Learning, Representation, and Synthesis of Discrete Dynamical Systems in Continuous Recurrent Neural Networks *

C. Lee Giles[a,b] Christian W. Omlin[a]
[a]NEC Research Institute, 4 Independence Way, Princeton, NJ 08540
[b]Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742
E-mail: {giles,omlinc}@research.nj.nec.com

***Abstract*— This paper gives an overview on learning and representation of discrete-time, discrete-space dynamical systems in discrete-time, continuous-space recurrent neural networks. We limit our discussion to dynamical systems (recurrent neural networks) which can be represented as finite-state machines (e.g. discrete event systems [53]). In particular, we discuss how a symbolic representation of the learned states and dynamics can be extracted from trained neural networks, and how (partially) known deterministic finite-state automata (DFAs) can be encoded in recurrent networks. While the DFAs that can be learned exactly with recurrent neural networks are generally small (on the order of 20 states), there exist subclasses of DFAs with on the order of 1000 states that can be learned by small recurrent networks. However, recent work in natural language processing implies that recurrent networks can possibly learn larger state systems [35].**

I. Introduction

Answering the questions "What are the rules that govern the dynamics of a complex system?" and "How can a complex system with desired dynamics be designed?" is often very hard for systems with many degrees of freedom. However, the consensus is that methods for answering such questions become necessary for intelligent control applications [40]. In such applications, control strategies are not hard-wired at the time of system design; instead, the complexity or performance criteria dictate that control strategies be flexible, and that control systems learn from past experience and adapt to the current situation.

Symbolic dynamics has been shown to be an appropriate tool for analyzing the behavior of many different complex system, e.g. hybrid systems [51], optical systems [5], nonlinear Hamiltonian systems [64], general dynamical systems [10], and recently discrete-time, continuous-space recurrent neural networks [8, 15, 22, 43, 52, 62, 63]. In particular, extraction of symbolic knowledge from recurrent networks trained on message sequences was used to learn the structure of the computer interconnection network [25].

In this paper, we focus on the analysis and synthesis of discrete-time, discrete-space systems with discrete-time, continuous-space recurrent neural networks and primarily on our own work. Such networks are able to model a variety of dynamical processes, including systems that can be represented as deterministic finite-state automata (DFAs). Discrete event systems [27] are one such class of dynamical systems.

We will show how recurrent neural networks with a large number of degrees of freedom can be trained to correctly classify strings of some regular language which have been encoded as temporal sequences. Thus, design is performed through training. Contrary to popular belief, it is possible to extract a symbolic model of the learned knowledge by a network in the form of DFAs. We briefly summarize the DFA extraction algorithm and also address the quality of the extracted rules, i.e. how well the extracted automaton represents the (unknown) regular source grammar that generated the training data.

Recent theoretical results assert that neural networks are appropriate tools for refining initial do-

* Proceedings of the IEEE Workshop on Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems, Monterey, CA, August 27-29, 1995.

main knowledge [1, 19]. Given partial prior knowledge about some application, performance can be significantly improved when that prior knowledge is used effectively, e.g. by initializing a network with the knowledge prior to training.

We summarize theoretical results which show that it is possible to encode discrete-time, discrete-space dynamical systems in discrete-time, continuous-space recurrent networks such that the behavior of the system and its neural network model is identical, even for very large systems and even in the presence of noise.

Finally, we show that the recurrent neural network approach to modeling dynamical systems can scale well for two subclasses of DFAs.

Motivation for this work comes from the increased use of neural networks implemented as VLSI [39, 58]. DFAs as neural networks and vice versa could have significant implications for design and effective implementation in this area.

II. Regular Languages

Since we will be training on strings of regular languages, we give a brief description of regular grammars; see [28] for more details. Regular languages represent the smallest and simplest class of formal languages in the Chomsky hierarchy and are generated by regular grammars. A regular grammar G is a quadruple $G =< S, N, T, P >$ where S is the start symbol, N and T are respectively nonterminal and terminal symbols 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 $L(G)$. A deterministic finite-state automaton (DFA) M is the recognizer of each regular language L; $L(G) = L(M)$. Formally, a DFA M is a 5-tuple $M =< \Sigma, Q, R, F, \delta >$ where $\Sigma = \{a_1, \ldots, a_K\}$ is the alphabet of the language L, $Q = \{s_1, \ldots, s_{N_s}\}$ is a set of states, $R \epsilon Q$ is the start state, $F \subseteq Q$ is a set of accepting states and $\delta : Q \times \Sigma \rightarrow Q$ define 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 entire string x has been read by M. Alternatively, a DFA M can be interpreted as grammar which generates the regular language $L(M)$.

