Equivalence in Knowledge Representation: Automata, Recurrent Neural Networks, and Dynamical Fuzzy Systems

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

[a] Department of Computer Science, University of Stellenbosch, South Africa

[b] NEC Research Institute, Princeton, NJ 08540

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

E-mail: omlin@cs.sun.ac.za {giles,karvel}@research.nj.nec.com

Abstract

Neuro-fuzzy systems - the combination of artificial neural networks with fuzzy logic - have become useful in many application domains. However, conventional neuro-fuzzy models can have inadequate representational power for applications which require context and state (e.g., speech, handwriting, control). Some of these applications can be readily modeled in the form of finite-state automata. Previously, it was proved that deterministic finite-state automata (DFAs) can be stably synthesized or mapped into recurrent neural networks by programming the networks' weights. Based on those results, a synthesis method is proposed for mapping fuzzy finite-state automata (FFAs) into recurrent neural networks. Importantly, this mapping is suitable for direct implementation in VLSI, i.e. the encoding of FFAs as a generalization of the encoding of DFAs. The synthesis method requires FFAs to undergo a transformation prior to being mapped into recurrent networks. The neurons are provided with an enriched functionality that enables the accommodation of a fuzzy representation of FFA states. This enriched neuron functionality also permits fuzzy parameters of FFAs to be directly represented as parameters of the neural network. In this paper we prove the stability of fuzzy finite-state dynamics of constructed neural networks for finite values of network weight and, through simulations, give empirical validation of the proofs.

1 Introduction

1.1 Motivation

There has been an increased interest in hybrid systems as more applications using hybrid models emerge [6, 18]. However, there are many definitions of hybrid systems [23, 25, 40, 46] (also see papers in this issue). One example of an hybrid system is in the combination of artificial neural networks and fuzzy systems (see [2, 24, 40]. Fuzzy logic [51] provides a mathematical foundation for approximate reasoning; fuzzy logic has proven very successful in a variety of applications [1, 5, 9, 11, 14, 21, 27, 31, 41, 49]. The parameters of adaptive fuzzy systems have clear physical meanings which facilitates the choice of their initial values. Furthermore, rule-based information can be incorporated into fuzzy systems in a systematic way. Fuzzy finite-state automata (FFAs) have a long history [43, 50] presenting a systematic synthesis method. Their potential as design tools for modeling a variety of systems is beginning to be exploited [28, 34]. Such systems have two major characteristics: (1) the necessity of state information where the current state of the system depends on past states and current inputs, and (2) the knowledge about the system's current state is vague or uncertain.

Artificial neural networks simulate on a small scale the information processing mechanisms found in biological systems which are based on the cooperation of neurons that perform simple operations, and on their ability to learn from examples. Recurrent neural networks (RNNs) are dynamic networks with temporal state representations and are computationally quite powerful [45]. This makes RNNs appropriate tools for modeling and learning many diverse dynamical problems from control to signal processing and times series forecasting (for more information on neural networks, see books such as Bishop [3] and Haykin [22].

The proofs of representational properties of AI and machine learning structures are important for a number of reasons. Many users of a model want guarantees about what it can theoretically do, i.e. its performance and capabilities; others need this for justification of use and acceptance. The capability of *representing* FFAs can be viewed as a foundation for the problem of *learning* FFAs from examples (if a network cannot represent FFAs, then it certainly will have difficulty in learning them). A stable encoding of knowledge means that the model will give the correct answer independent of when the system is used or how long it is used. This can lead to robustness that is noise independent. Finally, with the extraction of knowledge from trained neural networks, the methods discussed here could potentially be applied to incorporating and refining a priori fuzzy knowledge in recurrent neural networks [32].

1.2 Background

A variety of implementations of FFAs have been proposed [19, 26], some in digital systems. However, this to our knowledge is the first proof that such implementations in sigmoid activation RNNs are stable, i.e. guaranteed convergence to the correct prespecified membership. This proof is based on previous work of stably mapping deterministic finite-state automata (DFAs) in recurrent neural networks reported in [35]. In contrast to DFAs, a *set* of FFA states can be occupied to *varying degrees* at any point in time; this fuzzification of states generally reduces the size of the model, and the dynamics of the system being modeled is often more accessible to a direct interpretation.

From a control perspective, fuzzy finite-state automata have been shown to be useful for modeling fuzzy dynamical systems, often in conjunction with recurrent neural networks [8, 28, 29, 30]. There has been a lot of interest in learning, synthesis, and extraction of finite-state automata in recurrent neural networks [7, 10, 13, 15, 16, 42, 48, 52]. A variety of neural network implementations of FFAs have been proposed [19, 20, 26, 47]. We have previously shown how fuzzy finite-state automata (FFAs) can be mapped into recurrent neural networks with second-order weights using a *crisp* representation of FFA states [39]. That encoding required a transformation of a FFA into a deterministic finite-state automaton (DFA) which computes the membership functions for strings; it is only applicable to a restricted class of FFAs which have *final states*. The transformation of a fuzzy automaton into an equivalent deterministic acceptor generally increases the size of the automaton and thus the network size. Furthermore, the fuzzy transition memberships of the original FFA undergo modifications in the transformation of the original FFA into an equivalent DFA which is suitable for implementation in a second-order recurrent neural network. Thus, the direct correspondence between system and network parameters is lost; which may obscure the natural fuzzy description of systems being modeled.

The existence of a crisp recurrent network encoding for all FFAs raises the question whether recurrent networks can also be *trained* to compute the fuzzy membership function, and how they represent FFA states internally. Based on our theoretical analysis, we know that they have the ability to represent FFAs in the form of equivalent deterministic acceptors. Recent work reported in [4] addresses these questions. Instead of augmenting a second-order network with a linear output layer for computing the fuzzy string membership as suggested in [39], the authors chose to assign the fuzzy string memberships μ_i occurring in the training set to individual output neurons. Thus, they transformed the fuzzy inference problem into a binary inference problem with multiple classes. This approach lessens the burden on the training, and improves the accuracy and robustness of string membership computation. Apart from the use of multiple classes, training networks to compute the fuzzy string membership was identical with training networks to behave like DFAs. The authors empirically verified through knowledge extraction techniques [7, 36] that recurrent networks trained on fuzzy strings develop a crisp internal representation of FFAs, i.e. that they represent FFAs in the form of equivalent deterministic acceptors. Interestingly, previous theoretical analysis [39] had *correctly predicted* this type of knowledge representation of such trained networks.

1.3 Overview of Paper

Here, we present a method for encoding FFAs using a *fuzzy* representation of states. [1] The objectives of the FFA encoding algorithm are (1) ease of encoding FFAs into recurrent networks, (2) the direct representation of "fuzziness", i.e. the fuzzy memberships θ of individual transitions in FFAs are also parameters in the recurrent networks, and (3) achieving a fuzzy representation by making only minimal changes to the underlying architecture used for encoding DFAs (and crisp FFA representations).

Representation of FFAs in recurrent networks requires that the internal representation of FFA states and state transitions be stable for indefinite periods of time. We will demonstrate how the stability analysis for neural DFA encodings carries over to and generalizes the analysis of stable neural FFA representations.

The capability of *representing* FFAs can be viewed as a foundation for the problem of *learning* FFAs from examples (if a network cannot represent FFAs, then it certainly cannot learn them either). Since DFAs are used in high-level VLSI design, and recurrent neural networks can be directly implemented in chips, this approach could be useful for applications where FFAs are used in conjunction with recurrent networks and VLSI [44, 33] or the method could be applied to incorporate a priori knowledge into recurrent neural networks used for knowledge refinement [17].

The remainder of this paper is organized as follows: Fuzzy finite-state automata are introduced in Section 2. The fuzzy representation of FFA states and transitions in recurrent networks are discussed in Section 3. The mapping fuzzy automata $\rightarrow$ recurrent network requires that FFAs be transformed into equivalent FFAs which eliminate ambiguities among the fuzzy transitions memberships that lead to a single FFA state; the transformation algorithm presented in Section 4. The recurrent network architecture for representing FFAs is described in Section 5. The stability of the encoding is derived in Section 6. A discussion of simulation results in Section 7, and a summary of the results and possible directions for future research in Section 8 conclude this paper.

[1] For reasons of completeness, we have included the main results from [35] which laid the foundations for this and other papers [38, 39].

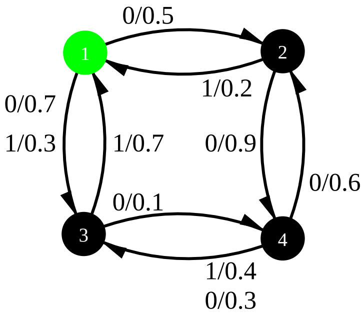


Figure 1: **Example of a Fuzzy Finite-State Automaton:** A fuzzy finite-state automaton is shown with weighted state transitions. State 1 is the automaton's start state. A transition from state q_j to q_i on input symbol a_k with weight θ is represented as a directed arc from q_j to q_i labeled a_k/θ. Note that transitions from states 1 and 4 on input symbols '0' are fuzzy ($\delta(1,0,.) = \{2,3\}$ and $\delta(4,0,.) = \{2,3\}$).

2 Fuzzy Finite-State Automata

In this section, we give a formal definition of FFAs [12] and illustrate the definition with an example.

Definition 2.1 *A fuzzy finite state automaton (FFA) M is a 6-tuple $M =< \Sigma, Q, R, Z, \delta, \omega >$ where $\Sigma = \{a_1, \ldots, a_m\}$ is the alphabet, $Q = \{q_1, \ldots, q_n\}$ is a set of states, $R \in Q$ is the automaton's fuzzy start state* [2] *, Z is a finite output alphabet, $\delta : \Sigma \times Q \times [0,1] \rightarrow Q$ is the fuzzy transition map, and $\omega : Q \rightarrow Z$ is the output map.*

Weights $\theta_{ijk} \in [0,1]$ define the 'fuzziness' of state transitions, i.e. a FFA can simultaneously be in different states with a different degree of certainty. The particular output mapping depends on the nature of the an application. Since our goal is to construct a fuzzy representation of FFA states and their stability over time, we will ignore the output mapping ω for the remainder of this discussion, and not concern ourselves with the language $L(M)$ defined by M. For a possible definition, see e.g. [12]. An example of a FFA over the input alphabet $\{0,1\}$ is shown in Figure 1.

3 Representation of Fuzzy States

3.1 Preliminaries

The current fuzzy state of a FFA M is a collection of states $\{q_i\}$ of M which are occupied with different degrees of fuzzy membership. A fuzzy representation of FFA states thus requires that knowledge about the

[2] In general, the start state of a FFA is fuzzy, i.e. it consists of a set of states that are occupied with varying memberships. It has been shown that a restricted class of FFAs whose initial state is a single crisp state is equivalent with the class of FFAs described in Definition 2.1 [12]. The distinction between the two classes of FFAs is irrelevant in the context of this paper.

