Pergamon

Neural Networks, Vol. 9, No. 1, pp. 41-52, 1996
Copyright © 1996 Elsevier Science Ltd. All rights reserved
Printed in Great Britain
0893-6080/96 $15.00+.00

0893-6080(95)00086-0

CONTRIBUTED ARTICLE

Extraction of Rules from Discrete-time Recurrent Neural Networks

CHRISTIAN W. OMLIN AND C. LEE GILES

NEC Research Institute

(*Received* 28 *August* 1992; *revised and accepted* 6 *June* 1995)

Abstract—*The extraction of symbolic knowledge from trained neural networks and the direct encoding of (partial) knowledge into networks prior to training are important issues. They allow the exchange of information between symbolic and connectionist knowledge representations. The focus of this paper is on the quality of the rules that are extracted from recurrent neural networks. Discrete-time recurrent neural networks can be trained to correctly classify strings of a regular language. Rules defining the learned grammar can be extracted from networks in the form of deterministic finite-state automata (DFAs) by applying clustering algorithms in the output space of recurrent state neurons. Our algorithm can extract different finite-state automata that are consistent with a training set from the same network. We compare the generalization performances of these different models and the trained network and we introduce a heuristic that permits us to choose among the consistent DFAs the model which best approximates the learned regular grammar.*

Keywords—Recurrent neural networks, Grammatical inference, Regular languages, Deterministic finite-state automata, Rule extraction, Generalization performance, Model selection, Occam's razor.

1. INTRODUCTION

There has been much interest in the integration and extraction of knowledge from neural networks (Towell et al., 1990; Frasconi et al., 1991; Hanson & Burr, 1991; Giles et al., 1992; Watrous & Kuhn, 1992; Fu, 1994). One reason is that the lack of understanding of the rules that underlie neural network performance has limited their use in some application domains. Another is that some intelligent processing naturally requires the use of symbolic and rule-based knowledge. Recurrent neural networks with discrete-time inputs readily lend themselves to certain types of knowledge encoding and extraction—in particular the ordered triple of a discrete Markov {state; input $\rightarrow$ next-state} process (Giles et al., 1992; Omlin & Giles, 1992). What this paper addresses is the validity and usefulness of extracting this information based on a representation of the neurons' activation.

Acknowledgements: The authors acknowledge useful discussions with E. B. Baum, K. Lang, and L. Valiant.

Requests for reprints should be sent to C. L. Giles, NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA; E-mail: giles@research.nj.nec.com

1.1. Background

Recently, the training of recurrent neural networks that recognize finite state languages has been discussed by several authors (Cleeremans et al., 1989; Williams & Zipser, 1989; Elman, 1990; Giles et al., 1992; Watrous & Kuhn, 1992). The motivation for this work has been multifold; from understanding how these neural networks process natural language to quantifying and understanding their computational characteristics. Cleeremans (1989) and Elman (1990) trained recurrent networks by predicting the next symbol using a truncation of the backward recurrence. Cleeremans et al. (1989) concluded that the hidden unit activations represented past histories and that clusters of these activations can represent the states of the generating automaton. The work reported by Giles et al. (1992) is an extension of Cleeremans et al. (1989) in that complete deterministic finite-state automata are extracted from recurrent networks. Watrous and Kuhn (1992) implemented an alternative approach to state machine extraction. In related work, Crutchfield and Young (1991) extract finite-state automata from general dynamical systems though the systems do not have dynamical inputs.

1.2. Motivation

Grammars and automata are intimately related.

Grammars generate the languages that automata recognize. The computational power of recurrent networks has been proven by relating these networks to automata. Recurrent networks have recently been shown to be computationally quite powerful, i.e., Turing equivalent (Siegelmann & Sontag, 1992). The representational limitations of recurrent network models have been explored by showing their relationships to automata (Goudreau et al., 1994, Giles et al., 1995b). Thus, it is natural to see how well recurrent networks learn what they can be proven to represent.

