Constructing Deterministic Finite-State Automata in Recurrent Neural Networks[†]

Christian W. Omlin [a,b], C. Lee Giles [a,c]

[a] NEC Research Institute, 4 Independence Way, Princeton, NJ 08540

[b] CS Department, Rensselaer Polytechnic Institute, Troy, NY 12180

[c] UMIACS, U. of Maryland, College Park, MD 20742

Abstract

Recurrent neural networks that are *trained* to behave like deterministic finite-state automata (DFA's) can show deteriorating performance when tested on long strings. This deteriorating performance can be attributed to the instability of the internal representation of the learned DFA states. The use of a sigmoidal discriminant function together with the recurrent structure contribute to this instability. We prove that a simple constructive algorithm can *construct* second-order recurrent neural networks with a sparse interconnection topology and sigmoidal discriminant function such that the internal DFA state representations are stable, i.e. the constructed network correctly classifies strings of *arbitrary length.* The algorithm is based on encoding strengths of weights directly into the neural network. We derive a relationship between the weight strength and the number of DFA states for robust string classification. For a DFA with n states and m input alphabet symbols, the constructive algorithm generates a "programmed" neural network with $O(n)$ neurons and $O(mn)$ weights. We compare our algorithm to other methods proposed in the literature.

1 INTRODUCTION

Recurrent neural networks can be trained to behave like deterministic finite-state automata (DFA's) [3, 4, 7, 8, 19, 20, 22]. The dynamical nature of recurrent networks can cause the internal representation of learned DFA states to deteriorate for long strings [23]; therefore, it can be difficult to make predictions about the generalization performance of trained recurrent networks. Recently, we have developed a simple method for encoding partial DFA's (state transitions) into recurrent neural networks [9, 18]. The goal was to demonstrate that prior knowledge can decrease the learning time significantly compared to learning without any

[0 †] Technical Report No. 94-3, Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180.

prior knowledge. The training time improvement was 'proportional' to the amount of prior knowledge with which a network was initialized. Important features of the encoding algorithm are the use of second-order weights, and the small number of weights that are programmed to achieve the desired network dynamics. When partial symbolic knowledge is encoded into a network in order to improve training, programming as few weights as possible is desirable because it leaves the network with many unbiased adaptable weights. This is important when a network is used for domain theory revision [15, 21], where the prior knowledge is not only incomplete, but may also be incorrect [10, 17].

Methods for constructing DFA's in recurrent networks where neurons have *hard-limiting* discriminant functions have been proposed [1, 14, 16]. This paper is concerned with neural network implementations of DFA's where *continuous sigmoidal* discriminant functions are used.

Our method is an alternative to an algorithm for constructing DFA's in recurrent networks with first-order weights proposed by Frasconi et al. [4, 5, 6]. A short introduction to finite-state automata will be followed by a review of the method by Frasconi et al. We will prove that our method can implement any deterministic finite-state automaton in second-order recurrent neural networks such that the behavior of the DFA and the constructed network are identical. Finally, we will compare DFA encoding algorithm with other methods proposed in the literature.

2 FINITE STATE AUTOMATA

Regular languages represent the smallest class of formal languages in the Chomsky hierarchy [12]. Regular languages are generated by regular grammars. A regular grammar G is a quadruple $G = < S, N, T, P >$ where S is the start symbol, N and T are non-terminal and terminal symbols, respectively, and P are productions of the form $A \rightarrow a$ or $A \rightarrow aB$ where $A, B \,\epsilon\, N$ and $a \,\epsilon\, T$. The regular language generated by G is denoted $L(G)$.

Associated with each regular language L is a deterministic finite-state automaton (DFA) M which is an acceptor for the language $L(G)$, i.e. $L(G) = L(M)$. DFA M accepts only strings which are a member of the regular language $L(G)$. Formally, a DFA M is a 5-tuple $M = < \Sigma, Q, R, F, \delta >$ where $\Sigma = \{a_1, \ldots, a_m\}$ is the alphabet of the language L, $Q = \{q_1, \ldots, q_n\}$ is a set of states, $R \,\epsilon\, Q$ is the start state, $F \subseteq Q$ is a set of accepting states and $\delta : Q \times \Sigma \rightarrow Q$ defines state transitions in M. A string x is accepted by the DFA M and hence is a member of the regular language $L(M)$ if an accepting state is reached after the string x has been read by M. Alternatively, a DFA M can also be considered a generator which generates the regular language $L(M)$.

3 FIRST-ORDER NETWORKS

This section summarizes work done by Frasconi et al. on implementing DFA's in recurrent neural networks. For details of the algorithms and the proofs see [4, 5, 6]. The work in [6] is an extension of [4, 5] which restricted the class of automata that could be encoded into recurrent networks to DFA's without cycles (except self-loops). The authors were focusing on automatic speech recognition as an application of implementing DFA's in recurrent neural networks. The constructed recurrent network becomes part of a K-L(priori-Knowledge and Learning) architecture consisting of two cooperating subnets devoted to explicit and learned rule representation, respectively, and whose outputs feed into a third subnet that computes the external output.

Each neuron in the first-order network computes the following function:

$$S_i^{(t+1)} = g(a_i(t)) = tanh(\frac{a_i(t)}{2}), \qquad a_i(t) = \sum_j W_{ij} S_j^{(t)} + \sum_k W_{ik} I_k^{(t)}, \tag{1}$$

where $S_j^{(t)}$ and $I_k^{(t)}$ represent the output of other state neurons and input neurons, respectively. For convenience of implementing a DFA in a recurrent network, a unary encoding is used to represent both inputs and DFA states; the state vectors representing successive DFA states $q(t)$ and $q(t+1)$ have a Hamming distance of 1. This requires a transformation of the original DFA into an equivalent DFA with more states which is suitable for the neural network implementation. In addition to the recurrent state neurons, there are three feed-forward layers of continuous state neurons that are used to construct the DFA state transitions. These continuous neurons implement boolean-like OR and AND functions by constraining the incoming weights. When a DFA state transition $\delta(q_j, a_k) = q_i$ is performed, the neuron corresponding to DFA state q_j switches from a high positive to a low negative output signal and the neuron corresponding to DFA state q_i changes its output signal from a low negative to a high positive value. The main characteristic of the constructed neural networks is the variable duration of the switching of state neurons, which is controlled by the self-recurrent weights W_{ii} and the input from other neurons. This is a desired property for the intended application. The authors prove that their proposed network construction algorithm can implement any DFA with n states and m input symbols using a network with no more than $2mn - m + 3n$ continuous neurons and no more than $m(n^2 + m + 5n - 5) + 6n$ weights.

4 SECOND-ORDER NETWORKS

The algorithm used here to construct DFA's in networks with second-order weights has also been used to encode partial prior knowledge to improve convergence time [9, 18], and to perform rule correction [10, 17].