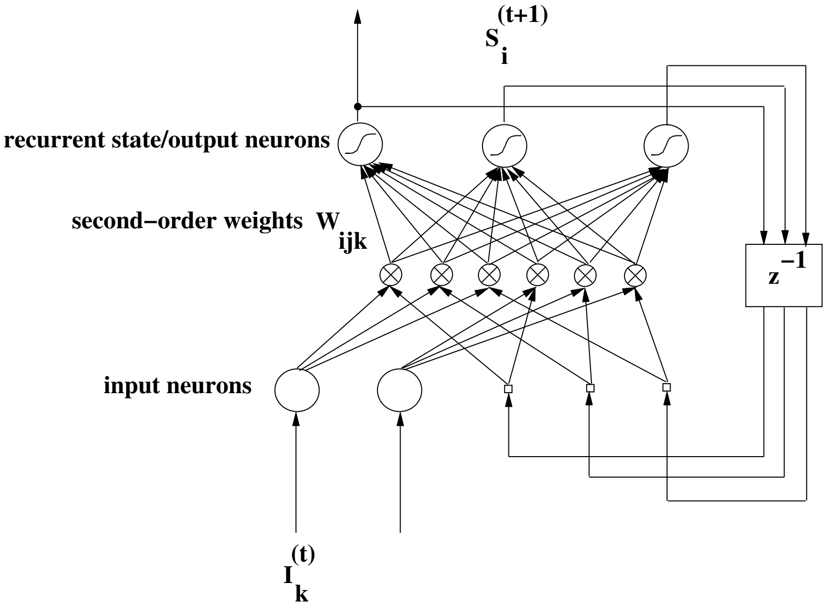


Figure 2: **Recurrent Network Architecture for Deterministic Finite-State Automata:** The recurrent state neurons are connected and implement the stable finite-state dynamics. One of the recurrent neurons also is the dedicated network output neuron (i.e., the neuron which by its output value classifies whether or not a given string is a member of a regular language).

memberships with which states $\{q_i\}$ are occupied be available. This requirement dictates the representation of the current fuzzy state in a recurrent neural network. Since the method for encoding FFAs in recurrent neural networks is a generalization of the method for encoding DFAs, we will briefly discuss the DFA encoding algorithm.

3.2 DFA Encoding Algorithm

We make use of an algorithm used for encoding deterministic finite-state automata (DFAs) [35, 38]. For encoding DFAs, we used discrete-time, second-order recurrent neural networks with sigmoidal discriminant functions which update their current state according to the following equations:

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

where b_i is the bias associated with hidden recurrent state neurons S_i, W_{ijk} is a second-order weight, and I_k denotes the input neuron for symbol a_k. The indices i, j, and k run over all state and input neurons, respectively. The product $S_j^{(t)} I_k^{(t)}$ directly corresponds to the state transition $\delta(q_j, a_k) = q_i$. The architecture is illustrated in Figure 2.

DFAs can be encoded in discrete-time, second-order recurrent neural networks with sigmoidal discriminant functions such that the DFA and constructed network accept the same regular language [35]. The desired finite-state dynamics are encoded into a network by programming a small subset of all available weights to values $+H$ and $-H$; this leads to a nearly orthonormal internal DFA state representation for sufficiently

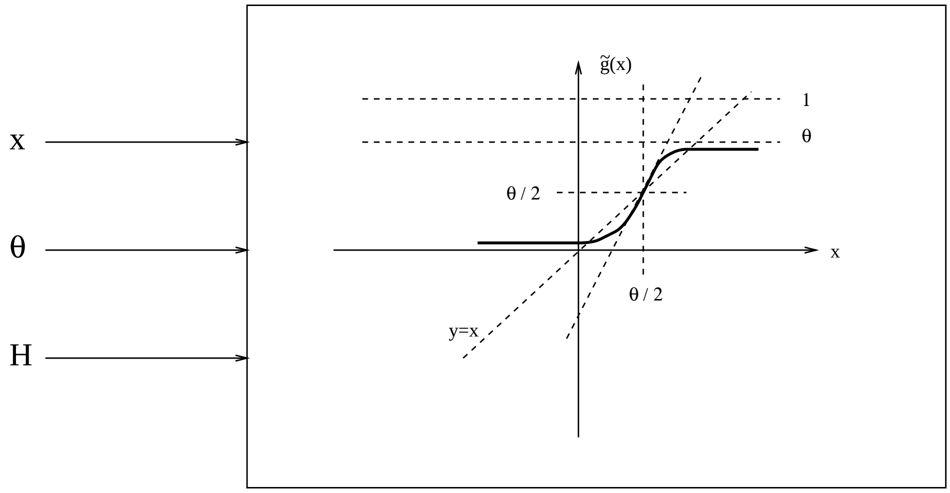


Figure 3: **Neuron Discriminant Function for Fuzzy States**: A neuron receives as input the collective signal x from all other neurons, the weight strength H, and the fuzzy transition membership θ to compute the function $\tilde{g}(x, H, \theta) = \frac{\theta}{1+e^{H(\theta-2x)/2\theta}}$. Thus, the sigmoidal discriminant function used to represent FFA states has variable output range.

large values of H. Since the magnitude of all weights in a constructed network are equal to H, the equation governing the dynamics of a constructed network is of the special form

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

where x is the input to neuron S_i. We have previously demonstrated that the stability of the internal DFA state representation is *independent* of the string length. In constructed networks, the recurrent state neurons always operate near their saturation regions for sufficiently large values of H; as a consequence, the internal DFA state representation remains indefinitely stable. We will apply the same tools and techniques to prove stability of the internal representation of fuzzy states in recurrent neural networks.

3.3 Recurrent State Neurons with Variable Output Range

We extend the functionality of recurrent state neurons in order to represent fuzzy states as illustrated in Figure 3. The main difference between the neuron discriminant function for DFAs and FFAs is that the neuron now receives as inputs the weight strength H, the signal x which represents the collective input from all other neurons, and the transition weight θ_{ijk}, where $\delta(q_j, a_k, \theta_{ijk}) = q_i$; we will denote this triple with (x, H, θ_{ijk}). The value of θ_{ijk} is different for each of the states that collectively make up the current fuzzy network state. This is consistent with the definition of FFAs.

The following generalized form of the sigmoidal discriminant function $g(.)$ will be useful for representing

FFA states:

$$S_i^{(t+1)} = \tilde{g}(x, H, \theta_{ijk}) = \frac{\theta_{ijk}}{1 + e^{H(\theta_{ijk} - 2x)/2\theta_{ijk}}} \tag{3}$$

Compared to the discriminant function $g(.)$ for the encoding of DFAs, the weight H which programs the network state transitions is strengthened by a factor $1/\theta_{ijk}$ $(0 < \theta_{ijk} \leq 1)$; the range of the function $\tilde{g}(.)$ is squashed to the interval $[0, \theta_{ijk}]$, and it has been shifted towards the origin. Setting $\theta_{ijk} = 1$ reduces the function (3) to the sigmoidal discriminant function (2) used for DFA encoding.

More formally, the function $\tilde{g}(x, H, \theta)$ has the following important invariant property which will later simplify the analysis:

Lemma 3.1 $\tilde{g}(\theta x, H, \theta) = \theta \; \tilde{g}(x, H, 1) = \theta \; g(x, H)$.

Proof: $\tilde{g}(\theta x, H, \theta) = \dfrac{\theta}{1 + e^{H(\theta - 2\theta x)/2\theta}} = \dfrac{\theta}{1 + e^{H(1-2x)/2}} = \theta \; \tilde{g}(x, H, 1) = \theta \; g(x, H)$.

□

Thus, $g(x, H)$ can be obtained by scaling $\tilde{g}(x, H, 1)$ uniformly in the $x-$ and $y-$directions by a factor θ.

The above property of $\tilde{g}$ allows a stability analysis of the internal FFA state representation similar to the analysis of the stability of the internal DFA state representation to be carried out.

3.4 Programming Fuzzy State Transitions

Consider state q_j of FFA M and the fuzzy state transition $\delta(q_j, a_k, \{\theta_{ijk}\} = \{q_{i_1} \ldots q_{i_r}\})$. We assign recurrent state neuron S_j to FFA state q_j and neurons $S_{i_1} \ldots S_{i_r}$ to FFA states $q_{i_1} \ldots q_{i_r}$. The basic idea is as follows: The activation of recurrent state neuron S_i represents the certainty θ_{ijk} with which some state transition $\delta(q_j, a_k, \theta_{ijk}) = q_i$ is carried out, i.e. $S_i^{t+1} \simeq \theta_{ijk}$. If q_i is not reached at time $t+1$, then we have $S_i^{t+1} \simeq 0$.

We program the second-order weights W_{ijk} as follows:

$$W_{ijk} = \begin{cases} +H & \text{if } q_i \in \delta(q_j, a_k, \theta_{ijk}) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

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

$$b_i = -H/2 \ if \ \ q_i \in M. \tag{6}$$

Setting W_{ijk} to a large positive value will ensure that S_i^{t+1} will be arbitrarily close to θ_{ijk} and setting W_{jjk} to a large negative value will guarantee that the output S_j^{t+1} will be arbitrarily close to 0. This is the same technique using for programming DFA state transitions in recurrent networks [35] and for encoding partial prior knowledge of a DFA for rule refinement [37].

4 Automata Transformation

4.1 Preliminaries

The above encoding algorithm leaves open the possibility for ambiguities when a FFA is encoded in a recurrent network as follows: Consider two FFA states q_j and q_l with transitions $\delta(q_j, a_k, \theta_{ijk}) = \delta(q_l, a_k, \theta_{ilk}) = q_i$ where q_i is one of all successor states reached from q_j and q_l, respectively, on input symbol a_k. Further assume that q_j and q_l are members of the set of current FFA states (i.e., these states are occupied with some fuzzy membership). Then, the state transition $\delta(q_j, a_k, \theta_{ijk}) = q_i$ requires that recurrent state neuron S_i have dynamic range $[0, \theta_{ijk}]$ while state transition $\delta(q_l, a_k, \theta_{ilk}) = q_i$ requires that state neuron S_i asymptotically approach θ_{ilk}. For $\theta_{ijk} \neq \theta_{ilk}$, we have ambiguity for the output range of neuron S_i:

Definition 4.1 *We say an ambiguity occurs at state q_i if there exist two states q_j and q_l with $\delta(q_j, a_k, \theta_{ijk}) = \delta(q_l, a_k, \theta_{ilk}) = q_i$ and $\theta_{ijk} \neq \theta_{ilk}$. A FFA M is called ambiguous if an ambiguity occurs for any state $q_i \in M$.*

4.2 Transformation Algorithm

That ambiguity could be resolved by testing all possible paths through the FFA and identify those states for which the above described ambiguity can occur. However, such an endeavor is computationally expensive. Instead, we propose to resolve that ambiguity by transforming any FFA M.

Before we state the transformation theorem, and give the algorithm, it will be useful to define the concept of equivalent FFAs.

