Fault-Tolerant Implementation of Finite-State Automata in Recurrent Neural Networks

C.W. Omlin [a,b], C.L. Giles [a,c]

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

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

[c] UMIACS, University of Maryland, College Park, MD 20742

Abstract

Recently, we have proven that the dynamics of any deterministic finite-state automata (DFA) with n states and m input symbols can be implemented in a sparse second-order recurrent neural network (SORNN) with $n+1$ state neurons and $O(mn)$ second-order weights and *sigmoidal* discriminant functions [5]. We investigate how that constructive algorithm can be extended to fault-tolerant neural DFA implementations where faults in an analog implementation of neurons or weights do not affect the desired network performance. We show that tolerance to weight perturbation can be achieved easily; tolerance to weight and/or neuron stuck-at-zero faults, however, requires duplication of the network resources. This result has an impact on the construction of neural DFAs with a dense internal representation of DFA states.

1 INTRODUCTION

Fault-tolerance is often mentioned as a desirable property of neural networks. However, neural networks are not inherently tolerant to faults in their internal structure; they can be made tolerant to certain types of faults either by providing multiple copies of the network resources [7] or by training them under conditions which emulate faults [8]. Neural networks may be able to recover from faults in their internal structure through retraining.

Recently, we have proven that deterministic-finite state automata (DFAs) can be implemented in sparse recurrent neural networks with second-order weights (SORNNs) such that the dynamics of the constructed network remains stable; we extend that result and show that the regular languages $L(DFA)$ and $L(SORNN)$ recognized by the DFA M and the SORNN constructed from M are identical. We investigate how the SORNN implementation of DFAs can be made robust to certain faults in the network's internal structure.

The remainder of this paper is organized as follows: A short introduction to regular languages and deterministic finite-state automata in section 2 is followed by a summary of the results that establish stable encoding of finite-state dynamics in SORNNs. We prove that the regular languages accepted by a DFA M and the SORNN constructed from M are identical. In section 4, we define the fault models we are considering. After a brief discussion of tolerance to weight perturbation and the effects of weight and/or neuron stuck-at-zero faults in sections 5 and 6, respectively, we discuss different designs for tolerance to weight and/or neuron-stuck-at-zero faults in sections 7 and 8. In particular, some of the results in section 8 will demonstrate the limitations of the representation of DFAs in SORNNs. A summary and directions for future research conclude this paper.

2 FINITE STATE AUTOMATA

Regular languages represent the smallest class of formal languages in the Chomsky hierarchy [2]. 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. P are productions of the form $A \rightarrow a$ or $A \rightarrow aB$ where $A, B \in N$ and $a \in T$. 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 $M =< \Sigma, Q, q_1, F, \delta >$ 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, $q_1 \in Q$ 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 DFA IMPLEMENTATION IN SPARSE SORNNs

3.1 Network Construction

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

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

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 encoding algorithm for sparse SORNNs assumes that a unary encoding is used for the input symbols. A special neuron S_0 represents the output (accept/reject) of a RNN. A network accepts a string if the value of S_0 at the end of the string is greater than 0.5; otherwise, the string is rejected.

In order to construct a neural network implementation of a DFA, we need to program the DFA state transitions and the classification of DFA state as either rejecting or accepting states.

Consider a transition $\delta(q_j, a_k) = q_i$. We arbitrarily identify DFA states s_j and s_i with state neurons S_j and S_i, respectively. One method of representing this transition is to have state neuron S_i have a high output ≈ 1 and state neuron S_j have a low output ≈ 0 after the input symbol a_k has entered the network via input neuron I_k. A possible implementation is to set the weights W_{jjk} and W_{ijk} accordingly:

$$W_{ijk} = +H \;\; if \;\; \delta(q_j, a_k) = q_i \tag{2}$$

$$W_{jjk} = \begin{cases} +H & \text{if } \delta(q_j, a_k) = q_j \\ -H & \text{otherwise} \end{cases} \tag{3}$$

$$W_{0jk} = \begin{cases} +H & \text{if } \delta(q_j, a_k) \,\epsilon\, F \\ -H & \text{otherwise} \end{cases} \tag{4}$$

$$b_i = -H/2 \tag{5}$$

Setting W_{ijk} to a large positive value will ensure that S_i^{t+1} will be high and setting W_{jjk} to a large negative value will guarantee that the output S_j^{t+1} will be low. All other weights are set to small random values. In addition to the encoding of the known DFA states, we also need to program the response neuron, indicating whether or not a DFA state is an accepting state. Assuming that a special end symbol e marks the end of each strings, we program the weight W_{0ie} as follows: If state q_0 is an accepting state, then we set the weight W_{0ie} to a large positive value; otherwise, we will initialize the weight W_{0ie} to a large negative value. (The end symbol is not a crucial component of the rule insertion algorithm.) We define the values for the programmed weights as a rational number H, and let large *programmed* weight values be $+H$ and small values $-H$. We will refer to H as the *strength* of a rule.

We set the value of the biases b_i of all state neurons to $-H/2$. This ensures that all state neurons which do not correspond to the previous or the current DFA state have a low output. Before reading a new string, the initial network state is set to

$$\mathbf{S^0} = (S_0^0, S_1^0, \ldots, S_N^0) = (S_0^0, 1, 0, \ldots, 0)$$

with $S_0^0 = 1$ if $q_1 \in F$ and $S_0^0 = 0$ otherwise.

Thus, the rule insertion algorithm defines a nearly *orthonormal internal representation* of all DFA states.

3.2 Network Dynamics

The equation governing the dynamics of a SORNN take on a special form when a network is constructed to behave like a DFA where

(1) most weights are zero,

(2) a neuron receives 'substantial' input only from one other neuron due to the desired nearly orthonormal network state which corresponds to the internal DFA state representation, and

(3) non-zero weights and biases take only on values $\pm H$ and $-H/2$, respectively.

The neurons of a constructed SORNN undergo state changes of the following types:

low $\rightarrow$ *high*:

$$S_i^{t+1} = h(S_j^t * + \sum_{S_l \epsilon C_{i,l}} S_l^t - S_i^t, H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{6}$$

$$S_i^{t+1} = h(S_j^t + \sum_{S_l \epsilon C_{i,l}} S_l^t + S_i^t, H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{7}$$

high $\rightarrow$ *high*:

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

high $\rightarrow$ *low*:

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

low $\rightarrow$ *low*:

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

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

where

$$C_{i,l} = \{S_j \mid W_{ijl} = H, l \neq i, l \neq j\} \tag{12}$$

The inputs I_k^t are not shown explicitly since we assume that each input symbol is assigned a separate input neuron in a one-hot encoding. The DFA state transitions corresponding to these types of neuron state changes are illustrated in figure 1.

The terms $S_j^t * H$ are derived from the network construction algorithm: $-H/2$ causes neuron outputs which do not correspond to the current DFA state to be reset to a small value. The term $S_j^t * H$ where S_j^t is a high signal represents the *principal contribution* to the neuron S_i^{t+1} which is responsible for driving the output of neuron S_i^{t+1} high when the network executes a DFA state transition $\delta(q_j, a_k) = q_i$. All other terms are the *residual contributions* to the input of neuron S_i^{t+1} where the signal S_l^t is low. 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 (6) and (7) 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 (7) applies; otherwise, there is a residual low signal S_i^t trying to drive S_i^{t+1} low and equation (6) applies. Similarly, either equation (10) or (11) 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 observations about the equations (6)-(11) which will simplify the analysis.

Observation 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 transition 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, an analysis that shows stability of high and low signals for these types of state transitions implies the stability of high and low signals for state transitions of type $high \rightarrow high$ and $high \rightarrow low$, respectively.

Observation 3.2 *Of the two possible equations (6) and (7) for state transitions $low \rightarrow high$, the former equation also covers the latter equation.*

If high signals can be kept stable for equation (6), then they certainly can also be kept stable for equation (7). No separate analysis is necessary for thetwo equations.