4.1 Network Construction

We use discrete-time, recurrent networks with weights W_{ijk} to implement DFA's. A network accepts a time-ordered sequence of inputs and evolves with dynamics defined by the following equations:

$$S_i^{(t+1)} = h(a_i(t)) = \frac{1}{1+e^{-a_i(t)}}, \qquad a_i(t) = b_i + \sum_{j,k} W_{ijk} S_j^{(t)} I_k^{(t)}, \tag{2}$$

where b_i is the bias associated with hidden recurrent state neurons S_i; I_k denotes the input neuron for symbol a_k. The product $S_j^{(t)} I_k^{(t)}$ directly corresponds to the state transition $\delta(q_j, a_k) = q_i$. The goal is to achieve a nearly orthonormal internal representation of the DFA states with the desired network dynamics. For the purpose of illustration, we assume that a unary encoding is used for the input symbols. A special neuron S_0 represents the output (accept/reject) of the network after an input string has been processed. Given a DFA with n states and m input symbols, a network with n+1 recurrent state neurons and m input neurons is constructed. The algorithm consists of two parts: programming weights of a network to reflect DFA state transitions $\delta(q_j, a_k) = q_i$ and programming the output of the response neuron for each DFA state (figure 1). Neurons S_j and S_i correspond to DFA states q_j and q_i, respectively. The weights W_{jjk}, W_{ijk} and biases b_i are programmed as follows:

$$W_{ijk} = +H \ \ if \ \ \delta(q_j, a_k) = q_i \tag{3}$$

$$W_{jjk} = \begin{cases} +H & \text{if } \delta(q_j, a_k) = q_j \\ -H & \text{otherwise} \end{cases} \tag{4}$$

$$W_{0jk} = \begin{cases} +H & \text{if } \delta(q_j, a_k) \in F \\ -H & \text{otherwise} \end{cases} \tag{5}$$

$$b_i = -H/2 \ \ for \ all \ state \ neurons \ S_i \tag{6}$$

For each DFA state transition, at most three weights of the network have to be programmed. The initial state $\mathbf{S}^0$ of the network is

$$\mathbf{S}^0 = (S_0^0, 1, 0, 0, \ldots, 0)$$

The initial value of the response neuron S_0^0 is 1 if the DFA's initial state q_0 is an accepting state and 0 otherwise. The network rejects a given string if the value of the output neuron S_0^t at the end of the string is less or equal 0.5; otherwise, the network accepts the string.

4.2 Internal State Representation and Network Performance

When a recurrent network is *trained* to correctly classify a set of example strings, it can be observed that the networks' generalization performance on long strings which the network was not explicitly trained on

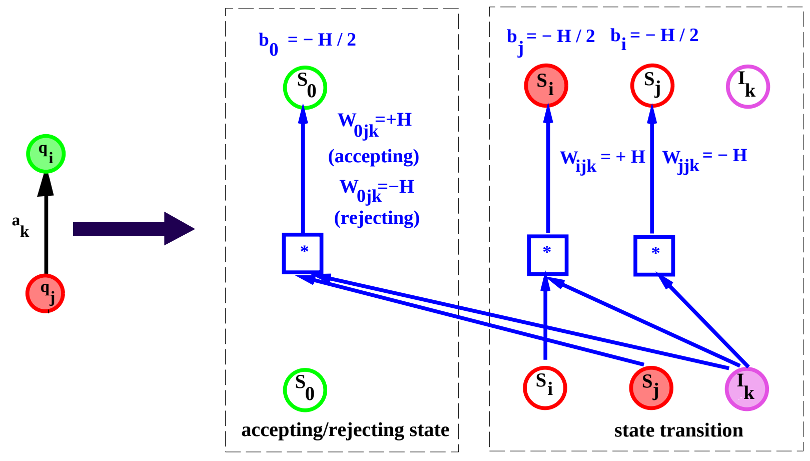

Figure 1: **Rule Insertion.** A known DFA transition $\delta(q_j, a_k) = q_i$ is programmed into a network.

deteriorates with increasing string length. This deteriorating performance can be explained by observing that the internal DFA state representation becomes unstable with increasing string length due to the network's dynamical nature and the sigmoidal discriminant function. This phenomenon has also been observed by Zeng et al. [23].

For a second-order network, whether or not a *constructed* recurrent network implements the desired DFA, i.e. whether the output of the the network and the DFA are identical for all input strings, depends on the value of H. The network dynamics preserves the internal nearly orthonormal DFA state representation only if the weights are programmed such that the outputs of all state neurons are close to 0 and 1, respectively. This calls for large values of the rule strength H. In order to demonstrate that the internal representation can be made sufficiently stable, we show test results on very long strings.

4.3 Experiment

We encoded a randomly generated, minimized 100-state DFA with alphabet $\Sigma = \{0, 1\}$ into a recurrent network with 101 state neurons (figure 2). The graph in figure 3 shows the generalization performance of the network constructed with varying rule strength $H = \{0.0, 0.1, 0.2, \ldots, 7.0\}$ on randomly chosen 1000 strings each of length 1000. At the end of each string, the network's classification was '1' (member of the regular language) if $S_0^{1000} > 0.5$, and '0' (not a member of the regular language) otherwise. We observe that the network performance monotonically improves with increasing value of H and that the network makes no classification errors for $H > 6.3$. The following analysis will show why this is the case.

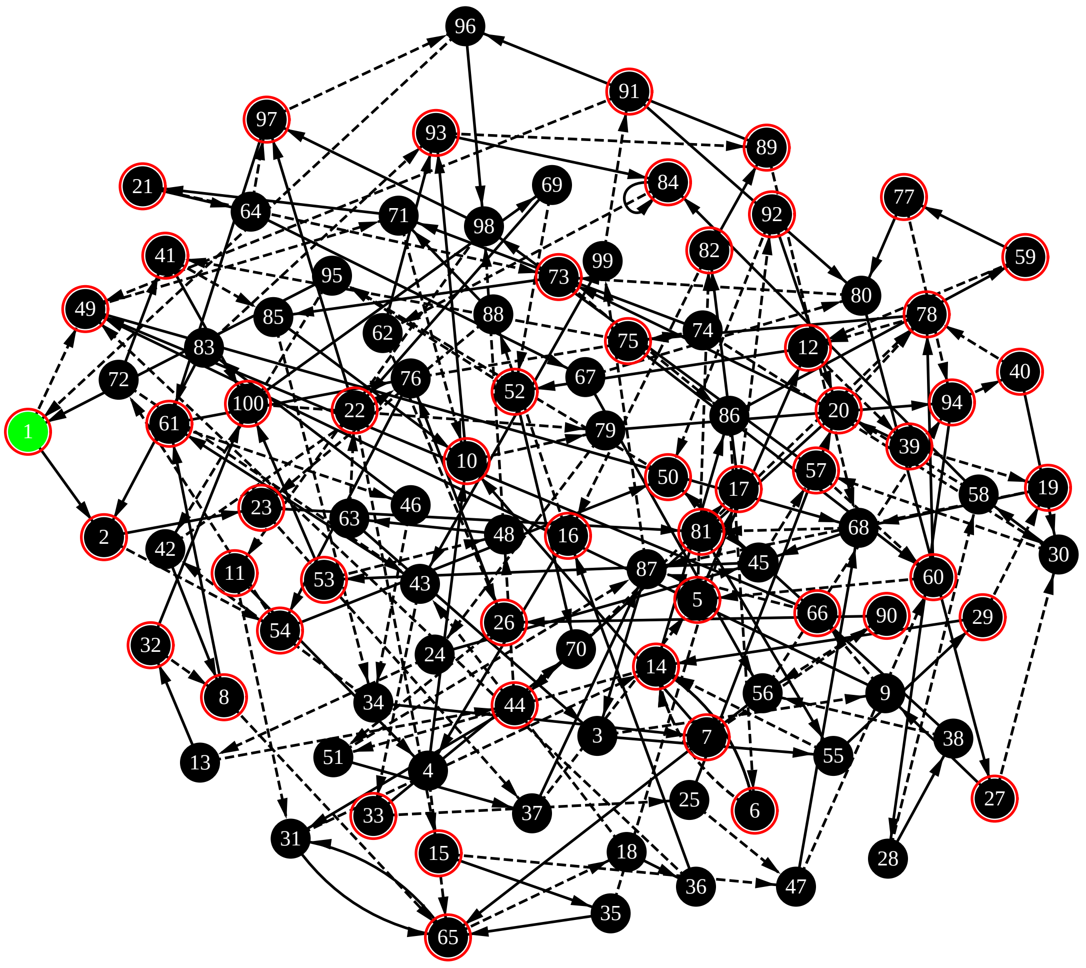

Figure 2: **Randomly Generated DFA:** The minimized DFA has 100 states and alphabet $\Sigma = \{0, 1\}$. State 1 is the start state. States with and without double circles are accepting and rejecting states, respectively.