Regular languages and DFAs have proven useful in the analysis and design of discrete event systems for control and manufacturing processes [42, 53] and of course in high level VLSI design [4].

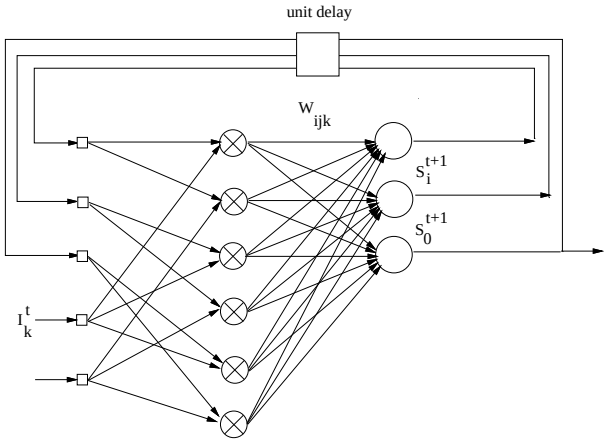

Figure 1: Second-order fully-connected recurrent neural network.

III. Recurrent Neural Networks

A. Network Architecture

Recurrent neural networks have been shown to be at least computationally equivalent to Turing machines [60, 59] and represent canonical forms of automata [43]. Their computational power and training ability make them useful tools for modeling nonlinear dynamical systems [13, 41]. DFAs can be represented in many discrete-time, recurrent network architectures [14, 16, 22, 29, 52]. (To review the large variety of recurrent neural network architectures, please see the papers in [21]. We choose for ease of representation networks with second-order weights W_{ijk} shown in figure 1. The continuous network dynamics are described by the following equations:

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

where b_i is the bias associated with hidden recurrent state neurons S_i; I_k denotes input neurons; g is the nonlinearity; and a_i is the activation of the ith neuron.

An aspect of the second order recurrent neural network is that the product $S_j^t I_k^t$ in the recurrent network directly corresponds to the state transition $\delta(q_j, a_k) = q_i$ in the DFA. After a string has been processed, the output of a designated neuron S_0 decides whether the network accepts or rejects a string. The network accepts a given string if the value of the output neuron S_0^t at the end of the string is greater than some preset value such as 0.5; otherwise, the network rejects the string. For the remainder of this

paper, we assume a one-hot encoding for input symbols a_k, i.e. $I_k^t \in \{0, 1\}$.

B. Training Algorithm

For any iterative training algorithm based on optimization, we must specify the training data, the form in which it will be presented, the error function that is to be minimized, the weight update algorithm and the training criterion, i.e. when to stop training.

The training data consists of strings that are members of some regular language (positive example strings) and strings which are not member of that language (negative example strings) along with a boolean label indicating membership in the unknown regular language. Example strings are presented as inputs to the network one symbol at a time. We use a one-hot encoding for input symbols, i.e. each symbol of the alphabet is assigned its own input neuron. Training is performed by updating the network weights at the end of each sample string presentation. We use a gradient descent optimization algorithm for finding a minimum on the network error surface which is defined by a quadratic cost function [55]. As with any optimization method based on gradient-descent, the training algorithm is prone to finding local minima for which the network does not correctly classify the training data. However, other methods such as simulated annealing are computationally prohibitive [31]. We have found that an incremental learning strategy whereby we start training on the shortest strings first and gradually train on more strings until the network correctly classifies the entire training set facilitates convergence. Theoretical results indicate that training with this incremental learning strategy is more likely to succeed compared to training on the entire data set from the start [6].

IV. Extraction of Symbolic Knowledge

Symbolic rules about the learned grammar can be extracted in the form of DFAs [8, 12, 22, 46, 62, 63]. The extraction algorithms are generally based on the hypothesis that the outputs of the recurrent state neurons tend to cluster when the network is well-trained and that these clusters correspond to the states of the learned DFA. Thus, rule extraction is reduced to finding clusters in the output space of state neurons and transitions between clusters. many clustering methods exist [22, 62, 63, 66]. The clustering method we use is based on partitioning; partitions and transitions between partitions correspond to DFA states and state transitions, respectively.