Definition 4.2 *Consider a FFA M that is processing some string $s = \sigma_1\sigma_2 \ldots \sigma_L$ with $\sigma_i \in \Sigma$. As M reads each symbol σ_i, it makes simultaneous weighted state transitions $\Sigma \times Q \times [0,1]$ according to the fuzzy transition map $\delta(q_j, a_k, \theta_{ijk}) = q_i$. The set of distinct weights $\{\theta_{ijk}\}$ of the fuzzy transition map at time t is called the active weight set.*

Note that the active weight set can change with each symbol σ_i processed by M. We will define what it means for two FFAs to be equivalent:

Input: FFA $M =< \Sigma, Q, R, Z, \delta, \omega >$ with $\Sigma = \{a_1, \ldots, a_M\}$ and $Q = \{q_1, \ldots, q_N\}$.

Output: FFA $M' =< \Sigma, Q', R', Z, \delta', \omega >$ with $\Sigma = \{a_1, \ldots, a_M\}$ and $Q' = \{q_1, \ldots, q_N, q_{N+1}, \ldots, q_X\}$ with the properties

(1) $M \equiv M'$ and

(2) there exist no two states q_j and q_l in M' with $\delta(q_j, a_k, \theta_{ijk}) = \delta(q_l, a_k, \theta_{ilk}) = q_i$ with $\theta_{ijk} \neq \theta_{ilk}$.

Algorithm:

1. $X \leftarrow N$; $list \leftarrow Q$;
while $list \neq \emptyset$ **do**
2. $list \leftarrow list \setminus \{q_i\}$;
for $k = 1 \ldots M$ **do**
3. visit $\leftarrow \emptyset$;
for $j = 1 \ldots N$ **do**
4. **if** $\delta(q_j, a_k, \cdot) = q_i$ **then** visit $\leftarrow$ visit $\cup$ $\{q_j\}$;
end
5. class $\leftarrow \{q_l \in$ visit $\mid \delta(q_l, a_k, \theta_{ilk}) = q_i$ with $\theta_{ilk} = \theta_{ik}\}$;
6. visit $\leftarrow$ visit $\setminus$ {class};
while class $\neq \emptyset$ **do**
7. class $\leftarrow \{q_l \in$ visit $\mid \delta(q_l, a_k, \theta_{ilk}) = q_i$ with $\theta_{ilk} = \theta_{ik}\}$;
8. visit $\leftarrow$ visit $\setminus$ {class};
9. $X \leftarrow X + 1$;
10. $Q \leftarrow Q \cup \{q_X\}$; /* create new FFA state q_X */
for each q_j **in** class **do**
11. $\delta(q_j, a_k, \theta_{ijk}) \leftarrow q_X$; /* change transition */
for $l = 1 \ldots N$ **do**
for $k = 1 \ldots M$ **do**
12. $\delta(q_X, a_k, \theta_{lXk}) \leftarrow \delta(q_i, a_k, \theta_{lik})$; /* implies $\theta_{lXk} \leftarrow \theta_{lik}$) */
end
end
end
end
end
end

Figure 4: **Algorithm for FFA Transformation.**

Definition 4.3 *Two FFAs M and M' with alphabet Σ are called* equivalent *if their active weight sets are at all times identical for any string $s \in \Sigma^*$.*

We will prove the following theorem:

Theorem 4.1 *Any FFA M can be transformed into an equivalent, unambiguous FFA M'.*

The trade-off for making the resolution of ambiguities computationally feasible is an increase in the number of FFA states. The algorithm that transform a FFA M into a FFA M' such that $L(M) = L(M')$ is shown in Figure 4. Before we prove the above theorem, we will discuss an example of FFA transformation.

4.3 Example

Consider the FFA shown in Figure 5a with four states and input alphabet $\Sigma = \{0, 1\}$; state q_1 is the start statẽfootnote The FFA shown in Figure 5a is a special case in that it does not contain any fuzzy transitions. Since the objective of the transformation algorithm is to resolve ambiguities for states q_i with $\delta(\{q_{j_1}, \ldots, q_{j_r}\}, a_k, \{, \theta_{ij_1k}, \ldots, \theta_{ij_rk}\}) = q_i$, fuzziness is of no relevance; therefore, we omitted it for reasons of simplicity.. The algorithm initializes the variable 'list' with all FFA states, i.e., list=$\{q_1, q_2, q_3, q_4\}$. First, we notice that no ambiguity exists for input symbol '0' at state q_1 since there are no state transitions $\delta(., 0, .) = q_1$. There exist two state transitions which have state q_1 as its target, i.e. $\delta(q_2, 1, 0.2) = \delta(q_3, 1, 0.7) = q_1$. Thus, we set the variable $visit = \{q_2, q_3\}$. According to Definition 4.1, an ambiguity exists since $\theta_{121} \neq \theta_{131}$. We resolve that ambiguity by introducing a new state q_5 and setting $\delta(q_3, 1, 0.7) = q_5$. Since $\delta(q_3, 1, 0.7)$ no longer leads to state q_1, we need to introduce new state transitions leading from state q_5 to the target states $\{q\}$ of all possible state transitions: $\delta(q_1, ., .) = \{q_2, q_3\}$. Thus, we set $\delta(q_5, 0, \theta_{250}) = q_2$ and $\delta(q_5, 1, \theta_{351}) = q_3$ with $\theta_{250} = \theta_{210}$ and $\theta_{351} = \theta_{311}$. One iteration through the outer loop thus results in the FFA shown in Figure 5b. Consider Figure 5d which shows the FFA after 3 iterations. State q_4 is the only state left which has incoming transitions $\delta(., a_k, \theta_{4.k}) = q_4$ where not all values $\theta_{4.k}$ are identical. We have $\delta(q_2, 0, 0.9) = \delta(q_6, 0, 0.9) = q_4$; since these two state transition do not cause an ambiguity for input symbol '0', we leave these state transitions as they are. However, we also have $\delta(q_2, 0, \theta_{420}) = \delta(q_3, 0, \theta_{430}) = \delta(q_7, 0, \theta_{470}) = q_4$ with $\theta_{430} = \theta_{470} \neq \theta_{420} = 0.9$. Instead of creating new states for both state transitions $\delta(q_3, 0, \theta_{430})$ and $\delta(q_7, 0, \theta_{470})$, it suffices to create one new state q_8 and to set $\delta(q_3, 0, 0.1) = \delta(q_7, 0, 0.1) = q_8$. States q_6 and q_7 are the only possible successor states on input symbols '0' and '1', respectively. Thus, we set $\delta(q_8, 0, 0.6) = q_6$ and $\delta(q_8, 1, 0.4) = q_7$. There exist no more ambiguities and the algorithm terminates (Figure 5e).

4.4 Properties of the Transformation Algorithm

We have shown with an example how the algorithm transforms any FFA M into a FFA M' without ambiguities. We now need to show that the algorithm correctly transforms M into M', i.e. we need to show that M and M' are equivalent. In addition, we also need to demonstrate that the algorithm terminates for any

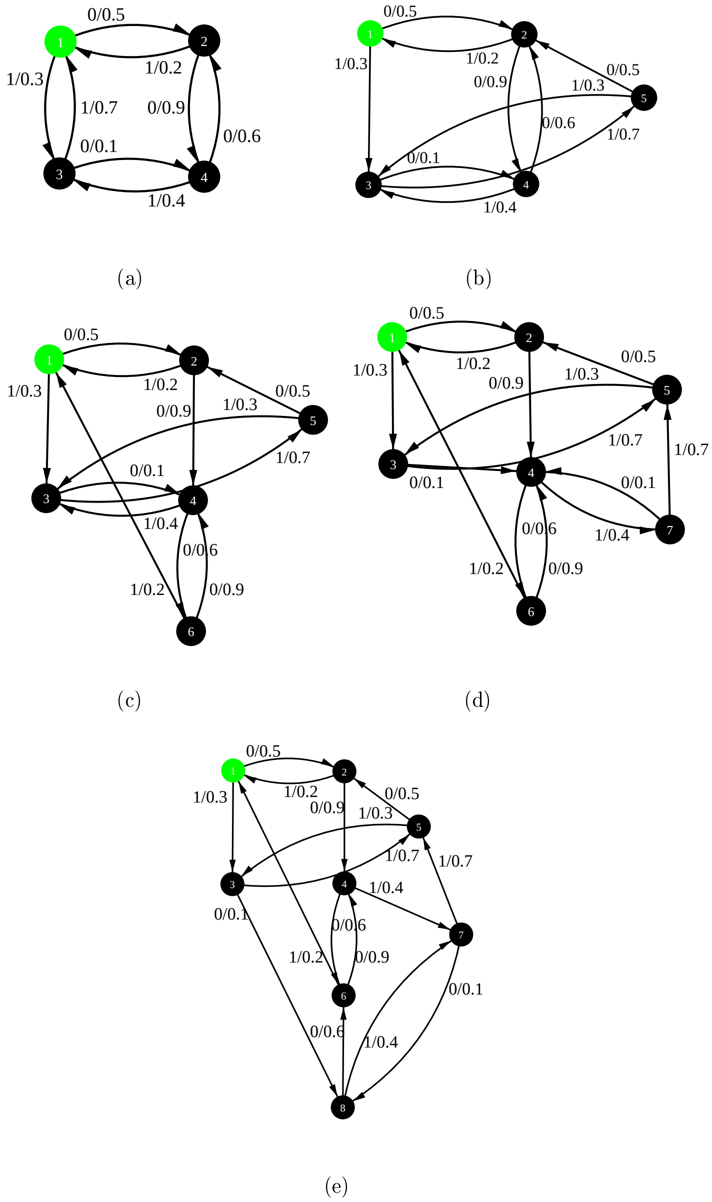


Figure 5: **Example of FFA Transformation:** Transition weight ambiguities are resolved in a sequence of steps: (a) the original FFA; there exist ambiguities for all four states; (b) the ambiguity of transition from state 3 to state 1 on input symbol 1 is removed by adding a new state 5; (c) the ambiguity of transition from state 4 to state 2 on input symbol 0 is removed by adding a new state 6; (d) the ambiguity of transition from state 4 to state 3 on input symbol 1 is removed by adding a new state 7; (e) the ambiguity of transition from states 3 and 7 - both transition have the same fuzzy membership - to state 4 is removed by adding a new state 8.

