Fuzzy Finite-state Automata Can Be Deterministically Encoded into Recurrent Neural Networks

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

[a] NEC Research Institute, Princeton, NJ 08540

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

Technical Report CS-TR-3599 and UMIACS-96-12

Abstract

There has been an increased interest in combining fuzzy systems with neural networks because fuzzy neural systems merge the advantages of both paradigms. On the one hand, parameters in fuzzy systems have clear physical meanings and rule-based and linguistic information can be incorporated into adaptive fuzzy systems in a systematic way. On the other hand, there exist powerful algorithms for training various neural network models. However, most of the proposed combined architectures are only able to process static input-output relationships, i.e. they are not able to process temporal input sequences of arbitrary length. Fuzzy finite-state automata (FFAs) can model dynamical processes whose current state depends on the current input and previous states. Unlike in the case of deterministic finite-state automata (DFAs), FFAs are not in one particular state, rather each state is occupied to some degree defined by a membership function. Based on previous work on encoding DFAs in discrete-time, second-order recurrent neural networks, we propose an algorithm that constructs an augmented recurrent neural network that encodes a FFA and recognizes a given fuzzy regular language with arbitrary accuracy. We then empirically verify the encoding methodology by measuring string recognition performance of recurrent neural networks which encode large randomly generated FFAs. In particular, we examine how the networks' performance varies as a function of synaptic weight strength.

1 Introduction

1.1 Fuzzy Systems and Neural Networks

There has been an increased interest in combining artificial neural networks and fuzzy systems (see [4] for a collection of papers). Fuzzy logic [55] provides a mathematical foundation for approximate reasoning; fuzzy logic controllers have proven very successful in a variety of applications [6, 23, 24, 34]. 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.

Artificial neural networks emulate on a small scale the information processing mechanisms found in biological systems which are based on the cooperation of neurons which perform simple operations and on their ability to learn from examples. Artificial neural networks have become valuable computational tools in their own right for tasks such as pattern recognition, control, and forecasting.

Fuzzy systems and multilayer perceptrons are computationally equivalent, i.e. they are both universal approximators [8, 50]. Recurrent neural networks have been shown to be computationally equivalent with Turing machines [43]; whether or not recurrent fuzzy systems are also Turing equivalent remains an open question. While the methodologies underlying fuzzy systems and neural networks are quite different, their functional forms are often similar. The development of powerful learning algorithms for neural networks has been beneficial to the field of fuzzy systems which adopted some learning algorithms; e.g. there exists a backpropagation training algorithms for fuzzy logic systems which are similar to the training algorithms for neural networks [17, 51].

1.2 Fuzzy Knowledge Representation in Neural Networks

In some cases, neural networks can be structured based on the principles of fuzzy logic [16, 36]. Neural network representations of fuzzy logic interpolation have also been used within the context of reinforcement learning [3].

A typical fuzzy neural network used for intelligent control is shown in figure 1. Typically, such networks are initialized with linguistic rules of the form

$$\text{IF } x_1 \text{ is } A_1 \text{ AND } x_2 \text{ is } A_3 \text{ THEN } y_1 \text{ is } C_1$$

where A_1, A_3 and C_1 are fuzzy sets and x_1, x_2, and y_1 are linguistic input and output variables, respectively. The network has an input layer consisting of real-valued input variables (e.g. linguistic variables), a fuzzification layer which maps input values x_i to fuzzy sets A_i, an interpolation layer which computes the conjunction of all antecedent conditions in a rule (e.g. differential softmin operation), a defuzzification layer which computes the output for a given rule (e.g. mean of maximum method), and an output layer which combines the recommendations from all fuzzy control rules in the rule base (e.g. weighted sum). Thus, fuzzy

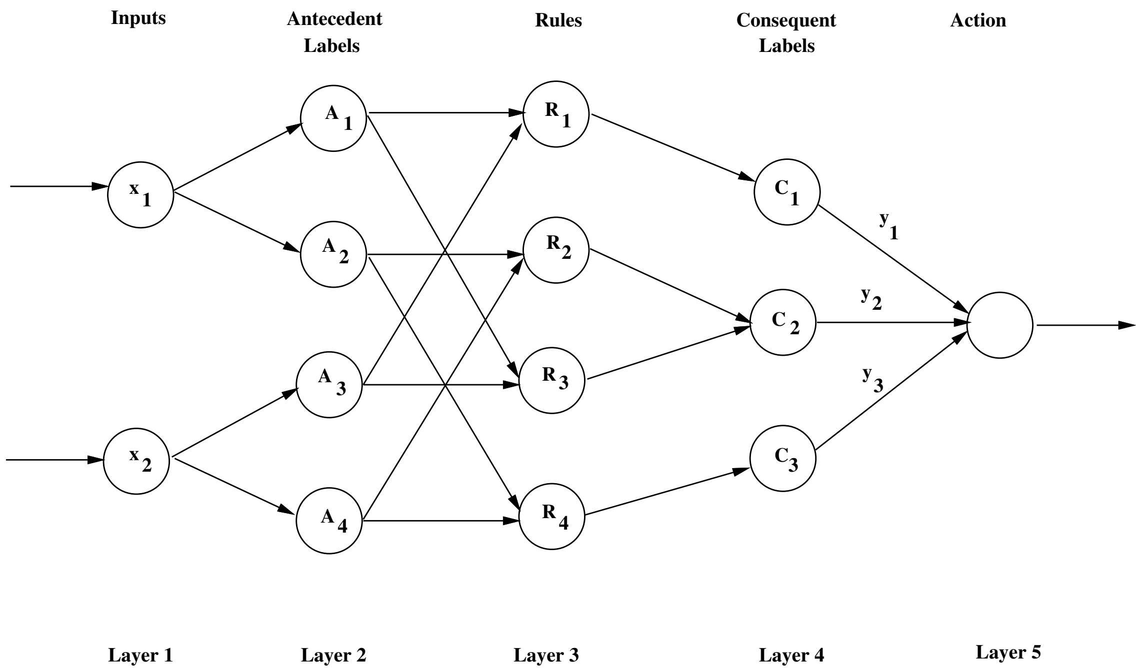


Figure 1: **Fuzzy Feedforward Network**: A feedforward network used in intelligent control is initialized with rules of the form IF x_1 is A_1 AND x_2 is A_3 THEN y_1 is C_1. The operations performed by the different layers are: layer 1: real-valued input variables x_1 and x_2, layer 2: mapping input variables into fuzzy sets (fuzzification), layer 3: compute conjunction of all antecedent conditions in a rule, layer 4: compute values for consequent labels (defuzzification), and label 5: combine recommendations from all fuzzy control rules in the rule base

neural networks play the role of *fuzzy logic interpolation engines.* [1] The rules are then fine tuned using a standard training algorithm for multilayer perceptrons. The extraction of fuzzy if-then-rules from trained multilayer perceptrons has also been investigated [16, 20, 29, 30].