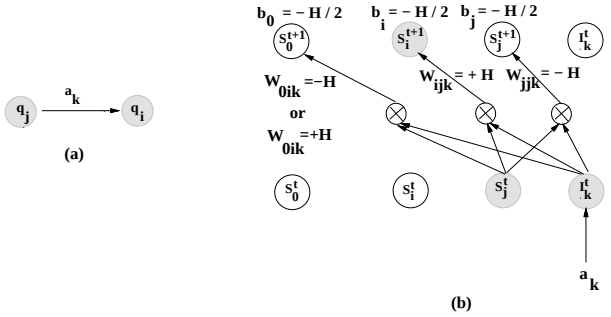

Figure 2: (a) A known DFA transition is programmed into a network. (b) Recurrent network unfolded over two time steps t and $t+1$. The insertion algorithm consists of two parts: Programming the network state transition and programming the output of the response neuron. Neurons S_i and S_j correspond to DFA states q_i and q_j, respectively; I_k denotes the input neuron for symbol a_k. Programming the weights W_{ijk}, W_{jjk} and biases b_i and b_j as shown in the figure ensures a nearly orthonormal internal representation of DFA states q_i and q_j (active/inactive neurons are illustrated by shaded/white circles). The value of the weight W_{0jk} connected to the response neuron S_0 depends on whether DFA state q_i is an accepting or rejecting state. H denotes the rule strength. The operation $S_i \cdot I_k$ is shown as $\otimes$.

Our algorithm employs a dynamical state space exploration along with pruning heuristics to make the extraction computationally feasible [22]. The cluster analysis is followed by a standard DFA minimization procedure [28]. Different minimized DFA's may be extracted for different partitionings. We have empirical evidence suggesting that, among the many possible DFA's, the DFA extracted with the coarsest partitioning which is consistent with the training data best models the knowledge learned by the neural network. Let M_q denote the minimized DFA extracted with partitioning parameter q. We extract DFA's $M_2, M_3, M_4, \ldots$; as the best representation of the network's knowledge, we choose the first DFA M_q which is consistent with the training data [50].

It is important to note that the above knowledge extraction algorithm may not capture the true long-term behavior of a trained network for an arbitrary number of time steps [44], i.e. several DFA states that are extracted with the above heuristic may de-

generate to one state due to the presence of attracting fixed points. It has also been argued from a dynamical systems point of view that the DFA extraction approach may be problematic [34]. However, our empirical results show the approach works well for extracting a model of the unknown regular source grammar. The mathematical foundation of DFA extraction based on partitioning of a network's state space has been subsequently established in [7, 44].

The approach for learning and extracting the grammatical rules from trained networks has been extended to context-free languages [28] by augmenting the basic recurrent network architecture with an external continuous stack [11, 24, 61, 65]. The augmented architecture implements a neural network pushdown automaton.

V. Synthesis of Large Systems

There is an increased interest in implementing neural network architectures in analog VLSI [3, 26, 30, 32, 38, 39, 54, 56]. Analog implementations offer the advantage of speed and low power consumption. Several methods for mapping DFAs into recurrent neural network architectures have been proposed [2, 16, 18, 36, 45]. Our algorithm takes advantage of the second-order structure of the recurrent network model which allows for a natural mapping of DFA state transitions into network state changes.

A. DFA Encoding Algorithm

Our encoding algorithm illustrated in figure 2 achieves a nearly orthonormal internal representation of the desired DFA dynamics; it constructs a network with $n+1$ recurrent state neurons (including the output neuron) and m input neurons from a DFA with n states and m input symbols. There is a one-to-one correspondence between state neurons S_i and DFA states q_i. For each DFA state transition $\delta(q_j, a_k) = q_i$, we set W_{ijk} to a large positive value $+H$. The self connection W_{jjk} is set to $-H$ (i.e. neuron S_j changes its state from high to low) except for state transitions $\delta(q_j, a_k) = q_j$ (self-loops) where W_{jjk} is set to $+H$ (i.e. state of neuron S_j remains high). Furthermore, if state q_i is an accepting state, then we program the weight W_{0jk} to $+H$; otherwise, we set W_{0jk} to $-H$. We set the bias terms b_i of all state neurons S_i to $-H/2$. For each DFA state transition, at most three weights of the network have to be programmed. The initial state $\mathbf{S}^0$ of the network is $\mathbf{S}^0 = (S_0^0, 1, 0, 0, \ldots, 0)$. The value of the response neuron S_0^0 is 0 if the DFA's initial state q_0 is a rejecting state and 1 otherwise. All weights that are not set to $-H$, $-H/2$ or $+H$ are set to zero.