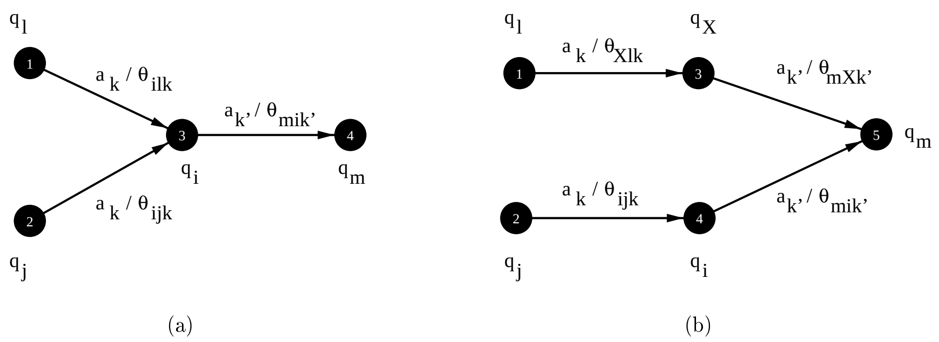


Figure 6: **Resolution of Ambiguities:** The transition ambiguity from states q_l and q_j to state q_i on input symbol a_k is resolved by adding a new state q_X and adjusting the transition as shown.

input M.

First, we prove the following property of the transformation algorithm:

Lemma 4.1 *Resolution of an ambiguity does not result in a new ambiguity.*

Proof: Consider the situation illustrated in Figure 6a. Let q_i, q_j, q_l, q_m be four FFA states and let there be an ambiguity at state q_i on input symbol a_k, i.e. $\delta(q_j, a_k, \theta_{ijk}) = \delta(q_l, a_k, \theta_{ilk}) = q_i$ with $\theta_{ijk} \neq \theta_{ilk}$. Furthermore, let $\delta(q_i, a_{k'}, \theta_{mik'}) = q_m$. The ambiguity is resolved by creating a new state q_X. We arbitrarily choose the state transition $\delta(q_l, a_k, \theta_{ilk}) = q_i$ and set $\delta(q_l, a_k, \theta_{Xlk}) = q_X$ with $\theta_{Xlk} = \theta_{ilk}$. This removes the ambiguity at state q_i. We now need to introduce a new state transition $\delta(q_X, a'_k, \theta_{mXk'}) = q_m$. By observing that $\theta_{mXk'} = \theta_{mik}$ we conclude that no new ambiguity has been created at state q_m following the resolution of the ambiguity at state q_i.

□

We observe that M' is not unique, i.e. the order in which states are visited and the order in which state transition ambiguities are resolved determine the final FFA M'. Consider the FFA in Figure 5a. In our example, if we had chosen to change transition $\delta(q_2, 1, 0.2) = q_1$ instead of state transition $\delta(q_3, 1, 0.7) = q_1$, then the resulting FFA M' would have been different. However, all possible transformations M' share a common invariant property:

Lemma 4.2 *The number of states in M' is constant regardless of the order in which states are visited and state transition ambiguities are resolved.*

Proof: To see that the lemma's claim holds true, we observe that resolving an ambiguity consists of creating a new state for each set of states $\{q_j\}$ with $\delta(q_j, a_k, \theta_{ijk}) = q_i$ with $\forall j \neq j' : \theta_{ijk} \neq \theta_{ij'k}$. Since resolving the ambiguity for any state q_i does not introduce new ambiguities (see Lemma 4.1), the number of newly created states only depends on the number FFA states with ambiguities.

□

The following definitions will be convenient:

Definition 4.4 *The* <u>*outdegree* $d_{out}(q_i)$</u> *of a state q_i in FFA M is the maximum number of states q_j for which we have $\delta(q_i, a_k, \theta_{ijk}) = \{q_j\}$ for fixed a_k with $\theta_{jik} > 0$ where the maximum is taken over all symbols a_k. The* <u>*maximum outdegree* $D_{out}(M)$</u> *of some FFA M is the maximum over all $d_{out}(q_i)$ with $q_i \in M$.*

Definition 4.5 *The* <u>*indegree* $d_{in}(q_i)$</u> *of a state q_i in FFA M is the maximum number of states q_j for which we have $\delta(\{q_j\}, a_k, \theta_{ijk}) = q_i$ for fixed a_k with $\theta_{ijk} > 0$ where the maximum is taken over all symbols a_k. The* <u>*maximum indegree* $D_{in}(M)$</u> *of some FFA M is the maximum over all $d_{in}(q_i)$ with $q_i \in M$.*

We can give a very loose upper bound for the number of states in M' as follows:

Lemma 4.3 *For a FFA M with N states and K input symbols, the transformed FFA has at most $D_{in}KN(N-1)$ states.*

Proof: Consider some arbitrary state q_i of M. It can have at most $D_{in}N$ incoming transitions for input symbol a_k. The resolution of ambiguity for state q_i requires that all but one transition $\delta(., a_k, \theta_{i.k})$ lead to a new state. In the case where the fuzzy transition memberships $\theta_{i.k}$ are all different, $N-1$ new states are created per ambiguous state. Thus, for K input symbols, at the most $D_{in}KN(N-1)$ new states are created.

□

The results in Table 1 show the size of randomly generated FFAs M with input alphabet $\{0,1\}$, the maximum outdegree $D_{out}(M)$, the upper bound on the size of transformed FFAs M' and average and standard deviation of actual sizes for transformed FFAs M' taken over 100 experiments. The random FFAs M were generated by connecting each state of M to at most D_{out} other states for given input symbol. We observe that the average actual size of transformed FFAs depends on the maximum outdegree $D_{out}(M)$ and appears to be linear in N and D_{out}. Lemma 4.3 has the following corollary:

Corollary 4.1 *The FFA transformation algorithm terminates for all possible FFAs.*

Proof: The size of the set *list* in the algorithm decreases monotonically with each iteration. Thus, the outer while loop terminates when list $= \emptyset$. Likewise, the inner while loop terminates since there are only a finite

size of M	$D_{out}(M)$	upper limit on size of M'	average size of M'	standard deviation
10	1	180	12	2
	2	360	16	5
	3	540	19	15
	4	720	25	29
	5	900	28	84
20	1	760	25	6
	2	1520	32	19
	3	2280	40	40
	4	3040	50	191
	5	3800	68	278
30	1	1740	38	7
	2	3480	49	27
	3	5220	61	64
	4	6960	84	266
	5	8700	111	578
40	1	2400	51	6
	2	4800	65	29
	3	7200	85	104
	4	9600	117	342
	5	12000	154	1057
50	1	4900	65	14
	2	9800	84	41
	3	14700	107	217
	4	19600	154	704
	5	24500	198	1478
100	1	19800	129	26
	2	39600	161	64
	3	59400	215	285
	4	78800	309	1845
	5	98600	401	3916

Table 1: **Scaling of Transformed FFAs:** The results show the increase of the size of FFA M due to its transformation into a FFA M' without ambiguities as a function of the size of M and the maximum outdegree $D_{out}(M)$. The FFAs were randomly generated and the average was computed over 100 transformations. The average size of transformed FFA appears to be linear in N and D_{out}.

number of states q_l in the set 'class' and the size of that set monotonically decreases with each iteration. Thus, the algorithm terminates.

$\square$

We now return to the proof of Theorem 4.1. We have already proven that the applying the FFA transformation algorithm results in a FFA where no ambiguities exist. It is easy to see that the a transformed FFA M' is equivalent with the original FFA M since no new fuzzy transition memberships have been added, and the algorithm leaves unchanged the order in which FFA transitions are executed. This completes the proof of Theorem 4.1.

$\square$

The above transformation algorithm removes all ambiguities for incoming transitions. However, a minor adjustment for the neural FFA encoding is needed: Given a FFA state q_i with $\delta(q_j, a_k, \theta_{ijk}) = q_i$ and $\delta(q_j, a_k, .) \neq q_i$, the corresponding weight W_{iik} is set to $-H$. We also need to specify an implicit value $\theta_{iik} > 0$ for the neural FFA encoding even though we have $\theta_{iik} = 0$ in the FFA. In order to be consistent with regard to neurons with variable output range, we set $\theta_{iik} = \theta_{ijk}$.

5 Network Architecture

The architecture for representing FFA is shown in Figure 7. A layer of sparsely connected recurrent neurons implements the finite state dynamics. Each neuron S_i of the state transition module has a dynamical output range $[0, \theta_{ijk}]$ where θ_{ijk} is the rule weight in the FFA state transition $\delta(q_j, a_k, \theta_{ijk}) = q_i$. Notice that each neuron S_i is only connected to pairs (S_i, I_k) for which $\theta_{ijk} = \theta_{ij'k}$ since we assume that M is transformed into an equivalent, unambiguous FFA M' prior to the network construction. The weights W_{ijk} are programmed as described in Section 3.4. Each recurrent state neurons receives as inputs the value S_j^t and an output range value θ_{ijk}; it computes its output according to Equation (3).

6 Network Stability Analysis

6.1 Preliminaries

In order to demonstrate how the FFA encoding algorithm achieves stability of the internal FFA state representation for indefinite periods of time, we need to understand the dynamics of signals in a constructed recurrent neural network.

We define stability of an internal FFA state representation as follows:

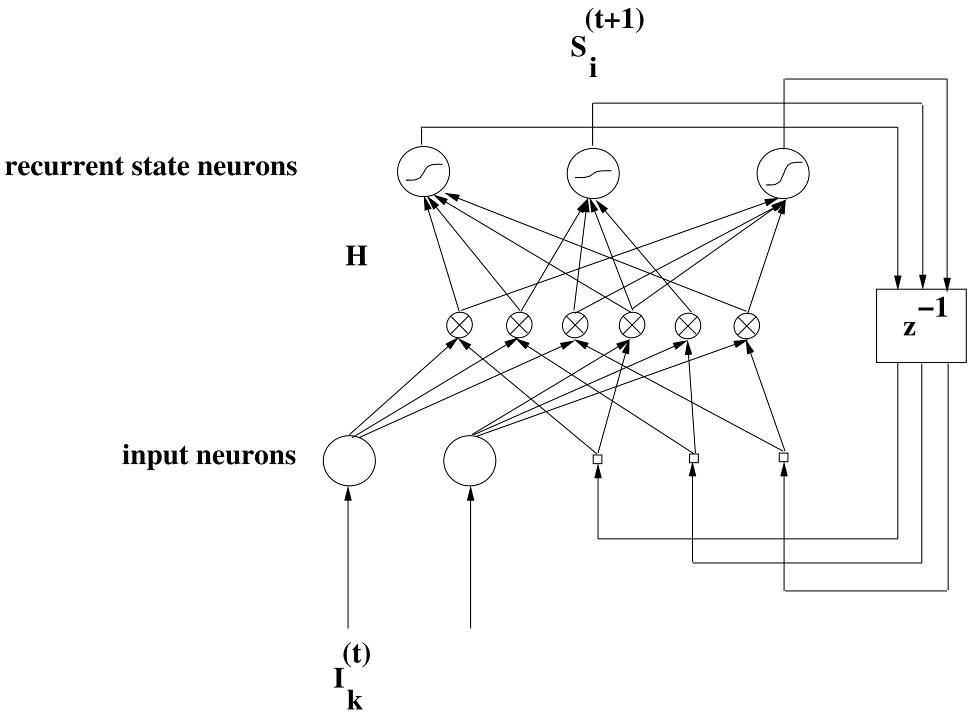


Figure 7: **Network Architecture for FFA Representation**: The architecture for representing FFA differs from that for DFAs in that (1) the recurrent state neurons have variable output range, (2) the resolution of ambiguities causes a sparser interconnection topology, and (3) there is no dedicated output neuron.