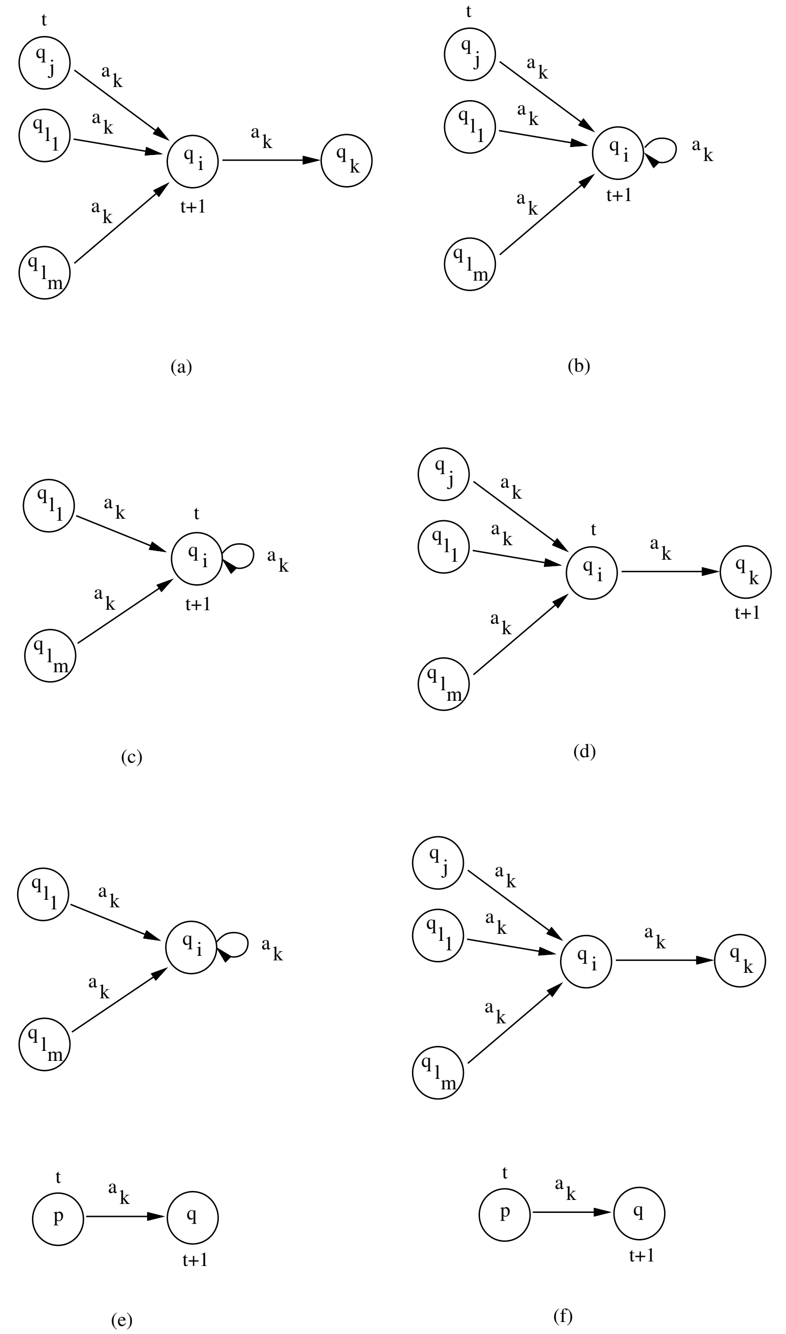

Figure 1: **Neuron State Changes and Corresponding DFA State Transitions**: The figure (a)-(f) illustrate the DFA state transitions corresponding to all possible state changes of neuron S_i; the DFA state(s) participating in the current transitions are marked with t and $t+1$. (a) $low \rightarrow high$ (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) $low \rightarrow low$ (no self-loop on q_i) (f) $low \rightarrow low$ (with self-loop on q_i). 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 $S_{l_1}, \ldots, S_{l_m}$.

Observation 3.3 *Of the two possible equations (10) and (11) for state transitions low $\rightarrow$ low, the latter equation also covers the former equation.*

An argument similar to that given for validity of observation 3.2 can be given.

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

low $\rightarrow$ low:

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

low $\rightarrow$ high:

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

Thus, the network equation (1) takes on the special form

$$a_i \equiv -H/2 + Hx_i \;\; with \;\; x_i = \sum_j S_j^t \tag{15}$$

since all but one input neuron have value 0 at any given time t. Notice that the number of terms in the sum $\sum_j S_j^t$ is equal to the number of DFA states q_j for which there are transitions $\delta(q_j, a_k) = q_i$ for the current input symbol a_k.

Thus, we can rewrite the operation performed by each state neuron more compactly as

$$S_i = h(H(2x_i - 1)/2) = h(x_i, H) = \frac{1}{1 + e^{H(1-2x_i)/2}} \tag{16}$$

The ideal case where neurons do not receive residual inputs from other neurons, successive network state changes can be expressed as the iteration of the function $h^t(x, H)$

$$h^t(x, H) \;\; = \;\; \begin{cases} h(x, H) & t = 1 \\ h(h^{t-1}(x, H), H) & t > 1 \end{cases} \tag{17}$$

The function $h(x, H)$ has the following useful property:

Lemma 3.1 *For $H > 4$, $h(x, H)$ has three fixed points $\phi^0 = 0.5, \phi^-$, and ϕ^+ with $0 < \phi^- < \phi^0 < \phi^+ < 1$. Furthermore, for $x < \phi^0$ and $\phi^0 < x$, $h^p(x, H)$ converges to ϕ^- and ϕ^+, respectively.*

Stability of ϕ^0 is obvious. ϕ^- and ϕ^+ can shown to be the minima of an appropriate Lyapunov function [1]. From the symmetry of $h(x, H)$, it follows that $\phi^- + \phi^+ = 1$.

We can now define a new function $h^t_\Delta(x, H)$ which takes the residual inputs into consideration. We assume that each neuron receives residual inputs from all other neurons. For a DFA, that means that there are transitions from each state to all other states on every input symbol. Although this is generally not the case, this worst case covers all possible DFAs and simplifies the analysis.

Since the initial output value of all state neurons except the neuron assigned to a DFAs start state are zero, the residual inputs under this worst case assumption are identical for all neurons; let Δx denote the residual neuron inputs. Then, the function $h^t_\Delta(x, H)$ is defined as

$$h^t_\Delta(x, H) \quad = \quad \begin{cases} h(x, H) & t = 1 \\ h(h^{t-1}_\Delta(x + \Delta x, H) + \Delta x, H) & t > 1 \end{cases} \tag{18}$$

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

We can quantify Δx for the case of low signals:

Lemma 3.2 *The low signals are bounded from above by the fixed point ϕ^-_Δ of the function*

$$h^t_{\Delta^-}(x, H) \quad = \quad \begin{cases} h(0, H) & t = 1 \\ h(n \cdot h^{t-1}_{\Delta^-}(x, H), H) & t > 1 \end{cases} \tag{19}$$

Proof: We prove the lemma by induction on t.

For $t = 1$, we have $h^1_{\Delta^-}(x, H) = h(0, H)$ since the initial output of all neurons except one 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 $h^t_{\Delta^-}(x, H)$. 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 equal to n times the values of the low signals at the current time step t which is equal to $h^t_{\Delta^-}(x, H)$. 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 $h^{t+1}_{\Delta^-} = h(n \cdot h^t_{\Delta^-}(x, H), H)$.

Similarly, we can quantify high signals:

Lemma 3.3 *The high signals are bounded from below by the fixed point ϕ^+_Δ of the function*

$$h^t_{\Delta^+}(x, H) \quad = \quad \begin{cases} h(1, H) & t = 1 \\ h(h^{t-1}_{\Delta^+}(x, H) - h^{t-1}_{\Delta^-}(x, H), H) & t > 1 \end{cases} \tag{20}$$

Proof: For the basis of the induction proof, we note that all state neurons with the exception of the neuron corresponding to the start state have value 0. Thus, the high signal of the the start state's successor state (start state and its successor state are identical in case of self loop) will have a high signal equal to $h(1, H)$.

Assuming the high signal at time step t is equal to $h^t_{\Delta^+}(x, H)$, 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 weighted by $+H$ and and a low signal weighted by $-H$. In this worst case, there is only one DFA transition $\delta(q_j, a_k) = q_i$ for chosen q_i and a_k; thus, $\sum_{S^t_l \epsilon C_{i,l}} = 0$. Any positive terms would only strengthen the high signal S^{t+1}_i. Furthermore, S^t_i has a negative contribution to the net input of S^{t+1}_i; if there were a transition $\delta(q_i, a_k) = q_i$ (self-loop), then the contribution $+S^t_i * H$ would only strengthen the high signal S^{t+1}_i.

Notice that the functions $h^t_{\Delta^-}(x, H)$ and $h^t_{\Delta^+}(x, H)$ have identical fixed points ϕ^-_Δ and ϕ^+_Δ since they are defined in terms of the same function $h(x, H)$. They only differ in their initial values for x and the residual input Δx.

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^t_j are strengthened to $h(-H/2)$ when there exists no state transition $\delta(., a_k) = q_j$. In that case, the neuron S^t_j receives no inputs from any of the other neurons; its output becomes less than ϕ^- since $h(0, H) < \phi^-$ for $H > 4$. Similarly, high signals S^t_i 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 $S^t_j > \phi^-$ or $S^t_j < \phi^+$ and only for a finite amount of time. For the majority of neurons we have $S^t_j \leq \phi^-$ and $S^t_j \geq \phi^+$. Since constructed SORNNs 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. Scaling issues are discussed elsewhere [6].

3.3 Stable Finite-State Dynamics

Stability of low and high signals is defined as follows:

Definition 3.1 *An encoding of DFA states in a SORNN is called stable if all the low and high signals are less and larger than 0.5, respectively.*

For given n, there exists a value $H > 4$ which causes the functions defined in equations (19) and (20) to converge toward their low and high fixed points, respectively.

Consider equation (13). In order for the low signal to remain less 0.5, the argument of $h_\Delta^t(.)$ must be less than 0.5 for all values of t. Thus, we require the following invariant property of the residual inputs:

$$-\frac{H}{2} + H * n * \phi_\Delta^- < \frac{1}{2} \tag{21}$$

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

Lemma 3.4 *The low signals S^t are always less to 0.5 if*

$$n < \frac{1}{2\phi_\Delta^-(H)}(1 + \frac{1}{H})$$

Thus, the lemma puts restrictions on the size n of the network for a given H. The location of the fixed point ϕ_Δ^- is determined by H; thus, we write $\phi_\Delta^-(H)$.