B. Learning with Prior Knowledge

Empirical studies have shown that partial prior knowledge of a DFA (states and transitions) can significantly improve the training time [23]. Recurrent networks can even perform rule revision, i.e. refine incomplete initial rules [17, 36, 37, 57] and correct incorrect prior knowledge through learning from data [48].

C. Stability of Designed Networks

The following theorem asserts that time-discrete, continuous recurrent neural networks can represent DFAs [47]:

Theorem C.1 *Let $L(M_{DFA})$ denote the regular language accepted by some DFA with n states over an alphabet $\Sigma = \{a_1, \ldots, a_K\}$. Then, a second-order recurrent neural network (RNN) with $n+1$ sigmoidal state neurons and K input neurons ("one-hot encoding") can be constructed such that $L(M_{DFA}) = L(M_{RNN})$ for a finite value of the rule strength H.*

The significance of that theorem is that it is possible for a nonlinear discrete dynamical system to be modeled exactly by a nonlinear continuous dynamical system, even in the presence of noise [45, 49]. Thus, discrete nonlinear systems can be mapped onto analog VLSI designs.

D. Scaling to Large Networks

It would seem only natural that the value of the rule strength H scale with increasing network size n. While there exist degenerate cases where we can observe such a scaling behavior, the rule strength H is largely independent of the size of 'average' DFAs [49].

VI. Large Systems

To date, it has been very hard to train recurrent networks on strings generated by DFAs with more than about 20 states. The complexity and poor convergence properties of training algorithms are one reason why learning regular languages with recurrent neural networks does not scale well with DFA size. While partial prior knowledge about the DFA can significantly improve the convergence time, it does not overcome the scaling problem, i.e. learning of large DFA with on the order of hundreds of states.

However, it has recently been reported that certain subclasses of DFAs with on the order of hundreds and even thousands of states can be learned rather

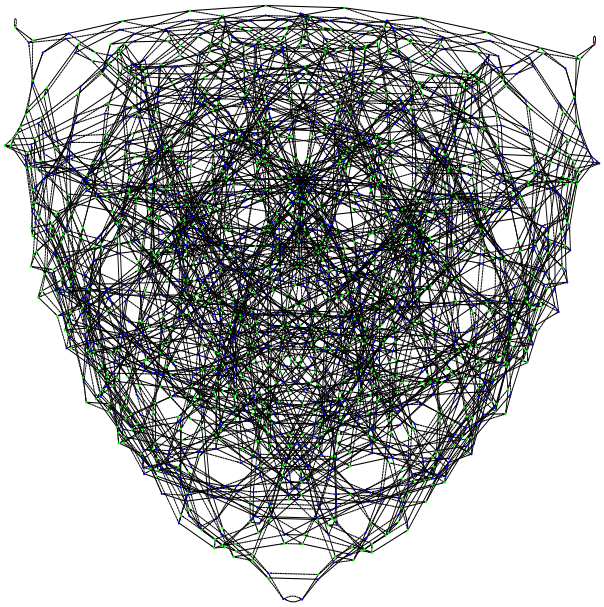

Figure 3: 2048 state FSM learned by a neural network.

easily with restricted feedback recurrent neural networks [9, 20].

A. Finite Memory Machines

DFAs with the property that their present state can always be uniquely determined from the knowledge of the last p inputs and the last q outputs for all possible sequences of length $max(p, q)$ are called *finite memory machines* (FMMs) [33]. The size of the largest learned machine had approximately 2000 states [20]. The characteristic property of large DMMs that can be learned easily with recurrent networks is that they can be implemented in sequential machines with little logic.

B. Definite Memory Machines

FMMs with input order $p > 0$ and output order $q = 0$ are called *definite memory machines* (DMMs) [33]. They require no feedback from the output and can be implemented with feedforward neural networks. Studies have shown that DMMs can also readily learned. DMMs with up to 2048 states (see figure 3) were learned perfectly (i.e. the network correctly classifies all strings of arbitrary length) with relatively little training data [9].

VII. Conclusions

We summarized methods for learning, extraction, and encoding of discrete dynamical processes in recurrent neural networks with continuous discriminant function. Extraction of symbolic knowledge from a trained network is useful since it makes the knowledge which is stored in a network's weights accessible to inspection, analysis, refinement and manipulation. Prior knowledge encoded in the network can be checked and can significantly reduce the learning time when used effectively. The stable insertion of automata (and their rules) into recurrent neural networks is important for robust computing and could possibly in neural network VLSI design.