Definition 6.1 *An fuzzy encoding of FFA states with transition weights $\{\theta_{ijk}\}$ in a second-order recurrent neural network is called* <u>stable</u> *if only state neurons corresponding to the set of current FFA states have an output greater than $\theta_{ijk}/2$ where θ_{ijk} is the dynamic range of recurrent state neurons, and all remaining recurrent neurons have low output signals less than $\theta_{ijk}/2$ for all possible input sequences.*

It follows from that definition that there exists an upper bound $0 < \phi^- < \theta_{ijk}/2$ for low signals and a lower bound $\theta_{ijk}/2 < \phi^+ < \theta_{ijk}$ for high signals in networks that represent stable FFA encodings. The ideal values for low and high signals are 0 and θ_{ijk}, respectively.

A detailed analysis of all possible network state changes in [35] revealed that, for the purpose of demonstrating stability of internal finite-state representations, it is sufficient to consider the following two worst cases: (1) A neuron which does not correspond to a current fuzzy automaton state receives the same residual low input from all other neurons that it is connected to, and that value is identical for all neurons. (2) A neuron which changes its output from low to high at the next time step receives input only from one other neuron (i.e. the neuron which corresponds to the current fuzzy automaton state), and it may inhibit itself. In the case of FFAs, a neuron S_i undergoing a state change from $S_i^t \approx 0$ to $S_I^{t+1} \approx \theta_{ijk}$ may receive principal inputs from more than one other neuron. However, any additional input only serves to strengthen high signals. Thus, the case of a neuron receiving principal input from exactly one other neuron represents a worst case.

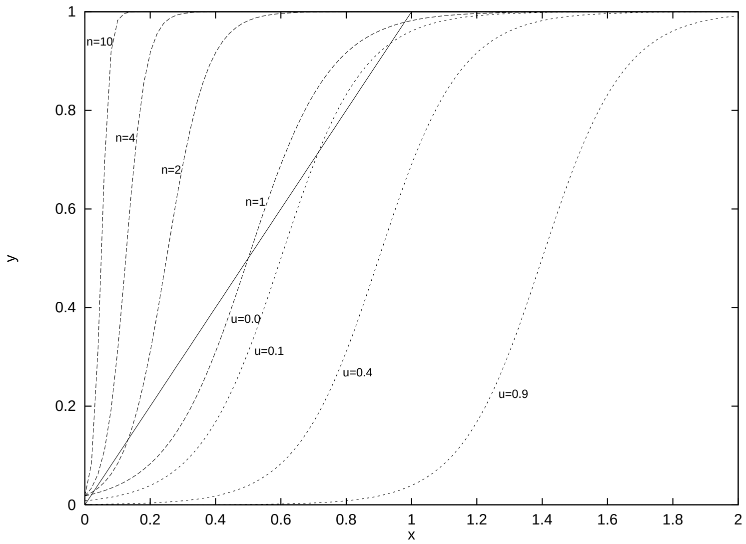


Figure 8: **Fixed Points of the Sigmoidal Discriminant Function:** Shown are the graphs of the function $f(x, H, 1, r) = \frac{1}{1+e^{H(1-2rx)/2}}$ (dashed graphs) for $H = 8$ and $r = \{1, 2, 4, 10\}$ and the function $p(x, u) = \frac{1}{1+e^{H(1-2(x-u))/2}}$ (dotted graphs) for $H = 8$ and $u = \{0.0,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

6.2 Fixed Point Analysis for Sigmoidal Discriminant Function

Here, we summarize without proofs some of the results that we used to demonstrate stability of neural DFA encodings; details of the proofs can be found in [35].

In order to guarantee low signals to remain low, we have to give a tight upper bound for low signals which remains valid for an arbitrary number of time steps:

Lemma 6.1 *The low signals are bounded from above by the fixed point* $[\phi_f^-]_\theta$ *of the function* f

$$\begin{cases} f^0 = 0 \\ f^{t+1} = \tilde{g}(r \cdot f^t) \end{cases} \tag{7}$$

where $[\phi_f^-]_\theta$ represents the fixed point of the discriminant function $\tilde{g}()$ with variable output range θ, and r denotes the maximum number of neurons that contribute to a neuron's input. For reasons of simplicity, we will write ϕ_f^- for $[\phi_f^-]_\theta$ with the implicit understanding that the location of fixed points depends on the particular choice of θ. This lemma can easily be proven by induction on t.

It is easy to see that the function to be iterated in Equation (7) is $f(x, H, \theta, r) = \dfrac{\theta}{1 + e^{H(\theta-2rx)/2\theta}}$. The graphs of the function for $\theta = 1.0$ are shown in Figure 9 for different values of the parameter r. It is obvious that the location of fixed points depends on the particular values of θ. We will show later in this section that the conditions that guarantee the existence of one or three fixed points are *independent* of the parameter θ.

The function $f(x, H, \theta, r)$ has some desirable properties:

Lemma 6.2 *For any $H > 0$, the function $f(x, H, \theta, r)$ has at least one fixed point ϕ_f^0.*

Lemma 6.3 *There exists a value $H_0^-(r)$ such that for any $H > H_0^-(r)$, $f(x, \theta, r)$ has three fixed points $0 < \phi_f^- < \phi_f^0 < \phi_f^+ < \theta$.*

Lemma 6.4 *If $f(x, \theta, r)$ has three fixed points ϕ_f^-, ϕ_f^0, and ϕ_f^+, then*

$$\lim_{t \to \infty} f^t = \begin{cases} \phi_f^- & x_0 < \phi_f^0 \\ \phi_f^0 & x_0 = \phi_f^0 \\ \phi_f^+ & x_0 > \phi_f^0 \end{cases} \tag{8}$$

The above lemma can be proven by defining an appropriate Lyapunov function P and showing that P has minima at ϕ_f^- and ϕ_f^+.

The basic idea behind the network stability analysis is to show that neuron outputs never exceed or fall below some fixed points ϕ^- and ϕ^+, respectively. The fixed points ϕ_f^- and ϕ_f^+ are only valid upper and lower bounds on low and high signals, respectively, if convergence toward these fixed points is monotone. The following corollary establish monotone convergence of f^t towards fixed points:

Corollary 6.1 *Let $f^0, f^1, f^2, \ldots$ denote the finite sequence computed by successive iteration of the function f. Then we have $f^0 < f^1 < \ldots < \phi_f$ where ϕ_f is one of the stable fixed points of $f(x, H, \theta, r)$.*

With these properties, we can quantify the value $H_0^-(r)$ such that for any $H > H_0^-(r)$, $f(x, \theta, r)$ has three fixed points. The low and high fixed points ϕ_f^- and ϕ_f^+ are the bounds for low and high signals, respectively. The larger r, the larger H must be chosen in order to guarantee the existence of three fixed points. If H is not chosen sufficiently large, then f^t converges to a unique fixed point $\theta/2 < \phi_f < \theta$. The following lemma expresses a quantitative condition which guarantees the existence of three fixed points:

Lemma 6.5 *The function $f(x, H, \theta, r) = \dfrac{\theta}{1 + e^{H(\theta - 2rx)/2\theta}}$ has three fixed points $0 < \phi_f^- < \phi_f^0 < \phi_f^+ < \theta$ if H is chosen such that*

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

where x satisfies the equation

$$r = \frac{\theta^2}{2x(\theta + (\theta - x)\ log(\frac{\theta - x}{x}))}$$

Proof: Fixed points of the function $f(x, H, \theta, r)$ satisfy the equation $\frac{\theta}{1 + e^{H(\theta - 2rx)/2\theta}} = x$. Given the parameter r, we must find a minimum value $H_0^-(r)$ such that $f(x, H, \theta, r)$ has three fixed points. We can think of x, r,

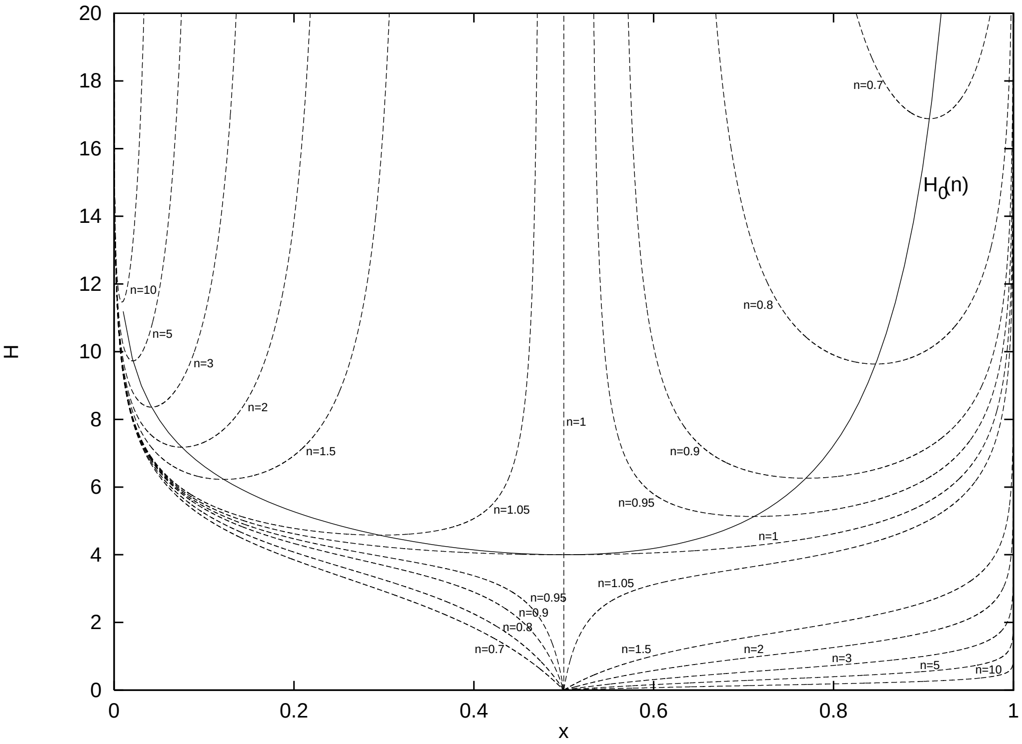


Figure 9: **Contour Plot of $f(x, H, \theta, r) = x$:** The contour plots (dotted graphs) show therelationship between H and x for various values of r and fixed value $\theta = 1$. If H is chosen such that $H > max(H_0^-(r), H_0^+(r))$ (solid graphs), then a line parallel to the x-axis intersects the surface satisfying $f(x, H, \theta, r) = x$ in three points which are the fixed points of $f(x, \theta, r)$.

and H as coordinates in a three-dimensional Euclidean space. Then the locus of points (x, r, H) satisfying relation the above equation 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, the fixed point equation cannot be solved explicitly for x as a function of r and H. However, it can be solved for either of the other parameters, giving the intersections with lines parallel to the r-axis or the H-axis:

$$r = r(x, \theta, H) = \frac{\theta}{2x} - \frac{\theta\ log(\frac{\theta - x}{x})}{Hx} \tag{9}$$

$$H = H(r, \theta, x) = \frac{2\theta\ log(\frac{\theta - x}{x})}{\theta - 2rx} \tag{10}$$

The contours of these functions show the relationship between H and x when r is fixed (Figure 11). 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 $f(x, H, \theta, r) = x$. Solving $\frac{\partial r(x,\theta,H)}{\partial x} = 0$, we obtain the conditions of the lemma.