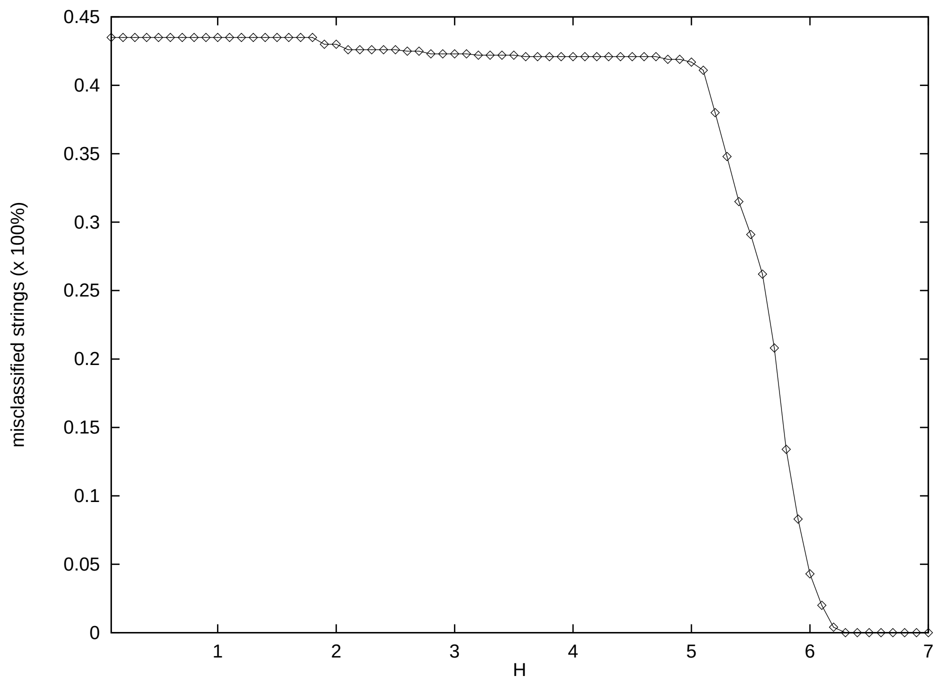

Figure 3: **Network Classification Performance:** The network classification performance on a data set as a function of the rule strength H (in 0.1 increments) is shown. The data set consisted of 1000 randomly chosen strings of length 1000; their labels were assigned by a randomly generated 100-state DFA. The classification performance is poor for small values of the rule strength H. The network's performance dramatically improves for $5 < H < 6$ to perfect classification for $H > 6.3$.

5 ANALYSIS

5.1 Motivation

When a constructed network processes a string, the state neurons go through a sequence of state changes. These state changes can be represented as iterations of the discriminant function $h(.)$ for each state neuron. A constructed network will only correctly classify strings of arbitrary length if its internal DFA state representation remains sufficiently stable. One way to achieve representation stability is to show that the iteration of the discriminant function $h(.)$ has desired properties such as fixed points. Thus, we will first investigate under what conditions the discriminant function $h(.)$ has fixed points. Recall that the recurrent network changes its state according to equation (2). From the encoding algorithm, we can derive a more precise form of the equation governing the dynamics of the constructed network:

$$S_i^{(t+1)} = h(x, H) = \frac{1}{1 + e^{H/2(1-2x)}} \tag{7}$$

The term $-H/2$ comes from the bias which is the same for all state neurons; Hx is the weighted sum feeding into neuron $S_i^{(t+1)}$. The graph of the function $h(.)$ is shown in figure 4. The discriminant function has properties which are important to the construction of recurrent networks with $h(x, H)$ as its discriminant function.

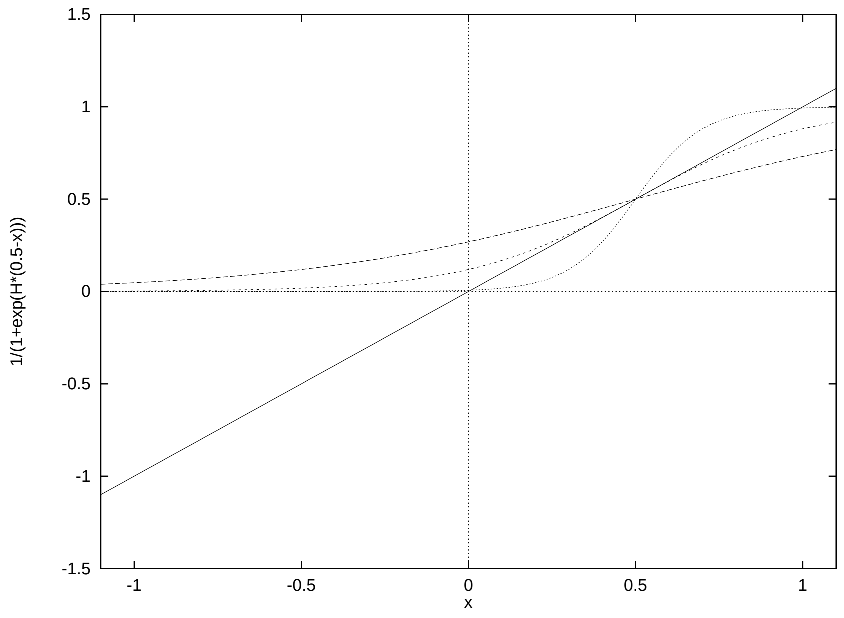

Figure 4: **Fixed Points of** $h(x, H)$**:** The graph of $1/(1+exp(H/2(1-2x)))$ is shown for $H = \{3, 4, 7\}$. The function has three fixed points $\phi^0 = 0.5, \phi^-$, and ϕ^+ as shown by the intersection with $y = H/4\ x$. $\phi^0 = 0.5$ is the only fixed point for $H \leq 4$. For $H > 4$, ϕ^0 is unstable; starting in the neighbourhood of ϕ^0, the stable fixed points ϕ^- and ϕ^+ can be reached.

5.2 Fixed Point Analysis

We investigate under what conditions $h(x, H)$ converges toward fixed points.

Lemma 5.2.1 *For $0 < H < 4$, $h(x, H)$ has the following fixed point:*

$$\phi^0 = 0.5$$

Furthermore, $h(x, H)$ converge to ϕ^0 for any choice of a start value x_0.

Proof: We observe that $h(x, H)$ is a *contractive mapping* for $0 < H < 4$, i.e. for any choice of x_1 and x_2 we have

$$|h(x_1, H) - h(x_2, H)| \leq c\ \ |x_1 - x_2|$$

with $0 \leq c < 1$ because the derivative $h'(x, H) \leq 1$ for all choices of x (figure 4). By the Contraction Mapping Theorem [2], $h(x, H)$ has only one fixed point at $\phi^0 = 0.5$ and the iteration $h(h(\ldots(x, H)\ldots) = h^p(x, H)$ converges to ϕ^0 for any choice of x.

The above lemma has the following corollary:

Corollary 5.2.1 *A recurrent network constructed from a DFA M with $H \leq 4$ will only misclassify strings that are members of the regular language $L(M)$.*

Proof: For some strings which are members of $L(M)$, the fixed point $\phi^0 = 0.5$ may be reached, leading to a misclassification of positive strings.