Adaptive fuzzy systems have proven to be very powerful tools for solving control problems. Their parameters often have clear physical meanings which facilitates the choice of their initial values. Although there has been much interest and success in combining fuzzy systems with neural networks, the efforts have focused on static models, i.e. approximating unknown dynamical systems with static input output mappings. However, there exist applications which require state information to be available across indefinite periods of time. Thus, an interesting and open issue is how fuzzy rules and automata can be embedded in recurrent neural networks.

References

[1] Y. Abu-Mostafa, "Learning from hints in neural networks," *Journal of Complexity*, vol. 6, p. 192, 1990.

[2] R. Alquezar and A. Sanfeliu, "An algebraic framework to represent finite state automata in single-layer recurrent neural networks," *Neural Computation*, 1995. In press.

[3] J. Alspector and R. Allen, "A neuromorphic VLSI learning system," in *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference* (P. Losleben, ed.), (Cambridge), pp. 313–349, MIT Press, 1987.

[4] P. Ashar, S. Devadas, and A. Newton, *Sequential Logic Synthesis*. Norwell, MA: Kluwer Academic Publishers, 1992.

[5] J. Bajer and P. Lisonek, "Symbolic computation approach to nonlinear dynamics," *Journal of Modern Optics*, vol. 38, no. 4, p. 719, 1991.

[6] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

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

[8] 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.

[9] D. Clouse, C. Giles, B. Horne, and G. Cottrell, "Learning large debruijn automata with feedforward neural networks," Tech. Rep. CS94-398, Computer Science and Engineering, University of California at San Diego, La Jolla, CA, 1994.

[10] J. Crutchfield and K. Young, "Computation at the onset of chaos," in *Proceedings of the 1988 Workshop on Complexity, Entropy and the Physics of Information* (W. Zurek, ed.), (Redwood City, CA), pp. 223–269, Addison-Wesley, 1991.

[11] S. Das, C. Giles, and G. Sun, "Learning context-free grammars: Limitations of a recurrent neural network with an external stack memory," in *Proceedings of The Fourteenth Annual Conference of the Cognitive Science Society*, (San Mateo, CA), pp. 791–795, Morgan Kaufmann Publishers, 1992.

[12] 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.), pp. 19–26, San Mateo, CA: Morgan Kaufmann, 1994.

[13] K. Doya, "Universality of fully-connected recurrent neural networks," tech. rep., Department of Biology, University of California, San Diego, La Jolla, CA 92093, 1993.

[14] J. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine Learning*, vol. 7, no. 2/3, pp. 195–226, 1991.

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

[16] 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.

[17] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Unified integration of explicit rules and learning by example in recurrent networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 2, pp. 340–346, 1995.

[18] 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.

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

[20] C. Giles, B. Horne, and T. Lin, "Learning a class of large finite state machines with a recurrent neural network," *Neural Networks*, 1995. In press.

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

[22] 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, pp. 393–405, 1992.

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

[24] C. Giles, G. Sun, H. Chen, Y. Lee, and D. Chen, "Higher order recurrent networks & grammatical inference," in *Advances in Neural Information Processing Systems 2* (D. Touretzky, ed.), (San Mateo, CA), pp. 380–387, Morgan Kaufmann Publishers, 1990.

[25] M. Goudreau and C. Giles, "Using recurrent neural networks to learn the structure of interconnection networks," *Neural Networks*, 1995. In press.

[26] H. Graf, L. Jackel, and W. Hubbard, "VLSI implementation of a neural network model," *Computer*, vol. 21, no. 3, pp. 41–49, 1988.

[27] Y.-C. Ho, ed., *Discrete Event Dynamic Systems, Analyzing Complexity and Performance in the Modern World.* Piscataway, NJ: IEEE Press, 1992.

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

[29] B. Horne and C. Giles, "An experimental comparison of recurrent neural networks," in *Advances in Neural Information Processing Systems 7* (G. Tesauro, D. Touretzky, and T. Leen, eds.), MIT Press, 1995. To appear.

[30] M. Jabri and B. Flower, "Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks," *Neural Computation*, vol. 3, no. 4, pp. 546–565, 1991.

[31] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[32] C. Koch, "Seeing chips: Analog VLSI circuits for computer vision," *Neural Computation*, vol. 1, no. 2, pp. 184–200, 1989.

[33] Z. Kohavi, *Switching and Finite Automata Theory.* New York, NY: McGraw-Hill, Inc., second ed., 1978.