$\square$

Even though the location of fixed points of the function f depends on H, r, and θ, we will use $[\phi_f]_\theta$ as a generic name for any fixed point of a function f.

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

Lemma 6.6 *The high signals are bounded from below by the fixed point* $[\phi_h^+]_\theta$ *of the function*

$$\begin{cases} h^0 = 1 \\ h^{t+1} = \tilde{g}(h^t - f^t) \end{cases} \tag{11}$$

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

Lemma 6.7 *Let* $[\phi_f]_\theta$ *denote the fixed point of the recursive function* f*, i.e.* $\lim_{t\to\infty} f^t = [\phi_f]_\theta$*. Then the recursively defined function* p

$$\begin{cases} p^0 = 1 \\ p^{t+1} = \tilde{g}(g^t - [\phi_f]_\theta) \end{cases} \tag{12}$$

has the property that $\forall t : p^t \leq h^t$.

Then, we have the following sufficient condition for the existence of two stable fixed point of the function defined in Equation (11):

Lemma 6.8 *Let the iterative function* p^t *have two stable fixed points and* $\forall t : p^t \leq h^t$*. Then the function* h^t *has also two stable fixed points.*

The above lemma has the following corollary:

Corollary 6.2 *A constructed network's high signals remain stable if the the sequence* $p^0, p^1, p^2, \ldots$ *converges towards the fixed point* $\theta/2 < [\phi_p^+]_\theta < \theta$.

Since we have decoupled the iterated function h^t from the iterated function f^t by introducing the iterated function p^t, we can apply the same technique to p^t for finding conditions for the existence of fixed points as in the case of f^t. In fact, the function that when iterated generates the sequence $p^0, p^1, p^2, \ldots$ is defined by

$$p(r, \theta, x) = \frac{\theta}{1 + e^{H(\theta - 2(x - [\phi_f^-]_\theta))/2\theta}} = \frac{\theta}{1 + e^{H'(\theta - 2r'x))/2\theta}} \tag{13}$$

with

$$H' = H(1 + 2[\phi_f^-]_\theta), \;\; r' = \frac{1}{1 + 2[\phi_f^-]_\theta} \tag{14}$$

We can iteratively compute the value of $[\phi_p]_\theta$ for given parameters H and r. Thus, we can repeat the original argument with H' and r' in place of H and r to find the conditions under which $p(r, x)$ and thus $g(r, x)$ have three fixed points.

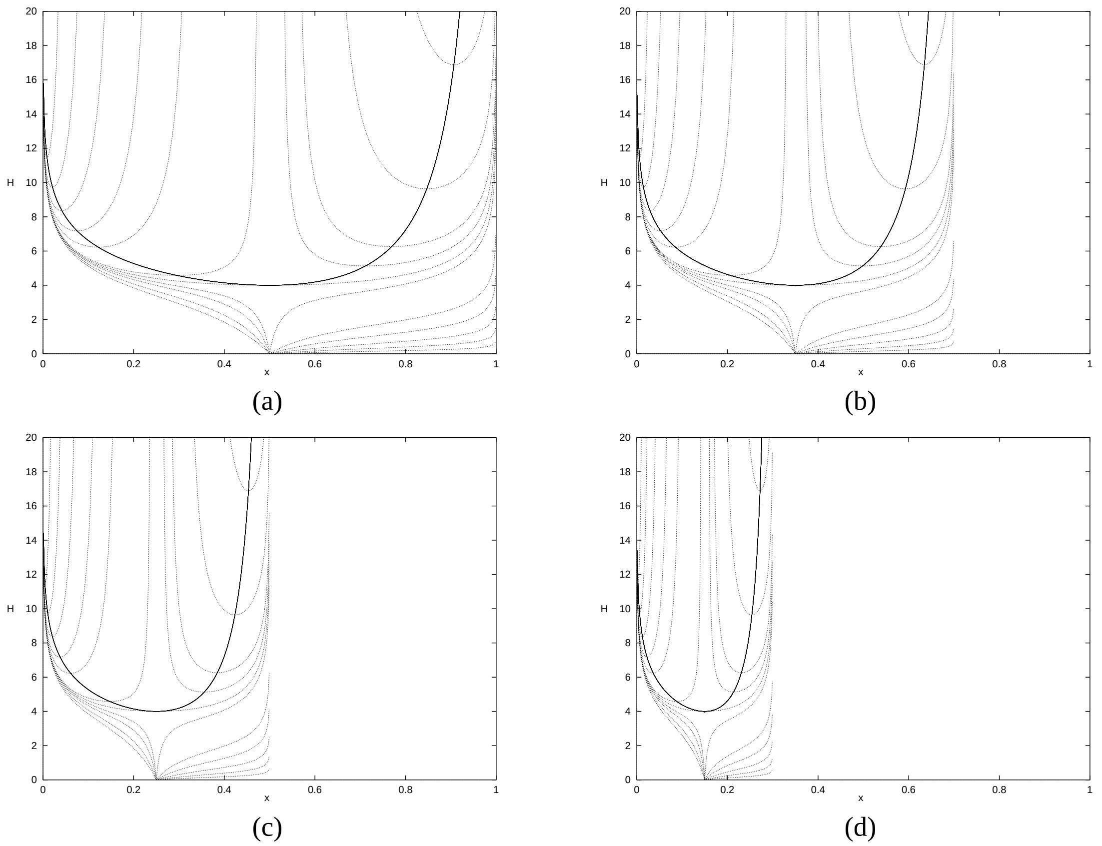


Figure 10: **Invariant Fixed Points:** The contour plots illustrating the existence and location of fixed points of the function $\tilde{g}(x, H, \theta, r) = \frac{\theta}{1+e^{H(\theta-2rx)/2\theta}}$ are shown for (a) $\theta = 1.0$, (b) $\theta = 0.7$, (c) $\theta = 0.5$, and (d) $\theta = 0.3$. The location of fixed points depends on the value of θ, but the condition on H and r for the existence of one vs. two stable fixed points is independent of θ. The scaling of the graphs illustrates that invariant property.

Lemma 6.9 *The function* $p(x, [\phi_f^-]_\theta) = \dfrac{1}{1+e^{H(\theta-2(x-[\phi_f^-]_\theta))/2\theta}}$ *has three fixed points* $0 < [\phi_p^-]_\theta < [\phi_p^0]_\theta < [\phi_p^+]_\theta < 1$ *if* H *is chosen such that*

$$H > H_0^+(r) = \frac{2(\theta + (\theta - x)\ log(\frac{\theta - x}{x}))}{(1 + 2[\phi_f^-]_\theta)(\theta - x)}$$

where x *satisfies the equation*

$$\frac{1}{1 + 2[\phi_f^-]_\theta} = \frac{\theta^2}{2x(\theta + (\theta - x)\ log(\frac{\theta - x}{x}))}$$

□

Since there is a collection of fuzzy transition memberships θ_{ijk} involved in the algorithm for constructing fuzzy representations of FFAs, we need to determine whether the condition of Lemmas 6.5 and 6.9 hold true for all rule weights θ_{ijk}. The following corollary establishes a useful invariant property of the function $H_0(x, r, \theta)$:

Corollary 6.3 *The value of the minima $H(x,r,\theta)$ only depends on the value of r and is independent of the particular values of θ:*

$$\inf H(x,r,\theta) = \inf \frac{2\,\theta\, log(\frac{\theta-x}{x})}{\theta - 2rx} = H_0(r) \tag{15}$$

Proof: The term $log(\frac{\theta-x}{x})$ scales the function $H(x,r,\theta)$ along the x-axis. We introduce a scaling factor ϵ and set $\theta' = \epsilon\,\theta$ and $x' = \epsilon\, x$. Then, Equation (10) becomes

$$H_\epsilon(x',r,\theta') = \frac{2\,\epsilon\,\theta\, log(\frac{\epsilon\,\theta-\epsilon\,x}{\epsilon\,x})}{\epsilon\,\theta - 2r\,\epsilon\,x} = \frac{2\,\epsilon\,\theta\, log(\frac{\theta-x}{x})}{\epsilon(\theta - 2rx)} = \frac{2\,\theta\, log(\frac{\theta-x}{x})}{\theta - 2rx} = H(x,r,\theta). \tag{16}$$

for fixed r. Thus the values of $H(x,r,\theta)$ are identical for fixed values of r, and their local minima have the same values independent of θ.

The relevance of the above corollary is that there is no need to test conditions for all possible values of θ in order to guarantee the existence of fixed points. The graphs in Figure 11 illustrate that invariant property of the sigmoidal discriminant function.

We can now proceed to prove stability of low and high signals, and thus stability of the fuzzy representation of FFA states, in a constructed recurrent neural network.

6.3 Network Stability

The existence of two stable fixed points of the discriminant function is only a necessary condition for network stability. We also need to establish conditions under which these fixed points are upper and lower bounds of stable low and high signals, respectively.

Before we define and derive the conditions for network stability, it is convenient to apply the result of Lemma 3.1 to the fixed points of the sigmoidal discriminant function (Section 3.3):

Corollary 6.4 *For any value θ with $0 < \theta \leq 1$, the fixed points $[\phi]_\theta$ of the discriminant function*

$$\frac{\theta}{1 + e^{H(\theta - 2rx)/2\theta}}$$

have the following invariant relationship:

$$[\phi]_\theta = \theta\ [\phi]_1$$

Proof: By definition, fixed points ϕ of $\tilde{g}(.)$ have the property that $[\phi]_\theta = \tilde{g}[(\phi)]_\theta$. According to Lemma 3.1, we also have

$$[\phi]_\theta = \tilde{g}([\phi]_\theta) = \tilde{g}([\phi]_\theta, H, \theta) = \theta\ \tilde{g}(\theta[\phi]_1, H, 1) = \theta\ \tilde{g}([\phi]_1) = \theta\ [\phi]_1$$

because the invariant scaling property applies to all points of the function $\tilde{g}$, including its fixed points. Thus, we do not have to consider the conditions separately for all values of θ that occur in a given FFA.