There exist applications where the variables of linguistic rules are recursive, i.e. the rules are of the form

$$\text{IF } x(t-1) \text{ is } \alpha \text{ AND } u(t-1) \text{ is } \beta \text{ THEN } x(t) \text{ is } \gamma$$

where $u(t-1)$ and $x(t-1)$ represent input and state variables, respectively. The value of the state variable $x(t)$ depends on both the input $u(t-1)$ and the previous state $x(t-1)$. Clearly, feedforward neural networks do not have the computational capabilities to represent such recursive rules when the depth of the recursion is not known a priori. Recurrent neural networks have the ability to store information over indefinite periods of time and are thus potentially useful for representing recursive linguistic rules.

[1] The term *fuzzy inference* is also often used to describe the function of a fuzzy neural network. We choose the term *fuzzy logic interpolation* in order to distinguish between the function of fuzzy neural networks and fuzzy logic inference where the objective is to obtain some properties of fuzzy sets $B_1, B_2, \ldots$ from properties of fuzzy sets $A_1, A_2, \ldots$ with the help of an inference scheme $A_1, A_2, \ldots \rightarrow B_1, B_2, \ldots$ which is governed by a set of rules [45, 46].

A large class of problems where the current state depends on both the current input and the previous state can be modeled by finite-state automata or their equivalent grammars. It has been shown that recurrent neural networks can represent finite-state automata [5, 7, 9, 11, 12, 15, 33, 37, 47, 48, 53, 56]. Thus, it is only natural to ask whether recurrent neural networks can also represent *fuzzy* finite-state automata (FFAs) and thus be used to implement recognizers of fuzzy regular grammars.

Fuzzy grammars have been found to be useful in a variety of applications such as in the analysis of X-rays [35], in digital circuit design [27], and in the design of intelligent human-computer interfaces [41]. The fundamentals of FFAs have been in discussed in [14, 40, 54] without presenting a systematic method for machine synthesis. Neural network implementations of fuzzy automata have been proposed in the literature [18, 19, 25, 49]. The synthesis method proposed in [18] uses digital design technology to implement fuzzy representations of states and outputs. In [49], the implementation of a Moore machine with fuzzy inputs and states is realized by training a feedforward network explicitly on the state transition table using a modified backpropagation algorithm. The fuzzification of inputs and states reduces the memory size that is required to implement the automaton in a microcontroller, e.g. antilock braking systems. In related work, an algorithm for implementing weighted regular languages in neural networks with probabilistic logic nodes was discussed in [26]. A general synthesis method for synchronous fuzzy sequential circuits has been discussed in [52]. A synthesis method for a class of discrete-time neural networks with multilevel threshold neurons with applications to gray level image processing has been proposed in [42].

1.3 Outline of Paper

The purpose of this paper is to show that recurrent networks that can represent DFAs can be easily modified to accommodate FFAs. We briefly review deterministic, finite-state automata and their implementation in recurrent neural networks in section 2. The extension of DFAs to FFAs is discussed in section 3. In section 4, we show how FFAs can be implemented in recurrent networks based on previous work on the encoding of DFAs [32, 31, 33]. In particular, our results show that FFAs can be encoded into recurrent networks such that a constructed network assigns membership grades to strings of arbitrary length with arbitrary accuracy. Notice that we do not claim that such a representation can be *learned*. Simulation results in section 5 validate our theoretical analysis. A summary and directions for future work in section 6 conclude this paper.

2 Finite-state Automata and Recurrent Neural Networks

Here we discuss the relationship between finite-state automata and recurrent neural networks necessary to mapping fuzzy automata into recurrent networks. Most of this can be found in detail in [31] and briefly with experimental verification in [33].

2.1 Deterministic Finite-state Automata

Regular languages represent the smallest class of formal languages in the Chomsky hierarchy [21]. Regular languages are generated by regular grammars.

Definition 2.1 *A regular grammar G is a quadruple $G =< S, N, T, P >$ where S is the start symbol, N and T are non-terminal and terminal symbols, respectively, and P are productions of the form $A \rightarrow a$ or $A \rightarrow aB$ where $A, B \,\epsilon\, N$ and $a \,\epsilon\, T$.*

The regular language generated by G is denoted $L(G)$.

Associated with each regular language L is a deterministic finite-state automaton (DFA) M which is an acceptor for the language $L(G)$, i.e. $L(G) = L(M)$. DFA M accepts only strings which are a member of the regular language $L(G)$.

Definition 2.2 *A DFA M is a 5-tuple $M =< \Sigma, Q, R, F, \delta >$ where $\Sigma = \{a_1, \ldots, a_m\}$ is the alphabet of the language L, $Q = \{q_1, \ldots, q_n\}$ is a set of states, $R \,\epsilon\, Q$ is the start state, $F \subseteq Q$ is a set of accepting states and $\delta : Q \times \Sigma \rightarrow Q$ defines state transitions in M.*

A string x is accepted by the DFA M and hence is a member of the regular language $L(M)$ if an accepting state is reached after the string x has been read by M. Alternatively, a DFA M can also be considered a generator which generates the regular language $L(M)$.

2.2 Network Construction

Various methods have been proposed for implementing DFAs in recurrent neural networks [1, 2, 12, 13, 22, 28, 32]. We use discrete-time, second-order recurrent neural networks with sigmoidal discriminant functions which update their current state according to the following equations:

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

where b_i is the bias associated with hidden recurrent state neurons S_i; I_k denotes the input neuron for symbol a_k. The product $S_j^{(t)} I_k^{(t)}$ directly corresponds to the state transition $\delta(q_j, a_k) = q_i$.

We have recently proven that 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 [31]. The desired finite-state dynamics are encoded into a network by programming a small subset of all available weights to values $+H$ and $-H$ leading to a nearly orthonormal internal DFA state representation where only one state neuron that corresponds to the current DFA state has a output signal ≈ 1; all other state neurons have output signal ≈ 0. Similarly, the weights of a network's output neuron

S_0 are programmed to $+H$ or $-H$ for correct string classification. The network construction algorithm depends on this nearly orthonormal internal DFA state representation for programming DFA state transitions. Instability of the internal representation leads to misclassification of strings.