Lemma 5.2.2 *For $H \geq 4$, $h(x, H)$ has three fixed points $\phi^0 = 0.5$, ϕ^- and ϕ^+.*

Proof: The lemma is proven by defining an appropriate Lyapunov function V(.) and showing that $V(.)$ decreases toward the minima ϕ^- and ϕ^+. The details of the proof can be found in [6].

For $H = 4$, the fixed points are $\phi^0 = \phi^- = \phi^+ = 0.5$. For $H > 4$, the fixed point $\phi^0 = 0.5$ is unstable, i.e. starting from $x \neq \phi^0$, the state trajectory converges to one of the two stable points ϕ^- and ϕ^+. In fact, the following holds true:

Lemma 5.2.3 *for $x < \phi^0$ and $\phi^0 < x$, $h^p(x, H)$ (with $H > 4$) converges to ϕ^- and ϕ^+, respectively.*

Proof: Starting with $x < \phi^0$, $h^p(x, H)$ cannot converge to ϕ^+, because this would require to cross the line $y = H/4 * x$; this is impossible since the derivative $h'(x, H)$ is less than $H/4$ for all values except for ϕ^0 (figure 4). A similar argument can be made for the case $x > \phi^0$.

Lemma 5.2.4 *For arbitrary $H > 0$, the two fixed points ϕ^- and ϕ^+ are related as follows:*

$$\phi^- + \phi^+ = 1$$

Proof: This relationship follows from the symmetry of $h(x, H)$ around $\phi^0 = 0.5$.

The graph of the iterated function $h^p(x, H)$ with initial values 0 and 1 is shown in figure 5 for different values of H. The iteration $h^p(\mathbf{S}^0, H)$ represents the DFA state transitions $\delta(\ldots\delta(q_0, a^1), a^2, \ldots, a^p) = q_j$.

5.3 Worst Case Analysis

Having found conditions under which the sigmoidal discriminant function $h(x, H)$ reaches its fixed points ϕ^-, ϕ^0, and ϕ^+, we will now analyze under what conditions a constructed network implements a given DFA. The main result will show how large a network may be for fixed H such that the DFA and the constructed network have identical behavior, i.e. the network accepts strings iff the DFA accepts the same strings; the length of the strings is *arbitrary*.

From the DFA encoding algorithm, we can derive four different types of neuron state changes:

low $\rightarrow$ *high*:

$$S_i^{t+1} = h(-H/2 + S_j^t * H + \sum_{S_l \epsilon C_{i,l}} S_l^t * H - S_i^t * H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{8}$$

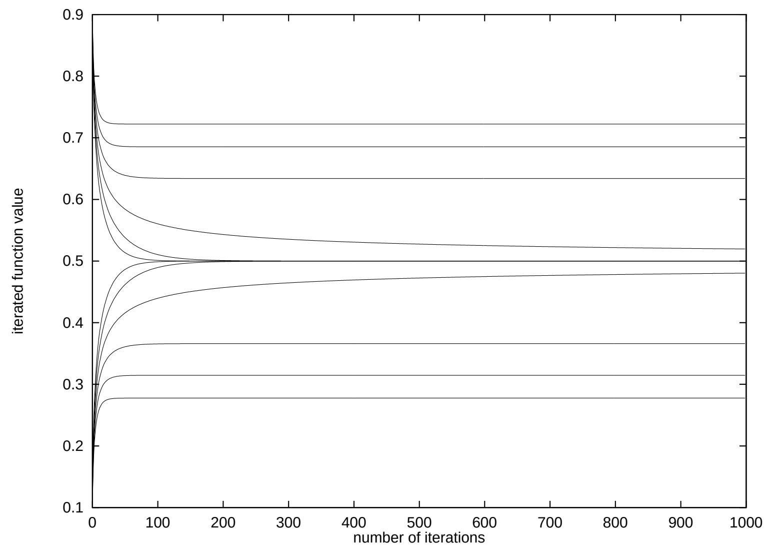

Figure 5: **Rate of Convergence of $h^p(x, H)$:** The graphs show for values of $H = \{3.8, 3.9, 4.0, 4.1, 4.2, 4.3\}$ the convergence of $h^p(x, H)$ toward the fixed points. For $H \leq 4$, $h^p(x, H)$ converges toward $\phi^0 = 0.5$ for any starting value of x. For $H > 4$, $h^p(x, H)$ converges toward ϕ^- and ϕ^+ for starting values $x = 0$ and $x = 1$, respectively.

$$S_i^{t+1} = h(-H/2 + S_j^t * H + \sum_{S_l \epsilon C_{i,l}} S_l^t * H + S_i^t * H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{9}$$

high $\rightarrow$ *high*:

$$S_i^{t+1} = h(-H/2 + S_i^t * H + \sum_{S_l \epsilon C_{i,l}} S_l^t * H) \quad (S_i^t : high, S_l^t : low) \tag{10}$$

high $\rightarrow$ *low*:

$$S_i^{t+1} = h(-H/2 - S_i^t * H + \sum_{S_l \epsilon C_{i,l}} S_l^t * H) \quad (S_i^t : high, S_l^t : low) \tag{11}$$

low $\rightarrow$ *low*:

$$S_i^{t+1} = h(-H/2 - S_i^t * H + \sum_{S_l \epsilon C_{i,l}} S_l^t * H) \quad (S_i^t, S_l^t : low) \tag{12}$$

$$S_i^{t+1} = h(-H/2 + S_i^t * H + \sum_{S_l \epsilon C_{i,l}} S_l^t * H) \quad (S_i^t, S_l^t : low) \tag{13}$$

where

$$C_{i,l} = \{S_j \mid W_{ijl} = H\} \tag{14}$$

The terms $-H/2$ and $S_j^t * H$ are derived from the network construction algorithm; they represent the *principal contribution* to the neuron S_i^{t+1}; all other terms are the *residual contributions* to the input of neuron S_i^{t+1}. The term $\sum S_l^t * H$ contributes to the total input of state neuron S_i^{t+1} if there are other transitions $\delta(q_l, a_k) = q_i$ in the DFA from which the recurrent network is constructed. Since there is a one-to-one correspondence between state neurons and DFA states, there will always be a negative contribution $-S_i^t * H$ for the current DFA state transition $\delta(q_j, a_k) = q_i$, i.e. only S_i^t can drive the signal S_i^{t+1} low. Equations (8) and (9) only differ with respect to the sign of the residual input $S_i^t * H$. If there is a state transition $\delta(q_i, a_k) = q_i$, then equation (9) applies; otherwise, there is a residual low signal S_i^t trying to drive S_i^{t+1} low and equation (8) applies. Similarly, either equation (12) or (13) is chosen for state transitions of the type *low* $\rightarrow$ *low*. The above equations account for all possible contributions to the net input of all state neurons because the encoding algorithm constructs a *sparse* recurrent network.

Before we proceed, we make some justified assumptions which will simplify the analysis.

Assumption 5.3.1 *An analysis for state transitions of type low $\rightarrow$ high and low $\rightarrow$ low also covers state transitions of type high $\rightarrow$ high and high $\rightarrow$ low, respectively.*

The state transitions types *high* $\rightarrow$ *high* and *high* $\rightarrow$ *low* cause stronger high and low signals, respectively, than the state transitions types *low* $\rightarrow$ *high* and *low* $\rightarrow$ *low*. Thus, no separate analysis will be necessary.

Assumption 5.3.2 *Of the two possible equations (8) and (9) for state transitions low $\rightarrow$ high, the former equation also covers the latter equation.*

If the internal state representations can be kept stable for equation (8), then it certainly can also be kept stable for equation (9). No separate analysis is necessary for the two equations.

Assumption 5.3.3 *Of the two possible equations (12) and (13) for state transitions low $\rightarrow$ low, the latter equation also covers the former equation.*

An argument similar to that given for validity of assumption 5.2.2 can be given.