□

We now redefine stability of recurrent networks constructed from DFAs in terms of fixed points:

Definition 6.2 *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* $[\phi_f^0]_{\theta_i}$*, and all the high signals are greater than* $[\phi_h^0]_{\theta_i}$ *for all* θ_i *of all state neurons* S_i*.*

We have simplified $\theta_{i..}$ to θ_i because the output of each neuron S_i has a fixed upper limit θ for a given input symbol, regardless which neurons S_j contribute residual inputs. We note that this new definition is stricter than that we gave in Definition 6.1. In order for the low signal to remain stable, the following condition has to be satisfied:

$$-\frac{H}{2} + Hr[\phi_f^-]_{\theta_j} < [\phi_f^0]_{\theta_j} \tag{17}$$

Similarly, the following inequality must be satisfied for stable high signals:

$$-\frac{H}{2} + H[\phi_h^+]_{\theta_j} - H[\phi_f^-]_{\theta_i} > [\phi_h^0]_{\theta_i} \tag{18}$$

The above two inequalities must be satisfied for all neurons at all times. Instead of testing for all values θ_{ijk} separately, we can simplify the set of inequalities as follows:

Lemma 6.10 *Let* θ_{min} *and* θ_{max} *denote the minimum and maximum, respectively, of all fuzzy transition memberships* θ_{ijk} *of a given FFA* M*. Then, inequalities (17) and (18) are satisfied for all transition weights* θ_{ijk} *if the inequalities*

$$-\frac{H}{2} + Hr[\phi_f^-]_{\theta_{max}} < [\phi_f^0]_{\theta_{min}} \tag{19}$$

$$-\frac{H}{2} + H[\phi_h^+]_{\theta_{min}} - H[\phi_f^-]_{\theta_{max}} > [\phi_h^0]_{\theta_{max}} \tag{20}$$

are satisfied.

Proof: Consider the two fixed points $[\phi_f^-]_{\theta_{min}}$ and $[\phi_h^-]_{\theta_{max}}$. According to Corollary 6.4, we have

$$[\phi_f^-]_{\theta_{min}} = \theta_{min}[\phi_f^-]_1 < \theta_{ijk}[\phi_f^-]_1 < \theta_{max}[\phi_f^-]_1 = [\phi_f^-]_{\theta_{max}}$$

Thus, if inequalities (19) and (20) are not violated for $[\phi_f^-]_{\theta_{min}}$ and $[\phi_f^-]_{\theta_{max}}$, then they will not be violated for $\theta_{min} \leq \theta_{ijk} \leq \theta_{max}$ either.

□

We can rewrite inequalities (19) and (20) as

$$-\frac{H}{2} + Hr\,\theta_{max}[\phi_f^-]_1 < \theta_{min}[\phi_f^0]_1 \tag{21}$$

and

$$-\frac{H}{2} + H\theta_{min}[\phi_h^+]_1 - H\theta_{max}[\phi_f^-]_1 > \theta_{max}[\phi_h^0]_1 \tag{22}$$

Solving inequalities (21) and (22) for $[\phi_f^-]_1$ and $[\phi_h^+]_1$, respectively, we obtain conditions under which a constructed recurrent network implements a given FFA. These conditions are expressed in the following theorem:

Theorem 6.1 *For some given unambiguous FFA M with n states and m input symbols, let r denote the maximum number of transitions to any state over all input symbols of M. Furthermore, let θ_{min} and θ_{max} denote the minimum and maximum, respectively, of all transitions weights θ_{ijk} in M. Then, a sparse recurrent neural network with n state and m input neurons can be constructed from M such that the internal state representation remains stable if*

$$\begin{aligned}
&(1)\ [\phi_f^-]_1 < \frac{1}{r\,\theta_{max}}\left(\frac{1}{2} + \theta_{min}\frac{[\phi_f^0]_1}{H}\right)\\
&(2)\ [\phi_h^+]_1 > \frac{1}{\theta_{min}}\left(\frac{1}{2} + \theta_{max}[\phi_f^-]_1 + \frac{[\phi_h^0]_1}{H}\right)\\
&(3)\ H > max(H_0^-(r), H_0^+(r)) \ .
\end{aligned}$$

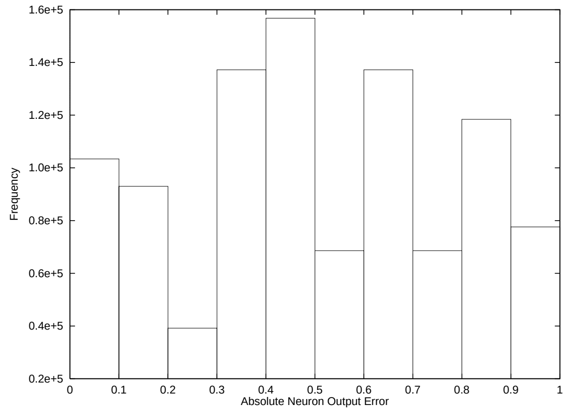
Furthermore, the constructed network has at most $3mn$ second-order weights with alphabet $\Sigma_w = \{-H, 0, +H\}$, $n+1$ biases with alphabet $\Sigma_b = \{-H/2\}$, and maximum fan-out $3m$.

□

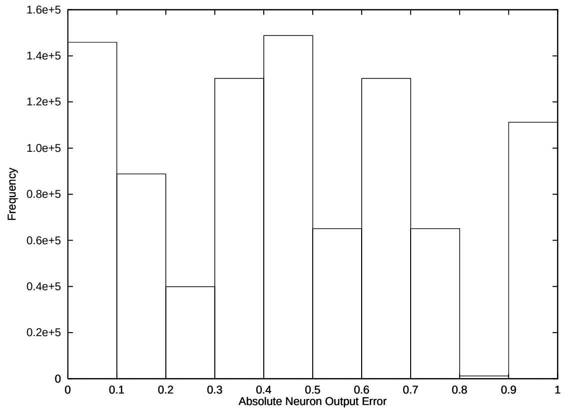
For $\theta_{min} = \theta_{max} = 1$, conditions (1)-(3) of the above theorem reduce to those found for stable DFA encodings [35]. This is consistent with a crisp representation of DFA states.

7 Simulations

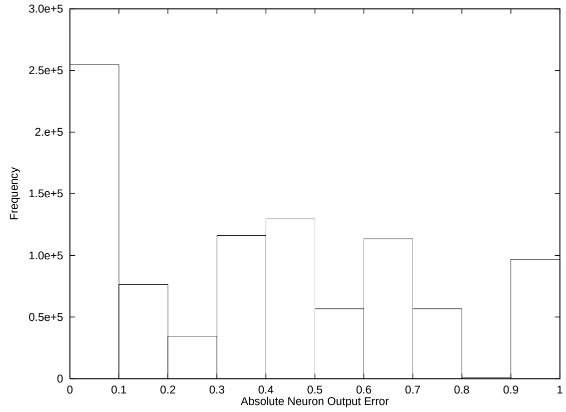
In order to validate our theory, we constructed a fuzzy encoding of a randomly generated FFA with 100 states (after the execution of the FFA transformation algorithm) over the input alphabet $\{0, 1\}$. We randomly assigned weights in the range $[0, 1]$ to all transitions in increments of 0.1. The maximum indegree was $D_{in}(M) = r = 5$. We then tested the stability of the fuzzy internal state representation on 100 randomly generated strings of length 100 by comparing, at each time step, the output signal of each recurrent state neuron with its ideal output signal (since each recurrent state neuron S_i corresponds to a FFA state q_i, we know the degree to which q_i is occupied after input symbol a_k has been read: either 0 or θ_{ijk}). A histogram of the differences between the ideal and the observed signal of state neurons for selected values of the weight strength H over all state neurons and all tested strings is shown in Figure 12. As expected, the error decreases for increasing values of H. We observe that the number of discrepancies between the desired



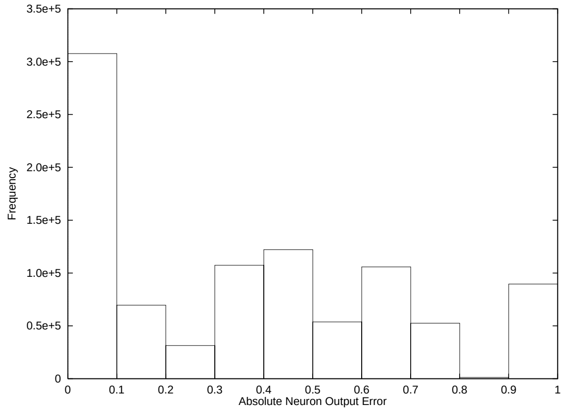
(a)



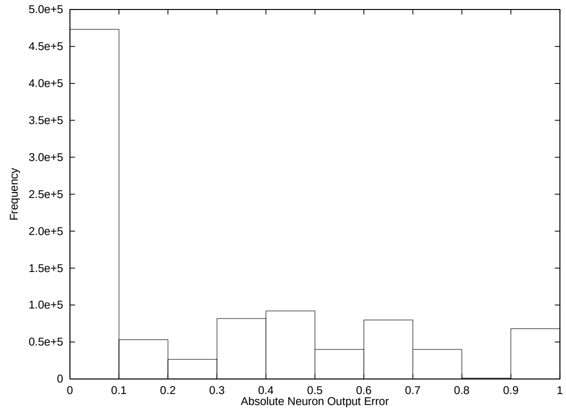
(b)



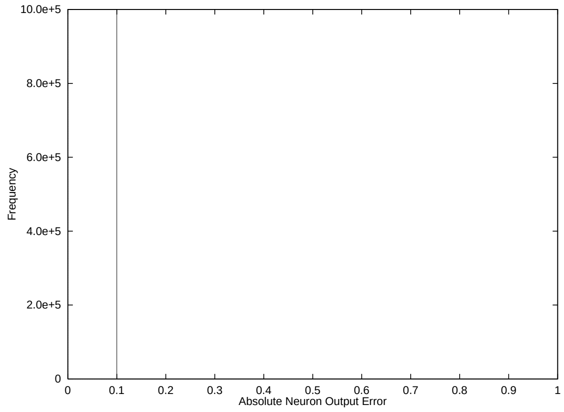
(c)



(d)



(e)



(f)

Figure 11: **Stability of FFA State Encoding:** The histogram shows the absolute neuron output error of a network with 100 neurons that implements a randomly generated FFA, and reads 100 randomly generated strings of length 100 for different values of the weight strength H. The error was computed by comparing, at each time step, the actual with the desired output of each state neuron. The distribution of neuron output signal errors are for weight strengths (a) $H = 6.0$, (b) $H = 9.0$, (c) $H = 9.60$, (d) $H = 9.65$, and (e) $H = 9.70$, and (f) $H = 9.75$.

and the actual neuron output decreases 'smoothly' for the shown values of H (almost no change can be observed for values up to $H = 6$). The most significant change can be observed by comparing the histograms for $H = 9.7$ and $H = 9.75$: The existence of significant neuron output errors for $H = 9.7$ suggests that the internal FFA representation is highly unstable. For $H \geq 9.75$, the internal FFA state representation becomes stable. This discontinuous change can be explained by observing that there exists a critical value $H_0(r)$ such that the number of stable fixed points also changes discontinuously from one to two for $H < H_0(r))$ and $H > H_0(r))$, respectively (see Figure 10). The 'smooth' transition from large output errors to very small errors for most recurrent state neurons (Figures 12a-e) can be explained by observing that not all recurrent state neurons receive the same number of residual inputs; some neurons may not receive any residual input for some given input symbol a_k at time step t; in that case, the low signals of those neurons are strengthened to $\tilde{g}(0, H, \theta_{i.k}) \simeq 0$ (note that strong low signals imply strong high signals by Lemma 6.7).