This being said, it is not evident that recurrent networks are the best tools for learning regular grammars. However, regular language inference requires state information to be stored over indefinite periods of time and no feature extraction is necessary for learning. Thus, grammatical inference is a good test bed in which to investigate issues related to the representation of symbolic knowledge in recurrent networks and to explore their computational capabilities. For example, Giles and Omlin (1994) show how grammar learning can be useful in quantifying the computational power of a pruning algorithm of recurrent neural networks. Jim et al. (1995) show that training noisy recurrent networks with grammatical strings is a good measure of the effect of different types of noise insertion methodologies on networks generalization and convergence performance. The performance of recurrent networks on long temporal sequences is naturally explored by using grammatical strings (Manolios & Fanelli, 1994). Frasconi et al. (1995) used grammar learning and representation to explore a new recurrent network architecture. Furthermore, recent results by Giles et al. (1995a) and Clouse et al. (1994) imply that neural networks under certain constraints can learn extremely large grammars and may be competitive with other methods.

The extraction of (symbolic) rules from both feed-forward as well as recurrent neural networks has become an active area of research (Cleeremans et al., 1989; Towell et al., 1990; Giles et al., 1992; Fu, 1994). The following question is the focus of our work: How good are the rules extracted from recurrent networks? This is an important one and, to our knowledge, has not been discussed anywhere else in the literature.

1.3. Overview

We train discrete second-order recurrent neural networks to recognize strings of a regular language given a finite set of positive and negative example strings. The DFAs extracted from trained networks depend on a clustering parameter and different, minimized DFAs may be obtained for different values of the clustering parameter. Several among the different DFAs may correctly classify all strings of the training set. Given a set of extracted DFAs which may accept different regular languages, the question arises which DFA best approximates the unknown source grammar, i.e., which extracted DFA assigns the same label as the unknown source grammar to as many strings as possible. A *model selection heuristic* is introduced which allows us to choose among the different DFAs the one that best models the regular language.

The remainder of this paper is organised as follows: we give a brief introduction to regular languages and grammatical inference in Section 2, followed by a presentation of the neural network architecture and training in Section 3; we also include some generalization results of trained networks on long test strings. The rule extraction algorithm and the problem of selecting among the extracted DFAs the best model are discussed in detail in Section 4. A summary of the empirical results, followed by a short discussion of related work and possible future research directions conclude this paper.

2. REGULAR LANGUAGES

2.1. Definitions

Regular languages represent the smallest class of formal languages in the Chomsky hierachy (Hopcroft & Ullman, 1979). Regular languages are generated by regular grammars. A regular grammar G is a 4-tuple $G = \langle S, N, T, P\rangle$ 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 $A, B \in N$ and $a \in T$. The regular language generated by G is denoted $L(G)$.

Associated with each regular language L is a deterministic finite-state automaton (DFA) M which is an acceptor for the language $L(G)$, i.e., $L(G) = L(M)$. DFA M accepts only strings which are a member of the regular language $L(G)$. Formally, a DFA M is a 5-tuple $M = \langle\Sigma, Q, R, F, \delta\rangle$ where $\Sigma = \{a_1, \dots, a_k\}$ is the alphabet of the language L, $Q = \{s_1, \dots, s_M\}$ is a set of states, $R \in Q$ is the start state, $F \subseteq Q$ is a set of accepting states and δ: $Q \times \Sigma \rightarrow Q$ 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 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. Grammatical Inference

Grammatical inference, the problem of inferring grammar(s) from samples of strings of an unknown regular language, is an NP-complete problem (Gold, 1978; Angluin & Smith, 1983). The problem of grammatical inference can be stated informally as follows. Given a finite set of strings along with a label indicating whether or not a string is a member of the language, find the rules which define the language. For example, we may be given the following two sets of positive and negative example strings:

Positive examples	*Negative examples*
10010	100010
1011101	001110001
00	000
111	100110111000
100100100111	1001001000111
1010011	100001
001111	00001111
101010	1010001010

In this example the rule might be to reject strings which contain three or more consecutive 0s and to accept all other strings. It is intuitively clear that the inferred grammar will better model the underlying language if more example strings — both positive and negative — are provided as input to the inference algorithm.

Our algorithm extracts rules from (trained) networks in the form of DFAs, which can be readily transformed into a regular grammar where each state of the DFA corresponds to a non-terminal symbol and the DFA transitions correspond to production rules.