The encoding algorithm leads to the following special form of the equation governing the network dynamics:

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

where x is the input to neuron S_i.

2.3 Network Stability

There exist only two kinds of signals in a constructed neural network that models a DFA: Recurrent state neurons have high output signals only when they correspond to the current DFA state; all other recurrent neurons have low output signals. The stability of the internal DFA representation depend on the value of the weight strength H used to program the state transitions. If H is chosen too small, then the internal DFA representation becomes unstable, i.e. state neurons S_i which do not correspond to the current DFA state q_i no longer have output signals ≈ 0. Since our goal is to have a constructed neural network exactly model the state transitions of some DFA, the problem is to find a value H_0 such that for $H > H_0$, the internal DFA state representation remains stable for strings of *arbitrary* length. We achieve this goal by proving that there exist appropriate upper and lower bounds for low and high signals, respectively which, if sufficiently tight, guarantee the stability of low and high signals.

In the remainder of this section, we state results which establish that stability of the internal representation can be achieved. The proofs of these results can be found in [31].

The terms *principal* and *residual inputs* will be useful for the following discussion:

Definition 2.3 *Let S_i be a neuron with low output signal S_i^t and S_j be a neuron with high output signal S_j^t. Furthermore, let $\{S_l\}$ and $\{S_{l'}\}$ be sets of neurons with output signals $\{S_l^t\}$ and $\{S_{l'}^t\}$, respectively, for which $W_{ilk} \neq 0$ and $W_{il'k} \neq 0$ for some input symbol a_k and assume $S_j \in \{S_l\}$. Then, neurons S_i and S_j receive <u>principal inputs</u> of opposite signs from neuron S_j and <u>residual inputs</u> from all other neurons S_l and $S_{l'}$, respectively, when the network executes the state transition $\delta(q_j, a_k) = q_i$.*

Low signals are particularly sensitive to becoming corrupted because even though a neuron S_l may not correspond to a DFA state q_i when a network executes a DFA state transition $\delta(q_j, a_k) = q_i$, it may still receive residual inputs from other neurons if state transitions $\delta(q_x, a_k) = q_l$ exist. Over several time steps, neuron S_l then computes an *iteration* of residual inputs which can ultimately lead to the situation where the low signal S_l^t converges toward a high output. Similarly, high signals can become corrupted.

The following lemma establishes an upper bound for low signals in a constructed network:

Lemma 2.1 *The low signals are bounded from above by the fixed point ϕ_f of the function f*

$$\begin{cases} f^0 = 0 \\ f^{t+1} = h(r \cdot f^t) \end{cases} \tag{3}$$

This lemma can be proven by induction on t. It is assumed that each neuron receives residual inputs from no more than r other neurons. Such an upper bound r obviously exists for any constructed network.

It is easy to see that the function to be iterated in equation (3) is $f(x,r) = \frac{1}{1+e^{(H/2)(1-2rx)}}$. The graphs of the function $f(x,r)$ are shown in figure 2 for different values of the parameter r.

The function $f(x,r)$ has some desirable properties [31]:

Lemma 2.2 *For any $H > 0$, the function $f(x,r)$ has at least one fixed point ϕ_f^0.*

If $f(x,r)$ has only one fixed point for chosen r and H, then the accumulation of residual inputs in neurons with initial output signals $S_l^0 = 0$ causes the iteration f^t to converge toward a value ≈ 1. This can cause instability of the internal DFA state representation since this convergence happens simultaneously for all recurrent state neurons.

The following lemma states that more desirable fixed points may exist:

Lemma 2.3 *There exists a value $H_0^-(r)$ such that for any $H > H_0^-(r)$, $f(x,r)$ has three fixed points $0 < \phi_f^- < \phi_f^0 < \phi_f^+ < 1$.*

The following lemma shows the convergence property of iterations of the function $f(x,r)$:

Lemma 2.4 *If $f(x,r)$ has three fixed points ϕ_f^-, ϕ_f^0, and ϕ_f^+, then*

$$\lim_{t\to\infty} f^t = \begin{cases} \phi_f^- & x_0 < \phi_f^0 \\ \phi_f^0 & x_0 = \phi_f^0 \\ \phi_f^+ & x_0 > \phi_f^0 \end{cases} \tag{4}$$

The above lemma can be proven by defining an appropriate Lyapunov function L and showing that L has minima at ϕ_f^- and ϕ_f^+ and that f^t converges toward one of these minima. Notice that the fixed point ϕ_f^0 is unstable.

Iteration towards a fixed point is only a necessary condition for the stability of low signals. In addition, convergence must occur monotonically:

Lemma 2.5 *Let $f^0, f^1, f^2, \ldots$ denote the finite sequence computed by successive iteration of the function f. Then we have $f^0 < f^1 < \ldots < \phi_f^-$.*

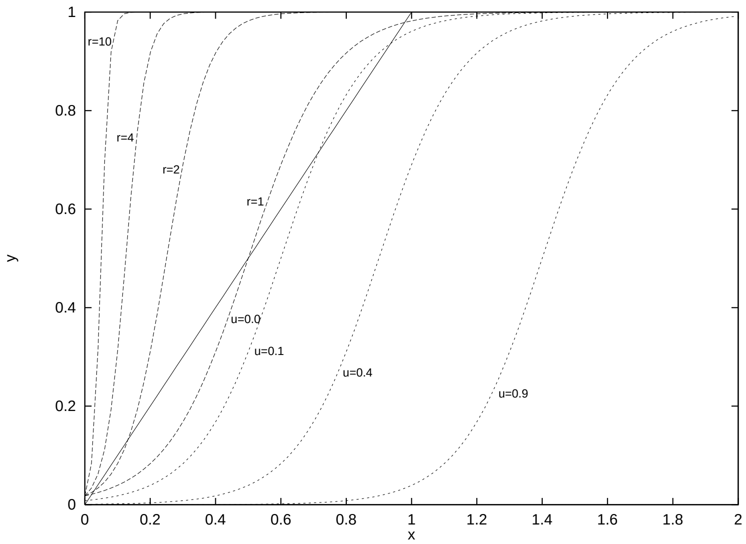


Figure 2: **Fixed Points of the Sigmoidal Discriminant Function**: Shown are the graphs of the function $f(x,r) = \frac{1}{1+e^{H(1-2rx)/2}}$ (dashed graphs) for $H = 8$ and $r = \{1,2,4,10\}$ and the function $p(x,u) = \frac{1}{1+e^{H(1-2(x-u))/2}}$ (dotted graphs) for $H = 8$ and $u = \{0.0, 0.1, 0.4, 0.9\}$. 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 $u = \{0.6, 0.9\}$.

