Constructing Deterministic Finite-State Automata in Recurrent Neural Networks

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

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

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

{omlinc, giles}@research.nj.nec.com

U. of Maryland TR UMIACS-TR-95-50 and CS-TR-3460

Revised February, 1996

Abstract

Recurrent neural networks that are *trained* to behave like deterministic finite-state automata (DFAs) 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 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

1.1 Motivation

Recurrent neural networks are neural network models which have feedback in the network architecture and, thus, have the power to represent and learn state processes. This feedback property enables such neural nets to be used in problems and applications which require state representation: speech processing, plant control, adaptive signal processing, time series prediction, engine diagnostics etc. (for example see the recent special issue on dynamically-driven recurrent neural networks [10]). For enhanced performance, some of these neural network algorithms are mapped directly into VLSI designs [20, 28].

Neural networks readily enhance their performance by having a priori knowledge about the problem to be solved encoded or used in the neural network [8, 27]. This work discusses how a priori finite state automata rules can be encoded into a recurrent neural network with sigmoid activation neurons in such a way that arbitrary long string sequences are always correctly recognized - a stable encoding of rules. Such rule encoding has been shown to speed up convergence time and to permit rule refinement i.e., correction of incorrect rules through later training. Thus, this encoding methodology would permit rules to be mapped into neural network VLSI chips, offering the potential of greatly increasing the versatility of neural network implementations.

1.2 Background

Recurrent neural networks can be trained to behave like deterministic finite-state automata (DFAs) [4, 6, 9, 11, 25, 26, 31]. The dynamical nature of recurrent networks can cause the internal representation of learned DFA states to deteriorate for long strings [32]; 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 DFAs (state transitions) into recurrent neural networks [12, 24]. The goal was to demonstrate that prior knowledge can decrease the learning time significantly compared to learning without any 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 [19, 27, 30], where the prior knowledge is not only incomplete, but may also be incorrect [13, 22].

Methods for constructing DFAs in recurrent networks where neurons have *hard-limiting* discriminant functions have been proposed [1, 18, 21]. This paper is concerned with neural network implementations of DFAs where *continuous sigmoidal* discriminant functions are used.

Stability of an internal DFA state representation implies that the output of the sigmoidal state neurons assigned to DFA states saturate at high gain; a constructed *discrete-time* network thus has stable periodic orbits. A saturation result has previously been proven for *continuous-time* networks [14]; for sufficiently high gain, the output along a stable limit cycle is saturated almost all the time. There is no known analog of this for stable periodic orbits of discrete-time networks. The only known stability result asserts that for a broad class of discrete-time networks where *all* output neurons are either self-inhibiting or self-exciting, outputs at stable fixed points saturate at high gain [15]. Our proof of stability of an internal DFA state representation establishes such a result for a special case of discrete-time recurrent networks.

Our method is an alternative to an algorithm for constructing DFAs in recurrent networks with first-order weights proposed by Frasconi et al. [6, 7]. 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 [16]. 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 DFAs in recurrent neural networks. For details of the algorithms and the proofs see [6, 7]. The work in [7] is an extension of [6] which restricted the class of automata that could be encoded into recurrent networks to DFAs without cycles (except selfloops). The authors were focusing on automatic speech recognition as an application of implementing DFAs 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:

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

where $S_j^{(t)}$ and $I_k^{(t)}$ represent the output of other state neurons and input neurons, respectively, and V_{ij} and W_{ik} are their corresponding weights. 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 AND and OR 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.

An example of a DFA and its implementation in a recurrent network are shown in figures 1 and 2, respectively. The DFA in figure 1a is modified such that the state vectors of adjacent DFA states have Hamming distance 1. This can be achieved by introducing a temporary state between any two states whose Hamming distance is larger than 1; the code of that temporary state becomes the logical OR of the two original DFA states. Further modifications to the original DFA may be necessary prior to constructing a recurrent network: Consider two mutually consecutive DFA states q_i and q_j with $\delta(q_i, a_k) = q_j$ and $\delta(q_j, a_k) = q_i$; then the introduction of temporary states leads to ambiguity. This ambiguity can be resolved by augmenting the number of DFA states. Applying these modifications to the DFA shown in figure 1a results in the DFA shown in figure 1b.

The recurrent network implementation of that DFA is shown in figure 2. 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.

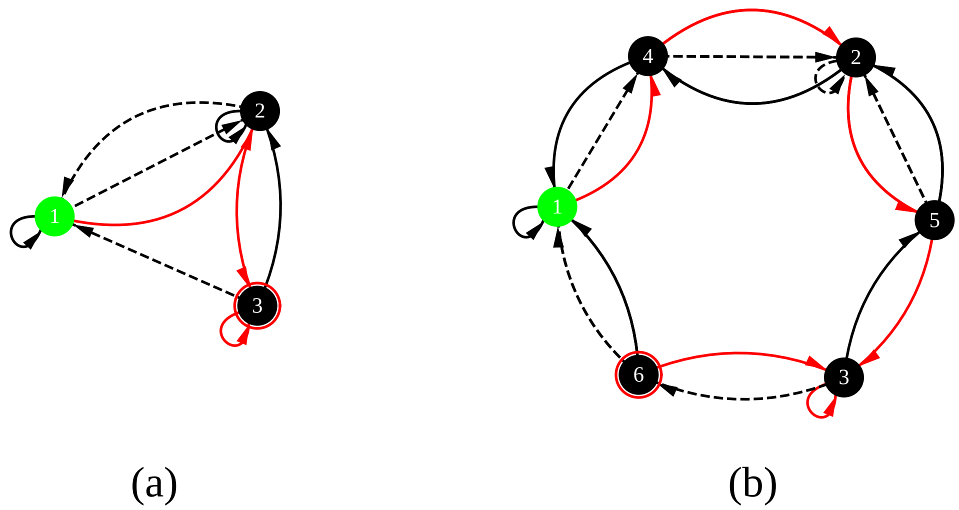

Figure 1: **Example of DFA Modification**: (a) A DFA with 3 input symbols; state 1 is the start state, state 3 is the only accepting state. (b) The original DFA (a) has been modified such that the codes of adjacent DFA states have Hamming distance 1 and any ambiguities due to mutually consecutive states have been removed.

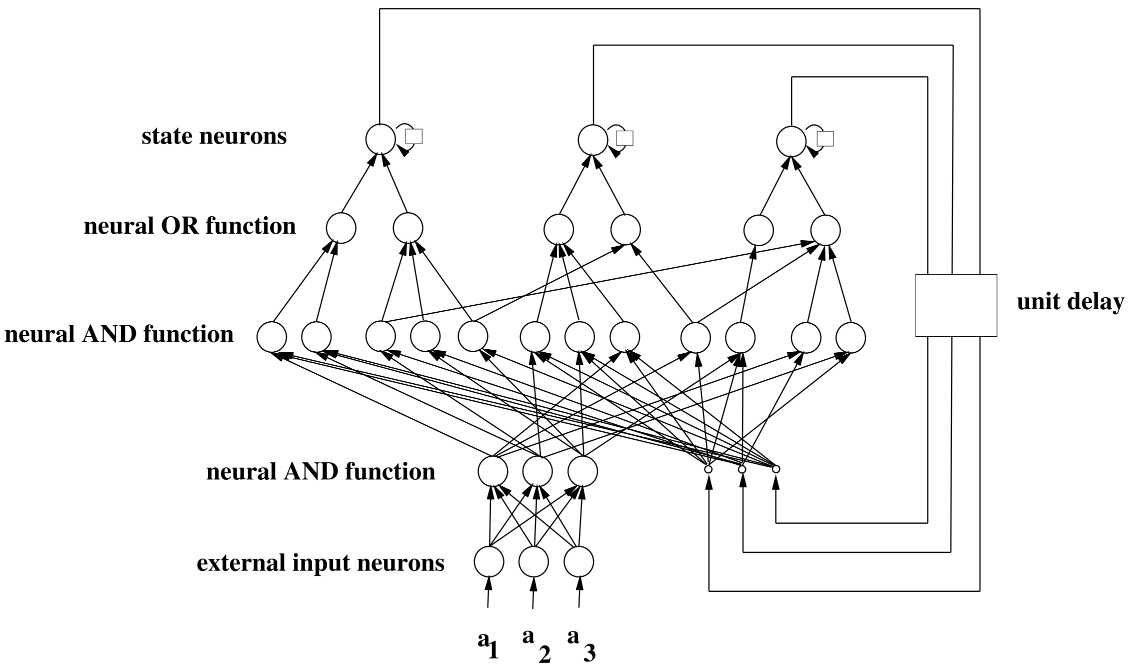

Figure 2: **Example of First-Order Network Construction**: The non-recurrent neurons compute boolean-like AND and OR functions. Signals along recurrent feedback connections are delayed by one time step. The weights are computed by solving linear constraints whose solutions guarantee that the neurons compute boolean-like AND- and OR-functions and desired state changes of the recurrent neurons.

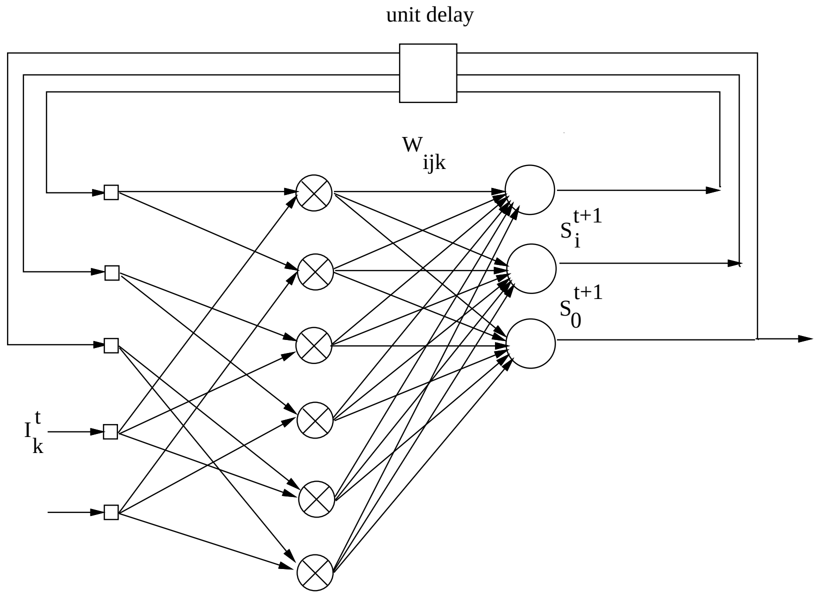

Figure 3: **Second-Order Network**: The external inputs are encoded across the input neurons I_k^t. The values of the state neurons S_j^t are fed back in parallel through a unit delay prior to the presentation of the next input symbol. The weighted sums of products $S_j^t I_k^t$ (denoted by $\otimes$), are fed through a sigmoidal discriminant function $\tau()$ to compute the next network state S_i^{t+1}.

4 SECOND-ORDER NETWORKS

The algorithm used here to construct DFAs in networks with second-order weights has also been used to encode partial prior knowledge to improve convergence time [12, 24], and to perform rule correction [13, 22].

4.1 Network Construction

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

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

where b_i is the bias associated with hidden recurrent state neurons S_i; I_k denotes the input neuron for symbol a_k and W_{ijk} is the corresponding weight (figure 3). 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 (figure 4): 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. Neurons S_j and S_i correspond to DFA states q_j and q_i, respectively. The weights W_{jjk}, W_{ijk}, W_{0jk}, and biases b_i are programmed with weight strength H as follows:

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

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

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

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

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.

With the above DFA encoding algorithm, equation (2) governing the dynamics of a constructed recurrent neural network takes on the special form

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

where $\alpha_i(t-1)$ is the net input to state neuron S_i. For the analysis, it will be convenient to use the same notation $h(.)$ for sigmoidal discriminants with different arguments, i.e. we will use $h(.)$ to stand for the generic sigmoidal discriminant function

$$h(x, H) = \frac{1}{1 + e^{H(1-2x)/2}} \tag{8}$$

We will explicitly state what the arguments in various sections of this paper.