[34] J. Kolen, "Recurrent networks: State machines or iterated function systems?," in *Proceedings of the 1993 Connectionist Models Summer School* (M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, eds.), (Hillsdale, NJ), Lawrence Erlbaum, 1994.

[35] S. Lawrence, C. Giles, and S. Fong, "On the applicability of neural network and machine learning methodologies to natural language processing," Tech. Rep. UMIACS-TR-95-64, University of Maryland, College Park, MD 20742, 1995.

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

[37] R. Maclin and J. Shavlik, "Refining domain theories expressed as finite-state automata," in *Proceedings of the Eighth International Workshop on Machine Learning (ML'91)* (L. Birnbaum and G. Collins, eds.), (San Mateo, CA), Morgan Kaufmann, 1991.

[38] N. May and D. Hammerstrom, "Fault simulation of a wafer-scale integrated neural network," *Technical Report: CS/E-88-020*, 1988.

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

[40] A. Meystel, "Multiscale models and controllers," in *Proceedings of IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, pp. 13–26, 1994.

[41] K. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. on Neural Networks*, vol. 1, no. 1, p. 4, 1990.

[42] A. Nerode and W. Kohn, "Models for hybrid systems: Automata, topologies, controllability, observability," Tech. Rep. TR 93-28, Mathematical Sciences Institute, Cornell University, Ithaca, New York 14850, June 1993.

[43] O. Nerrand, P. Roussel-Ragot, G. D. L. Personnaz, and S. Marcos, "Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms," *Neural Computation*, vol. 5, pp. 165–197, 1993.

[44] P. T. no, 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.

[45] C. Omlin and C. Giles, "Constructing deterministic finite-state automata in recurrent neural networks," Tech. Rep. UMIACS-TR-95-50 and CS-TR-3460, University of Maryland, College Park, MD 20742, 1995.

[46] C. Omlin and C. Giles, "Extraction of rules from discrete-time recurrent neural networks," *Neural Networks*, 1995. Accepted for publication.

[47] C. Omlin and C. Giles, "Fault-tolerant implementation of finite-state automata in recurrent neural networks," Tech. Rep. 95-3, Computer

Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180, 1995.

[48] C. Omlin and C. Giles, "Rule checking with recurrent neural networks," *IEEE Transactions on Knowledge and Data Engineering*, 1995. in press.

[49] C. Omlin and C. Giles, "Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants," *Neural Computation*, 1995. Accepted for publication.

[50] C. Omlin, C. Giles, and C. Miller, "Heuristics for the extraction of rules from discrete-time recurrent neural networks," in *Proceedings International Joint Conference on Neural Networks 1992*, vol. I, pp. 33–38, June 1992.

[51] P. Peleties and R. DeCarlo, "Analysis of a hybrid system using symbolic dynamics and Petri nets," *Automatica*, vol. 30, no. 9, p. 1421, 1991.

[52] J. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, no. 2/3, pp. 227–252, 1991.

[53] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[54] A. Rao, M. Walker, L. Clark, L. Akers, and R. Grondin, "VLSI implementation of neural classifiers," *Neural Computation*, vol. 2, no. 1, pp. 35–43, 1990.

[55] R.J.Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Backpropagation: Theory, Architectures and Applications* (Y. Chauvin and D. E. Rumelhart, eds.), ch. 13, pp. 433–486, Hillsdale, N.J.: Lawrence Erlbaum Publishers, 1995.

[56] S. Satyanarayana, Y. Tsividis, and H. P. Graf, "A reconfigurable analog VLSI neural network chip," in *Proceedings of NIPS-89*, pp. 758–768, 1989.

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

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

[59] H. Siegelmann, B. Horne, and C. Giles, "Computational capabilities of recurrent narx neural networks," *IEEE Trans. on Systems, Man and Cybernetics*, 1996. accepted.

[60] 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.

[61] G. Sun, C. Giles, H. Chen, and Y. Lee, "The neural network pushdown automaton: Model, stack and learning simulations," Tech. Rep. UMIACS-TR-93-77, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, 1993.

[62] P. Tino and J. Sajda, "Learning and extracting initial mealy automata with a modular neural network model," *Neural Computation*, vol. 7, pp. 822–844, 1995.

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

[64] K. Yip, "Understanding complex dynamics by visual and symbolic reasoning," *Artificial Intelligence*, vol. 51, no. 1-3, p. 179, 1991.

[65] Z. Zeng, R. Goodman, and P. Smyth, "Discrete recurrent neural networks for grammatical inference," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 320–330, 1994.

[66] 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.