A similar analysis can be carried out for state transitions of equation (14). In the worst case, the following inequality must be satisfied for stable high signals:

$$-\frac{H}{2} + S_j^t * H - S_i^t * H > \frac{1}{2} \tag{22}$$

The high signal S_j^t and the low signal S_i^t converge toward ϕ_Δ^+ and ϕ_Δ^-, respectively. Therefore, the inequality simplifies to:

$$-\frac{H}{2} + H * \phi_\Delta^+ - H(1 - \phi_\Delta^+) > \frac{1}{2} \tag{23}$$

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

Lemma 3.5 *The high signals S^t are always greater than 0.5 if*

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

We have the following theorem:

Theorem 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 is the current DFA state and $S_i < 0.5$ otherwise if*

$$n < \frac{1}{2\phi_\Delta^-(H)}(1 + \frac{1}{H}) \quad \textit{with} \quad \phi_\Delta^+(H) > \frac{1}{4}(3 + \frac{1}{H})$$

for a proper choice of H. H, $\phi_\Delta^-(H)$ and $\phi_\Delta^+(H)$ can be computed by iterating $h_\Delta(0, H)$ until $h_\Delta^t(0, H) = h_\Delta^{t+1}(0, H) = \phi_\Delta^-(H)$.

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.

Although we have stated the conditions for stable low and high signals separately, the condition for stable signals can be simplified as follows:

Corollary 3.1 *If all neurons in a SORNN (not including the output neuron) have at least two weights $W_{ixk} = W_{iyk} = H$ for all input symbols a_k, then the internal representation of a DFA remains stable if*

$$\phi_\Delta^-(H) < \frac{1}{2n}(1 + \frac{1}{H})$$

Proof: Substituting $1 - \phi_\Delta^+$ for ϕ_Δ^- in theorem 3.1 we get

$$1 - \phi_\Delta^+ < \frac{1}{2n}(1 + \frac{1}{H}) \tag{24}$$

$$1 - \frac{1}{2n}(1 + \frac{1}{H}) < \phi_\Delta^+ \tag{25}$$

$$\frac{2n-1}{2n} + \frac{2n-1}{2Hn} < \phi_\Delta^+ \tag{26}$$

But stable high signals require

$$\frac{3}{4} + \frac{1}{4H} < \phi_\Delta^+ \tag{27}$$

Comparing inequalities (26) and (27), we conclude that the former implies the latter for $n \geq 2$.

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 3.2 *Let $L(DFA)$ denote the language accepted by a DFA M with n states and let $L(SORNN)$ be the language accepted by the fully-recurrent SORNN constructed from M; then, we have $L(SORNN) = L(DFA)$ if*

$$\phi_\Delta^-(H) < \frac{1}{2n}(1 + \frac{1}{H}) \quad \textit{and} \quad \phi_\Delta^+(H) > \frac{1}{4}(3 + \frac{1}{H})$$

Proof: We already established conditions for stable finite-state dynamics; we just have to give additional conditions for correct string classification.

For the case of an ungrammatical strings, the following condition must be satisfied:

$$-\frac{H}{2} - H * \phi_\Delta^+ + (n-1) * H * \phi_\Delta^- < \frac{1}{2} \tag{28}$$

where we have made the usual simplification about the convergence of the outputs to the fixed points ϕ_Δ^- and ϕ_Δ^+; furthermore, we assume that only one DFA state is a rejecting state; then the output neuron's residual inputs from all other state neurons is positive, weakening the intended high 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:

$$-\frac{H}{2} + H * \phi_\Delta^+ - (n-1) * H * \phi_\Delta^- > \frac{1}{2} \tag{29}$$

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

$$-2 * H * \phi_\Delta^+ + 2 * (n-1) * H * \phi_\Delta^- < 0 \tag{30}$$

Solving for ϕ_Δ^-, we get the following condition for the correct output of a network:

$$\phi_\Delta^- < \frac{1}{n} \tag{31}$$

Thus we have the following conditions for stable low signals and correct string classification: for $(H > 1)$.

$$\phi_\Delta^-(H) \quad < \quad \begin{cases} \frac{1}{2n}\,(1 + \frac{1}{H}) < \frac{1}{n} \quad for \quad H > 1 & \text{(dynamics)} \\ \frac{1}{n} & \text{(classification)} \end{cases} \tag{32}$$

Comparing these two conditions, it follows that the condition for stable low signals implies the condition for correct string classification. Notice that the condition for stable high signals also has to be satisfied. Since there does not exist a closed form for expressing ϕ_Δ^- as a function of H, we leave these two conditions independent. This completes the proof of the corollary.

Thus, we can now give a complete algorithm for constructing a SORNN from from a given DFA which recognizes the same regular language as the DFA (figure 2).

4 FAULT MODELS

The specific faults than can occur in a hardware implementation depend on whether a neural network is implemented in digital or analog technology. The results of a study on fault models in VLSI wafer neural networks can be found in [4]. In this paper, we are only concerned with faults as they occur in analog implementations. Analog implementations of neural networks have the advantage of lower power consumption compared to digital implementations.

We will consider the following three fault-models:

Input: DFA $(\Sigma, Q, q_1, F, \delta)$ with $\Sigma = \{a_1, a_2, \ldots, a_K\}$, $Q = \{q_1, q_2, \ldots, q_M\}$, q_1 is the start state, $F \subseteq Q$ is the set of accepting states, $\delta : \Sigma \times Q \rightarrow Q$ are the state transitions.

Output: SORNN with $L(SORNN) = L(DFA)$ which is unique up to labeling of neurons.

Algorithm Initialization:

1. choose $N > M$ neurons with sigmoidal discriminant function $(x) = \frac{1}{1+e^{-x}}$
2. for each $a_k \in \Sigma$ construct an input vector $(0, \ldots, 0, I_{k-1}, 1, I_{k+1}, 0, \ldots, 0)$ of length K
3. choose weight strength H such that

$$\phi_{\Delta}^{-}(H) < \frac{1}{2n}(1 + \frac{1}{H}) \quad \text{with} \quad \phi_{\Delta}^{+}(H) > \frac{1}{4}(3 + \frac{1}{H})$$

where $\phi_{\Delta}^{-}(H)$ and $\phi_{\Delta}^{+}(H)$ are the fixed points of the discriminant function $h_{\Delta}(x, H)$

Network Construction:

4. for $i = 1 \ldots N$

   $b_i = -H/2$;

   for $j = 1 \ldots N$

   for $k = 1 \ldots K$

   $W_{ijk} = +H$ if $\delta(q_j, a_k) = q_i$; $W_{jjk} = -H$ if $\delta(q_j, a_k) \neq q_j$;

   $W_{0jk} = +H$ if $\delta(q_j, a_k) \in F$; $W_{0jk} = -H$ if $\delta(q_j, a_k) \notin F$;

Network Initialization:

5. Before reading a new string, set the initial network state to

   $\mathbf{S^0} = (S_0^0, S_1^0, \ldots, S_N^0) = (S_0^0, 1, 0, \ldots, 0)$

   with $S_0^0 = 1$ if $q_1 \in F$ and $S_0^0 = 0$ otherwise;

Network Dynamics:

6. $S_i^{t+1} = g(b_i + \sum_{j,k} W_{ijk}\, S_j^t\, I_k^t)$

Network Performance:

7. The output of S_0^f after reading a string s of length f is 1 if $s \in L(DFA)$ and 0 otherwise.

Network Complexity:

8. number of neurons: M+1; number of weights: O(MN); maximum fan-out: 3M

   weight alphabet $\Sigma_W = \{-H, -H/2, 0, H\}$

Figure 2: **Algorithm for Encoding Arbitrary DFAs in SORNNs**

(1) the value of a fabricated weight may not be identical to the weight value of the design, i.e. $\overline{W}_{ijk} = W_{ijk} * (1 \pm \varepsilon_{ijk})$ if $W_{ijk} \neq 0$, where ε_{ijk} stands for the percentage of perturbation of some weight W_{ijk}; ε_{ijk} may be different for different weights, but we assume that there exists a *known* maximum value ε such that for all individual values ε_{ijk}, we have $0 \leq \varepsilon_{ijk} \leq \varepsilon$.

(2) the output of a neuron may be stuck at zero ('neuron stuck-at-zero') which effectively removes the neuron from the network

(3) a 'weight-stuck-at-zero' effectively removes the faulty weight from the network.

All faults are likely to occur during the fabrication of an analog implementation. If the implementation of the weights is analog and no special circuitry is added to the design, then the weights will also be sensitive to temperature variations [3].