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 deteriorates with increasing string length. This deteriorating performance can be explained by observing

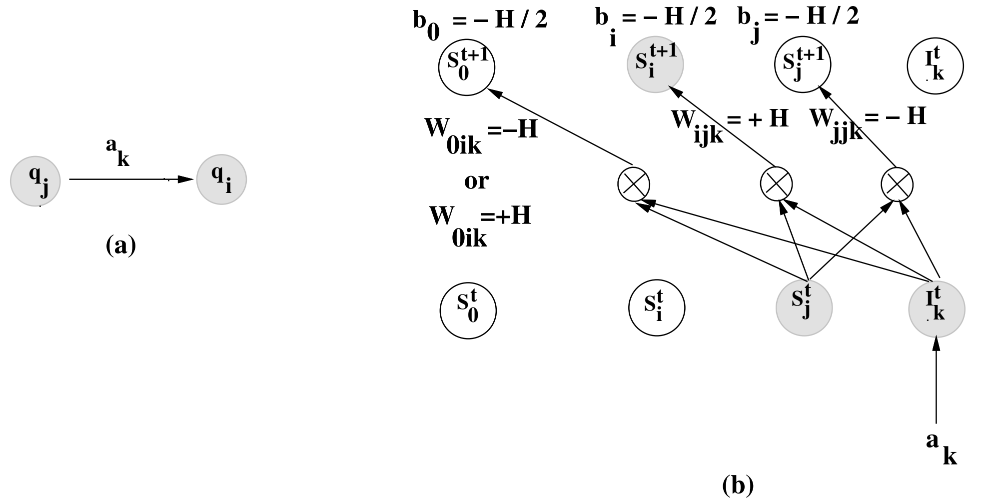

Figure 4: **Encoding of Rules in Second-Order Networks**: (a) A known DFA transition $\delta(q_j, a_k) = q_i$ is programmed into a network. (b) Recurrent network unfolded over two time steps t and $t+1$, i.e. the duration of the state transition $\delta(q_j, a_k) = q_i$. The insertion algorithm consists of two parts: Programming the network state transition and programming the output of the response neuron. Neurons S_i and S_j correspond to DFA states q_i and q_j, respectively; I_k denotes the input neuron for symbol a_k. Programming the weights W_{ijk}, W_{jjk} and biases b_i and b_j as shown in the figure ensures a nearly orthonormal internal representation of DFA states q_i and q_j (active/inactive neurons are illustrated by shaded/white circles). The value of the weight W_{0jk} connected to the response neuron S_0 depends on whether DFA state q_i is an accepting or rejecting state. H denotes the rule strength. The operation $S_i \cdot I_k$ is shown as $\otimes$.

that the internal DFAstate 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 others [3, 29, 32].

We encoded a randomly generated, minimized 100-state DFA with alphabet $\Sigma = \{0, 1\}$ into a recurrent network with 101 state neurons. The graph in figure 5 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.

5 ANALYSIS

Whether or not a *constructed* second-order recurrent network implements a 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

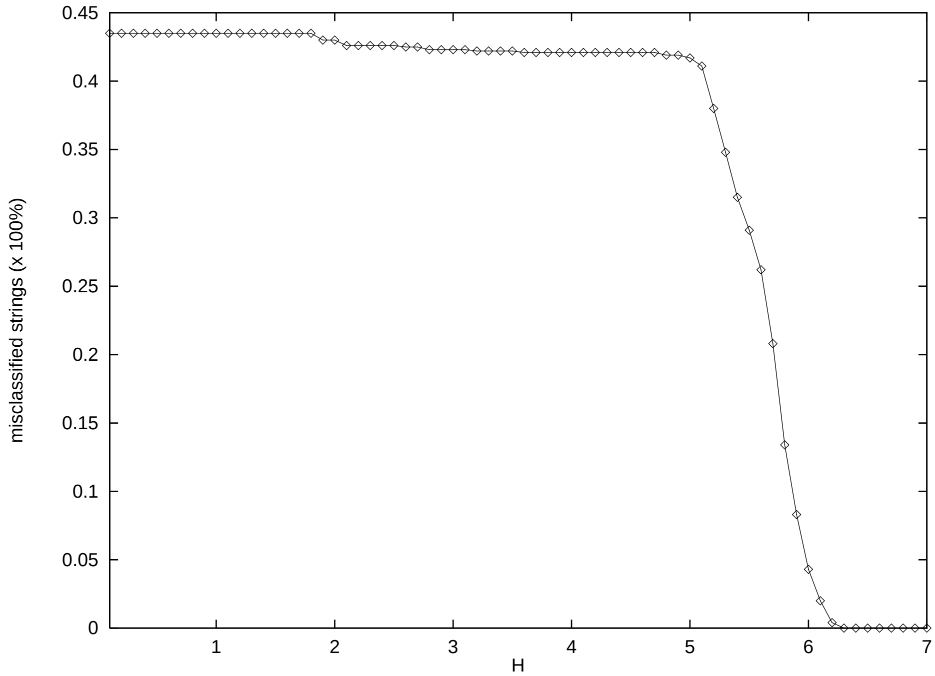

Figure 5: **Network Classification Performance:** The network classification performance on a data set as a function of the rule strength H (in 0.1 increments) is shown. The data set consisted of 1000 randomly chosen strings of length 1000; their labels were assigned by a randomly generated 100-state 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 $H > 6.3$.

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. Our experiment has shown that $H > 6.3$ was sufficient for a particular large network to classify long strings, i.e. the finite-state dynamics remained sufficiently stable and allowed the network to correctly classify strings based on its state after 1,000 time steps. The goal of this analysis is to demonstrate that a network's finite-state dynamics can be made stable with finite weight strength H for strings of arbitrary length and arbitrary DFAs.

5.1 Overview

What follows is a description of the analysis and proofs in this section.

There exist two kinds of signals in a network that implements a given DFA: At any given time step, exactly one recurrent neuron which represents the current DFA state has a high output signal; that high signal drives the output state change of itself and at most one other recurrent neuron at the next time step. Low output signals of all other recurrent neurons act as perturbations on the finite-state dynamics, i.e. on the intended orthonormal internal DFA representation. Thus, we can view the state changes that neurons undergo while a network is processing a string as component-wise iterations of the discriminant function (section 5.2). The key to our DFA encoding algorithm is the use of the sigmoidal discriminant function, in particular its convergence toward stable fixed points under iteration. Thus, we discuss in section 5.3 some relevant properties of the sigmoidal discriminant function.

In order to quantitatively assess the perturbation caused by the low signals, we analyze in section 5.4 all possible neuron state changes, i.e. neuron state changes from low output to high output, high output to high output, high output to low output, and low output to high output. Two of these transition types further subdivide into separate cases for total of six types of neuron state changes. For a worst case analysis, it will suffice to consider only two of the six cases, by observing that stability of the internal DFA state representation for these two cases implies stability of low and high signals for all types of neuron state transitions (section 5.5).

In section 5.6, we derive upper and lower bounds for low and high signals, respectively, for arbitrary string lengths. These bounds are obtained by assuming the worst case of maximal perturbation of the desired finite-state dynamics, i.e. all neurons receive low signals as inputs from all other neurons. The bounds are the two stable fixed points of the sigmoidal discriminant function. The weight strength H must be chosen such that low signals converge toward the low fixed point and high signals converge toward the high fixed point. They represent the worst cases, i.e. low and high signals are usually less and greater than the low and high fixed points, respectively.

In order for a network to implement the desired finite-state dynamics, it will suffice to require that the total neuron input never exceed or fall below a certain threshold for low and high signals, respectively. In section 5.7, we derive conditions in the form of upper and lower bounds for the values of low and high fixed points, respectively, of the discriminant function which guarantee stable finite-state dynamics for arbitrary string length; the former will depend on the network size, whereas the latter is independent of the network size.

In general, the conditions of section 5.7 are too strict, i.e. a network will correctly classify strings of arbitrary length for a much smaller value of the weight strength H. In section 5.8, we relax the worst case condition where each neuron receives inputs from all other neurons by limiting the number of neurons from which all neurons receive inputs. The proof will proceed as in the worst case analysis.

Finally in section 5.9, we discuss the case of fully recurrent networks where only a small subset of weights are programmed to $+H$ or $-H$ and all other second-order weights are initialized to random values drawn from an interval $[-W, W]$ with arbitrary distribution. We give implicit bounds on the size of W for given weight strength H and network size that guarantee correct string classification. A comparison of our DFA encoding algorithm with other methods that have been proposed in the literature and open problems for further research conclude this section.

5.2 Network Dynamics as Iterated Functions

When a network processes a string, the state neurons go through a sequence of state changes. The network state at time t is computed from the network state at time $t-1$, the current input and the weights. Since the discriminant function $h(.)$ is fixed, these network state changes can be represented as iterations of $h(.)$ for each state neuron:

$$S_i^t = h^t(x_i^t, H) \quad = \quad \begin{cases} S_i^0 & t = 0 \\ h(h^{t-1}(x_i^{t-1}, H), H) & t > 0 \end{cases} \tag{9}$$

A network will only correctly classify strings of arbitrary length if its internal DFA state representation remains sufficiently stable. Stability can only be guaranteed if the neurons are shown to operate near their saturation regions for sufficiently high gain of the sigmoidal discriminant function $h(.)$. One way to achieve stability is thus to show that the iteration of the discriminant function $h(.)$ converges toward its fixed points in these regions, i.e. points for which we have i.e. $h(x, H) = x$. This observation will be the basis for a quantitative analysis which establishes bounds on the network size and the weight strength H which guarantee stable internal representation for arbitrary DFAs.

5.3 Properties of Sigmoidal Discriminants

We present some useful properties of the sigmoidal discriminant function $h(x, H) = h(x, H) = \frac{1}{1+e^{H(1-2nx)/2}}$ since this special form of the discriminant will occur throughout the remainder of this paper.

First, we define the concept of fixed points of a function [2]:

Definition 5.3.1 *Let $f : X \rightarrow X$ be a mapping on a metric space (X, d). A point $x_f \in X$ such that $f(x_f) = x_f$ is called a <u>fixed point</u> of the mapping.*

We are interested in a particular kind of fixed point:

Definition 5.3.2 *A fixed point x_f is called <u>stable</u> if there exists an interval $I =]a, b[\in X$ with $x_f \in I$ such that the iteration of f converges toward x_f for any start value $x_0 \in I$.*

Continuous functions $f : X \rightarrow X$ have the following useful property:

Theorem 5.3.1 (Brouwer's Fixed Point Theorem) *: Under a continuous mapping $f : X \rightarrow X$, there exists at least one fixed point.*

Thus, the function $h(.)$ has the following property:

Corollary 5.3.1 *The sigmoidal function $h(.)$ has at least one fixed point.*

The following lemma describes further useful properties of the function $h(.)$:

Lemma 5.3.1 *(1) $h(x,H)$ is monotonically increasing*

(2) $lim_{x\to-\infty}h(x,H)=0$ and $lim_{x\to\infty}h(x,H)=1$

(3) $h'(\frac{1}{2n},H)=\frac{Hn}{4}$

(4) $h'(x,H)<h'(\frac{1}{2n},H)$ for $x\neq\frac{1}{2n}$

Proof:

(1) We have $h'(x,H)=\frac{nHe^{H(1-2x)/2}}{(1+e^{H(1-2x)/2})^2}$ which is positive for any choice of x.

(2) The term $e^{H(1-2x)/2}$ goes to 0 as x goes to ∞. Thus, $h(x,H)$ asymptotically approaches 1. Similarly, $e^{H(1-2x)/2}$ goes to ∞ as x goes to $-\infty$. Thus, $h(x,H)$ asymptotically approaches 0.

(3) Substituting $\frac{1}{2n}$ for x, it follows that $h'(\frac{1}{2n},H)=\frac{Hn}{4}$.