Thus, we are left with only the two following types of state transitions which represent the worst cases:

low $\rightarrow$ *low*:

$$S_i^{t+1} = h(-H/2 + S_i^t * H + \sum_{S_l \epsilon C_{i,l}} S_l^t * H) \quad (S_i^t, S_l^t : low) \tag{15}$$

low $\rightarrow$ *high*:

$$S_i^{t+1} = h(-H/2 + S_j^t * H + \sum_{S_l \epsilon C_{i,l}} S_l^t * H - S_i^t * H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{16}$$

We will make a simplifying assumption about the values of low and high signals:

Assumption 5.3.4 *The low signal S_l^t in state transitions of type low $\rightarrow$ low converges toward the fixed point ϕ^-; the high signal S_j^t in transitions of type low $\rightarrow$ high converges toward fixed point ϕ^+.*

The network's initial state is $\mathbf{S}^0 = (S_0^0, 1, 0, \ldots, 0)$; the value of S_0^0 depends on whether the start state is an accepting or rejecting state. The low and high signals of all state neurons will converge toward the fixed points ϕ^- and ϕ^+, respectively; these fixed points will be reached for sufficiently long strings according to lemma 5.2.3.

In order for the internal DFA state representation to remain stable, the low and high signals must remain sufficiently stable, i.e. the low and high signals must be less or equal and larger than 0.5, respectively.

Consider equation (15). In order for the low signal to remain less or equal 0.5, the argument of $h(.)$ must be less or equal 0.5. From equation (15) and under assumption 5.3.4, we have the following condition for stable low signals:

$$-\frac{H}{2} + H * n * \phi^- \leq \frac{1}{2} \tag{17}$$

Solving the above inequality leads to the following lemma for the preservation of low signals:

Lemma 5.3.1 *The low signals S^t are always less or equal to 0.5 if*

$$n \leq \lfloor \frac{1}{2\phi^-(H)}(1 + \frac{1}{H}) \rfloor$$

Thus, the lemma puts restrictions on the size n of the network for a given H. The location of the fixed point ϕ^- is determined by H; thus, we write $\phi^-(H)$. The graph in figure 6 shows the maximum allowed network size n as a function of the rule strength H; the allowed size grows exponentially with increasing rule strength.

A similar analysis can be carried out for state transitions of equation (16). In the worst case, the following inequality must be satisfied for stable high signals:

$$-\frac{H}{2} + S_j^t * H - S_i^t * H > \frac{1}{2} \tag{18}$$

For the left-hand side in the above inequality, we assumed that there is only one DFA transition $\delta(q_j, a_k) = q_i$ for chosen q_i and a_k; thus, $\sum_{S_l^t \epsilon C_{i,l}} = 0$. Any positive terms would only strengthen the high signal S_i^{t+1}. Furthermore, S_i^t has a negative contribution to the net input of S_i^{t+1}; if there were a transition $\delta(q_i, a_k) = q_i$ (self-loop), then the contribution $+S_i^t * H$ would only strengthen the high signal S_i^{t+1}. Thus, we are considering the worst possible case for high signals for all recurrent state neurons. The high signal S_j^t and the low signal S_i^t converge toward ϕ^+ and ϕ^-, respectively. Therefore, the inequality simplifies to:

$$-\frac{H}{2} + H * \phi^+ - H(1 - \phi^+) > \frac{1}{2} \tag{19}$$

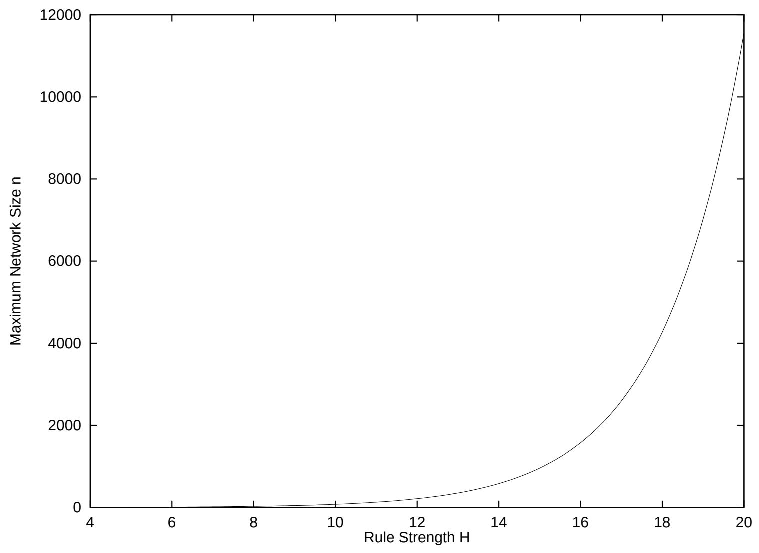

Figure 6: **Maximum Allowed Network Size for Preservation of Low Signals**: The graph shows for different values of H the maximum allowed number of nodes in a constructed network such that the low signals remain stable.

Solving for ϕ^+ results in the following condition for stable high signals:

Lemma 5.3.2 *The high signals S^t are always greater than 0.5 if*

$$\phi^+(H) > \frac{1}{4}(3 + \frac{1}{H})$$

The left hand-side of the above inequality converges toward 1 and the right-hand side decays exponentially toward 3/4 with increasing values of H. Thus, there exists a value H_{min} such that the above inequality is always satisfied (figure 7) for all values $H > H_{min}$.

The condition for guaranteed low signals explicitly depends on the network size, whereas the condition for guaranteed high signals does not. This asymmetry arises from the simplifying assumptions in the analysis: The residual inputs contribute to weakening low signals (i.e. higher low signals) and thus had to be included in the analysis; they strengthen high signals (i.e. higher high signals) and could thus be omitted in proof of the stabiblity of high signals.

We can now state the following theorem:

Theorem 5.3.1 *A sparse recurrent neural network RNN with $n+1$ sigmoidal state neurons, m input neurons, at most $3mn$ second-order weights with alphabet $\Sigma_w = \{-H, 0, +H\}$ $(4 < H_{min} < H < H_{pred})$, $n+1$ biases with alphabet $\Sigma_b = \{-H/2\}$, and maximum fan-out $3m$ can be constructed from a DFA M with n states and m input symbols such that the internal state representation remains stable, i.e. $S_i > 0.5$ when q_i*

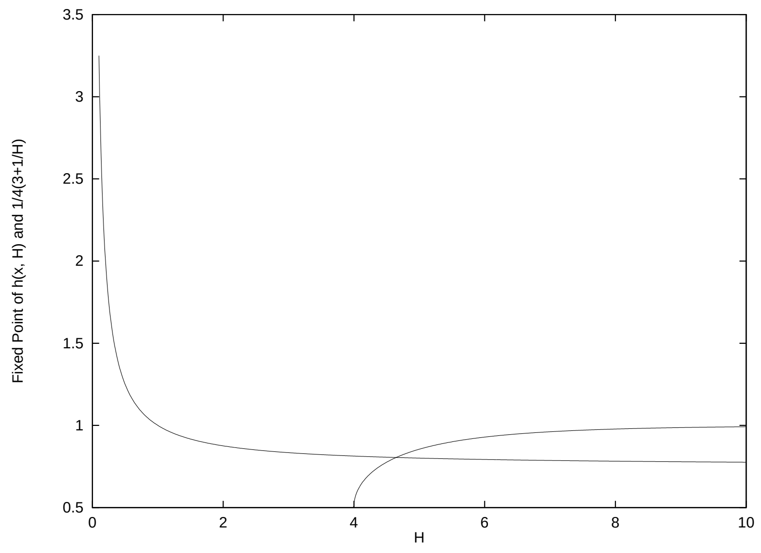

Figure 7: **Condition for Preservation of High Signals:** $\phi^{+}(H)$ converges toward 1 and 1/4 (3+1/H) converges exponentially toward 3/4 as H increases. Thus, the condition for strong high signals is satisfied for some values of $\phi^{+}(H) > H_{min}$.

is the current DFA state and $S_i \leq 0.5$ otherwise if

$$n \leq \lfloor \frac{1}{2\phi^{-}(H)}(1+\frac{1}{H}) \rfloor \quad with \quad \phi^{+}(H) > \frac{1}{4}(3+\frac{1}{H})$$

for a proper choice of H. H, $\phi^{-}(H)$ and $\phi^{+}(H)$ can be computed by iterating $h(0,H)$ until $h^{p}(0,H) = h^{p+1}(0,H) = \phi^{-}(H)$.