5 WEIGHT PERTURBATION

Recall the network state transitions and their equations where preservation of low and high signals implied preservation of all signals and thus stability of the internal DFA state representation:

low $\rightarrow$ *low*:

$$S_i^{t+1} = h(S_i^t + \sum_{S_l \in C_{i,l}} S_l^t, H) \tag{33}$$

low $\rightarrow$ *high*:

$$S_i^{t+1} = h(S_j^t + \sum_{S_l \in C_{i,l}} S_l^t - S_i^t, H) \tag{34}$$

where the set $C_{i,l}$ represents all state neurons S_l that are connected to neuron S_i. For the weight perturbation model, all weights of a constructed network differs from their ideal value. Since all weights have values $+H$ or $-H$, the fault model for perturbed weights can be rewritten:

$$H_i = H * (1 \pm \varepsilon_i) \tag{35}$$

We further assume that there exists a maximum perturbation ε, i.e. some values of H_i may be closer to the ideal value H obtained from the encoding algorithm, but no value H_i differ from H by more than ε. We choose the sign of ε such that the intended target signal S_i^{t+1} weakens, i.e. intended low and high target signals become larger and smaller, respectively; this represents a worst case with respect to the weight perturbation fault model.

Then, equations (33) and (34) can be expressed as follows:

low $\rightarrow$ *low*:

$$S_i^{t+1} = h(-H*(1-\varepsilon)/2 + S_i^t * H * (1+\varepsilon) + \sum_{S_l \epsilon C_{i,l}} S_l^t * H * (1+\varepsilon)) \tag{36}$$

low $\rightarrow$ *high*:

$$S_i^{t+1} = h(-H*(1+\varepsilon)/2 + S_j^t * H * (1-\varepsilon) + \sum_{S_l \epsilon C_{i,l}} S_l^t * H * (1-\varepsilon) - S_i^t * H * (1+\varepsilon)) \tag{37}$$

For preservation of all signals, low and high signals must at all times be less and greater than 0.5, respectively. Under the assumption that low and high neuron outputs converge toward fixed points ϕ_Δ^- and ϕ_Δ^+, respectively, and that each neuron receives inputs from all other neurons except the special response neuron S_0, equation (36) leads to the following inequality for guaranteed low signals:

$$-\frac{H*(1-\varepsilon)}{2} + H*(1+\varepsilon)*n*\phi_\Delta^- < \frac{1}{2} \tag{38}$$

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

Lemma 5.1 *The low signals S^t are always less or equal to 0.5 if*

$$n < \frac{1}{2\phi_\Delta^-(H,\varepsilon)(1+\varepsilon)}((1-\varepsilon) + \frac{1}{H})$$

The fixed point ϕ_Δ^- now depends on H *and* on ε; thus, we write $\phi_\Delta^-(H,\varepsilon)$. We evaluate the fixed points ϕ_Δ^- at $H(1-\varepsilon)$; for any $H' > H(1-\varepsilon)$, we have $\phi_\Delta^-(H') < \phi_\Delta^-(H(1-\varepsilon))$ and consequently $n' > n$. Different weights may have different actual values H_i and thus different fixed points may be reached; but for all variations ε_i, we have $\varepsilon_i < \varepsilon$ and thus $\phi_\Delta^-(H,\varepsilon_i) < \phi_\Delta^-(H,\varepsilon)$. Thus, it is sufficient to evaluate the fixed point as chosen above to give a conservative estimate for the maximum fixed point $\phi_\Delta^-(H,\varepsilon) < \frac{1}{2}$ that can be reached and thus the maximum allowed network size for preservation of low signals.

Similarly, equation (37) leads to the following inequality if we assume that neuron outputs converge toward the fixed points ϕ_Δ^- and ϕ_Δ^+, respectively:

$$-\frac{H*(1+\varepsilon)}{2} + H*(1-\varepsilon)*\phi_\Delta^+ - \phi_\Delta^- * H*(1+\varepsilon) > \frac{1}{2} \tag{39}$$

We have ignored the term $\sum_{S_l \epsilon C_{i,l}} S_l^t * H * (1-\varepsilon)$ in equation (37) because a positive contribution to the net input of neuron S_i^{t+1} only strengthens the high signal and is not necessary to prove preservation of high signals.

Using the identity $\phi_\Delta^- = 1 - \phi_\Delta^+$ and solving the inequality (39) for ϕ_Δ^+, we get the following condition for guaranteed high signals:

Lemma 5.2 *The high signals S^t are always greater than 0.5 if*

$$\phi_\Delta^+(H, \varepsilon) > \frac{1}{4}(3(1+\varepsilon) + \frac{1}{H})$$

Again, we evaluate the fixed point ϕ_Δ^+ at $H(1-\varepsilon)$; for any choice $H' > H(1-\varepsilon)$, we have $\phi_\Delta^+(H') > \phi_\Delta^+(H(1-\varepsilon))$. The condition of lemma 5.2 certainly applies for any choice $H' > H(1-\varepsilon)$ if it applies to $\phi_\Delta^+(H(1-\varepsilon))$.

We can now state the following theorem:

Theorem 5.1 *Consider 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 $3m$ that has been constructed from a DFA M with n states and m input symbols. Assuming that the values H_i of all implemented weights differ from their theoretical value by no more than ε_i, where $H_i = H(1 \pm \varepsilon_i)$ and the maximum weight variation is bounded by ε with $\varepsilon_i \leq \varepsilon$, there exists a value H for given ε such that the internal state representation remains stable, i.e. $S_i > 0.5$ when q_i is the current DFA state and $S_i < 0.5$ otherwise if*

$$n < \frac{1}{2\phi_\Delta^-(H(1-\varepsilon))(1+\varepsilon)}((1-\varepsilon) + \frac{1}{H}) \quad \textit{with} \quad \phi_\Delta^+(H(1-\varepsilon)) > \frac{1}{4}(3(1+\varepsilon) + \frac{1}{H})$$

Proof: Our analysis assumed the worst case for weak low and high signals in an imperfect implementation of the weights H; 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 even in the presence of weight perturbations.

It follows that a SORNN can implement any DFA even in the presence of perturbations of the weights due to either imperfect fabrication or temperature sensitivity since the condition for stable dynamics implies the condition for correct classification. It should come as no surprise that the conditions for theorem 5.1 reduce to those of theorem 3.1 for perfect implementation of all weights ($\varepsilon = 0$).

6 Faults in Neurons and Weights

We will discuss in this section how neuron and weight stuck-at-zero faults affect the performance of constructed networks.

The effects of a neuron stuck-at-zero fault may disable a constructed network's so that it can no longer distinguish positive from negative example strings as expressed in the following theorem:

Theorem 6.1 *Consider a recurrent network constructed from a DFA M with $L(M) = L(SORNN)$. If the output of at least one neuron of the SORNN is stuck at zero, then there exists a prefix string p such that any string $px \in L(M)$ of arbitrary length is misclassified.*

Proof: Let S_f denote the faulty neuron. Then we choose the prefix p such that DFA state q_i corresponding to state neuron $S_i = S_f$ is reached from the start state: $\delta(q_1, p) = q_i$. According to the DFA encoding algorithm, the weights feeding into S_f are then programmed such that $S_f^{|p|}$ has a high output; however we have $S_f^t = 0$ for any t since S_f is faulty. However, with $S_f^{|p|} = 0$, the state neurons S_j^τ for $\tau > p$ corresponding to DFA states q_j reached during subsequent state transitions $\delta(q_i, x)$ will not be high, since $S_f^{|p|} = 0$ cannot drive any other state neurons high (including itself). Thus, the network will 'starve' and the outputs of all 'healthy' state neurons (including the special response neuron S_0) will converge toward a common fixed point $0 < \phi_\Delta^- < 0.5$ (faulty state neurons have output 0). Therefore, for any strings $px \in L(M)$, the constructed network will misclassify px since we require $S_0 > 0.5$ for positive example strings.

The effects of weight-stuck-at-zero faults can be equally catastrophic. A weight-stuck-at-zero translates into a weight programmed to $+H$ or $-H$ stuck at zero. Weights $+H$ are only used for a state neuron to drive another (or possibly the same) state neuron to a high output; $-H$ is only used to drive the output of a high neuron low and only S_i^t can drive S_i^{t+1} low. Thus, we have to distinguish two different cases of weight-stuck-at-zero faults.

Let us first consider faulty weights which are programmed to $+H$. Recall two types of network state transitions:

low $\rightarrow$ *low*:

$$S_i^{t+1} = h(S_i^t + \sum_{S_l \in C_{i,l}} S_l^t, H) \quad (S_i^t, S_l^t : low) \tag{40}$$

low $\rightarrow$ *high*:

$$S_i^{t+1} = h(S_j^t + \sum_{S_l \in C_{i,l}} S_l^t - S_i^t, H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{41}$$

and one of the two remaining transition types:

high $\rightarrow$ *high*:

$$S_i^{t+1} = h(S_i^t + \sum_{S_l \in C_{i,l}} S_l^t, H) \quad (S_j^t : high, S_l^t, S_i^t : low) \tag{42}$$

Assuming that weight faults affect state transitions expressed by (42), state transitions of type (41) will also be affected. Thus, it suffices to consider state transitions of type (42).