(4) We compute $h''(z,H)$ by computing the derivative of $h'(x,H)$. For reasons of simplicity, we set $z=H(1-2nx)/2$ and obtain

$$h''(z,H)=\frac{nH(e^z-e^3z)}{(1+e^{2z})^2}$$

In order to find the maximum of $h'(z,H)$, we set $h''(z,H)=0$ and obtain $z=0$. Solving for x we find a maximum of $h'(x,H)$ for $x=\frac{1}{2n}$. We will prove the following conjecture in section 5.6:

Conjecture 5.3.1 *There exists a value $H_0^-(n)$ such that for any $H>H_0^-(n)$, $h(.)$ has three fixed points $0<\phi_h^-<\phi_h^0<\phi_h^+<1$.*

The fixed points of $h(.)$ have the following useful property:

Lemma 5.3.2 *If the function $h(x,H)$ has three fixed points $\phi_h^-<\phi_h^0<\phi_h^+$, then the fixed points ϕ_h^- and ϕ_h^+ are stable.*

Proof: The lemma is proven by defining an appropriate Lyapunov function P(.) and showing that $P(.)$ decreases toward the minima ϕ_h^- and ϕ_h^+ under the iteration $h^p(x,H)$ [7]:

Let $P(x_i)=(x_i-\phi_h^+)^2$. It follows that ΔP decreases only if x_i approaches the fixed point ϕ_h^+. To see this, we compute

$$\Delta P=P(h(x_i,H))-P(x_i)=(h(x_i,H)-\phi_h^+)^2-(x_i-\phi_h^+)^2=(h(x_i,H)+x_i-2\phi_h^+)(h(x_i,H)-x_i).$$

If $x_i > h(x_i, H) > 0$, then $x_i > \phi_h^+$. Therefore, the first factor is positive and consequently $\Delta P < 0$. Conversely, if $0 < x_i < h(x_i, H)$ then $x_i < \phi_h^+$ and $h(x_i, H) < \phi_h^+$. Therefore, the first factor is negative and we have $\Delta P < 0$ again. Hence, the stability of ϕ_h^+ follows for each $x_i \in (\phi_h^0, \infty)$. A similar argument can be made for the stability of the other fixed point ϕ_h^-.

For the remainder of this paper, we are mainly interested in the two stable fixed points. The following lemma concerning stable fixed points will be useful:

Lemma 5.3.3 *A point x is a stable fixed point of a continuous function $f : X \to X$ if and only if $|f'(x)| < 1$.*

Proof: Let ϕ_f be a stable fixed point of some continuous function f. Choose some value x_0 sufficiently close to ϕ_f. By definition of the stable fixed point ϕ_f, we have $\lim_{t\to\infty} f^t(x_0) = \phi_f$. However, that is only possible if the distance $|x_t - \phi|$ decreases monotonically under the iteration of f which requires $|f'(\phi_f - \varepsilon)| < 1$ and $|f'(\phi_f + \varepsilon)| < 1$ for an arbitrary small ε-neigborhood of ϕ_f. Conversely, with $|f'(\phi_f - \varepsilon)| < 1$ and $|f'(\phi_f + \varepsilon)| < 1$, the distance $|x_t - \phi|$ decreases monotonically under the iteration of f; in the limit, the fixed point ϕ_f is reached.

The above lemma has the following corollary:

Corollary 5.3.2 *The iteration of the function $h(.)$ converges monotonically to one of its fixed points ϕ_h.*

Proof: Since ϕ_h^- and ϕ_h^+ are stable fixed points of the function $h(.)$, we have $\left|h'(\phi_h^-)\right| < 1$ and $\left|h'(\phi_h^+)\right| < 1$. Furthermore since $h(.)$ is a monotone continuous function, we have $0 < h'(\phi_h^-) < 1$ and $0 < h'(\phi_h^+) < 1$. This precludes the possibility of alternating convergence toward the fixed point ϕ_h. According to lemma 5.3.2, the $\Delta P(x)$ only decreases if the iteration $h(x)$ approaches a fixed point ϕ_h. This concludes the proof of the corollary.

The following lemma concerning convergence behavior of $h(.)$ will be useful:

Lemma 5.3.4 *If the function $h(.)$ has three fixed points, then the iteration $h^0, h^1, h^2, \ldots$ of the function $h(.)$ converges to the following fixed points:*

$$\lim_{t\to\infty} h^t = \begin{cases} \phi_h^- h^0 < \phi_h^0 \\ \phi_h^0 h^0 = \phi_h^0 \\ \phi_h^+ h^0 > \phi_h^0 \end{cases} \tag{10}$$

Proof: We assume that $h(.)$ has three fixed points. Convergence toward ϕ_h^0 for $h^0 = \phi_h^0$ is trivial since ϕ_h^0 is a fixed point of $h(.)$. Since $h(.)$ is a bounded function with $\lim_{x\to-\infty} h(.) = 0$ and $\lim_{x\to\infty} h(.) = 1$ (lemma 5.3.1), it follows that $x > h(x, H)$ for $x \in]-\infty, \phi_h^-] \cup]\phi_h^0, \phi_h^+[$ and $x < h(x, H)$ for $x \in]\phi_h^-, \phi_h^0[\cup]\phi_h^+, \infty[$.

According to corollary 5.3.2,, iteration of $h(.)$ converges monotonically to one of its fixed points. Thus, the iteration $h^0, h^1, h^2, \ldots$ of $h(.)$ has to converge toward ϕ_h^- and ϕ_h^+ for $h^0 < \phi_h^0$ and $h^0 > \phi_h^0$, respectively.

5.4 Quantitative Analysis of Network Dynamics

For the remainder of this discussion, we will use S_i and S_i^t to denote the neuron corresponding to DFA state q_i and the output value (or signal) of neuron S_i, respectively. Under the assumption that all neurons operate near their saturated regions, each neuron can send two kinds of signals to other neurons:

(1) High signals: If neuron S_i^t represents the current DFA state q_i, then S_i^t will be high ($S_\cdot^t$: high).

(2) Low signals; Neurons S_j^t which do not represent the current DFA state have a low output ($S_\cdot^t$: low).

Recall that the arguments of the discriminant function $h(x, H)$ were the sum of unweighted signals x and the weight strength H. We now expand the term x to account for the different kinds of signals that are present in a neural DFA.

We now define a new function $h_\Delta(x_i, H)$ which takes the residual inputs into consideration. let Δx_i denote the residual neuron inputs to neuron S_i^t. Then, the function $h_\Delta(x_i, H)$ is recursively defined as

$$h_\Delta^t(x_i^t, H) \quad = \quad \begin{cases} 0 & t = 0 \\ h(h_\Delta^{t-1}(x_i^{t-1}, H) + \Delta x_i^t, H) & t > 0 \end{cases} \qquad (11)$$

The initial values for low and high signals are $x_i = 0$ and $x_i = 1$, respectively.

The magnitude of the residual inputs Δx_i depend on the coupling between recurrent state neurons. Neurons which are connected to a large number of other neurons will receive a larger residual input than neurons which are connected to only a few other neurons. Consider the neuron S_m which receives a residual input Δx_m from the most number r of neurons, i.e. $\Delta x_i \leq \Delta x_m$. In order to show network stability, it suffices to assume the worst case where all neurons receive the same amount of residual input for given time index t, i.e. Δx_m^t. This assumption is valid since the initial value for all neurons except the neuron corresponding to a DFA's start state is 0.

Consider the DFA state transition $\delta(q_j, a_k) = q_i$; let neurons S_i and S_j correspond to DFA states q_i and q_j, respectively, and assume that $\delta(q_i, a_k) \neq q_i$. There may be other states $q_l \in \{q_{l_1}, \ldots, q_{l_m}\}$ which have q_i as their successor state (figure 8a): $\delta(q_l, a_k) = q_i$. Thus, neurons S_l are connected to neuron S_i via weights $W_{ilk} = H$ and neuron S_i is connected to itself via weight $W_{iik} = -H$. Since all network signals S_i^t lie in the interval $]0, 1[$, neuron S_i receives input not only from neuron S_j, but also small, but not negligible inputs from neurons S_l at the time the network executes the DFA state transition $\delta(q_j, a_k) = q_i$. Signals S_i^t and S_j^t are low and high, respectively, prior to executing $\delta(q_j, a_k) = q_i$; after execution, S_i^{t+1} will be a high signal, whereas S_j^{t+1} will be a low signal. Thus, the equation governing the neuron state change *low* $S_i^t \rightarrow$ *high* S_i^{t+1} can be expressed as:

$$S_i^{t+1} = h(S_j^t + \sum_{S_l \in C_{i,k}} S_l^t - S_i^t, H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{12}$$

where

$$C_{i,k} = \{S_l \mid W_{ilk} = H, l \neq i, l \neq j\} \tag{13}$$

Notice the term $-S_i^t$ which weakens the high signal S_i^{t+1}. At the same time, the terms S_l^t strengthen the high signal if there exist DFA state transitions $\delta(q_l, a_k) = q_i$. For reasons of clarity, we simplified the products $S_j^t \cdot I_k^t$ to S_j^t since we have $I_x^t = \delta_{xk}$ where δ denotes the Kronecker delta.

For the case shown in figure 8b where there exists a DFA transition $\delta(q_i, a_k) = q_i$ (self-loop) which is not the DFA state transition that the network currently executes, everything remains the same as above except that neuron S_i is connected to itself via weight $W_{iik} = H$. This leads to a slightly different form of the equation governing the neuron state change *low* $S_i^t \rightarrow$ *high* S_i^{t+1}:

$$S_i^{t+1} = h(S_j^t + \sum_{S_l \in C_{i,k}} S_l^t + S_i^t, H) \quad (S_j^t : high; S_i^t, S_l^t : low) \tag{14}$$

For the case where the current DFA state transition is $\delta(q_i, a_k) = q_i$ (self-loop, figure 8c), the output of neuron S_i remains high; the equation governing the neuron state change *high* $S_i^t \rightarrow$ *high* S_i^{t+1} becomes

$$S_i^{t+1} = h(S_i^t + \sum_{S_l \in C_{i,k}} S_l^t, H) \quad (S_i^t : high, S_l^t : low) \tag{15}$$

The neuron S_j will change from a high signal S_j^t to a low signal S_j^{t+1} when the network executes the DFA state transition $\delta(q_j, a_k) = q_i$ with $q_j \neq q_i$ (figure 8d). Thus, neuron S_j has necessarily a self-connecting weight $W_jjk = -H$. The equation governing the neuron state change *high* $S_j^t \rightarrow$ *low* S_j^{t+1} then becomes

$$S_j^{t+1} = h(-S_j^t + \sum_{S_l \in C_{i,k}} S_l^t, H) \quad (S_j^t : high, S_l^t : low) \tag{16}$$

Under the assumption of a nearly orthonormal internal DFA state representation, the neurons S_l with $l \neq i, l \neq j$ will undergo state changes *low* $S_l^t \rightarrow$ *low* S_l^{t+1}. These neurons may also receive residual inputs from other neurons with low output signals. According to whether neurons S_l have self-connections programmed to H or $-H$, the equations governing neuron state changes *low* $S_l^t \rightarrow$ *low* S_l^{t+1} become

$$S_l^{t+1} = h(S_l^t + \sum_{S_{l'} \in C_{i,k}} S_{l'}^t, H) \quad (S_l^t, S_{l'}^t : low) \tag{17}$$

$$S_l^{t+1} = h(-S_l^t + \sum_{S_{l'} \in C_{i,k}} S_{l'}^t, H) \quad (S_l^t, S_{l'}^t : low) \tag{18}$$

The above equations account for all possible contributions to the net input of all state neurons.