With these properties, we can quantify the value $H_0^-(r)$ such that for any $H > H_0^-(r)$, $f(x,r)$ has three fixed points. 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 $0.5 < \phi_f < 1$. The following lemma expresses a quantitative condition which guarantees the existence of three fixed points:

Lemma 2.6 *The function* $f(x,r) = \dfrac{1}{1+e^{(H/2)(1-2rx)}}$ *has three fixed points* $0 < \phi_f^- < \phi_f^0 < \phi_f^+ < 1$ *if* H *is chosen such that*

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

where x *satisfies the equation*

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

The contour plots in figure 3 show the relationship between H and x for various values of r. If H is chosen such that $H > H_0(r)$, then three fixed points exist; otherwise, only a single fixed point exists. The number and the location of fixed points depends on the values of r and H. Thus, we write $\phi_f^-(r,H)$, $\phi_f^0(r,H)$, and $\phi_f^+(r,H)$, to denote the stable low, the unstable, and the stable high fixed point, respectively. We will use ϕ_f as a generic name for any fixed point of a function f.

Similarly, we can obtain a lower bound for high signals in a constructed network:

Lemma 2.7 *Let* ϕ_f *denote the fixed point of the recursive function* f, *i.e.* $\lim_{t\to\infty} f^t = \phi_f$. *Then the high signals in a constructed recurrent neural network are bounded from below by the fixed point* ϕ_g^+ *of the recursively*

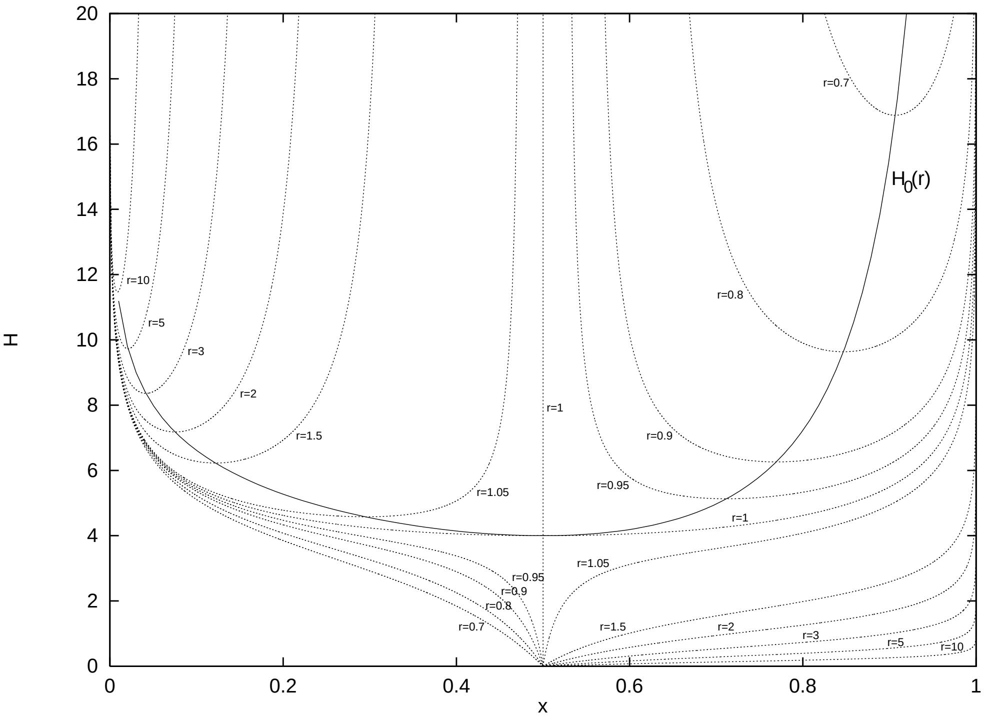


Figure 3: **Existence of Fixed Points**: The contour plots of the function $h(x,r) = x$ (dotted graphs) show the relationship between H and x for various values of r. If H is chosen such that $H > H_0(r)$ (solid graph), then a line parallel to the x-axis intersects the surface satisfying $h(x,r) = x$ in three points which are the fixed points of $h(x,r)$.

defined function g

$$\begin{cases} g^0 = 1 \\ g^{t+1} = h(g^t - \phi_f^-) \end{cases} \tag{5}$$

This can be proven by induction on t. Notice that we assume that the iteration f^t converges toward ϕ_f^-. The graphs of the function $g(x,u) = g(x,\phi_f^-)$ for some values of u are shown in figure 2.

We can apply the same technique for finding conditions for the existence of fixed points of $g(x,u)$ as in the case of $f(x,r)$. In fact, the function that when iterated generates the sequence $g^0, g^1, g^2, \ldots$ is defined by

$$g(x,u) = g(x,\phi_f) = \frac{1}{1+e^{(H/2)(1-2(x-\phi_f^-))}} = \frac{1}{1+e^{(H'/2)(1-2r'x))}} \tag{6}$$

with

$$H' = H(1+2\phi_f), \ \ r' = \frac{1}{1+2\phi_f} \tag{7}$$

Since we can iteratively compute the value of ϕ_f for given parameters H and r, we can repeat the original argument with H' and r' in place of H and r to find the conditions under which $g(r,x)$ has three fixed points. This results in the following lemma:

Lemma 2.8 *The function* $g(x,\phi_f^-) = \dfrac{1}{1+e^{(H/2)(1-2(x-\phi_f^-))}}$ *has three fixed points* $0 < \phi_g^- < \phi_g^0 < \phi_g^+ < 1$ *if* H *is chosen such that*

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

where x satisfies the equation

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

We now define stability of recurrent networks constructed from DFAs:

Definition 2.4 *An encoding of DFA states in a second-order recurrent neural network is called stable if all the low signals are less than $\phi_f^0(r, H)$, and all the high signals are greater than $\phi_g^0(r, H)$.*

The following result states conditions under which a constructed recurrent network implements a given DFA can be obtained by assuming a worst case where all neurons are assumed to contribute to the instability of low and high signals:

Theorem 2.1 *For some given DFA 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. Then, a sparse recurrent neural network with $n+1$ sigmoidal state neurons and m input neurons can be constructed from M such that the internal state representation remains stable if the following three conditions are satisfied:*

$$(1) \quad \phi_f^-(r, H) < \frac{1}{r}(\frac{1}{2} + \frac{\phi_f^0(r, H)}{H})$$

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

$$(3) \quad H > max(H_0^-(r), H_0^+(r))$$

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

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

The above conditions implicitly put lower bounds on the magnitude of H which guarantee stable finite-state dynamics for a network of given size. As such, they represent worst cases, i.e. the finite-state dynamics of a given neural network implementation may remain stable for smaller values of H even for very large networks [33].