Let the weight $+H$ in the term $S_i^t * H$ in equations (40) and (42) be stuck at zero. Then, in order for this single weight-stuck-at-zero fault not to affect the operation of the entire network, the corresponding state transition must remain unaffected. This requires the term $-H/2 + \sum_{S_l \in C_{i,l}} S_l^t * H$ to be less than 0.5 for state transitions of type (40) and greater than 0.5 for state transitions of type (42). Those are contradictory requirements which cannot be simultaneously fulfilled. The requirement for the preservation of low signals for networks with faulty weights $+H$ can be computed as

$$n < \frac{1}{2\phi_\Delta^-(H)}(1 + \frac{1}{H}) \tag{43}$$

However, this is also the condition for the preservation of low signals in networks without faults (theorem 9.1). Thus, we have proven the following lemma:

Lemma 6.1 *Let S_i be a neuron whose output is programmed via a weight $+H$. If that weight is stuck at zero, then the output of S_i will always be low and converge toward the fixed point ϕ_Δ^- with $0 < \phi_\Delta^- < 0.5$.*

Let us now consider the effects of stuck-at-zero faults for weights which are programmed to $-H$. There are two types of state transitions where a programmed weight $-H$ occurs:

low $\rightarrow$ *low*:

$$S_i^{t+1} = h(-S_i^t + \sum_{S_l \in C_{i,l}} S_l^t, H) \quad (S_i^t, S_l^t : low) \tag{44}$$

high $\rightarrow$ *low*:

$$S_i^{t+1} = h(-S_i^t + \sum_{S_l \in C_{i,l}} S_l^t, H) \quad (S_i^t : high, S_l^t : low) \tag{45}$$

Setting the term $-S_i^t * H$ equal to zero in both equations, we get the following condition for the preservation of low signals which is identical for equations (44) and (45):

$$-\frac{H}{2} + H * (n-1) * \phi_\Delta^- < \frac{1}{2} \tag{46}$$

where we assumed that low signals will converge toward the fixed point ϕ_Δ^-.

Solving inequality (46) for n yields the following lemma:

Lemma 6.2 *In a constructed network with $n+1$ state neurons (including the special response neuron S_0), low signals will be preserved when weights programmed to $-H$ are stuck at zero if the following condition is satisfied:*

$$n < \frac{1}{2\phi_\Delta^-(H)}(1 + \frac{1}{H}) + 1$$

Comparing the above condition with the the condition in theorem 3.1, we see that the above condition is always satisfied for a SORNN without faults which implements a given DFA.

We can now state the following theorem:

Theorem 6.2 *Consider a recurrent network constructed from a DFA M with $L(M) = L(RNN)$. If at least one of a constructed network's weight programmed to $+H$ is stuck at zero, then there exists a prefix p such that any string $px \in L(M)$ of arbitrary length is misclassified.*

Proof: We have shown that the output of a neuron programmed via a faulty weight will always be low and converge toward a fixed point $0 < \phi < 0.5$. A constructed network will 'starve', i.e. the outputs of all its state neurons (including the special response neuron S_0) will be low after the DFA state corresponding to the state neuron programmed via the faulty weight has been reached.

There exists an asymmetry regarding the misclassification of positive and negative example strings in recurrent networks with stuck-at-zero faults:

Corollary 6.1 *Consider a recurrent network constructed from a DFA M with $L(M) = L(RNN)$. Then negative example strings with $x \notin L(M)$ cannot be misclassified due to stuck-at-zero faults in neurons or weights.*

Proof: Negative example strings could only be misclassified if a faulty weight or neuron could cause the output of a 'healthy' state neuron to reach a fixed point $0.5 < \phi_\Delta^+ < 1$ when it should converge toward a fixed point $0 < \phi_\Delta^- < 0.5$. However, stuck-at-zero faults in neurons or weights can only cause the outputs of all neurons (including the special response neuron S_0) to converge toward ϕ_Δ^- with $0 < \phi_\Delta^- < 0.5$. Thus, negative example strings can never be misclassified due to a stuck-at-zero fault in a constructed network.

Thus, we have found that our network model is not inherently robust to faults. Stuck-at-zero faults are masked as long as the neurons and weights in question are not activated, i.e. certain states of a DFA are not visited. As soon as a faulty neuron or weight is used for any DFA transition, the network performance degrades catastrophically. Designs that tolerate such faults are the topic of the next section.

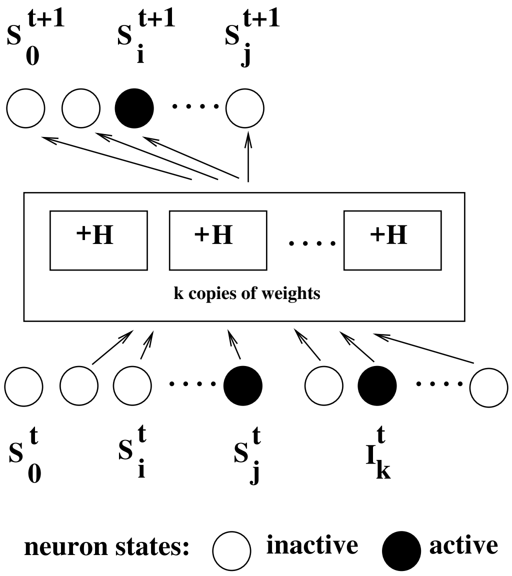

Figure 3: **Design for Tolerance to Faulty Weights**: Replicating the weights programmed to $+H$ k times allows a network to tolerate up to $k-1$ faulty weights per neuron. No special measures have to be taken to provide tolerance to faulty weights that are programmed to $-H$. The change of neuron activities during a DFA state transition $\delta(q_j, a_k) = q_i$ is shown.

7 Fault-Tolerant Design

In order for a network to tolerate stuck-at-zero faults for neurons or weights, a network's internal representation of DFA states and transitions has to be *distributed* over several neurons and weights. Typically, the resources of any (computational) system are replicated in order to achieve fault-tolerance. We will propose different designs for achieving fault-tolerance in recurrent networks.

7.1 Replicated Weights

A possible design for achieving tolerance to weight-stuck-at-zero faults is shown in figure 3. All weights are replicated k times. This design has the following property:

Theorem 7.1 *A recurrent network constructed from a DFA M such that $L(SORNN) = L(M)$ will tolerate exactly $k-1$ stuck-at-zero faults for weights per neuron if all the weights programmed to $+H$ are replicated k times.*

Proof: The proof of theorem 3.1 asserts stability of the internal DFA state representation for weight strength $+H$. Providing k copies of all weights $+H$ equals to constructing a network with weight strength $+kH$. Thus, the network will still implement a given DFA with $k-1$ faulty weights per neuron. It is worth noting that no replication of weights programmed to $-H$ is necessary; according to lemma 6.2, low signals are preserved even when those weights are faulty.

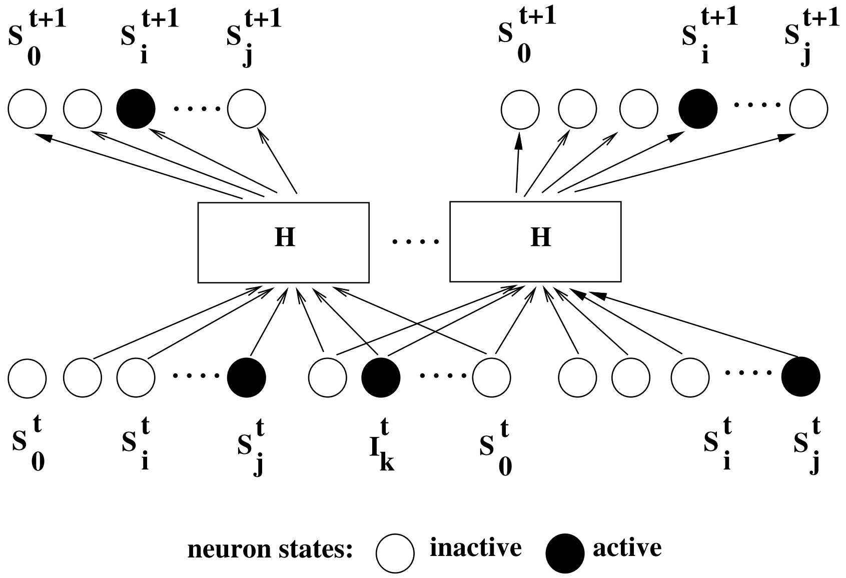

Figure 4: **Design for Tolerance to Faulty Neurons**: The entire network is replicated $2k + 1$ times, allowing k faulty neurons. Whether or not a string is accepted is decided based on a majority vote among the $2k + 1$ networks, of which at least $k + 1$ have to be flawless.

7.2 Replicated Networks

An alternative design of fault-tolerant networks is to replicate the entire network k times; thus, k *independent* networks decide whether or not to accept a string. This design is shown in figure 4. The following theorem expresses the level of fault-tolerance of such a system:

Theorem 7.2 *A system of $2k + 1$ independent recurrent networks constructed from a DFA M such that $L(SORNN) = L(M)$ will tolerate exactly k stuck-at-zero faults in neurons.*

Proof: A single neuron stuck-at-zero fault in any network can cause a wrong output at the special response neuron; thus, a positive string may be misclassified (theorem 6.2). Whether or not a string is accepted has to be decided based on a majority vote among the outputs of the independent network. In order to tolerate k wrong outputs caused by k faulty neurons (possibly in different networks), there must be at least $k + 1$ correct outputs from networks without faults. If a positive string is misclassified by k faulty networks, then the majority vote among the flawless networks will yield a correct answer.

8 Alternative Designs

As shown above, conservative approaches to designing fault-tolerant recurrent networks are effective. However, because all computational resources are replicated, it seems that the available computational resources are vastly underused. For instance, the use of pairwise orthogonal internal DFA state representation leads to a very sparse internal DFA state representation, i.e. many possible internal representations are not used at all. It is therefore of interest whether other binary internal representations for DFA states are possible. In the following sections, we will study properties of binary DFA encodings which are not pairwise orthogonal.