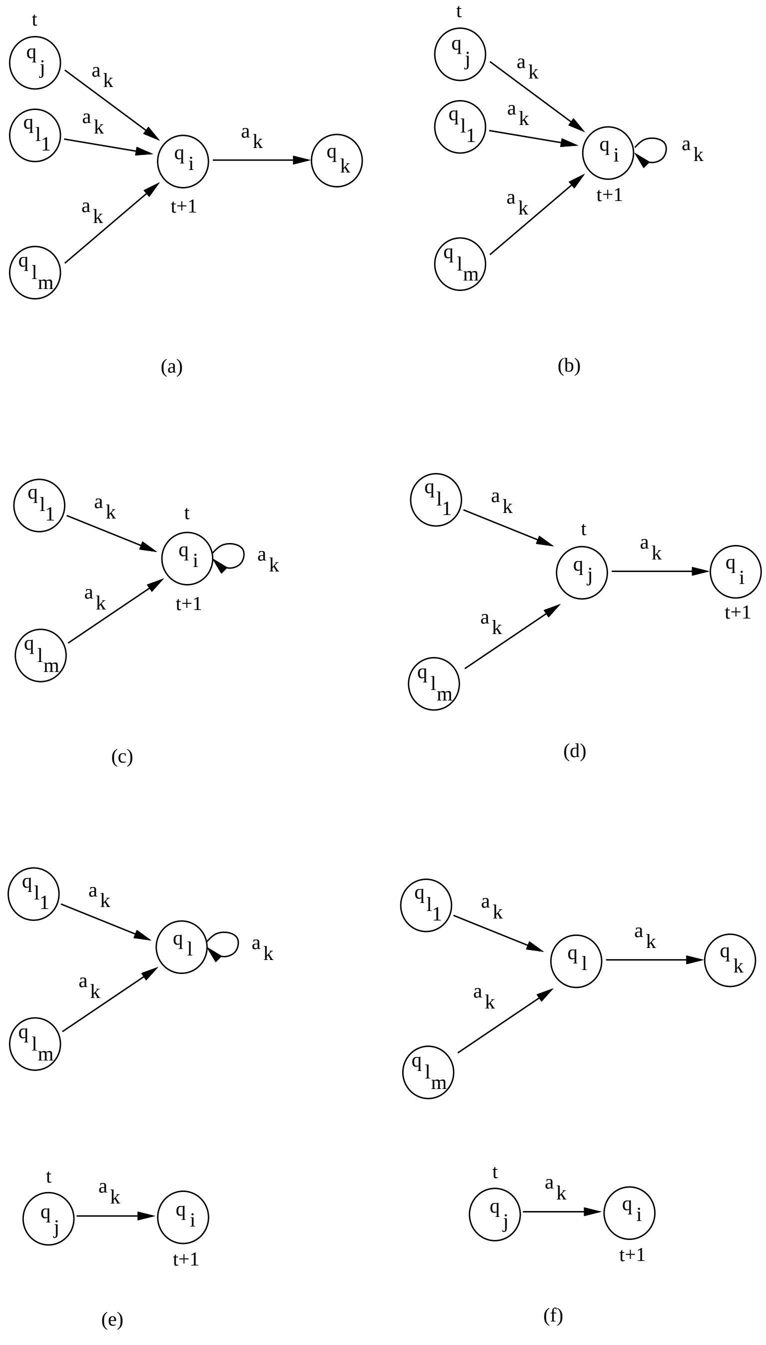

Figure 6: **Neuron State Changes and Corresponding DFA State Transitions**: The figure (a)-(f) illustrate the DFA state transitions corresponding to all possible state changes of neuron S_i; the DFA state(s) participating in the current transitions are marked with t and $t+1$. (a) $low \rightarrow high$ (no self-loop on q_i) (b) $low \rightarrow high$ (with self-loop on q_i) (c) $high \rightarrow high$ (necessarily a self-loop on q_i) (d) $high \rightarrow low$ (necessarily no self-loop on q_i) (e) $low \rightarrow low$ (with self-loop on q_l) (f) $low \rightarrow low$ (no self-loop on q_l). Notice that, even though state q_i is neither the source nor the target of the current state transition in cases (e) and (f), the corresponding state neuron S_i still receives residual inputs from state neurons $S_{l_1}, \ldots, S_{l_m}$.

5.5 Simplifying Observations

We will make some observations regarding the equations (12)-(18) which will simplify the stability analysis for recurrent networks.

For the remainder of this discussion, it will be convenient to use the terms *principal* and *residual inputs*:

Definition 5.5.1 *Let S_i be a neuron with low output signal S_i^t and S_j be a neuron with high output signal S_j^t. Furthermore, let $\{S_l\}$ and $\{S_{l'}\}$ be sets of neurons with output signals $\{S_l^t\}$ and $\{S_{l'}^t\}$, respectively, for which $W_{ilk} \neq 0$ and $W_{il'k} \neq 0$ for some input symbol a_k and assume $S_j \in \{S_l\}$. Then, neurons S_i and S_j receive principal inputs of opposite signs from neuron S_j and residual inputs from all other neurons S_l and $S_{l'}$, respectively, when the network executes the state transition $\delta(q_j, a_k) = q_i$.*

Notice that neuron S_i may receive a residual input signal S_i^t if there exists a DFA state transition $\delta(q_i, a_k) = q_i$ which is not the currently executed transition. Equations (12)-(18) clearly show that principal inputs are the high signals that drive network state changes whereas residual inputs are the low signals that perturb these ideal network state changes and can lead to the instability of DFA encodings.

Consider equations (12) and (14) governing neuron state changes *low* $S_i^t \rightarrow$ *high* S_i^{t+1}. In both equations, the principal inputs and the terms and the number of terms in the two sums are identical. They only differ with respect to the sign of the residual input S_i^t. Thus, the high signal S_i^{t+1} in equation (12) will be weaker than S_i^{t+1} in equation (14). Hence, if neurons can uphold the condition for stable DFA encodings under state change equation (12), they can certainly also satisfy that condition under state change equation (14) and no separate analysis is necessary.

Now consider the case where neurons undergo state changes *high* $S_i^t \rightarrow$ *high* S_i^{t+1} (equation (15)). Under the stated assumptions, that equation is identical with equation (14) except for the index of the high signal. Thus, the above argument also applies to equation (15).

Now consider the case where neurons undergo state changes *high* $S_i^t \rightarrow$ *low*, S_i^{t+1} (equation (16)). That equation is identical with equation (18) except that S_i^t is smaller than S_l^t (they are both negative) and thus results in a stronger low signal S_i^{t+1} than in equation (18). Thus, stability of low signals for state change equation (18) implies stability of low signals for equation (16)).

Consider equations (18) and (17). In both equations, the terms and the number of terms in the two sums are again identical, but the residual inputs S_i^t have opposite signs. Since the residual input S_i^t is negative in equation (18), the low signal S_i^{t+1} will be stronger than the low signal S_i^{t+1} in equation (17). Hence, if neurons can uphold the condition for stable DFA encodings under state change equation (17), they can certainly also satisfy that condition under state change equation (18) and no separate analysis is necessary.

Thus, the stability analysis for all possible neuron state changes reduces to analyzing stability under neuron state changes governed by the following two equations:

low $S_i^t \rightarrow$ *low* S_i^{t+1}:

$$S_i^{t+1} = h(S_i^t + \sum_{S_l \in C_{i,k}} S_l^t, H) \quad (S_i^t, S_l^t : low) \tag{19}$$

low $S_i^t \rightarrow$ *high* S_i^{t+1}:

$$S_i^{t+1} = h(S_j^t + \sum_{S_l \in C_{i,k}} S_l^t - S_i^t, H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{20}$$

5.6 Worst Case Analysis

In order to show network stability, it suffices to assume the worst case where all neurons receive the same amount of residual input for given time index t, i.e. Δx^t.

We can quantify Δx^t for the case of low signals as follows:

Lemma 5.6.1 *The low signals are bounded from above by the fixed point* ϕ_f^- *of the function* f

$$\begin{cases} f^0 = 0 \\ f^{t+1} = h(n \cdot f^t) \end{cases} \tag{21}$$

i.e. we have $\Delta x_i^{t+1} = n \cdot f^t$ since $x_i^0 = 0$ for low signals in equation (13).

Proof: We prove the lemma by induction on t. For $t = 1$, we have $h_\Delta^0(x_i^0, H) = 0 = f^0$ since the initial output of all neurons except the neuron corresponding to the initial DFA state is equal to 0. This establishes the basis of the induction.

Assume the hypothesis is correct for $t > 1$, i.e. all neurons with low outputs have value f^t. To see that the hypothesis holds for $t+1$, we observe that the input to all neurons which execute a state transition of type *low* $\rightarrow$ *low* during time step $t+1$ is in the worst case equal to n times the values of the low signals at the current time step t which is f^t. Hence the output of all neurons which do not correspond to the target DFA state of the DFA state transition at time $t+1$ is equal to

$$h_\Delta^{t+1} = h(h_\Delta^t(x_i^t, H) + \Delta x_i^t, H) = h(n \cdot h_\Delta^t(x, H), H) = h(n \cdot f^t)$$

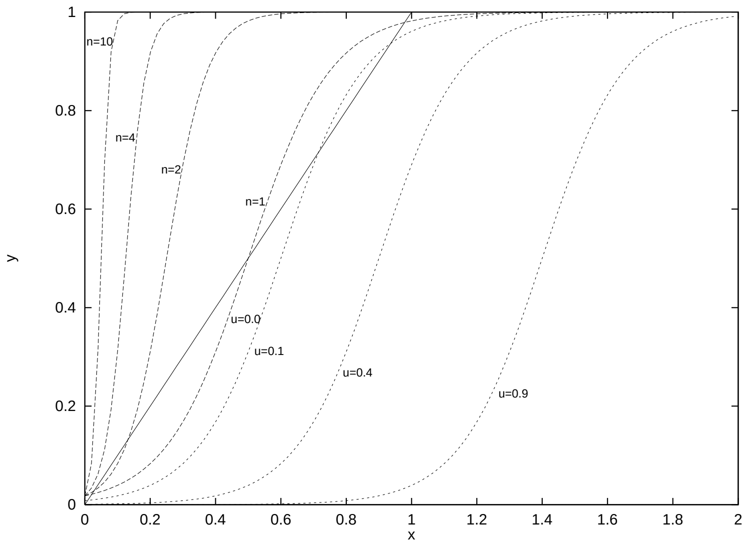

Figure 7: **Fixed Points of the Sigmoidal Discriminant Function**: Shown are the graphs of the function $f(x,n) = \frac{1}{1+e^{H(1-2nx)/2}}$ (dashed graphs) for $H = 8$ and $n = \{1,2,4,10\}$ and the function $p(x,u) = \frac{1}{1+e^{H(1-2(x-u))/2}}$ (dotted graphs) for $H = 8$ and $u = \{0.0, 0.1, 0.4, 0.9\}$. Their intersection with the function $y = x$ shows the existence and location of fixed points. In this example, $f(x,n)$ has three fixed points for $n = \{1,2\}$, but only one fixed point for $n = \{4,10\}$ and $p(x,u)$ has three fixed points for $u = \{0.0, 0.1\}$, but only one fixed point for $u = \{0.6, 0.9\}$.

since the principal input $h^t_\Delta(x^t_i, H)$ to neurons whose output remains low is equal to zero. This concludes the proof of the lemma.

It remains to be shown that, for given n, there exists a value $H > H_0^-(n)$ which makes Δx sufficiently small such that $f^t(x,H)$ converges toward its fixed points ϕ_f^-. If we choose $H < H_0^-(n)$, then $f^t(x,H)$ converges toward its only fixed point ϕ_f^+; in this case, the nearly orthonormal internal DFA state representation is no longer maintained and, as a consequence, such a network will generally misclassify strings.