Proof: Our analysis assumed the worst case for weak low and high signals; this analysis also covers the cases of strong low and high signals. Thus, we have shown that the internal state representation remains sufficiently stable in general. The value of H can be computed (lemma 5.2.3).

The experiments of section 4.3 showed that a randomly generated DFA with 100 states correctly classifies 1000 strings of length 1000 for $H_{exp} > 6.3$. We computed the value of H_{pred} which guarantees that the constructed network performs perfectly; the value $H_{pred} = 10.4$ guarantees perfect generalization. H_{pred} was computed from a worst case analysis; it should thus come as no surprise that $H_{min} < H_{exp} < H_{pred}$.

Since our analysis investigated the worst case and since we made no limits on the string length, we can now state that a recurrent network can be constructed from a DFA such that the behavior of the recurrent network and the DFA are the same:

Corollary 5.3.1 *Let $L(M_{DFA})$ denote the language accepted by a DFA M and let $L(M_{RNN})$ be the language accepted by the sparse RNN constructed from M; then, there exists a value of H ($4 < H_{min} < H < H_{pred}$)*

such that $L(M_{RNN}) = L(M_{DFA})$. The value H_{pred} can be computed from theorem 5.3.1.

Proof: Theorem 5.3.1 states that the internal DFA state representation remains stable for arbitrary string length. Thus, all DFA states are always distinguishable and the value of the special output neuron S_0 is greater than 0.5 if the string is accepted by the DFA and less or equal 0.5 otherwise.

5.4 Analysis for Partially Recurrent Networks

The above analysis was carried out for the worst case where a neuron receives residual inputs from all other neurons. The upper bound on the maximum number of recurrent state neurons can be increased when a network is constructed from a specific DFA. Consider a state q_i and let D_{ik} denote the number of states q_j such that $\delta(q_j, a_k) = q_i$ for each symbol a_k. Setting $D = \max\{D_{ik}\}$, each recurrent state neuron receives residual input from at most $\rho = \frac{D}{n}$ recurrent neurons for a chosen input symbol a_k. Thus, the number of terms in the sum $\sum H * S^t$ in reduces from n to $\rho * n$ in inequality (17):

$$-\frac{H}{2} + H * n * \rho * \phi^- \leq \frac{1}{2} \tag{20}$$

The analysis for guaranteed high signals is the same as in section 5.3, since S_i^t can drive S_i^{t+1} low and the positive contribution of other state neurons only strengthens the high signal S_i^{t+1}. Thus, we have the following theorem for the construction of recurrent networks for specific DFA's:

Theorem 5.4.1 *Let ρ denote the maximum fraction of DFA states q_j from which there are state transitions $\delta(q_j, a_k) = q_i$ for a fixed choice of a_k and any q_i. Then, a sparse recurrent neural network RNN with $n+1$ sigmoidal state neurons, m input neurons, at most $3mn$ second-order weights with alphabet $\Sigma_w = \{-H, 0, +H\}$ $(4 < H_{min} < H < H_{pred})$, $n+1$ biases with alphabet $\Sigma_b = \{-H/2\}$, and maximum fan-out $3m$ can be constructed from a DFA M with n states and m input symbols of states such that the internal state representation remains stable, i.e. $S_i > 0.5$ when q_i is the current DFA state and $S_i \leq 0.5$ otherwise if*

$$n_\rho \leq \lfloor \frac{1}{2\ \rho\ \phi^-(H)}(1 + \frac{1}{H}) \rfloor \quad \textit{with} \quad \phi_\rho^+(H) > \frac{1}{4}(3 + \frac{1}{H})$$

for a proper choice of H.

It follows that a recurrent network can be constructed from any DFA such that the languages accepted by the network and the DFA are identical.

5.5 Analysis for Fully Recurrent Networks

In the above analysis, the constructed networks were not fully connected. When recurrent networks are used for domain theory revision, fully recurrent networks are initialized with the available prior symbolic knowledge; partial or complete knowledge can be encoded. In the case of encoding a complete DFA, a small

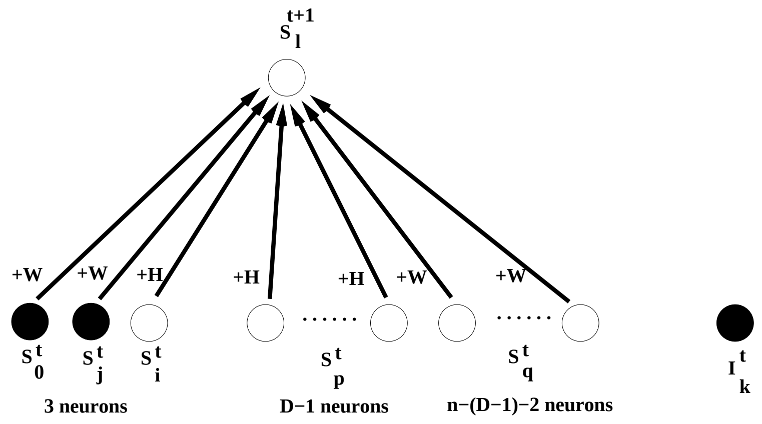

Figure 8: **Preservation of Low Signals:** The figure shows the input fed into neuron S_l^{t+1} from all other neurons for a chosen input symbol. For clarity, the operation $S_j I_k$ is omitted.

number of weights are programmed to values $+H$ and $-H$ according to the encoding algorithm. All other weights are usually initialized to small random values. We are interested in whether a recurrent network can be constructed such that the constructed network still implements a given DFA when the network is fully recurrent and the free weights (as opposed to the programmed weights) are initialized to random values drawn from the interval $[-W, W]$ according to some distribution.

Notice that the special response neuron S_0 also drives other neurons in a fully connected recurrent network. The following inequality has to be satisfied in order for low signals to be preserved:

$$-\frac{H}{2} + H * x \leq \frac{1}{2} \tag{21}$$

We will now derive a a condition for the preservation of low signals in fully recurrent networks (figure 8). Consider a DFA transition $\delta(q_j, a_k) = q_i$. All state neurons S_l^{t+1} which do not correspond to DFA state q_i should be low signals. Assuming the current state is an accepting state, neuron S_0^t has a high output signal and is weighted by $+W$. Neuron S_j^t has a high output since it corresponds to the current DFA state q_j; it is also weighted by $+W$. Neuron S_i^t is low since we are dealing with state transitions of type *low* $\longrightarrow$ *low* only; it has a weight $+H$. There are $\rho n - 1$ neurons with low outputs and weights $+H$. The remaining $n - (\rho n - 1) - 2$ neurons also have low outputs and are weighted by $+W$. Thus, the worst case expression for the net input to neuron S_l^{t+1} becomes:

$$\frac{W}{H}S_0^t + \frac{W}{H}S_j^t + S_i^t + (\rho n - 1)S_p^t + \frac{W}{H}(n - (\rho n - 1) - 2)S_q^t \tag{22}$$