Since deterministic and fuzzy finite-state automata share a common underlying structure expressed in terms of state transitions, we will be able to use the result on the stability of the network dynamics for DFAs to implement fuzzy finite-state automata.

3 Fuzzy Finite-state Automata

Here we formally define fuzzy finite-state automata (FFA) and give a simple example.

3.1 Definitions and Properties

We begin by defining the class of fuzzy automata for which we develop a synthesis method for recurrent neural networks:

Definition 3.1 *A fuzzy regular grammar $\widetilde{G}$ is a quadruple $\widetilde{G} =< S, N, T, P >$ where S is the start symbol, N and T are non-terminal and terminal symbols, respectively, and P are productions of the form $A \xrightarrow{\theta} a$ or $A \xrightarrow{\theta} aB$ where $A, B \,\epsilon\, N$, $a \,\epsilon\, T$ and $0 \leq \theta \leq 1$.*

Unlike in the case of DFAs where strings either belong or do not belong to some regular language, strings of a fuzzy language have graded membership:

Definition 3.2 *Given a regular fuzzy grammar $\widetilde{G}$, the membership grade $\mu_G(x)$ of a string $x \in T$ in the regular language $L(\widetilde{G})$ is the maximum value of any derivation of x, where the value of a specific derivation of x is equal to the minimum weight of the productions used:*

$$\mu_G(x) = \mu_G(S \stackrel{*}{\Rightarrow} x) = \max_{S \stackrel{*}{\Rightarrow} x} \; \min[\mu_G(S \rightarrow \alpha_1), \mu_G(\alpha_1 \rightarrow \alpha_2), \ldots, \mu_G(\alpha_m \rightarrow x)]$$

This is akin to the definition of stochastic regular languages [38] where the min-and max-operators are replaced by the product- and sum-operators, respectively. Both fuzzy and stochastic regular languages are examples of weighted regular languages [39].

Definition 3.3 *A fuzzy finite-state automaton (FFA) $\widetilde{M}$ is a 6-tuple $\widetilde{M} =< \Sigma, Q, Z, \widetilde{R}, \delta, \omega >$ where Σ, Q, and q_0 are the same as in DFAs; Z is a finite output alphabet, $\widetilde{R}$ is the fuzzy initial state, $\delta : \Sigma \times Q \times [0,1] \rightarrow Q$ is the fuzzy transition map and $\omega : Q \rightarrow Z$ is the output map.*

In this paper, we consider a restricted type of fuzzy automaton whose initial state is not fuzzy, and ω is a function from F to Z, where F is a non fuzzy subset of states, called *final states*. Any fuzzy automaton as described in definition 3.3 is equivalent to a restricted fuzzy automaton [10]. Notice that a FFA reduces to a conventional DFA by restricting the transition weights to 1.

As in the case of DFAs and regular grammars, there exist a correspondence between FFAs and fuzzy regular grammars [10]:

Theorem 3.1 *For a given fuzzy grammar $\widetilde{G}$, there exists a fuzzy automaton $\widetilde{M}$ such that $L(\widetilde{G}) = L(\widetilde{M})$.*

Our goal is to use only continuous (sigmoidal and linear) discriminant functions for the neural network implementation of FFAs. The following results greatly simplifies the encoding of FFAs in recurrent networks with continuous discriminant functions:

Theorem 3.2 *Given a regular fuzzy automaton $\widetilde{M}$, there exists a deterministic finite-state automaton M with output alphabet $Z \subseteq \{\theta : \theta \text{ is a production weight}\} \cup \{0\}$ which computes the membership function $\mu : \Sigma^* \rightarrow [0,1]$ of the language $L(\widetilde{M})$.*

The constructive proof can be found in [44]. An immediate consequence of this theorem is the following corollary:

Corollary 3.1 *Given a regular fuzzy grammar $\widetilde{G}$, there exist an equivalent unambiguous grammar G in which productions have the form $A \stackrel{1.0}{\rightarrow} aB$ or $A \stackrel{\theta}{\rightarrow} a$.*

3.2 Example

Consider a fuzzy regular grammar with non-terminal symbols $N = \{A, B\}$, terminal symbols $T = \{0, 1\}$ and the following production rules:

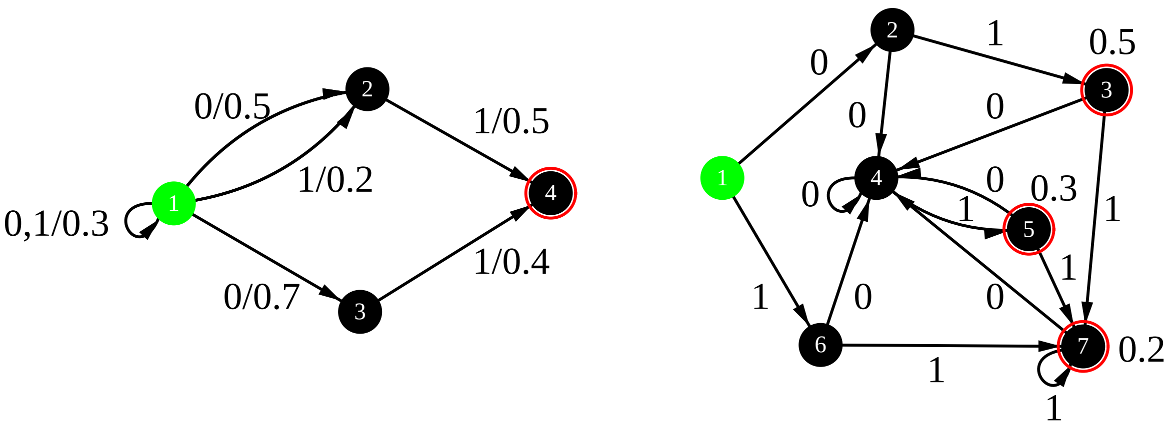
$$S \stackrel{0.3}{\rightarrow} 0S \quad S \stackrel{0.5}{\rightarrow} 0A \quad S \stackrel{0.7}{\rightarrow} 0B \quad S \stackrel{0.3}{\rightarrow} 1S \quad S \stackrel{0.2}{\rightarrow} 1A \quad A \stackrel{0.5}{\rightarrow} 1 \quad B \stackrel{0.4}{\rightarrow} 1$$

The FFA which accepts the strings generated by the above grammar is shown in figure 2a. Only transitions which correspond to the production rules are shown; implicitly, all other transitions leads to a rejecting garbage state. The deterministic acceptor of the FFA which computes the same string membership is shown in figure 2b.