It is easy to see that the function to be iterated in equation (20) is $f(x,n) = \dfrac{1}{1+e^{(H/2)(1-2nx)}}$. The graphs of the function are shown in figure 7 for different values of the parameter r. With these properties, we can quantify the value $H_0^-(n)$ such that for any $H > H_0^-(n)$, $f(x,n)$ has two stable fixed points. The low and high fixed points ϕ_f^- and ϕ_f^+ will be the bounds for low and high signals, respectively. The larger n, the larger H must be chosen in order to guarantee the existence of two stable fixed points. If H is not chosen sufficiently large, then f^t converges to a unique fixed point $0.5 < \phi_f < 1$. The following lemma expresses a quantitative condition which guarantees the existence of two stable fixed points:

Lemma 5.6.2 *The function* $f(x,n) = \dfrac{1}{1+e^{(H/2)(1-2nx)}}$ *has two stable fixed points* $0 < \phi_f^- < \phi_f^+ < 1$ *if* H *is chosen such that*

$$H > H_0^-(n) = \frac{2(1 + (1-x)\ log(\frac{1-x}{x}))}{1-x}$$

where x *satisfies the equation*

$$n = \frac{1}{2x(1 + (1-x)\ log(\frac{1-x}{x}))}$$

The contour plots in figure 7 show the relationship between H and x for various values of n. If H is chosen such that $H > H_0(n)$, then two stable fixed points exist; otherwise, only a single stable fixed point exists.

Proof: Fixed points of the function $f(x, n)$ satisfy the equation $\frac{1}{1+e^{(H/2)(1-2nx)}} = x$. Given the parameter n, we must find a minimum value $H_0^-(n)$ such that $f(x, n)$ has three fixed points. We can think of x, n, and H as coordinates in a three-dimensional Euclidean space. Then the locus of points (x, n, H) satisfying the relation

$$f(x, n) = \frac{1}{1 + e^{(H/2)(1-2nx)}} \tag{22}$$

is a curved surface. What we are interested in is the number of points where a line parallel to the x-axis intersects this surface.

Unfortunately, equation (21) cannot be solved explicitly for x as a function of n and H. However, it can be solved for either one of the other parameters, giving the intersections with lines parallel to the n-axis or the H-axis:

$$n = n(x, H) = \frac{1}{2x} - \frac{log(\frac{1-x}{x})}{Hx} \tag{23}$$

$$H = H(n, x) = \frac{2\ log(\frac{1-x}{x})}{1 - 2nx} \tag{24}$$

The contours of these functions show the relationship between H and x when n is fixed (figure 7). We need to find the point on each contour where the tangent is parallel to the x-axis, which will indicate where the transition occurs between one and three solutions for $f(n, x) = x$. Solving $\frac{\partial n(x,H)}{\partial x} = 0$, we obtain the conditions of the lemma.

The number and the location of fixed points depends on the values of n and H. Thus, we write $\phi_f^-(n, H)$, $\phi_f^0(n, H)$, and $\phi_f^+(n, H)$, to denote the stable low, the unstable, and the stable high fixed point, respectively.

Similarly, we can quantify high signals in a constructed network:

Lemma 5.6.3 *The high signals are bounded from below by the fixed point ϕ_g^+ of the function*

$$\begin{cases} g^0 = 1 \\ g^{t+1} = h(g^t - f^t) \end{cases} \tag{25}$$

Proof: For the basis of the induction proof, we note that the only neuron which has a high signal at time $t = 0$ is the neuron corresponding to the DFA's start state; its value is initialized to $g^0 = 1$.

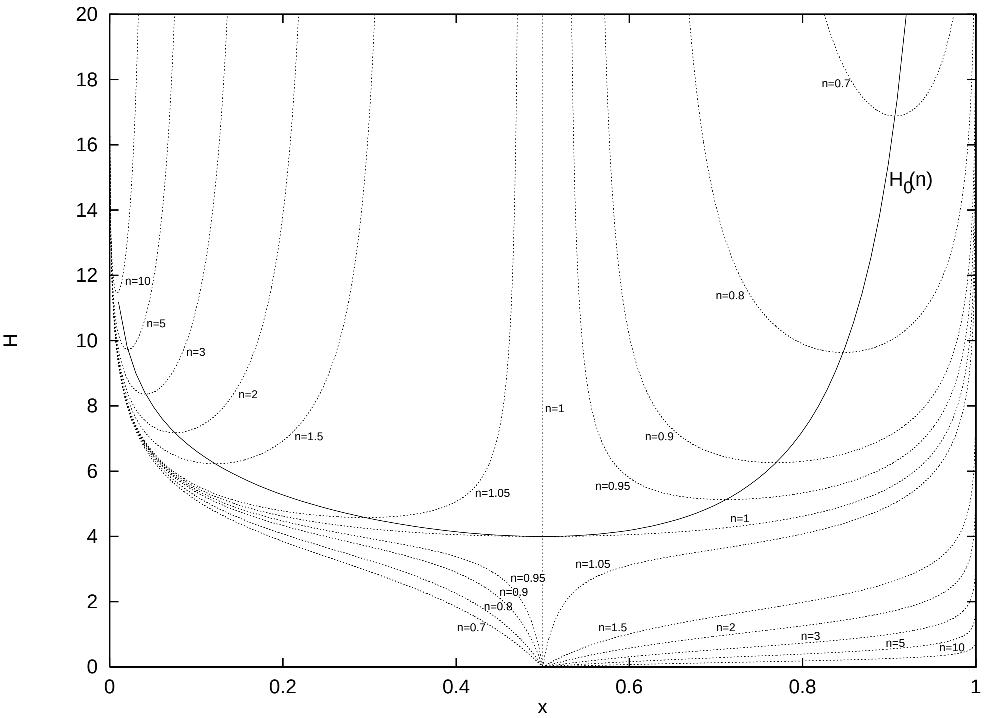

Figure 8: **Existence of Fixed Points**: The contour plots of the function $f(x,n) = x$ (dotted graphs) show the relationship between H and x for various values of n. If H is chosen such that $H > H_0(n)$ (solid graph), then a line parallel to the x-axis intersects the surface satisfying $f(x,n) = x$ in three points which are the fixed points of $h(x,n)$. For illustration purposes, contour plots for positive fractions are also shown even though n can only assume positive integer values in the context of DFA encoding in recurrent neural networks.

Assuming the high signal at time step t is equal to g^t, we observe that the neuron whose output is to be driven high on the next time step $t+1$ receives in the worst case a high input signal S_j^t weighted by $+H$ and and a low signal S_i^t weighted by $-H$. Thus, the input to a neuron undergoing a state transition *low* $\rightarrow$ *high* receives at time $t+1$ in the worst case the input $g^t - f^t$. Since the iteration f^t converges toward a fixed point ϕ_f, the sequence $g^0 > g^1 > g^2 > \ldots$ corresponding to the high signals at subsequent time steps converges monotonically toward a fixed point ϕ_g. This concludes the proof of this lemma.

Notice that the above recurrence relation couples f^t and g^t which makes it difficult if not impossible to find a function g(x, n) which when iterated gives the same values as g^t. However, we can bound the sequence $g^0, g^1, g^2, \ldots$ from below by a recursively defined function $\tilde{g}^t$ - i.e. $\forall t : \tilde{g}^t \leq g^t$ - which decouples g^t from f^t.

Lemma 5.6.4 *Let ϕ_f denote the fixed point of the recursive function f, i.e. $\lim_{t\to\infty} f^t = \phi_f$. Then the recursively defined function $\tilde{g}$*

$$\begin{cases} \tilde{g}^0 = 1 \\ \tilde{g}^{t+1} = h(\tilde{g}^t - \phi_f) \end{cases} \tag{26}$$

has the property that $\forall t : \tilde{g}^t \leq g^t$.

It is obvious that the function $\tilde{g}(x,u) = \dfrac{1}{1 + e^{H(1-2(x-u))/2}}$ is being iterated in equation (25). The graph of the function $\tilde{g}(x,u)$ for some values of u are shown in figure 7. The lemmas and corollaries of section 5.3 also apply to the function $\tilde{g}(x,u)$.

Proof: We prove this lemma by induction on t. For $t = 0$, we have $\tilde{g}^0 = g^0$. Let the induction hypothesis be true for t, i.e. $\forall t_i \leq t : \tilde{g}^t \leq g^t$. By the induction hypothesis, we have

$$1 - 2(\tilde{g}^t - \phi_f) \geq 1 - 2(g^t - \phi_f) \tag{27}$$

However, $f^t \leq \phi_f$ since the sequence $f^0, f^1, f^2, \ldots$ is monotonically increasing (corollary 5.3.2) . Thus, we observe that

$$1 - 2(g^t - \phi_f) \geq 1 - 2(g^t - f^t) \tag{28}$$

It thus follows that $\tilde{g}^{t+1} \leq g^{t+1}$. By the induction principle, we thus have $\forall t : \tilde{g}^t \leq g^t$.

The following corollary to lemma 5.3.2 becomes useful:

Corollary 5.6.1 *The function $h(x,H) = \dfrac{1}{1 + e^{H(1-2x)/2}}$ has three fixed points for $H > 4$.*

Proof: This represent the special case for $n = 1$. From lemma 5.3.1, it follows that $h'(0.5, H) = \dfrac{H}{4} > 1$ for $H > 4$. Thus, $h(.)$ crosses the function $y = x$ three times and thus as three fixed points two of which are stable (lemma 5.3.2).

The following lemma establishes a useful relationship between the functions $g(x, n)$ and $\tilde{g}(x, u)$:

Lemma 5.6.5 *Let the function $\tilde{g}(x, u)$ have two stable fixed points and let $\forall t : \tilde{g}^t \leq g^t$. Then the function $g(x, n)$ has also two stable fixed points.*

Proof: We have established in lemma 5.6.4 that $\tilde{g}^t \leq g^t$. Because the sigmoidal function h is monotonically increasing and bounded, it also follows that $g^t \leq h(.)$. Thus we have $\tilde{g}^t \leq g^t \leq h(.)$, i.e. the function $g()$ is bounded from below and above by $\tilde{g}$ and $h(.)$, respectively. Since $h(.)$ has two stable fixed points for any $H > 4$, it follows that $g(.)$ also has two stable fixed points if $\tilde{g}$ has two stable fixed points.

Since we have decoupled the iterated function g^t from the iterated function f^t by introducing the iterated function $\tilde{g}^t$, we can apply the same technique for finding conditions for the existence of fixed points of $\tilde{g}(x, u)$ as in the case of f^t. In fact, the function that when iterated generates the sequence $\tilde{g}^0, \tilde{g}^1, \tilde{g}^2, \ldots$ is defined by

$$\tilde{g}(x, u) = \tilde{g}(x, \phi_f^-) = \frac{1}{1 + e^{(H/2)(1-2(x-\phi_f^-))}} = \frac{1}{1 + e^{(H'/2)(1-2n'x))}} \tag{29}$$

with

$$H' = H(1 + 2\phi_f^-), \;\; n' = \frac{1}{1 + 2\phi_f^-} \tag{30}$$

Since we can iteratively compute the value of ϕ_f for given parameters H and r, we can repeat the original argument with H' and n' in place of H and r to find the conditions under which $\tilde{g}(n, x)$ and thus $g(n, x)$ have two stable fixed points. This results in the following lemma:

Lemma 5.6.6 *The function $\tilde{g}(x, \phi_f^-) = \dfrac{1}{1 + e^{(H/2)(1-2(x-\phi_f^-))}}$ has three fixed points $0 < \phi_{\tilde{g}}^- < \phi_{\tilde{g}}^0 < \phi_{\tilde{g}}^+ < 1$ if H is chosen such that*

$$H > H_0^+(n) = \frac{2(1 + (1-x)\, log(\frac{1-x}{x}))}{(1 + 2\phi_f^-)(1-x)}$$

where x satisfies the equation

$$\frac{1}{1 + 2\phi_f^-} = \frac{1}{2x(1 + (1-x)\, log(\frac{1-x}{x}))}$$

In practice, only few neurons ever exceed or fall below the fixed points ϕ^- and ϕ^+, respectively. Furthermore, the network has a built-in reset mechanism which allows low and high signals to be strengthened. Low signals S_j^t are strengthened to $g(-H/2)$ when there exists no state transition $\delta(., a_k) = q_j$. In that case, the neuron S_j^t receives no inputs from any of the other neurons; its output becomes less than ϕ^- since $g(-H/2) = h(0, H) < \phi^-$. Similarly, high signals S_i^t get strengthened if either low signals feeding into neuron S_i on a current state transition $\delta(\{q_j\}, a_k) = q_i$ have been strengthened during the previous time step or when the number of positive residual inputs to neuron S_i compensates for a weak high signal from neurons

$\{q_j\}$. Thus only a small number of neurons will have $S_j^t > \phi^-$ or $S_j^t < \phi^+$. For the majority of neurons we have $S_j^t \leq \phi^-$ and $S_i^t \geq \phi^+$ for low and high signals, respectively. Since constructed networks are able to regenerate their internal signals and since typical DFAs do not have the worst case properties assumed in this analysis, the conditions guaranteeing stable low and high signals are generally much too strong for some given DFA.