Assuming that all signals converge toward their respective fixed points, the above net input yields the following inequality for guaranteed low signals S_l^{t+1} in the worst case:

$$-\frac{H}{2} + 2W(1 - \phi^-) + H\phi^- + H(\rho n - 1)\phi^- + W(n - (\rho n - 1) - 2)\phi^- \leq \frac{1}{2} \tag{23}$$

Solving for n, we get the following lemma:

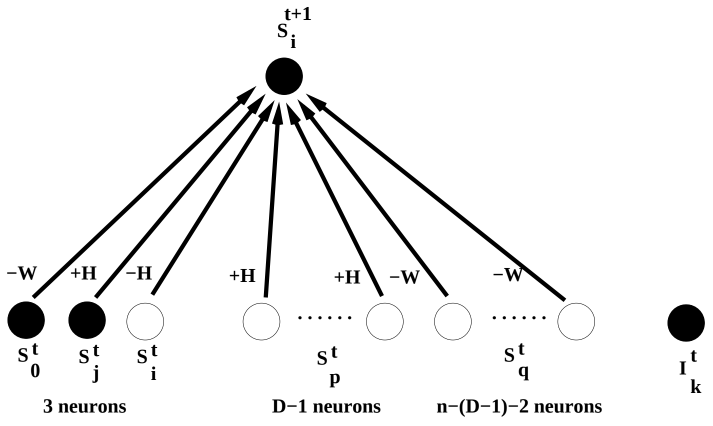

Figure 9: **Preservation of High Signals:** The figure shows the input fed into neuron S_i^{t+1} from all other neurons for a chosen input symbol. For clarity, the operation $S_j I_k$ is not shown.

Lemma 5.5.1 *The low signals S^t in a fully recurrent network are always less or equal to 0.5 if*

$$n_{\rho,W} \leq \lfloor \frac{1}{2\phi^-(H)} \frac{1 + H - 2W(2 - 3\phi^-(H))}{\rho H + W(1-\rho)} \rfloor$$

Notice that the above inequality reduces to the inequalities for the simpler cases discussed above.

Similarly, we can derive a condition for the preservation of high signals (figure 9). For a DFA state transition $\delta(q_j, a_k) = q_i$, state neuron S_j^t is a high signal and S_i^{t+1} should change from a low to a high signal. In the worst case, neuron S_i^{t+1} receives the following net input:

$$-\frac{W}{H}S_0^t + S_j^t - \frac{W}{H}S_i^t + \frac{W}{H}(\rho n - 1)S_p^t - \frac{W}{H}(n - (\rho n - 1) - 2)S_q^t \tag{24}$$

Assuming that all signals converge toward their respective fixed points, we get the following inequality for stable high signals:

$$-\frac{H}{2} - W\phi^+ + H\phi^+ + H(\rho n - 2)\phi^- - W(n - (\rho n - 1) - 2)\phi^- > \frac{1}{2} \tag{25}$$

Solving the above inequality for ϕ^- yields the following condition for stable high signals:

Lemma 5.5.2 *The high signals S^t in a fully recurrent network are always greater than 0.5 if*

$$\phi^+_{\rho,W}(H) > \frac{1}{2} \; \frac{1 + 5H - 2\rho n(H + W) + 2W(n-1)}{3H + W(n-2) - \rho n(H + W)}$$

Thus, we have proven the following theorem:

Theorem 5.5.1 *A fully recurrent neural network RNN with $n+1$ sigmoidal state neurons, m input neurons, at most $3mn$ second-order weights with alphabet $\Sigma_w = \{-H, 0, +H\}$ ($4 < H_{min} < H < H_{pred}$), $n+1$*

biases with alphabet $\Sigma_b = \{-H/2\}$, maximum fan-out $3m$, and random initial weights drawn from an arbitrary distribution in [-W, W] with $W < H$ can be constructed from a DFA M with n states and m input symbols such that the internal state representation remains stable, i.e. $S_i > 0.5$ when q_i is the current DFA state and $S_i \leq 0.5$ otherwise if

$$n_{\rho,W} \leq \lfloor \frac{1}{2\phi^-(H)} \frac{1 + H - 2W(2 - 3\phi^-(H))}{\rho H + W(1-\rho)} \rfloor \quad \textit{with} \quad \phi^+_{\rho,W}(H) > \frac{1}{2} \; \frac{1 + 5H - 2\rho n(H+W) + 2W(n-1)}{3H + W(n-2) - \rho n(H+W)}$$

There exist solutions for H, W, ρ, and $n_{\rho,W}$ such that the above two conditions are satisfied.

Theorems 5.3.1, 5.4.1 and 5.5.1 have the following corollary:

Corollary 5.5.1 *The maximum allowed network sizes n, n_ρ and $n_{\rho,W}$ are related as follows:*

$$n_\rho \geq n \quad \textit{and} \quad n_\rho \geq n_{\rho,W}$$

where equality only holds for $\rho = 1$ and $W = 0$.

We have shown that a fully recurrent network can be constructed from a DFA such that the languages accepted by the network and the DFA are identical *independent* of the distribution of the randomly initialized weights. The value W depends on the network size n, the value of ρ, and the magnitude of H.

One can view fully recurrent networks as sparse networks with noise in the programmed weights. From that point of view, the encoding algorithm constructs sparse networks which are to some extent tolerant to noise in the weights.

Both conditions in theorem 5.5.1 must be satisfied for a stable internal DFA representation. We cannot guarantee that a network constructed from a given DFA accepts the same language as the DFA if any one of the above conditions is violated. In fact, we believe that the following conjecture holds true:

Conjecture 5.5.1 *If either one of the two conditions is violated, then the languages accepted by the constructed network and the given DFA are not identical for an arbitrary distribution of the randomly initialized weights in the interval $[-W, W]$.*

5.6 Comparison with other Methods

Different methods [1, 6, 11, 14, 16] for encoding DFA's with n states and m input symbols in recurrent networks are summarized in table 2. The methods differ in the choice of the discriminant function (hard-limiting, sigmoidal, radial basis function), the size of the constructed network and the restrictions that are

author(s)	nonlinearity	order	# neurons	# weights	weight alphabet	fan-in limit	fan-out limit
Minsky (1967)	hard	first	$O(mn)$	$O(mn)$	$\Sigma_W = \{1, 2\}$	none	none
Alon et al. (1991)[1]	hard	first	$O(n^{3/4})$	-	no restriction	none	none
Alon et al. (1991)[1]	hard	first	$O(n)$	-	any restriction	none	yes
Frasconi et al. (1993)	sigmoid	first	$O(mn)$	$O(n^2)$	no restriction	none	none
Horne (1994)[2]	hard	first	$O(\sqrt{\frac{mn\log n}{\log m + \log n}})$	-	no restriction	none	none
Horne (1994)[2]	hard	first	$O(\sqrt{mn\log n})$	$O(mn\log n)$	$\Sigma_W = \{-1, 1\}$	none	none
Horne (1994)[2]	hard	first	$O(\frac{mn\log n}{\log m + \log n})$	$O(n)$	$\Sigma_W = \{-1, 1, 2\}$	2	none
Gori et al. (1994)[3]	sigmoid/radial	first	$O(n)$	$O(n^2)$	no restriction	none	none
Giles & Omlin (1994)[4]	sigmoid	second	$O(n)$	$O(mn)$	$\Sigma_W = \{-H, -H/2, +H\}$	none	$3m$

Table 2: **Comparison of different DFA Encoding Methods**: The different methods use different amounts and types of resources to implement a given DFA with n states and m input symbols. [1,2] There also exist lower bounds for the number of neurons necessary to implement any DFA. [2] The bounds for $\Sigma = \{0, 1\}$ have been generalized to arbitrary alphabet size m. [3] The authors use their network with sigmoidal and radial basis functions in multiple layers to train recurrent networks; however, their architecture could be used to directly encode a DFA in a network. [4] The rule strength H can be chosen according to the results in section 5.3.

imposed on the weight alphabet, the neuron fan-in and fan-out. The results in [14] improve the upper and lower bounds reported in [1] for DFA's with only two input symbols. Those bounds can be generalized to DFA's with m input symbols [13]. Among the methods which use continuous discriminant functions, our algorithm uses no more neurons than the best of all methods, and consistently uses fewer weights and smaller fan-out size than all methods.