Before we investigate alternative designs for fault-tolerance, we make the following observation which simplifies the task:

Corollary 8.1 *Tolerance to neuron-stuck-at-zero faults also implies tolerance to weight-stuck-at-zero faults.*

Proof: Fault-tolerance is achieved based on redundancy of the encoding of DFA states and state transitions. If a network can tolerate faulty neurons, then the healthy neurons and associated weights must compensate for the loss, i.e. a mechanism which recovers the state information and still performs the desired state transitions must be in place. The same mechanism will also allow the execution of state transitions if some weights that implement the state transition are faulty.

Thus, for the remainder of this paper, we will only be concerned with stuck-at-zero faults in neurons.

8.1 Properties of Binary DFA Encodings

Recall the equation which governs the dynamics of SORNNs with second-order weights; for a constructed network, they can be rewritten as

$$S_i^{(t+1)} = h(a_i(t)) = \frac{1}{1+e^{-a_i(t)}}, \qquad a_i(t) = -\frac{H}{2} + \sum_{j,k} \pm H S_j^{(t)} I_k^{(t)} \tag{47}$$

where the sign of H is determined by the DFA encoding algorithm. An analysis of a constructed SORNNs dynamical stability becomes feasible if the DFA representation is derived from fixed points of a SORNNs discriminant functions and state transitions are programmed in a uniform way; in our case, the function is a sigmoid. If we approximate the sigmoidal discriminant function with a hard-limiting threshold function, then equation (47) has a natural interpretation in Boolean logic where all variables can only assume binary values:

$$s_i^u = 0 \vee \bigvee_{j,k} w_{ijk} \wedge s_j^v \wedge i_k \tag{48}$$

where we let variables in lower case correspond to the binary interpretation of their corresponding variables in a SORNN. We have replaced the monotonically increasing time index with the label of the DFA state represented by the binary code. For an actual SORNN implementation of a DFA, the weights W_{ijk} will be set to $+H$ for $w_{ijk} = 1$ and to 0 or $-H$ for $w_{ijk} = 0$; however, it is sufficient to program only weights W_{iik} to $-H$ where appropriate. A binary encoding of a DFA state q_u is then a code $\mathbf{s}^u = (s_0^u, s_1^u, s_2^u, \ldots, s_n^u)$

where n is the length of the encoding and s_i^u is the i^{th} component of the encoding for state q_u. We will denote with $\mathcal{S} = \{s_i\}$ the encodings of all states of a DFA. The component s_0^u - the response component - indicates whether or not the DFA state represented by $(s_1^u, s_2^u, \ldots, s_n^u)$ is an accepting state. The code s^1 for a DFAs start state q_1 is the start code of the encoding. A state transition $\delta(q_v, a_k) = q_u$ then becomes a code transition ω with $\omega(s^v, a_k) = s^u$, where ω is implemented in the following way: The 'weight' w_{ijk} has value 1 if the components s_i^u and s_j^v both have value 1; we say components s_i^u and s_j^v are *connected* via w_{ijk}. If an input symbol a_k is presented, then the 'input' i_k has value 1 causing s_i^u to become 1; this operation is carried out simultaneously for all components s_i^u. The value of s_0^u is derived from the code s^v via weights $\{w_{0jk}\}$; s_0^u has value 1 if q_u is an accepting string and 0 otherwise.

Thus, an encoding of DFA M is a pair $(\mathcal{S}, \mathcal{W})$ where DFA states $Q = \{q_u\}$ naturally map into codes $\mathcal{S} = \{s^u\}$ and state transitions $\delta(q_v, a_k) = q_u$ map into code transitions $\omega(s^v, a_k) = s^u$.

One might think that the number n_{DFA} of DFA states that can be represented in $(\mathcal{S}, \mathcal{W})$ is equal to the number of different binary strings of length n_{DFA}. However, we will show in the following section that this is not the case; furthermore, we will prove that another encoding scheme ("n-choose-k") is in general not feasible either if the code length is to be minimal.

8.2 Realizable Binary DFA Encodings

From now on, we may write s^{v+1} for $\omega(s^v, a_k) = s^u$ if a_k is arbitrary; generally, we write $s^{v+\tau}$ for the code reached after τ symbols starting with code s^v.

Definition 8.1 *A binary encoding $(\mathcal{S}, \mathcal{W})$ 'realizes' a DFA if the encoding $\mathcal{W}$ of DFA state transitions guarantees that the codes s^{v+l} reached on successive code transitions $s^v \xrightarrow{a^\tau} s^{v+1} \xrightarrow{a^{\tau+1}} \cdots s^{v+l-1} \xrightarrow{a^{\tau+l}} s^{v+l}$ always unambiguously identify the current DFA state q_v.*

The following lemma shows that the encodings s^v for DFA states cannot be arbitrary:

Lemma 8.1 *Let m be the number of input symbols of a DFA M and $\delta(q_v, \{a_k\}) = \{q_{u_1}, q_{u_2}, \ldots, q_{u_m}\}$ with $q_{u_i} \neq q_{u_j}$, i.e. no two state transitions from a state q_v lead to the same state q_u. In order for a binary encoding $(\mathcal{S}, \mathcal{W})$ to realize a DFA M, there must be at least m codes $s^{u_1}, s^{u_2}, \ldots, s^{u_m}$ which are orthogonal to the encoding s^v of the current DFA state q_v.*

Proof: The lemma states that no two codes s^u and s^v can have any components in common if there exists a transition $\delta(q_v, a_k) = q_u$: $s_i^u \neq s_i^v$. Let us assume that there are l DFA states which form a cycle ('l-cycle') $q_v \xrightarrow{a^\tau} q_{v+1} \xrightarrow{a^{\tau+1}} q_{v+2} \xrightarrow{a^{\tau+3}} \cdots q_{v+l} \xrightarrow{a^{\tau+l}} q_v$ and that codes $s^{v+\tau}$ and $s^{v+\tau+1}$ have exactly one component $s_c^{v+\tau} = s_c^{v+\tau+1} = 1$ in common and that the index c be different for different pairs of codes $s^{v+\tau}$ and $s^{v+\tau+1}$. Then, components $s_c^{v+\tau}$ and $s_c^{v+\tau+1}$ are connected via weight $w_{cck} = 1$. After a l-cycle, the codes s^v that were activated during the cycle have become indistinguishable, because the common component $s_c^{v+\tau}$ is propagated through every code transition starting with $\tau = 0$. Thus, the number of components i with $s^{v+\tau} = 1$ monotonically increases until the codes $s^{v+\tau}$ have the same components in common. In the worst case, the encoding of all states become indistinguishable if a DFA with n states has a cycle of length n or if the codes $s^{v+\tau}$ changed all components $s^{v+\tau}$ in $(s_0^{v+\tau}, s_1^{v+\tau}, \ldots, s_n^{v+\tau})$ during code transitions in the l-cycle.

Notice that the above argument depends on the symbols $a^{\tau+l}$ being identical for all code transitions; if they are all identical, then the same weight w_{cck} will cause component $s_c^{v+\tau+1}$ to become 1 and never change change its value for the remainder of the l-cycle.

Thus, the above lemma establishes orthogonality of the encoding of states q_u and q_v in DFA state transitions $\delta(q_v, a_k) = q_u$ as a *necessary* condition for a realizable binary encoding of a DFA; that condition is also necessary for unique identification of DFA states from codes.

Lemma 8.2 *If the encodings s^u and s^v for all DFA state transition $\delta(q_v, a_k) = q_u$ with $q_v \neq q_u$ are orthogonal, then the current DFA state can always be unambiguously identified from the current encoding s^u.*

Consider a code transition $\omega(s^v, a_k) = s^u$. If the code s^u is orthogonal to code s^v, then encodings s^u and s^v never have any components s_i^u in common, i.e. component s_j^v always changes its value from 1 to 0 on a code transition $\omega(s^v, a_k) = s^u$. Thus, the current DFA state can always be identified unambiguously from the current encoding s^u. An exception are code transitions $\omega(s^v, a_k) = s^v$; in that case, the encoding does not change.

The following corollary provides a sufficient condition under which a binary DFA encoding is realizable:

Corollary 8.2 *If all the codes s^u of the states q_u of M are pairwise orthonormal, then there exists a binary encoding $(\mathcal{S}, \mathcal{W})$ of M.*

Proof: For each code s^v, there exist at least as many codes s^{u_i} that are orthonormal with s^v. Furthermore, each code s^v uniquely identifies a DFA state q^v since no other code s^u has any non-zero components in common with s^v.