5.7 Network Stability

We now define stability of recurrent networks constructed from DFAs:

Definition 5.7.1 *An encoding of DFA states in a second-order recurrent neural network is called* <u>*stable*</u> *if all the low signals are less than* $\phi_f^0(n, H)$*, and all the high signals are greater than* $\phi_g^0(n, H)$*.*

We have established in the previous section that the fixed points ϕ_f^- and ϕ_g^+ are the upper and lower bounds of low and highs signals, respectively. However, the bounds only hold if each neuron receives total input which does not exceed or falls below the values ϕ_f^0 and ϕ_g^0, respectively (lemma 5.3.4).

Consider equation (18). In order for the low signal to remain less than ϕ_f^0, the argument of $h(.)$ must be less than ϕ_f^0 for all values of t. Thus, we require the following invariant property of the residual inputs for state transitions of the type $low \rightarrow low$:

$$-\frac{H}{2} + H\, n\, \phi_f^- < \phi_f^0 \tag{31}$$

where we assumed that all low signals have the same value and that their maximum values is the fixed point ϕ_f^-. This assumption is justified since the output of all state neurons with low values are initialized to zero.

A similar analysis can be carried out for state transitions of equation (19). The following inequality must be satisfied for :

$$-\frac{H}{2} + H\phi_g^+ - H\phi_f^- > \phi_g^0 \tag{32}$$

where we assumed that there is only one DFA transition $\delta(q_j, a_k) = q_i$ for chosen q_i and a_k, and thus $\sum_{S_l \in C_{i,k}} = 0$.

Solving inequalities (30) and (31) for ϕ_f^- and ϕ_g^+, respectively, we obtain conditions under which a constructed recurrent network implements a given DFA. These conditions are expressed in the following theorem:

Theorem 5.7.1 *For some given DFA M with n states and m input symbols, a sparse recurrent neural network with $n+1$ sigmoidal state neurons and m input neurons can be constructed from M such that the internal state representation remains stable if the following three conditions are satisfied:*

$$\text{(1)} \quad \phi_f^-(n,H) < \frac{1}{n}\left(\frac{1}{2} + \frac{\phi_f^0(n,H)}{H}\right)$$

$$\text{(2)} \quad \phi_g^+(n,H) > \frac{1}{2} + \phi_f^-(n,H) + \frac{\phi_g^0(n,H)}{H}$$

$$\text{(3)} \quad H > max(H_0^-(n), H_0^+(n))$$

Furthermore, the constructed network has at most $3mn$ *second-order weights with alphabet* $\Sigma_w = \{-H, 0, +H\}$*,* $n+1$ *biases with alphabet* $\Sigma_b = \{-H/2\}$*, and maximum fan-out* $3m$*.*

The number of weights and the maximum fan-out follow directly from the DFA encoding algorithm.

Stable encoding of DFA state is a necessary condition for a neural network to implement a given DFA. The network must also correctly classify all strings. The conditions for correct string classification are expressed in the following corollary:

Corollary 5.7.1 *Let* $L(M_{DFA})$ *denote the regular language accepted by a DFA* M *with* n *states and let* $L(M_{RNN})$ *be the language accepted by the recurrent network constructed from* M*. Then, we have* $L(M_{RNN}) = L(M_{DFA})$ *if*

$$\text{(1)} \quad \phi_g^+(n,H) > \frac{1}{2}\left(1 + \frac{1}{n} + \frac{2\,\phi_g^0(n,H)}{H}\right)$$

$$\text{(2)} \quad H > max(H_0^-(n), H_0^+(n))$$

Proof: For the case of an ungrammatical strings, the input to the response neuron S_0 must satisfy the following condition:

$$-\frac{H}{2} - H\phi_g^+ + (n-1)H\phi_f^- < \frac{1}{2} \tag{33}$$

where we have made the usual simplification about the convergence of the outputs to the fixed points ϕ_f^- and ϕ_g^+. Furthermore, we assume that the state q_i of the state transition $\delta(q_j, a_k) = q_i$ is the only rejecting state; then the output neuron's residual inputs from all other state neurons is positive, weakening the intended low signal for the network's output neuron. Notice that the output neuron is the only neuron which can be forced toward a low signal by neurons other than itself.

A similar condition can be formulated for grammatical strings:

$$-\frac{H}{2} + H\phi_g^+ - (n-1)H\phi_f^- > \frac{1}{2} \tag{34}$$

The above two inequalities can be simplified into a single inequality:

$$-2H\phi_g^+ + 2(n-1)H\phi_f^- < 0 \tag{35}$$

Observing that $\phi_f^- + \phi_g^+ < 2$ and solving for ϕ_f^-, we get the following condition for the correct output of a network:

$$\phi_f^- < \frac{2}{n} \tag{36}$$

Thus we have the following conditions for stable low signals and correct string classification:

$$\phi_f^- \quad < \quad \begin{cases} \frac{1}{n}\ (\frac{1}{2} + \frac{\phi_f^0(n,H)}{H}) & \text{(dynamics)} \\ \\ \frac{2}{n} & \text{(classification)} \end{cases} \tag{37}$$

We observe that

$$\frac{1}{n}(\frac{1}{2} + \frac{\phi_f^0}{H}) > \frac{1}{2n}$$

Choosing $\phi_f^- < \dfrac{1}{2n}$ thus implies the condition for stable low signals. Substituting $\frac{1}{2n}$ for ϕ_f^- in inequality (36) yields condition (1) of the corollary.

5.8 Analysis for Partially Recurrent Networks

Although the DFA encoding algorithm constructs a recurrent network with sparse interconnections, the above analysis was carried out for a fully interconnected network where each neuron receives residual inputs from all other neurons, causing maximal perturbation to the stability of the internal DFA representation. As a result, the condition for stable low signals is rather strict, i.e. we may empirically find that a constructed network remains stable for values of the weight strength H which is considerably smaller than the value predicted by the theory. The question of how H scales with network size is important. The empirical results in [23] indicate that $H \approx 6$ for randomly generated DFAs independent of the size of the DFA. However, for DFAs where there exists one or several states q_i with a large number of states q_j for which $\delta(q_j, a_k) = q_i$ for some a_k, the value of H scales with the size of the DFA.

We will now show how the results of the above stability analysis can be improved by considering the stability of networks with sparse interconnections. While the results for stable finite-state dynamics of the constructed networks are similar, the result for correct classification of strings becomes independent of the weight value H.

Before we analyze the conditions for stable low and high signals, we introduce the following notations: Let D_{ik} denote the number of states q_j that make transitions to state q_i for input symbol a_k. We further define $D_i = max\{D_{ik}\}$ (maximum number of transitions to q_i over all input symbols) and $D = max\{D_i\}$ (maximum number of transitions to any state over all input symbols). Then, $\rho = D/n$ denotes the maximum fraction of all states q_j for which $\delta(\{q_j\}, a_k) = q_i$.

The analysis of the existence of fixed points of section 5.6 obviously also applies to the case of partially recurrent networks with $D = \rho n$ instead of n as the parameter of the sigmoidal function $h(.)$.

Thus, for the case of partially recurrent neural networks, we have the following conditions for stable finite-state dynamics:

Theorem 5.8.1 *For some given DFA M with n states and m input symbols, let D denote the maximum number of transitions to any state over all input symbols of M, and let $\rho = D/n$. Then, a sparse recurrent neural network with $n+1$ sigmoidal state neurons and m input neurons can be constructed from M such that the internal state representation remains stable if the following conditions are satisfied:*

$$\text{(1)} \quad \phi_f^-(\rho n, H) < \frac{1}{n}\left(\frac{1}{2} + \frac{\phi_f^0(\rho n, H)}{H}\right)$$

$$\text{(2)} \quad \phi_g^+(\rho n, H) > \frac{1}{2} + \phi_f^-(\rho n, H) + \frac{\phi_g^0(\rho n, H)}{H}$$

$$\text{(3)} \quad H > max(H_0^-(\rho n), H_0^+(\rho n))$$

Furthermore, the constructed network has at most $3mn$ second-order weights with alphabet $\Sigma_w = \{-H, 0, +H\}$ $(H > 4)$, $n+1$ biases $b_i = -H/2$, and maximum fan-out $3m$.

Stable encoding of DFA state is a necessary condition for a neural network to implement a given DFA. The network must also correctly classify all strings. The conditions for correct string classification are expressed in the following corollary:

Corollary 5.8.1 *For some given DFA M with n states, let D denote the maximum number of transitions to any state voer all input symbols of M, and let $\rho = D/n$. Let $L(M_{DFA})$ and $L(M_{RNN})$ denote the regular languages accepted by DFA M and a recurrent neural network constructed from M. Then, we have $L(M_{RNN}) = L(M_{DFA})$ if*

$$\text{(1)} \quad \phi_g^+(\rho n, H) > \frac{1}{2}\left(1 + \frac{1}{n} + \frac{2\,\phi_g^0(\rho n, H)}{H}\right)$$

$$\text{(2)} \quad H > max(H_0^-(\rho n), H_0^+(\rho n))$$

Proof: As in the worst case analysis, we obtain the following conditions for stable dynamics and correct string classification:

$$\phi_f^- \quad < \quad \begin{cases} \dfrac{1}{\rho n}\left(\dfrac{1}{2} + \dfrac{\phi_f^0(\rho n, H)}{H}\right) & \text{(dynamics)} \\ \\ \dfrac{2}{n} & \text{(classification)} \end{cases} \tag{38}$$

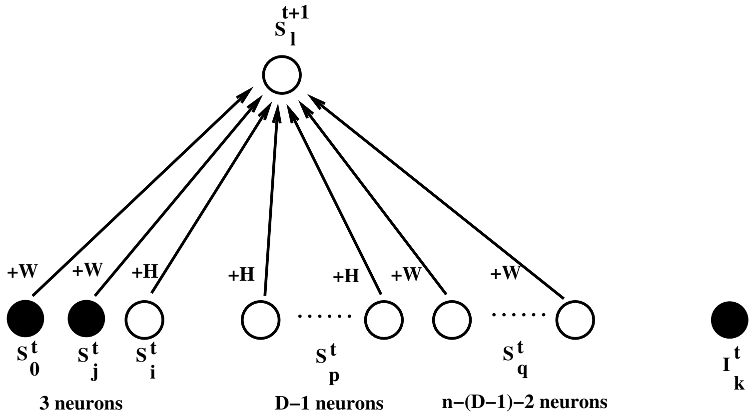

Figure 9: **Preservation of Low Signals:** The figure shows the input fed into neuron S_l^{t+1} from all other neurons for a chosen input symbol. For clarity, the operation $S_j \cdot I_k$ is omitted.

We observe that

$$\frac{1}{\rho n}(\frac{1}{2} + \frac{\phi_f^0}{H}) > \frac{1}{2\rho n} > \frac{1}{2n}$$

Choosing $\phi_f^- < \dfrac{1}{2n}$ thus implies the condition for stable low signals in partially recurrent networks. Substituting $\frac{1}{2n}$ for ϕ_f^- in inequality (37) yields condition (1) of the corollary.

