix gives

$$\frac{n2^n}{c(n+\log n)} \leq k$$

So that

$$k = \Omega\left(\frac{n2^n}{n+\log n}\right).$$

Since $n = \lceil \log m \rceil + 1 > \log m$ then

$$k = \Omega\left(\frac{(\log m) 2^{\log m}}{\log m + \log\log m}\right) = \Omega(m).$$

Q.E.D.

6 Summary

In summary, we provide new bounds on the node complexity of implementing FSMs with recurrent neural networks having hard–limiting nonlinearities. Specifically, we improved the best known upper bound for unrestricted networks from $O\left(m^{3/4}\right)$ to $O\left(\sqrt{m}\right)$. We also improved the best known bound for networks in which the weights are restricted to a finite set from $O(m)$ to $O\left(\sqrt{m \log m}\right)$. Previously, there was no known upper bounds when the nodes of the network had a finite fan–in. We showed that it is possible to implement arbitrary FSMs in a network with $O(m)$ nodes having a fan–in of 2. These upper bounds are somewhat remarkable since they match lower bounds developed by Alon *et al.* for fully connected recurrent networks when the size of the weight set or the fan–in of each node is finite.

Our upper bounds were all found using constructions which yield a multilayer feedforward section. However, we argue that such networks can be simulated by a fully–connected, single–layer network, like a Hopfield network, if one is willing to incur a slowdown. Our first two upper bounds involve a constant slowdown factor of four, which is reasonable since it is independent of the problem size. But for the case when the fan–in is restricted to two, the slowdown scales proportionally with the log of number of states in the FSM. We leave it as an open question if these slowdown factors can be improved, or if an FSM can be simulated at all in a fully–connected network with less slowdown when the fan–in is so severely constrained.

Although one might speculate that more complex networks might yield more efficient constructions, we showed that, at least when the state of the FSM is encoded optimally in a subset of $\lceil \log m \rceil$ nodes, that for a very generic architecture, these lower bounds do not change for restrictions on weights or fan–in. When the network is unrestricted, the lower bound matches our upper bound, but is higher than the lower bound obtained for fully–connected networks. We leave it as an open question if a sparser encoding of the state variables can lead to a more efficient implementation. However, we speculate that it would not.

Finally, one interesting aspect of this study is that there is really not much difference in efficiency

$\sqrt{\log m}$	m
1	2
2	16
3	512
4	65,536
5	33,554,432
6	68,719,476,736
7	562,949,953,421,312
8	18,446,744,073,709,551,616

Table 1: The growth of $\sqrt{\log m}$ as a function of m.

when the network is totally unrestricted and when there are severe restrictions placed on the weights, since there is only a $\Theta\left(\sqrt{\log m}\right)$ penalty for restricting the weights to either -1 or 1. To get some idea for how marginal this difference is, Table 1 shows the relationship between m and $\sqrt{\log m}$. Clearly, this term is negligible.

Acknowledgments

The authors would like to thank Greg Heileman, Chaouki Abdallah, Bernard Moret, and Hava Siegelmann for many helpful discussions on this material and for making several valuable suggestions on the manuscript.

Appendix A

The following inequality can be derived easily using integration techniques.

$$\log x! \geq x \log x - \frac{x-1}{\ln 2} \tag{A.1}$$

For $a, b \geq 2$ it can easily be shown that

$$\log(a+b) \leq \log a + \log b \tag{A.2}$$

Finally, for any constant $c \geq 1$ and for all $y \geq 2$ it can easily be shown (Horne and Hush, 1994) that

$$y \leq cx \log x \quad \Longrightarrow \quad \frac{y}{c \log y} \leq x \tag{A.3}$$

References

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

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

Elman, J. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

Giles, C., Sun, G., Chen, H., Lee, Y., and Chen, D. (1990). Higher order recurrent networks & grammatical inference. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2*, pages 380–387. Morgan Kaufmann.

Hopcroft, J. and Ullman, J. (1979). *Introduction to automata theory, languages, and computation.* Addison–Wesley, Reading, MA.

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science USA*, 79:2554–2558.

Horne, B. and Hush, D. (1994). On the node complexity of neural networks. *Neural Networks*, 7(9):1413–1426.

Jordan, M. and Rumelhart, D. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354.

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

Lupanov, O. (1972). Circuits using threshold elements. *Soviet Physics — Doklady*, 17(2):91–93.

Lupanov, O. (1973). The synthesis of circuits from threshold elements. *Problemy Kibernetiki*, 26:109–140. In Russian.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Minsky, M. (1967). *Computation: Finite and infinite machines.* Prentice–Hall, Englewood Cliffs.

Muroga, S. (1971). *Threshold Logic and Its Applications.* Wiley, New York, NY.

Narendra, K. and Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1:4–27.

Nerrand, O., Roussel-Ragot, P., Personnaz, L., Dreyfus, G., and Marcos, S. (1993). Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms. *Neural Computation*, 5(2):165–199.

Parberry, I. (1994). *Circuit complexity and neural networks.* MIT Press, Cambridge, MA.

Red'kin, N. (1972). Realization of boolean functions in a certain class of threshold element circuits. *Automation and Remote Control*, pages 1252–1256.

Robinson, A. and Fallside, F. (1988). Static and dynamic error propagation networks with application to speech coding. In Anderson, D., editor, *Neural Information Processing Systems*, pages 632–641, New York, NY. American Institute of Physics.

Siegelmann, H. and Sontag, E. (1991). Neural networks are universal computing devices. Technical Report SYCON–91–08, Rutgers Center for Systems and Control.

Siegelmann, H. and Sontag, E. (1992). On the computational power of neural networks. In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory*, pages 440–449. ACM Press.

Siegelmann, H. and Sontag, E. (1994). Analog computation via neural networks. *Theoretical Computer Science*, 131:331–360.

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

Watrous, R. and Kuhn, G. (1992). Induction of finite–state languages using second–order recurrent networks. *Neural Computation*, 4(3):406–414.

Williams, R. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Winder, R. (1963). Bounds on threshold gate realizability. *IEEE Transactions on Electronic Computers*, EC–12:561–564.

Winder, R. (1970). Threshold logic asymptotes. *IEEE Transactions on Computers*, C–19(4):349–353.