The above condition where any two codes are orthonormal is a sufficient condition for a realizable binary DFA encoding. Denser encodings might exist; a particular encoding depends on the DFA, but we suspect that finding such a denser encoding is a computationally hard problem. This leads to the following interesting question: What is the densest possible representation of codes? Binary codes of length $log(n)$ are needed to distinguish n codes. However, the following corollary to lemma 8.1 shows that codes of length $logn$ can generally not be for realizable DFA encodings:

Corollary 8.3 *There does not exist a minimal binary encoding* $(\mathcal{S}_{min}, \mathcal{W}_{min})$ *with codes* $(s_0^u, s_1^u, s_2^u, \ldots, s_{\log(n)}^u)$ *which can emulate any given DFA* M *with* n *states.*

Proof: We observe that any encoding $(\mathcal{S}, \mathcal{W})$ with $\log(n)$ components $(s_0^u, s_1^u, s_2^u, \ldots, s_{\log(n)}^u)$ cannot realize an arbitrary DFA M with n states because the necessary condition of lemma 8.1 is not satisfied. In a minimal binary encoding, there exists only one code s_i with $\omega(s^v, a_k) = s^u$ such that $s^u \cdot s^v = 0$; s^u and s^v are the complements of each other. Thus, encodings with code length $\log(n)$ exist only for DFAs with no more than two states.

Notice that the original DFA encoding algorithm constructed DFA state transitions as transitions between *unary* binary DFA state codes; these codes were mutually orthonormal and thus satisfied the necessary conditions for legal binary codes.

There exist DFAs which require pairwise orthonormal internal representation of DFA states. The following definition is useful:

Definition 8.2 *The graph* $G(M)$ *underlying a DFA* M *is a graph whose vertices are the states of* M *and whose directed edges are the state transitions of* M*. If there exist multiple DFA state transition* $\delta(q_v, \{a_{k_1}, a_{k_2}, \ldots, a_{k_p}\}) = q_u$*, then* $G(M)$ *has a single directed edge connecting vertices* u *and* v*.*

We now state the following corollary:

Corollary 8.4 *Any DFA with* n *states and at least* $n-1$ *input symbols whose underlying graph* $G(M)$ *is completely connected requires orthonormal codes of length* $n+1$ *(including the component for the special response neuron).*

Proof: If the graph $G(M)$ underlying a DFA M is completely connected, then the codes s^u and s^v for pairs of DFA state must be orthogonal according to lemma 8.1. The only codes which satisfy these conditions for all pairs of codes s^u and s^v, are orthonormal binary codes (assuming unary encoding).

The above corollary also gives a direct solution for a fault-tolerant RNN implementation of a DFA M whose underlying graph $G(M)$ is completely connected:

Corollary 8.5 *Consider a SORNN implementation of a DFA* M *whose underlying graph* $G(M)$ *is completely connected. Then, the constructed network will tolerate* $k-1$ *stuck-at-zero faults in neurons (and therefore in weights) only if the codes* s^u *have* k *components* $s_i^u = 1$ *and the codes of the DFA states* M *are mutually orthogonal.*

Proof: Since the intersection of the sets of components with $s_i^u = 1$ and $s_j^v = 1$ for DFA states q_u and q_v is empty for $u \neq v$, the current state can be uniquely identified even when $k-1$ neurons are faulty. It is clear that the constructed network accepts the same language $L(M)$ with and without faults.

Notice that the required implementation is different from the straight-forward implementation of a SORNN where the entire network had to be replicated $2k+1$ times in order to tolerate $k-1$ faults in neurons. Those were k independent networks (except for the output of the entire system, which requires a majority vote among the special response neurons of each network). In the above case, there are not k independent networks; it is a single network with kn neurons, but whose interconnection network is denser than that of $2k+1$ independent networks.

8.3 "n choose k" Encoding Algorithm

The investigation of the "n choose k" encoding algorithm is based on the premise that the representation of a current DFA state be distributed across at least k active state neurons if the constructed network is to tolerate $k-1$ faulty neurons.

Definition 8.3 *A (binary) (k,m,n)-encoding $(\mathcal{S}, \mathcal{W})$ is a realizable encoding $(\mathcal{S}, \mathcal{W})$ of a DFA M with m input symbols where each code s^u for qu has exactly k components $s_i^u = 1$ $(i > 0)$.*

Given a DFA M with n_{DFA} states and m input symbols, we construct a (k,m,n)-encoding $(\mathcal{S}, \mathcal{W})$ by choosing the length n_{RNN} (or simply n) of codes s^u such that

$$n_{DFA} \geq \left(\begin{array}{c} n \\ k \end{array} \right)$$

where exactly k components s_i^u have value 1 for each code s^u. Consider a code transition $\omega(s^v, a_k) = s^{u_k}$; according to lemma 8.1, a DFA encoding is realizable only if there exist codes $s^{u_1}, s^{u_2}, \ldots, s^{u_m}$ that are orthogonal to s^v. By construction, none of the codes s^{u_k} have any components in common with s^v. Since exactly k components of s^v are 1, there are only $n-k$ components left of which exactly k components for each of the m codes s^{u_k} must have value 1. Thus, we require that

$$m \leq \left(\begin{array}{c} n-k \\ k \end{array} \right)$$

which leads to the condition $n \geq 2k$ where equality only holds for DFAs with two states and one input symbol. Clearly, our choice for codes s^u satisfy the necessary conditions of a realizable $(\mathcal{S}, \mathcal{W})$ for any DFA M. The weights w_{ijk} are to be programmed to implement transitions $\omega(s^v, a_k) = s^u$ such that $(\mathcal{S}, \mathcal{W})$ emulates M.

Before we proceed to program the weights, we will discuss some interesting properties of "n choose k" codes for DFA states.

Definition 8.4 *The capacity $C_{S,W}(k,m,n)$ of an encoding $(\mathcal{S},\mathcal{W})$ is the maximum number of codes in $(\mathcal{S},\mathcal{W})$ for given $k, m,$ and n.*

The following corollary gives an upper bound on the size n_{DFA} of a DFA M that can be constructed:

Lemma 8.3 *A (k,m,n)-encoding $(\mathcal{S},\mathcal{W})$ with codes $s^u = (s_1^u, s_2^u, \ldots, s_n^u)$ has capacity*

$$C_{S,W}(k,m,n) = \left(\begin{array}{c} n \\ \lfloor \frac{n}{2} \rfloor \end{array} \right).$$

Proof: The proof follows from the design of the codes s^u and the identity $\left(\begin{array}{c} n \\ n-k \end{array} \right) = \left(\begin{array}{c} n \\ k \end{array} \right)$.

An encoding $(\mathcal{S},\mathcal{W})$ for a DFA M may have higher capacity than M has states, i.e. $C_{S,W}(k,m,n) \geq n_{DFA}$. A DFA M may not exhaust the capacity of its encoding $(\mathcal{S},\mathcal{W})$ because the binomial coefficients are not 'dense'. When we construct an encoding for a specific DFA with n_{DFA} states, we can always find values n_{RNN} and k such that

$$C_{S,W}(k,m,n_{RNN}) \; < \; n_{DFA} \; \leq \; C_{S,W}(k,m,n_{RNN}+1).$$

i.e. we may construct an encoding $(\mathcal{S},\mathcal{W})$ whose capacity is not fully exhausted by the DFA M the encoding was derived from.

We now turn to the problem of programming weights w_{ijk} for a "n choose k" encoding. Recall the binary approximation of network state changes (48):

$$s_i^u = 0 \vee \bigvee_{j,k} w_{ijk} \wedge s_j^v \wedge i_k \tag{49}$$

The following definition will be useful:

Definition 8.5 *We say a weight conflict occurs when some code transition $\omega(s^u, a_k) = s^{u'}$ requires $w_{ijk} = 0$ while some other code transition $\omega(s^v, a_k) = s^{v'}$ requires $w_{ijk} = 1$.*

Consider the DFA shown in figure 5 with 6 states whose alphabet consists of a single symbol: A possible state encoding for that DFAs state is shown in figure 6. When we attempt to program the weights w_{ijk} such that the desired code transitions are executed for each input symbol, we observe that conflicts occur for the following weights: w_{411}, w_{511}, w_{121}, w_{321}, w_{231}, w_{431}, w_{241}, w_{521}, w_{351} and w_{451}. In general, the following lemma holds true:

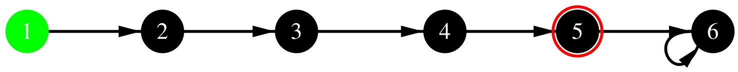

Figure 5: **A simple 'chain' DFA**: This DFA cannot be implemented with a "5 choose 2" algorithm such that no weight conflicts occur.

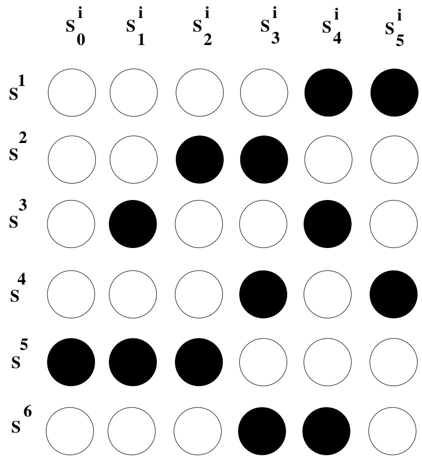

Figure 6: **Binary DFA Encoding**: The chosen binary "5 choose 2" algorithm will cause weight conflicts.