4 Recurrent Neural Network Architecture for Fuzzy Automata

We describe a method for encoding any fuzzy finite-state automata into a recurrent neural network. The result of theorem 2.1 concerning the stability of the programmed network dynamics applies to finite-state automata whose states are *crisp*, i.e. the degree with which a state is the automaton's current state is either 0 or 1. On the other hand, FFAs can be in several states at any given time with different degrees of vagueness; vagueness is specified by a real number from the interval $[0, 1]$.

Theorem 3.2 enables us to transform any FFA into a deterministic automaton which computes the same membership function $\mu : \Sigma^* \rightarrow [0, 1]$. We just need to demonstrate how to implement the computation of μ with continuous discriminant functions.



(a) (b)

Figure 4: **Transformation of a FFA into its corresponding DFA**: (a) A fuzzy finite-state automaton with weighted state transitions. State 1 is the automaton's start state; accepting states are drawn with double circles. Only paths that can lead to an accepting state are shown (transitions to garbage states are not shown explicitly). (b) corresponding deterministic finite-state automaton which computes the membership function strings. The accepting states are labeled with the degree of membership. Notice that all transitions in the DFA have weight 1.

For that purpose, we augment the network architecture used for encoding DFAs with additional weights which connect the recurrent state neurons to a linear output neuron. The recurrent neurons shown in figure 3 implement the desired finite-state dynamics, i.e. transitions between crisp states. We showed in section 2.3 how to make the finite-state dynamics stable for arbitrary string lengths. The weights connecting the recurrent state neurons with the linear output neuron are just the memberships assigned to the DFA states after the transformation of a FFA into an equivalent DFA. The algorithm for encoding FFAs in second-order recurrent neural networks is shown in figure 4.

Recall the upper and lower bounds on the low and high signals, respectively (lemmas 2.1 and 2.8). Let μ_i denote the graded memberships assigned to DFA states q_i. In the worst case, the network computes for a given string the fuzzy membership function

$$\mu_{RNN} = \mu_i\,\phi_g^+ + (n_{acc} - 1)\,\phi_f^-$$

where n_{acc} is the number of DFA states with $\mu_i > 0$.

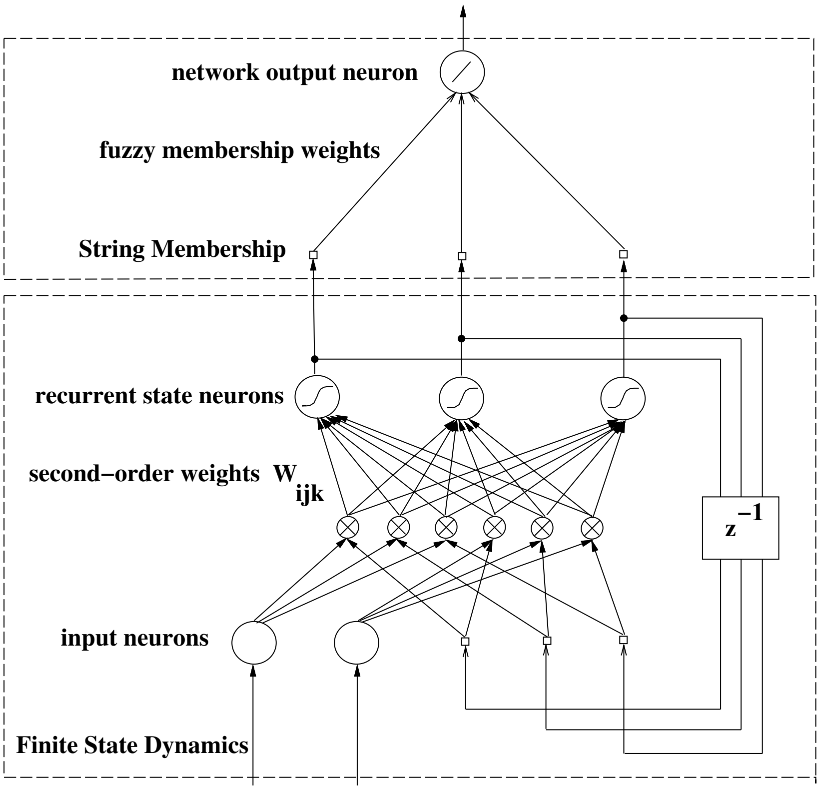


Figure 5: **Recurrent Network Architecture for Fuzzy Finite-state Automata**: The architecture consists of two parts: Recurrent state neurons encode the state transitions of the deterministic acceptor. These recurrent state neurons are connected to a linear output neuron which computes string membership.

Since ϕ_f^- and ϕ_g^+ converge toward 0 and 1, respectively for increasing values of H, μ_{RNN} converges toward μ_i. Notice that $|\mu_{RNN} - \mu_i|$ can be made arbitrarily small by increasing H.

5 Simulation Results

In order to empirically test our encoding methodology, we examine how well strings from a randomly generated FFAs are classified by a recurrent neural network in which the FFA is encoded. We randomly generated deterministic acceptors for fuzzy regular languages over the alphabet $\{0, 1\}$ with 100 states as follows: For each DFA state, we randomly generated a transition for each of the two input symbols to another state. Each accepting DFA state q_i was assigned a membership $0 < \mu_i < 1$; for all non-accepting states q_j, we set $\mu_j = 0$. We encoded these acceptors into recurrent networks with 100 recurrent state neurons, two input neurons (one for each of the two input symbols 0 and 1), and one linear output neuron. We measured their performance on 100 randomly generated strings of length 100 whose membership was determined from their deterministic acceptors. The graphs in figure 5 show the average absolute error of the network output as a function of the weight strength H used to encode the finite-state dynamics for DFAs where 1%, 5%, 20%, 30%, 50% and 100% of all states had labels $0 < \mu_i < 1$. We observe that the error exponentially decreases with increasing hint strength H, i.e. the average output error can be made arbitrarily small. The value of H

Input: FFA $M =< \Sigma, Q, R, F, Z, \delta, \omega >$ with $\Sigma = \{a_1, a_2, \ldots, a_K\}$, $Q = \{q_1, q_2, \ldots, q_N\}$, $R = \{q_1\}$ is the crisp start state, $F \subseteq Q$ is the set of accepting states, Z is the output alphabet, $\delta : \Sigma \times Q \rightarrow Q$ are the state transitions, $\omega : Q \rightarrow Z$ is the output map.