5.7 Open Problems

One of the theoretical results in [1] gives a lower bound of $\Omega(\sqrt{nlogn})$ on the number of *hard-limiting* neurons needed to implement a DFA with n states when the weight alphabet and the neuron *fan-in* are limited. Our encoding algorithm establishes without optimization an upper bound of $O(n)$ for *sigmoidal* neurons with limited *fan-out*. It would be interesting to investigate whether there is a lower bound and whether the upper bound can be made tighter. While n states can be encoded in only $\log n$ neurons using a binary encoding scheme, our encoding algorithm cannot encode arbitrary DFA's with only $\log n$ neurons; this can be shown on small example DFA's.

Our result about constructing DFA's in fully recurrent networks makes no assumptions about the distribution of the randomly initialized weights. It would be interesting to investigate whether a constructed network and its associated DFA accept the same language with some probability p if either one of the conditions for guaranteed stability of low and high signals were violated. A positive result would have to make some assumptions about the distribution of the randomly initialized weights.

6 CONCLUSION

We compared two different methods for encoding deterministic finite-state automata (DFA's) into recurrent neural networks with sigmoidal discriminant functions. The method proposed in [6] implements DFA's using linear programming and explicit implementation of state transitions implementing boolean-like functions with sigmoidal neurons. The authors give rigorous proofs about their neural network implementation of DFA's. An interesting characteristic of their approach is that state transitions usually take several time steps to complete.

We have proven that our encoding algorithm can implement any DFA with n states and m input symbols in a *sparse* recurrent network with $O(n)$ state neurons, $O(mn)$ weights and limited fan-out of size $O(m)$ such that the DFA and the constructed network accept the same regular language. The desired network dynamics is achieved by programming some of the weights to values $+H$ or $-H$. A worst case analysis has revealed a quantitative relationship between the rule strength H with which some weights are initialized and the maximum network size such that the network dynamics remains robust for arbitrary string length. For any chosen value $4 < H_{min} < H < H_{pred}$, there exists an upper bound on the network size which guarantees that the constructed network implements a given DFA; the value H_{min} is *independent* of the DFA to be implemented and H_{pred} can be computed for a given DFA. We extended our analysis to *fully recurrent* networks where free weights are initialized with random values drawn from a distribution in the interval $[-W, W]$ with $W < H$. We have proven that there exist fully-recurrent networks which accept the same language as the given DFA for *arbitrary* distribution of the randomly initialized weights $W > 0$.

This is only a proof of existence, i.e. we do not make any claims that such a solution can be *learned.*

Our algorithm for constructing DFA's in recurrent neural networks is more straightforward compared to the method proposed in [6]. By using second-order weights, we have adjusted the network architecture so that DFA state transitions are naturally mapped into network state transitions. Our networks need fewer nodes and weights than the implementation reported in [6]. The network model has not lost any of its computational capabilities by the introduction of second-order weights.

7 ACKNOWLEDGMENT

We would like to acknowledge useful discussions with B.G. Horne, L. R. Leerink, and T. Lin.

References

[1] N. Alon, A. Dewdney, and T. Ott, "Efficient simulation of finite automata by neural nets," *Journal of the Association for Computing Machinery*, vol. 38, no. 2, pp. 495–514, April 1991.

[2] M. Barnsley, *Fractals Everywhere.* San Diego, CA: Academic Press, 1988.

[3] J. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 1990.

[4] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "A unified approach for integrating explicit knowledge and learning by example in recurrent networks," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 1, p. 811, IEEE 91CH3049-4, 1991.

[5] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Unified integration of explicit rules and learning by example in recurrent networks," *IEEE Transactions on Knowledge and Data Engineering*, 1993. To appear.

[6] P. Frasconi, M. Gori, and G. Soda, "Injecting nondeterministic finite state automata into recurrent networks," tech. rep., Dipartimento di Sistemi e Informatica, Università di Firenze, Italy, Florence, Italy, 1993.

[7] C. Giles, D. Chen, C. Miller, H. Chen, G. Sun, and Y. Lee, "Second-order recurrent neural networks for grammatical inference," in *Proceedings of the International Joint Conference on Neural Networks 1991*, vol. II, pp. 273–281, July 1991.

[8] C. Giles, C. Miller, D. Chen, H. Chen, G. Sun, and Y. Lee, "Learning and extracting finite state automata with second-order recurrent neural networks," *Neural Computation*, vol. 4, no. 3, p. 380, 1992.

[9] C. Giles and C. Omlin, "Inserting rules into recurrent neural networks," in *Neural Networks for Signal Processing II, Proceedings of The 1992 IEEE Workshop* (S. Kung, F. Fallside, J. A. Sorenson, and C. Kamm, eds.), pp. 13–22, IEEE Press, 1992.

[10] C. Giles and C. Omlin, "Rule refinement with recurrent neural networks," in *Proceedings IEEE International Conference on Neural Networks (ICNN'93)*, vol. II, pp. 801–806, 1993.

[11] M. Gori, M. Maggini, and G. Soda, "Insertion of finite state automata in recurrent radial basis function networks," tech. rep., Dipartimento di Sistemi e Informatica, Università di Firenze, Italy, 1994.

[12] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1979.

[13] B. Horne. Personal Communication.

[14] B. Horne and D. Hush, "Bounds on the complexity of recurrent neural network implementations of finite state machines," in *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, 1994. To appear.

[15] R. Maclin and J. Shavlik, "Refining algorithms with knowledge-based neural networks: Improving the chou-fasman algorithm for protein folding," in *Computational Learning Theory and Natural Learning Systems* (S. Hanson, G. Drastal, and R. Rivest, eds.), MIT Press, 1992.

[16] M. Minsky, *Computation: Finite and Infinite Machines*, ch. 3, pp. 32–66. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1967.

[17] C. Omlin and C. Giles, "Rule checking with recurrent neural networks," *IEEE Transactions on Knowledge and Data Engineering*, 1993. accepted for publication.

[18] C. Omlin and C. Giles, "Training second-order recurrent neural networks using hints," in *Proceedings of the Ninth International Conference on Machine Learning* (D. Sleeman and P. Edwards, eds.), (San Mateo, CA), pp. 363–368, Morgan Kaufmann Publishers, 1992.

[19] J. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, pp. 227–252, 1991.

[20] D. Servan-Schreiber, A. Cleeremans, and J. McClelland, "Graded state machine: The representation of temporal contingencies in simple recurrent networks," *Machine Learning*, vol. 7, p. 161, 1991.

[21] G. Towell, J. Shavlik, and M. Noordewier, "Refinement of approximately correct domain theories by knowledge-based neural networks," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, (San Mateo, CA), p. 861, Morgan Kaufmann Publishers, 1990.

[22] R. Watrous and G. Kuhn, "Induction of finite-state languages using second-order recurrent networks," *Neural Computation*, vol. 4, no. 3, p. 406, 1992.

[23] Z. Zeng, R. Goodman, and P. Smyth, "Learning finite state machines with self-clustering recurrent networks," *Neural Computation*, vol. 5, no. 6, pp. 976–990, 1993.