8 Conclusions

Theoretical work that proves representational relationships between different computational paradigms is important because it establishes the equivalences of those models. Previously it has been shown that it is possible to deterministically encode fuzzy finite-state automata (FFA) in recurrent neural networks by transforming any given FFA into a deterministic acceptor which assign string membership [39]. In such a deterministic encoding, only the network's classification of strings is fuzzy, whereas the representation of states is *crisp*. The correspondence between FFA and network parameters - i.e. fuzzy transition memberships and network weights, respectively - is lost in the transformation.

Here, we have demonstrated analytically and empirically that it is possible to encode FFAs in recurrent networks *without* transforming them into deterministic acceptors. The constructed network directly represents FFA states with the desired fuzziness. That representation requires (1) a slightly increased functionality of sigmoidal discriminant functions (it only requires the discriminants to accommodate variable output range), and (2) a transformation of a given FFA into an equivalent FFA with a larger number of states. (We have found empirically that the increase in automaton size is roughly proportional $N * K$ where N and K are the automaton and alphabet size, respectively.) In the proposed mapping FFA $\rightarrow$ recurrent network, the correspondence between FFA and network parameters remains intact; this can be significant if the physical properties of some unknown dynamic, nonlinear system are to be derived from a trained network modeling that system. Furthermore, the analysis tools and methods used to demonstrate the stability of the crisp internal representation of DFA's carried over and generalized to show stability of the internal FFA representation.

We speculate that other encoding methods are possible and that it is an open question as to which encoding method are better. One could argue that, from a engineering point of view, it may seem more natural to use radial basis functions to represent fuzzy state membership (they are often used along with triangular and trapezoidal membership functions in the design of fuzzy systems) instead of sigmoidal discriminant functions (DFAs can be mapped into recurrent neural networks with radialbasis functions [15]). It is an open question how mappings of FFA into recurrent neural networks with radial basis discriminant functions would be implemented and how such mappings would compare to the encoding algorithm described in this work.

The usefulness of training recurrent neural networks with fuzzy state representation from examples to behave like a FFA - the variable output range θ can be treated as a variable parameter and an update rule similar to that for networks weights can be derived - and whether useful information can be extracted from trained networks has yet to be determined, but based on past applications of fuzzy dynamical systems is promising. In particular, it would be interesting to compare training and knowledge representation of networks whose discriminant functions have fixed and variable output ranges, respectively. Discriminant functions with variable neuron output range may open the door to novel methods for the extraction of symbolic knowledge from recurrent neural networks.

Finally, proofs of equivalence mappings such as these raises questions about computational equivalence in other hybrid models and can open the door to more computationally efficient and powerful algorithms for other applications.

9 Acknowledgments

We would like to acknowledge useful discussions with D. Handscomb and B.G. Horne.

References

[1] H. Berenji and P. Khedkar, "Learning and fine tuning fuzzy logic controllers through reinforcement," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 724–740, 1992.

[2] J. Bezdek, "Fuzzy logic and neural networks," *IEEE Trans. on Neural Networks*, vol. 3, 1992. Special Issue.

[3] C. Bishop, *Neural Networks for Pattern Recognition.* New York, N.Y.: Oxford University Press, 1995.

[4] A. Blanco, M. Delgado, and M. Pegalajar, "Fuzzy grammar inference using neural networks," tech. rep., Department of Computer Science and Artificial Intelligence, University of Granada, Spain, 1997.

[5] P. Bonissone, V. Badami, K. Chiang, P. Khedkar, K. Marcelle, and M. Schutten, "Industrial applications of fuzzy logic at General Electric," *Proceedings of the IEEE*, vol. 83, no. 3, pp. 450–465, 1995.

[6] L. Bookman and R. Sun, "Architectures for integrating symbolic and neural processes," *Connection Science*, vol. 5, no. 3,4, 1993. Special Issue.

[7] 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, pp. 1135–1178, 1996.

[8] F. Cellier and Y. Pan, "Fuzzy adaptive recurrent counterpropagation neural networks: A tool for efficient implementation of qualitative models of dynamic processes," *J. Systems Engineering*, vol. 5, no. 4, pp. 207–222, 1995.

[9] S. Chiu, S. Chand, D. Moore, and A. Chaudhary, "Fuzzy logic for control of roll and moment for a flexible wing aircraft," *IEEE Control Systems Magazine*, vol. 11, no. 4, pp. 42–48, 1991.

[10] A. Cleeremans, D. Servan-Schreiber, and J. McClelland, "Finite state automata and simple recurrent recurrent networks," *Neural Computation*, vol. 1, no. 3, pp. 372–381, 1989.

[11] J. Corbin, "A fuzzy logic-based financial transaction system," *Embedded Systems Programming*, vol. 7, no. 12, p. 24, 1994.

[12] D. Dubois and H. Prade, *Fuzzy sets and systems: theory and applications*, vol. 144 of *Mathematics in Science and Engineering*, pp. 220–226. Academic Press, 1980.

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

[14] L. Franquelo and J. Chavez, "Fasy: A fuzzy-logic based tool for analog synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 15, no. 7, p. 705, 1996.

[15] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, no. 1, pp. 5–32, 1996.

[16] 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.

[17] C. Giles and C. Omlin, "Extraction, insertion and refinement of symbolic rules in dynamically driven recurrent neural networks," *Connection Science*, vol. 5, no. 3 & 4, pp. 307–337, 1993.

[18] C. Giles, R. Sun, and J. Zurada, "Neural networks and hybrid intelligent models: Foundations, theory, and applications," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, 1998. Special Issue.

[19] J. Grantner and M. Patyra, "Synthesis and analysis of fuzzy logic finite state machine models," in *Proc. of the Third IEEE Conf. on Fuzzy Systems*, vol. I, pp. 205–210, 1994.

[20] J. Grantner and M. Patyra, "VLSI implementations of fuzzy logic finite state machines," in *Proceedings of the Fifth IFSA Congress*, pp. 781–784, 1993.

[21] T. L. Hardy, "Multi-objective decision-making under uncertainty fuzzy logic methods," Tech. Rep. TM 106796, NASA, Washington, D.C., 1994.

[22] S. Haykin, *Neural Networks, A Comprehensive Foundation*. New York, NY: Prentice Hall, 1998.

[23] J. Hendler, "Developing hybrid symbolic/connectionist models," in *Advances in Connectionist and Neural Computation Theory* (J. Barnden and J. Pollack, eds.), Ablex Publishing, 1991.

[24] C. S. Herrmann, "A hybrid fuzzy-neural expert system for diagnosis," in *Proc. of the Fourteenth International Joint Conf. on Artificial Intelligence*, vol. I, pp. 494–502, 1995.

[25] V. Honavar and L. Uhr, eds., *Artificial Intelligence and Neural Networks: Steps toward Principled Integration*. San Diego, CA: Academic Press, 1994.

[26] E. Khan and F. Unal, "Recurrent fuzzy logic using neural networks," in *Advances in fuzzy logic, neural networks, and genetic algorithms* (T. Furuhashi, ed.), Lecture Notes in AI, Springer Verlag, 1995.

[27] W. J. M. Kickert and H. van Nauta Lemke, "Application of a fuzzy controller in a warm water plant," *Automatica*, vol. 12, no. 4, pp. 301–308, 1976.

[28] E. Kosmatopoulos and M. Christodoulou, "Neural networks for identification of fuzzy dynamical systems: an application to identification of vehicle highway systems," tech. rep., Dept. of Electronic and Computer Engineering, Technical U. of Crete, 1995.

[29] E. Kosmatopoulos and M. Christodoulou, "Structural properties of gradient recurrent high-order neural networks," *IEEE Transactions on Circuits and Systems*, 1995.

[30] E. Kosmatopoulos, M. Polycarpou, M. Christodoulou, and P. Ioannou, "High-order neural networks for identification of dynamical systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 422–431, 1995.

[31] C. Lee, "Fuzzy logic in control systems: fuzzy logic controller," *IEEE Transactions on Man, Systems, and Cybernetics*, vol. SMC-20, no. 2, pp. 404–435, 1990.

[32] 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.

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

[34] S. Mensch and H. Lipp, "Fuzzy specification of finite state machines," in *Proceedings of the European Design Automation Conference*, pp. 622–626, 1990.

[35] C. Omlin and C. Giles, "Constructing deterministic finite-state automata in recurrent neural networks," *Journal of the ACM*, vol. 43, no. 6, pp. 937–972, 1996.

[36] C. Omlin and C. Giles, "Extraction of rules from discrete-time recurrent neural networks," *Neural Networks*, vol. 9, no. 1, pp. 41–52, 1996.

[37] 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.

[38] C. Omlin and C. Giles, "Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants," *Neural Computation*, vol. 8, no. 7, pp. 675–696, 1996.

[39] C. Omlin, K. Thornber, and C. Giles, "Fuzzy finite-state automata can be deterministically encoded into recurrent neural networks," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 1, pp. 76–89, 1998.

[40] M. Palaniswami, Y. Attikiouzel, and a. T. F. R.J. Marks, D. Fogel, eds., *Computational Intelligence: A Dynamic System Perspective.* Piscataway, NJ: IEEE Press, 1995.

[41] C. Pappis and E. Mamdani, "A fuzzy logic controller for a traffic junction," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-7, no. 10, pp. 707–717, 1977.

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

[43] E. Santos, "Maximin automata," *Information and Control*, vol. 13, pp. 363–377, 1968.

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

[45] H. Siegelmann and E. Sontag, "On the computational power of neural nets," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[46] R. Sun, "Learning, action, and consciousness: a hybrid approach towards modeling consciousness," *Neural Networks*, vol. 10, no. 7, pp. 1317–1332, 1997.

[47] F. Unal and E. Khan, "A fuzzy finite state machine implementation based on a neural fuzzy system," in *Proceedings of the Third International Conference on Fuzzy Systems*, vol. 3, pp. 1749–1754, 1994.

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

[49] X. Yang and G. Kalambur, "Design for machining using expert system and fuzzy logic approach," *Journal of Materials Engineering and Performance*, vol. 4, no. 5, p. 599, 1995.

[50] L. Zadeh, "Fuzzy languages and their relation to human and machine intelligence," Tech. Rep. ERL-M302, Electronics Research Laboratory, University of California, Berkeley, 1971.

[51] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[52] 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.