Output: Second-order recurrent neural network with $L(RNN) = L(FFA)$, i.e. the recurrent network assigns with arbitrary precision the same membership to all strings as the fuzzy automaton. The network is unique up to labeling of neurons.

Algorithm Initialization:

1. Transform M into a unique deterministic acceptor M' with $N' > N$ states that computes the string membership μ_G for arbitrary strings. Let M' be in state q'_i after some string s has been read; then the label $0 \leq \mu_i \leq 1$ associated with state q'_i indicates the membership assigned to s by M.
2. choose N' neurons with sigmoidal discriminant function $gx) = \frac{1}{1+e^{-x}}$ and one nonrecurrent output neuron with linear discrimiant function.
3. 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
4. choose weight strength H according to theorem 2.1

Network Construction:

5. 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 $\delta(q_j, a_k) \neq q_j$;
   - $W_{0ik} = \mu_i;$

Network Dynamics:

6. $S_i^{t+1} = g(b_i + \sum_{j,k,j>0} W_{ijk}\, S_j^t\, I_k^t)\ \ (i > 0)$

   $S_0^{t+1} = \sum_{j>0} \mu_j S_j^{t+1}$

Network Initialization:

7. Before reading a new string, set the initial network state to
   $\mathbf{S^0} = (S_0^0, S_1^0, \ldots, S_{N'}^0) = (S_0^0, 1, 0, \ldots, 0)$; the value of S_0^0 is computed as $S_0^0 = \mu_1$ (see network dynamics). .

Network Performance:

8. The output of S_0^f after reading a string s of length f is the string membership $\mu_G(s)$.

Network Complexity:

9. number of neurons: N'+1; number of weights: O(MN'); maximum fan-out: 3M
   weight alphabet $\Sigma_W = \{-H, -H/2, 0, H, \mu_1, \ldots, \mu_{N'}\}$

Figure 6: Algorithm for Encoding Arbitrary FFAs in Second-Order Recurrent Neural Networks

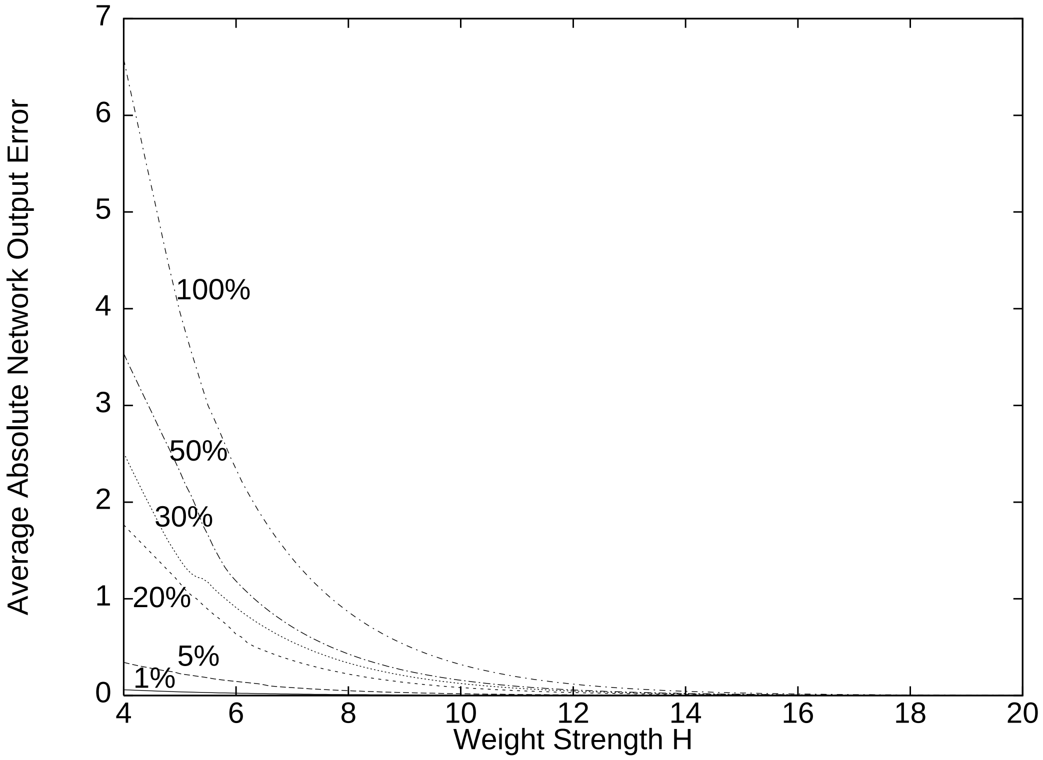


Figure 7: **Network Performance**: The graphs show average absolute error of the network output when tested on 100 randomly generated strings of length 100 as a function of the weight strength H used to encode the finite-state dynamics of randomly generated DFAs with 100 states. The percentages of DFA states with $\mu_i > 0$ were 1%, 5%, 20%, 30%, 50% and 100% respectively, of all DFA states.

for which the dynamics of all 6 DFAs remains stable for strings of arbitrary length is approximately $H \approx 9.8$.

6 Conclusions

We have proposed a method for representing fuzzy finite-state automata (FFAs) in recurrent neural networks with continuous discriminant functions. Based on a previous result on encoding stable representations of finite-state dynamics in recurrent networks, we have shown how FFAs can be encoded in recurrent networks that compute string membership functions with arbitrary accuracy. The method uses an algorithm which transforms FFAs into equivalent DFAs which compute fuzzy string membership. The fuzzy FFA states are transformed into crisp DFA states. A membership label μ_i with $0 < \mu_i \leq 1$ is associated with each accepting DFA state; nonaccepting DFA states have label $\mu_i = 0$. The membership of a string is equal to the membership label of the last visited DFA state.

A recurrent neural network is constructed from the original architecture used for DFA encodings by connecting the recurrent state neurons to a linear output neuron. The weights of these connections are set to the value of the membership labels of the DFA states. The accuracy of the computation of the string membership function depends on the network size, the number of DFA states which membership label $\mu_i > 0$, and the weight strength H used to encode the finite-state dynamics in the recurrent network. The larger H is chosen, the more accurate the network computes membership functions.