5.9 Analysis for Fully Recurrent Networks

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 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; these weights can be interpreted as *noise* on the neural DFA encoding. The following definition of noisy recurrent networks will be convenient:

Definition 5.9.1 *A noisy fully recurrent, second-order recurrent network (or just noisy network) is a constructed network where all weights w_{ijk} that are not programmed to either $+H$ or $-H$ are initialized to random values drawn from an interval $[-W, W]$ according to some distribtion.*

Notice that the above definition leaves open the possibility that the network is much larger than required for the implementation of a particular DFA. For the remainder of this section, we assume that the smallest possible network is to be constructed, i.e. there are no neurons which are not needed for the DFA encoding. Furthermore, we make the technical assumption that $H > W$; this is reasonable since the neural DFA encoding algorithms uses the weight strength H to achieve the encoding.
An analyis similiar to that of sections 5.6 and 5.7 will examine conditions under which a noisy recurrent network can implement a given DFA. Unlike in the case of sparse network, the special response neuron S_0 also drives the output of other neurons and every recurrent neuron receives residual inputs from all other neurons.

We will now derive conditions for the preservation of low and high signals in fully recurrent networks.

Consider a DFA transition $\delta(q_j, a_k) = q_i$. All state neurons S_l which do not correspond to DFA state q_i should be low signals (figure 9). Assuming the current state is an accepting state, neuron S_0 has a high output signal and is weighted by $+W$. Neuron S_j has a high output since it corresponds to the current DFA state q_j; it is also weighted by $+W$. Neuron S_i is low since we are dealing with state transitions of type *low* $S_i^t \rightarrow$ *low* S_l^{t+1} only; it has a weight $+H$. There are $D-1$ neurons with low outputs and weights $+H$. The remaining $n-(D-1)-2$ neurons also have low outputs and are weighted by $+W$. Thus, the worst case expression for the net input to neuron S_l becomes:

$$-H/2 + WS_0^t + WS_j^t + HS_i^t + H(D-1)S_p^t + W(n-(D-1)-2)S_q^t \tag{39}$$

We assume that all neurons which are affected by a transition of type *low* $\rightarrow$ *low* in receive in the worst case the same residual input and thus will have equal output signals f_W^t for all t. Similiarly, we will assume that the high output signals of the neuron corresponding to the current DFA state and the signal of a network's output neuron are identical and equal to g_W^t. We can then rewrite equation (38) as

$$-H/2 + Wg_W^t + Wg_W^t + Hf_W^t + H(D-1)f_W^t + W(n-(D-1)-2)f_W^t. \tag{40}$$

Similiarly, we express the worst case for a state transition of type *low* $\rightarrow$ *high*: For a DFA state transition $\delta(q_j, a_k) = q_i$, state neuron S_j has a high output signal S_j^t and S_i should change its output from a low signal S_i^t to a high signa S_i^{t+1}. In the worst case, neuron S_i receives the following net input:

$$-H/2 - WS_0^t + HS_j^t - WS_i^t + H(D-1)S_p^t - W(n-(D-1)-2)S_q^t \tag{41}$$

With the same assumption as above where high signals of the neuron corresponding to the current target state and the network's output neuron have the same value g_W^t, we can rewrite the above equation as

$$-H/2 - Wg_W^t + Hg_W^t - Hf_W^t + H(D-1)f_W^t - W(n-(D-1)-2)f_W^t \tag{42}$$

Since network state changes can be interpreted as iterated functions (section 5.2), we can give upper and lower bounds on the low and high signals, respectively, in a noisy fully recurrent network as follows:

Lemma 5.9.1 *The low and high signals f_W^t and g_w^t, respectively, in a noisy neural network are bounded from below and above, respectively, by the fixed points of the recursively defined functions:*

$$\begin{cases} f_W^0 = 0 \\ f_W^{t+1} = h(-H/2 + 2Wg_W^t + HDf_W^t + W(n-D-1)f_W^t)) \end{cases} \tag{43}$$

$$\begin{cases} g_W^0 = 1 \\ g_W^{t+1} = h(-H/2 + (H-W)g_W^t + H(D-2)f_W^t - W(n-D-1)f_W^t) \end{cases} \tag{44}$$

The induction proof of the above lemma is similar to the proof of lemmas 5.6.1 and 5.6.2, i.e. we assume the worst case where all neurons receive identical residual inputs. As in section 5.2, the existence of three fixed points for the functions f_W^t and g_w^t are a necessary condition for the stability of DFA encodings in noisy recurrent networks. However, the above equations couple the low and high signals f_W^t and g_W^t, respectively. This makes a fixed point analysis of these two functions impossible. Therefore, we will make use of the following lemma which will allow us to decouple the iterations of low and high signals and thus carry out the analysis:

Lemma 5.9.2 *Consider the recursively defined function $\tilde{f}_W^t$:*

$$\begin{cases} \tilde{f}_W^0 = 0 \\ \tilde{f}_W^{t+1} = h(-H/2 + 2W + HDf_W^t + W(n-D-1)f_W^t)) \end{cases} \tag{45}$$

If the function $\tilde{f}_W^t$ has two stable fixed points, then so does the function f_W^t.

Proof: The proof is the same as in lemma 5.6.3 and we will not repeat it here. We use the notation $\tilde{f}_W(x, n, D, W, H)$ to denote the function being iterated in this lemma. We note the relationship

$$h(x, H) = \frac{1}{1 + e^{H(1-2x)/2)}} \leq f_W() \leq \tilde{f}_W() \tag{46}$$

where equality holds for $D = 1$ and $W = 0$. Since $h(.)$ has two stable fixed points for $H > 4$ (see 5.6.1), it thus follows that $f_w()$ has two stable fixed points if $\tilde{f}_W$ has two stable fixed points.

Thus, we can analyze the existence of fixed points of the function f_W^t indirectly by examining the existence of fixed points of the function $\tilde{f}_W^t$ which is independent of the function g_W^t. Notice that we use the function $\tilde{f}_W^t$ to examine the existence of fixed points of f_W^t; for the analysis of network stability, we refer to the fixed point f_W^t.

Fixed points of the function $\tilde{f}_W(x, n, D, W, H)$ satisfy the equation

$$\tilde{f}_W(x, n, D, W, H) = \frac{1}{1 + e^{-(-H/2+2W+(D(H-W)+W(n-1))x}} = x \tag{47}$$

Solving the above equation for H, we obtain

$$H(x, n, D, W) = 2\ \frac{log(\frac{1-x}{x}) + xW(n-D) + W(2-x)}{1-2xD} \tag{48}$$

Solving $\frac{\partial H}{\partial x)} = 0$ for n and substituting in the above equation, we have proven the following lemma:

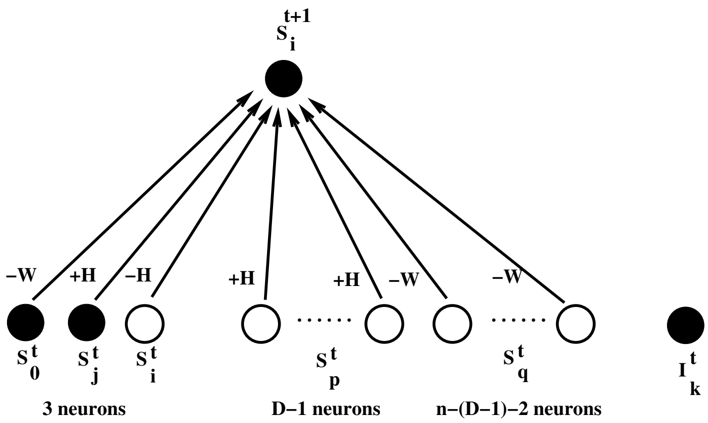

Figure 10: **Preservation of High Signals:** The figure shows the input fed into neuron S_i^{t+1} from all other neurons for a chosen input symbol. For clarity, the operation $S_j \cdot I_k$ is not shown.

Lemma 5.9.3 *The function* $\tilde{f}_W(x,n,D,W,H) = \dfrac{1}{1+e^{-(-H/2+2W+(D(H-W)+W(n-1))x)}}$ *has three fixed points* $0 < \phi^-_{f_W} < \phi^0_{f_W} < \phi^+_{f_W} < 1$ *if* H *is chosen such that*

$$H > H^-_W(n) = \frac{2(1+W-2x(1-x)\ log(\frac{1-x}{x}))}{1-x}$$

where x *satisfies the equation*

$$n = \frac{1+Wx(3D(x-1)-x)-2xD(1+(1-x)log(\frac{1-x}{x})}{Wx(1-x)}$$

We perform a similiar analysis for high signals in noisy recurrent networks. Recall that the high signals in a noisy recurrent network are bounded from below by the fixed points of the recursively defined function

$$\begin{cases} g^0_W = 1 \\ g^{t+1}_W = h(-H/2+(H-W)g^t_W + H(D-2)f^t_W - W(n-D-1)f^t_W) \end{cases} \tag{49}$$

We decouple g^{t+1}_W from f^t_W by introducing a new function $\tilde{g}_W$:

Lemma 5.9.4 *Consider the recursively defined function* $\tilde{g}_W$*:*

$$\begin{cases} \tilde{g}^0_W = 1 \\ \tilde{g}^{t+1}_W = h(-H/2+(H-W)\tilde{g}^t_W + (H(D-2)-W(n-D-1))\phi^+_{\tilde{f}^t_W})) \end{cases} \tag{50}$$

where $\phi^+_{\tilde{f}^t_W}$ *is the fixed point of the function* $\tilde{f}_W$*. If the function* $\tilde{g}^t_W$ *has three fixed points, then so does the function* g^t_W*.*

Proof: An argument similar to that in the proof of lemma 5.6.3 can be given. The only difference is that $h()$ and $\tilde{g}_W$ may both assume the roles of either lower or upper bound for g_W depending on the value of D.

In order to obtain conditions under which g_W has three fixed points, we examine the existence of fixed points of $\tilde{g}_W$.

Fixed points of the function $\tilde{g}_W(x, n, D, W, H)$ satisfy the equation

$$\tilde{g}_W(x, n, D, W, H) = \frac{1}{1 + e^{-(-H/2 + (H-W)x + (H(D-2) - W(n-D-1))\phi^+_{\tilde{g}_W})}} = x \tag{51}$$

Solving the above equation for H, we obtain

$$H(x, n, D, W) = 2\,\frac{log(\frac{1-x}{x}) - W(\phi^+_{\tilde{g}_W}(n-D-1) + x) +_{\tilde{g}_W} (n-D))}{1 - 2(x + 2\phi^+_{\tilde{g}_W}(D-2))} \tag{52}$$

Solving $\frac{\partial H}{\partial x)} = 0$ for n and substituting in the above equation, we have proven the following lemma:

Lemma 5.9.5 *The function* $\tilde{g}_W(x, n, D, W, H) = \dfrac{1}{1 + e^{-(-H/2 + (H-W)x + (D(H-W) + W(n-1))\phi^+_{\tilde{g}_W})}}$ *has three fixed points* $0 < \phi^-_{\tilde{g}_W} < \phi^0_{\tilde{g}_W} < \phi^+_{\tilde{g}_W} < 1$ *if* H *is chosen such that*

$$H > H^+_W(n) = \frac{1 + W(x(1-x))}{x(1-x}\big)$$

where x *satisfies the equation*

$$n = \frac{1 + 2D\phi^+_{\tilde{g}_W} + x(W(2\phi^+_{\tilde{g}_W}(1-x) - x) - 2) - 2x(1-x)log(\frac{1-x}{x})}{2Wx(1-x)\phi^+_{\tilde{g}_W}}$$

5.10 Network Stability

We now turn to the analysis of stable DFA encodings for noisy recurrent networks:

Definition 5.10.1 *An encoding of DFA states in a noisy, second-order recurrent neural network is called* <u>*stable*</u> *if all the low signals are less than* $\phi^0_{f_W}(n, D, W, H)$*, and all the high signals are greater than* $\phi^0_{g_W}(n, D, W, H)$.