Lemma 8.4 *Weight conflicts will always occur in a "n choose k" encoding of a DFA M if M contains a chain of length l $q_j \xrightarrow{a_k} q_{j+1} \xrightarrow{a_k} q_{j+2} \cdots \xrightarrow{a_k} q_{j+l}$ such that $l > \lfloor \frac{n}{k} \rfloor + 1$.*

Proof: Encode the first $\lfloor \frac{n}{k} \rfloor$ states of M such that all codes s^u are pairwise orthogonal, i.e. no two codes s^u and s^v have any two components such that $s_i^u = s_i^v = 1$. The code $s^{\lfloor \frac{n}{k} \rfloor + 1}$ for DFA state $q_{\lfloor \frac{n}{k} \rfloor + 1}$ must have at least one component $s_i^p = s_i^{\lfloor \frac{n}{k} \rfloor + 1} = 1$ in common with code s^p $(1 < p < \lfloor \frac{n}{k} \rfloor)$. Now consider programming the weights for the code transition $\omega(s^{\lfloor \frac{n}{k} \rfloor + 1}, a_k) = s^{\lfloor \frac{n}{k} \rfloor + 2}$. That transition will require $w_{ijk} = 1$ for some pair i, j where the code transition $\omega(s^p, a_k) = a^{p+1}$ required $w_{ijk} = 0$ and vice versa. Thus a weight conflict always occurs.

There are two possible ways to resolve weight conflicts without increasing the network size: One can either decide to set $w_{ijk} = 0$ or to set $w_{ijk} = 1$ when a conflict occurs for weight W_{ijk}. However, the codes for some DFA states will become indistinguishable as a result of either strategy to resolve weight conflicts.

We can now prove the following disappointing fact about an attempted "n choose k" DFA encoding algorithm:

Theorem 8.1 *Given an arbitrary DFA M with n_{DFA} states, there does not exist a "n choose k" encoding of M with $n < k\ n_{DFA} + 1$ state neurons.*

Proof: Suppose there exists a "n choose k" encoding of a DFA M with n_{DFA} states with $n < k\ n_{DFA} + 1$. Then, there are at least two states q_u and q_v whose codes s^u and s^v are not orthogonal, i.e. there exist a component $s_j^u = s_j^v = 1$. Consider two different codes s^{u+1} and s^{v+1} which each have exactly k components with value 1 not including the response component such that $\omega(s^u, a_k) = s^{u+1}$ and $\omega(s^v, a_k) = s^{v+1}$. The components $s_{i_1}^{u+1}, s_{i_2}^{u+1}, \ldots, s_{i_k}^{u+1}$ are programmed via some weights including the weights $w_{i_1 jk}, w_{i_2 jk}, \ldots, w_{i_k jk}$. But these weights will also cause the components $s_{i_1}^{v+1}, s_{i_2}^{v1}, \ldots, s_{i_k}^{v+1}$ to have value 1. Unless s^{u+1} and s^{v+1} are identical codes, this contradicts the assumption about the "n choose k" encoding because these weights will force s^{v+1} to have more than k components with value 1. Similarly, s^{u+1} will end up with more than k components equal to 1. Thus, there do not exist weights which execute the desired code transitions for "n choose k" encodings with $n < kn_{DFA}$.

The only possible design for a "n choose k" is shown in figure 7. It is similar to the design of figure 4; however, in the previous design the k networks worked independently, whereas this design has a denser interconnection network.

The above theorem has the following corollary:

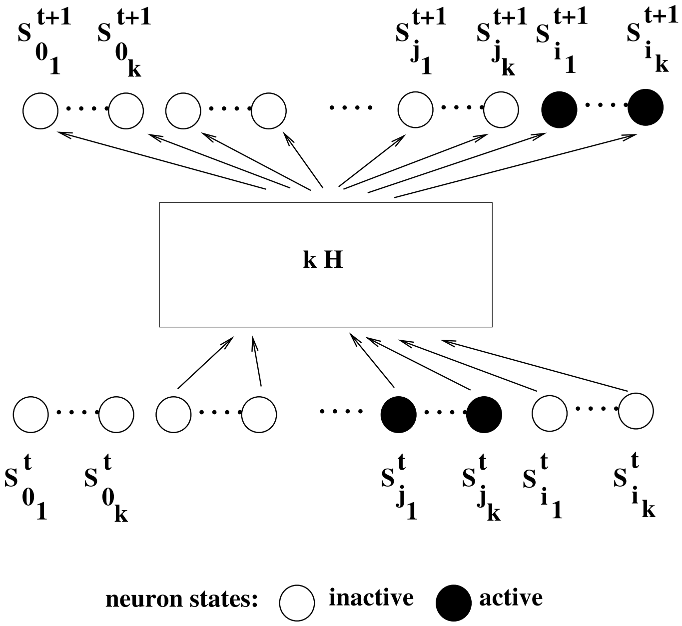

Figure 7: **Design for Tolerance to Faulty Neurons**: Each DFA state is allocated k state neurons with $S_{i_k} = 1$ such that the binary state encodings are pairwise orthogonal. The input neurons are omitted for clarity.

Corollary 8.6 *A "n choose k" encoding for a DFA M with n_{DFA} states which tolerates $k-1$ faulty neurons exists only for $n > k\ n_{DFA} + 1$.*

Proof: The proof follows from corollary 8.5 which proved the fault-tolerance of designs for DFAs whose underlying graph is completely connected.

9 Summary

We have investigated the level of fault-tolerance of different designs for implementing deterministic finite-state automata (DFAs) in second-order recurrent neural networks (SORNNs). We have considered weight perturbations and stuck-at-zero faults for weights and neurons. The original SORNN implementation of DFAs can easily be made tolerant to weight perturbation provided the maximum value of the perturbation is known. Various approaches for tolerance to stuck-at-zero faults for weights and neurons provide different levels of fault-tolerance. Since any fault-tolerant design has to based on a distribution of the resources used for the internal DFA state representation and transition, we investigated implementations based on binary encodings for DFAs and demonstrated under what conditions a binary encoding can realize any DFA. We proved that it is impossible to provide tolerance to stuck-at-zero faults for neurons without replicating the entire network. The attempt to reduce the network size while providing fault-tolerance was based on a "n choose k" encoding algorithm where k out of n recurrent state neurons were used to represent DFA states.

design	# neurons	# weights	# faulty neurons	# faulty weights
D1	$n+1$	$O(mn)$	0	0
D2	$n+1$	$O(kmn)$	0	$k-1$ [1]
D3	$(2k+1)(n+1)$	$O(kmn)$	k	k
D4	$k(n+1)$	$O(k^2mn)$	$k-1 \leq k' \leq (k-1)(n+1)$ [2]	$(k^2-1) \leq k' \leq (k^2-1)(n+1)$ [2]

Table 1: **Fault-Tolerance of Different Network Designs**: This table the number of faulty neurons and/or weights various designs can tolerate. The designs are (D1) original DFA encoding algorithm (D2) network with replicated weights (D3) replicated networks (D4) pairwise orthogonal "n choose k" encoding
[1] The design can tolerate $k-1$ faulty weights per neuron; the total number of faulty weights is $O(kmn)$.
[2] The design can tolerate a least $k-1$ faulty neurons and (k^2-1) faulty weights. If no more than $k-1$ neurons which uniquely identify the current DFA state are faulty, i.e. one of the k identifying neurons has a high output for all DFA states, then the network can tolerate a total of $(k-1)(n+1)$ faulty neurons. A similar explanation applies to faulty weights.

A summary of the network designs and their level of fault-tolerance is shown in table 1.

The representation of DFA states across state neurons has obviously an impact on the tolerance of neural DFAs to faults in their internal structure. For the construction of SORNNs from DFA, the internal representation has to be maximally sparse in order to allow for fault-tolerance.

On the other hand, it would be interesting to investigate how dense the internal DFA representation can be chosen for some given DFA, i.e. what is the smallest number of recurrent neurons necessary in order to implement some given DFA. We already discussed that it is impossible to construct a SORNN with $log(n)$ neurons for a DFA with n states except for some trivial DFAs. Finding a SORNN of minimal size as measured by the number of state neurons is probably a computationally hard problem. In fact, we believe that the following conjecture holds true:

Conjecture 9.1 *Constructing a SORNN of minimal size for some given DFA is a NP-complete problem.*

It may be possible that there exist recurrent networks with higher order product terms in the equation governing their dynamics for which fault-tolerant design require less than kn state neurons. However, an algorithm for constructing such networks is likely to be more complex and require more weights than the algorithm for constructing SORNNs.

10 ACKNOWLEDGMENT

We would like to acknowledge useful discussions with J. Giordmaine.

References

[1] 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.

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

[3] L. Leerink. Personal Communication.

[4] N. May and D. Hammerstrom, "Fault simulation of a wafer-scale integrated neural network," tech. rep., Oregon Graduate Center, 1988.

[5] C. Omlin and C. Giles, "Constructing deterministic finite-state automata in sparse recurrent neural networks," in *IEEE International Conference on Neural Networks (ICNN'94)*, pp. 1732–1737, 1994.

[6] C. Omlin and C. Giles, "Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants," Tech. Rep. UMIACS-TR-94-101, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, 1994.

[7] D. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," Tech. Rep. TR-92-CSE-26, Department of Electrical and Computer Engineering, University of Masssachusetts, Amherst, MA, 1992.

[8] C. Sequin and R. Clay, "Fault tolerance training improves generalization and robustness," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN'92), Baltimore, MD*, vol. 1, pp. 769–774, June 1992.