An interesting question is whether representations of FFAs can be *learned* through training on example strings and how weighted production rules are represented in trained networks. Such insight may lead to a more direct encoding of FFAs in recurrent networks without the additional step of transforming FFAs into equivalent DFAs which compute the same string membership functions, i.e. a *fuzzy representation* of states and outputs. This may lead to smaller analog VLSI implementations of finite-state controllers.

One problem with training fully recurrent networks with sigmoidal discriminant functions to behave like FFAs is the instability of learning algorithms based on gradient descent, i.e. it can become very difficult to train sigmoidal neurons to target values which are outside of the saturated regions of the discriminant function. This suggests the use of continuous multilevel threshold neurons [42] which also have the potential for stable internal DFA state representations. Whether training such networks is feasible remains an open question.

References

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

[2] R. Alquezar and A. Sanfeliu, "An algebraic framework to represent finite state machines in single-layer recurrent neural networks," *Neural Computation*, vol. 7, no. 5, p. 931, 1995.

[3] 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.

[4] J. Bezdek, ed., *IEEE Transactions on Neural Networks – Special Issue on Fuzzy Logic and Neural Networks*, vol. 3. IEEE Neural Networks Council, 1992.

[5] M. Casey, *Computation in discrete-time dynamical systems.* PhD thesis, Department of Mathematics, University of California at San Diego, La Jolla, CA, 1995.

[6] 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.

[7] 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.

[8] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, 1989.

[9] S. Das and M. Mozer, "A unified gradient-descent/clustering architecture for finite state machine induction," in *Advances in Neural Information Processing Systems 6* (J. Cowan, G. Tesauro, and J. Alspector, eds.), (San Francisco, CA), pp. 19–26, Morgan Kaufmann, 1994.

[10] 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.

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

[12] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite state automata in recurrent radial basis function networks," *Machine Learning*, 1995. In press.

[13] 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.

[14] B. Gaines and L. Kohout, "The logic of automata," *International Journal of General Systems*, vol. 2, pp. 191–208, 1976.

[15] 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.

[16] P. Goode and M. Chow, "A hybrid fuzzy/neural systems used to extract heuristic knowledge from a fault detection problem," in *Proceedings of the Third IEEE Conference on Fuzzy Systems*, vol. III, pp. 1731–1736, 1994.

[17] V. Gorrini and H. Bersini, "Recurrent fuzzy systems," in *Proceedings of the Third IEEE Conference on Fuzzy Systems*, vol. I, pp. 193–198, 1994.

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

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

[20] Y. Hayashi and A. Imura, "Fuzzy neural expert system with automated extraction of fuzzy if-then rules from a trained neural network," in *Proceedings of the First IEEE Conference on Fuzzy Systems*, pp. 489–494, 1990.

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

[22] B. Horne and D. Hush, "Bounds on the complexity of recurrent neural network implementations of finite state machines," in *Advances in Neural Information Processing Systems 6*, pp. 359–366, Morgan Kaufmann, 1994.

[23] 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.

[24] 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.

[25] S. Lee and E. Lee, "Fuzzy neural networks," *Mathematical Biosciences*, vol. 23, pp. 151–177, 1975.

[26] T. Ludermir, "Logical networks capable of computing weighted regular languages," in *Proceedings of the International Joint Conference on Neural Networks 1991*, vol. II, pp. 1687–1692, 1991.

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

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

[29] S. Mitra and S. Pal, "Fuzzy multilayer perceptron, inferencing and rule generation," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 51–63, 1995.

[30] T. Nishina, M. Hagiwara, and M. Nakagawa, "Fuzzy inference neural networks which automatically partition a pattern space and extract fuzzy if-then rules," in *Proceedings of the Third IEEE Conference on Fuzzy Systems*, vol. II, pp. 1314–1319, 1994.

[31] C. Omlin and C. Giles, "Constructing deterministic finite-state automata in recurrent neural networks," *Journal of the ACM.* Accepted for publication. A revised version of U. of Maryland TR UMIACS-TR-95-50.

[32] 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.

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

[34] 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.

[35] A. Pathak and S. Pal, "Fuzzy grammars in syntactic recognition of skeletal maturity from x-rays," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 5, pp. 657–667, 1986.

[36] C. Perneel, J.-M. Renders, J.-M. Themlin, and M. Acheroy, "Fuzzy reasoning and neural networks for decision making problems in uncertain environments," in *Proceedings of the Third IEEE Conference on Fuzzy Systems*, vol. II, pp. 1111–1125, 1994.

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

[38] M. Rabin, "Probabilistic automata," *Information and Control*, vol. 6, pp. 230–245, 1963.

[39] A. Salommaa, "Probabilistic and weighted grammars," *Information and Control*, vol. 15, pp. 529–544, 1969.

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

[41] H. Senay, "Fuzzy command grammars for intelligent interface design," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 5, pp. 1124–1131, 1992.

[42] J. Si and A. Michel, "Analysis and synthesis of a class of discrete-time neural networks with multilevel threshold neurons," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, p. 105, 1995.

[43] H. Siegelmann and E. Sontag, "Turing compatability with neural nets," *Applied Mathematics Letters*, vol. 4, no. 6, pp. 77–80, 1991.

[44] M. Thomason and P. Marinos, "Deterministic acceptors of regular fuzzy languages," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 228–230, 1974.

[45] K. Thornber, "The fidelity of fuzzy-logic inference," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 4, pp. 288–297, 1993.

[46] K. Thornber, "A key to fuzzy-logic inference," *International Journal of Approximate Reasoning*, vol. 8, pp. 105–121, 1993.

[47] P. Tino, B. Horne, and C.L.Giles, "Finite state machines and recurrent neural networks – automata and dynamical systems approaches," Tech. Rep. UMIACS-TR-95-1, Institute for Advance Computer Studies, University of Maryland, College Park, MD 20742, 1995.

[48] P. Tino and J. Sajda, "Learning and extracting initial mealy machines with a modular neural network model," *Neural Computation*, vol. 7, no. 4, pp. 822–844, 1995.

[49] 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.

[50] L.-X. Wang, *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[51] L.-X. Wang, "Fuzzy systems are universal approximators," in *Proceedings of the First International Conference on Fuzzy Systems*, pp. 1163–1170, 1992.

[52] T. Watanabe, M. Matsumoto, and M. Enokida, "Synthesis of synchronous fuzzy sequential circuits," in *Proceedings of the Third IFSA World Congress*, pp. 288–291, 1989.

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

[54] W. Wee and K. Fu, "A formulation of fuzzy automata and its applications as a model of learning systems," *IEEE Transactions on System Science and Cybernetics*, vol. 5, pp. 215–223, 1969.

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

[56] 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.