Assuming that all signals converge toward their respective fixed points, we get the following inequalities for stable low and high signals, respectively:

$$-H/2 + W\phi^+_{g_W} + W\phi^+_{g_W} + H\phi^-_{f_W} + H(Dn-1)\phi^-_{f_W} + W(n - (Dn-1) - 2)\phi^-_{f_W} < \phi^0_{f_W} \tag{53}$$

$$-H/2 - W\phi^+_{g_W} + H\phi^+_{g_W} - H\phi^-_{f_W} + W(Dn-1)\phi^-_{f_W} - W(n - (Dn-1) - 2)\phi^-_{f_W} > \phi^0_{g_W} \tag{54}$$

Solving the above inequalities for $\phi^-_{f_W}$ and $\phi^+_{g_W}$, respectively, we obtain the following result:

Theorem 5.10.1 *A noisy, fully recurrent neural network RNN with $n+1$ sigmoidal state neurons, m input neurons, at most $3mn$ second-order weights with alphabet $\Sigma_w = \{-H, 0, +H\}$, $n+1$ biases $b_i = -H/2$, 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 if*

(1) $\phi_{f_W}^{-} < \dfrac{2(H-W)\phi_{f_W}^{0} - 4W\phi_{g_W}^{0} + H(H-3W)}{2n((1-3D)W^2 + H((1-2D)W + HD)) + W(H+W)}$

(2) $\phi_{g_W}^{+} > \dfrac{2(H + n(W(1-4D)\phi_{f_W}^{0} + (2W(n(1-D)-1) + nDH)\phi_{g_W}^{0} + H(H(1+Dn) + W(n(2-3D)-1))}{2n((1-3D)W^2 + H((1-2D)W + HD)) + W(H+W)}$

(3) $H > max(H_W^{-}(D), H_W^{+}(D))$

Stability of the internal DFA state encoding in noisy recurrent networks for the correct classification of strings of arbitrary length. We also need to examine the conditions under which the output of the network is correct. However, correct labeling of rejecting and accepting network states in noisy recurrent networks is no different from the case of fully recurrent networks. The following inequalities which represent a worst case must be satisfied for correct classification of grammatical and ungrammatical strings, respectively:

$$-\frac{H}{2} + H\phi_{g_W}^{+} - (n-1)H\phi_{f_W}^{-} > \frac{1}{2} \tag{55}$$

$$-\frac{H}{2} - H\phi_{g_W}^{+} + (n-1)H\phi_{f_W}^{-} < \frac{1}{2} \tag{56}$$

These two equations can be simplified which yields the condition $\phi_{f_W}^{-} < \frac{2}{n}$ for correct string classification.

Thus, we have the following corollary for noisy recurrent networks:

Corollary 5.10.1 *For some given DFA M with n states, let D denote the maximum number of transitions to any state voer all input symbols of M, and let $\rho = D/n$. Let $L(M_{DFA})$ and $L(M_{RNN})$ denote the regular languages accepted by DFA M and a noisy recurrent neural network constructed from M. Then, we have $L(M_{RNN}) = L(M_{DFA})$ if*

(1) $\phi_{f_W}^{-} < \dfrac{2(H-W)\phi_{f_W}^{0} - 4W\phi_{g_W}^{0} + H(H-3W)}{2n((1-3D)W^2 + H((1-2D)W + HD)) + W(H+W)}$ (network dynamics)

(2) $\phi_{g_W}^{+} > \dfrac{2(H + n(W(1-4D)\phi_{f_W}^{0} + (2W(n(1-D)-1) + nDH)\phi_{g_W}^{0} + H(H(1+Dn) + W(n(2-3D)-1))}{2n((1-3D)W^2 + H((1-2D)W + HD)) + W(H+W)}$ (network dynamics)

(3) $\phi_{f_W}^{-} < \frac{2}{n}$ (string classification)

(4) $H > max(H_W^{-}(D), H_W^{+}(D))$ (existence of fixed points)

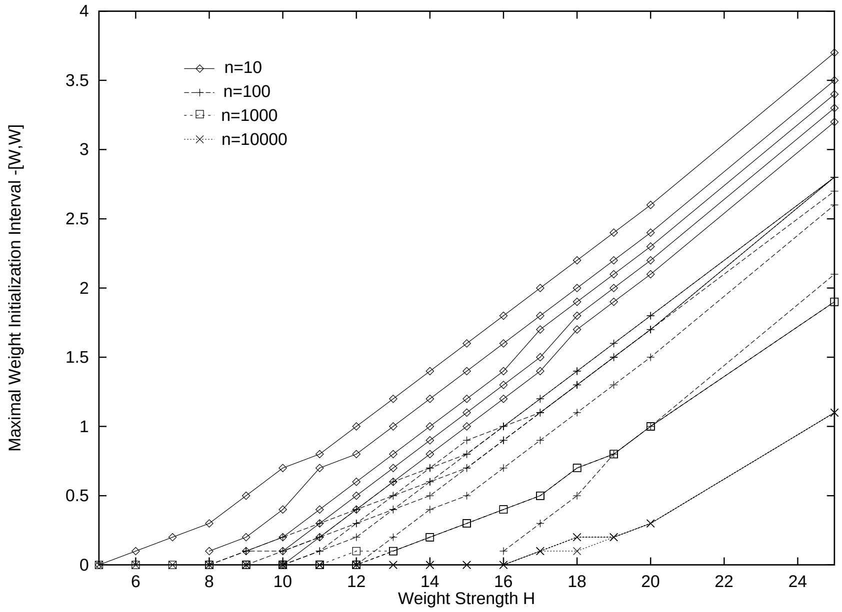

Figure 11: **Maximal Random Weight Initialization Interval:**
The curves show for different network sizes n the maximal allowable interval $[-W, W]$ from which values of the randomly initialized weights can be drawn as a function of H and ρ. The values for ρ_n were $\rho_{10} = \{0.1, 0.2, 0.3, 0.4, 0.5\}, \rho_{100} = \{0.01, 0.02, 0.03, 0.04\}, \rho_{1000} = \{0.001, 0.002, 0.003, 0.004\}$, and $\rho_{10000} = \{0.0001, 0.0002, 0.0003, 0.0004\}$.

The graphs in figure 11 show for different network sizes n the maximal allowable interval $[-W, W]$ from which values of the randomly initialized weights can be drawn as a function of H such that the internal DFA representation of constructed recurrent networks remains stable for strings of arbitrary length. The values of ρ_n for networks of size n were $\rho_{10} = \{0.1, 0.2, 0.3, 0.4, 0.5\}, \rho_{100} = \{0.01, 0.02, 0.03, 0.04\}, \rho_{1000} = \{0.001, 0.002, 0.003, 0.004\}$, and $\rho_{10000} = \{0.0001, 0.0002, 0.0003, 0.0004\}$. The conditions of theorem 5.9.1 where no longer satisfied for larger values of ρ_n. As expected, the size of the interval $[-W, W]$ increases with increasing value of the weight strength H for given network size n, i.e. the network tolerates more noise from the randomly initialized weights with increasing H. That interval becomes independent of the values of ρ and H for increasing network size. For fixed values of H, that interval becomes smaller with increasing network size, since the number of random weights which interfere with the internal DFA state representation also increases.

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.

All conditions in theorem 5.9.1 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 conditions is violated. In fact, we believe that the following conjecture holds true:

Conjecture 5.10.1 *If any one of the 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* $[-W, W]$.

5.11 Comparison with other Methods

Different methods [1, 7, 5, 18, 21] for encoding DFAs with n states and m input symbols in recurrent networks are summarized in table 1. 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 imposed on the weight alphabet, the neuron fan-in and fan-out. The results in [18] improve the upper and lower bounds reported in [1] for DFAs with only two input symbols. Those bounds can be generalized to DFAs with m input symbols [17]. 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.

5.12 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 DFAs with only $\log n$ neurons; this can be shown on small example DFAs.

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 (1996)[2]	hard	first	$O(\sqrt{\frac{mn\log n}{\log m+\log n}})$	-	no restriction	none	none
Horne (1996)[2]	hard	first	$O(\sqrt{mn\log n})$	$O(mn\log n)$	$\Sigma_W = \{-1,1\}$	none	none
Horne (1996)[2]	hard	first	$O(\frac{mn\log n}{\log m+\log n})$	$O(n)$	$\Sigma_W = \{-1,1,2\}$	2	none
Frasconi et al. (1996)[3]	sigmoid/radial	first	$O(n)$	$O(n^2)$	no restriction	none	none
Omlin & Giles[4]	sigmoid	second	$O(n)$	$O(mn)$	$\Sigma_W = \{-H,-H/2,+H\}$	none	$3m$

Table 1: **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 this paper.

6 CONCLUSION

We compared two different methods for encoding deterministic finite-state automata (DFAs) into recurrent neural networks with sigmoidal discriminant functions. The method proposed in [7] implements DFAs 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 DFAs. 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. This is only a proof of existence, i.e. we do not make any claims that such a solution can be *learned*. For any chosen value $H > 4$, 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 can be computed for a given DFA.

Our algorithm for constructing DFAs in recurrent neural networks is more straightforward compared to the method proposed in [7]. 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 [7]. The network model has not lost any of its computational capabilities by the introduction of second-order weights.

We are currently investigating how other kinds of knowledge which may be useful in building hybrid systems can be represented in second-order recurrent neural networks.

7 ACKNOWLEDGMENT

We would like to acknowledge useful discussions with D. Handscomb (Oxford University Computing Laboratory), and J.A. Giordmaine and B.G. Horne (NEC Research Institute).

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] M. Casey, "The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction," *Neural Computation*, vol. 8, no. 6, 1996. In Press.

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

[5] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite state automata in recurrent radial basis function networks," *Machine Learning*, 1996. In press.

[6] 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.

[7] 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.

[8] S. Geman, E. Bienenstock, and R. Dourstat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.

[9] 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.

[10] C. Giles, G. Kuhn, and R. Williams, "Dynamic recurrent neural networks: Theory and applications," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 153–156, 1994.

[11] 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.

[12] 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.

[13] 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.

[14] M. Hirsch, "Convergent activation dynamics in continuous-time neural networks," *Neural Networks*, vol. 2, pp. 331–351, 1989.

[15] M. Hirsch, "Saturation at high gain in discrete time recurrent networks," *Neural Networks*, vol. 7, no. 3, pp. 449–453, 1994.

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

[17] B. Horne. Personal Communication.

[18] B. Horne and D. Hush, "Bounds on the complexity of recurrent neural network implementations of finite state machines," *Neural Networks*, vol. 9, no. 2, pp. 243–252, 1996.

[19] R. Maclin and J. Shavlik, "Using knowledge-based neural networks to improve algorithms: Refining the Chou-Fasman Algorithm for Protein Folding," *Machine Learning*, vol. 11, pp. 195–215, 1993.

[20] C. Mead, *Analog VLSI and Neural Systems.* Reading, MA: Addison-Wesley, 1989.

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

[22] C. Omlin and C. Giles, "Rule revision with recurrent neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 1, pp. 183–188, 1996.

[23] C. Omlin and C. Giles, "Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants," *Neural Computation*, vol. 8, no. 4, 1996. In Press.

[24] 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.

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

[26] 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.

[27] J. Shavlik, "Combining symbolic and neural learning," *Machine Learning*, vol. 14, no. 3, pp. 321–331, 1994.

[28] B. J. Sheu, *Neural Information Processing and VLSI.* Boston, MA: Kluwer Academic Publishers, 1995.

[29] P. Tino, B. Horne, and C. Giles, "Fixed points in two–neuron discrete time recurrent networks: Stability and bifurcation considerations," Tech. Rep. UMIACS-TR-95-51, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, 1995.

[30] 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.

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

[32] 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.