3. RECURRENT NEURAL NETWORK

3.1. Architecture

Recurrent neural networks have been shown to have powerful capabilities for modeling many computational structures; an excellent discussion of recurrent neural network models and references can be found in Hertz et al. (1991). To learn grammars, we use a second-order recurrent neural network (Lee et al., 1986; Giles et al., 1990; Gun et al., 1990; Pollack, 1991). The network architecture is illustrated in Figure 1. This net has N recurrent hidden neurons labeled S_j; L special, nonrecurrent input neurons labeled I_k; and $N^2 \times L$ real-valued weights labeled W_{ijk}. As long as the number of input neurons is small compared to the number of hidden neurons, the complexity of the network only grows as $O(N^2)$, the same as a linear network. We refer to the values of the hidden neurons collectively as a state *vector* **S** in the finite N-dimensional space $[0,1]^N$. Note that the weights W_{ijk} modify a product of the hidden S_j and input I_k neurons. This quadratic form directly represents the state transition diagrams of a state process — $\{$*input, state*$\} \rightarrow \{$*next state*$\}$. This recurrent network accepts a time-ordered sequence of inputs and evolves with dynamics defined by the following equations:

$$S_i^{(t+1)} = g(\Xi_i), \qquad \Xi_i \equiv \sum_{j,k} W_{ijk} S_j^{(t)} I_k^{(t)}$$

where g is a sigmoid discriminant function. Each input string is encoded into the input neurons one character per discrete time step t. The above equation is then evaluated for each hidden neuron S_i to compute the next state vector **S** of the hidden neurons at the next time step $t+1$. With unary encoding the neural network is constructed with one input neuron

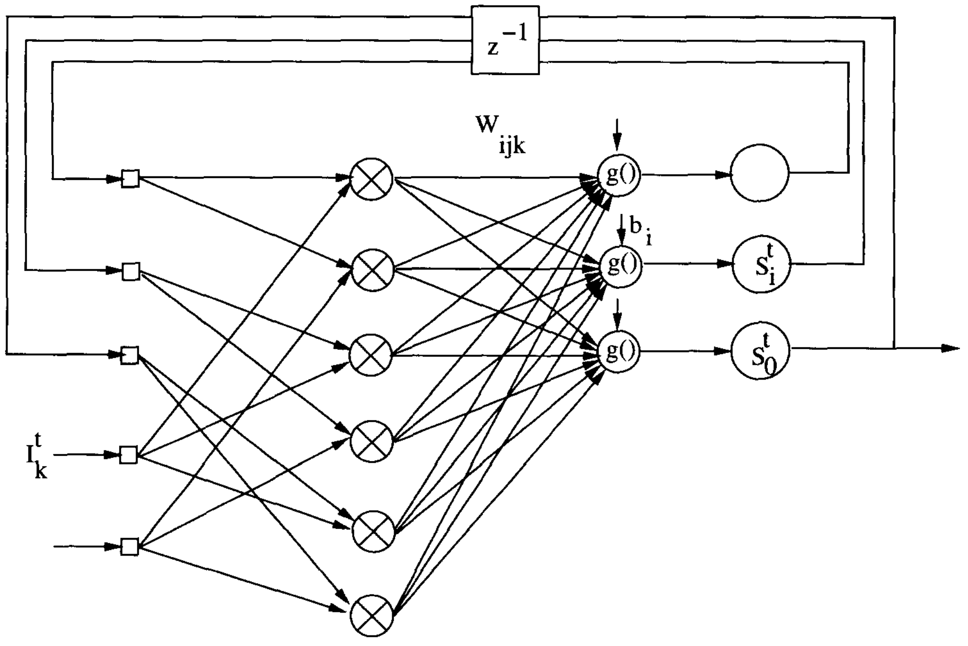

FIGURE 1. A second-order, single layer recurrent neural network. $S_i^{(t)}$ and $I_k^{(t)}$ represent the values of the *i*th and *k*th state and input neuron, respectively, at time *T*. $\otimes$ represents the operation $W_{ijk} \times S_j^{(t)} \times I_k^{(t)}$; $g()$ is the sigmoidal discriminant function.

TABLE 1
Network Generalization Performance

Test Set	Training Set Size	Average μ	90% Confidence Interval
Strings of length 1–15	1000	0.84%	$0.6\% \leqslant \mu \leqslant 1.00\%$
	700	1.63%	$1.26\% \leqslant \mu \leqslant 1.99\%$
	300	9.74%	$8.34\% \leqslant \mu \leqslant 11.14\%$
	100	38.09%	$36.23\% \leqslant \mu \leqslant 39.96\%$
	50	57.96%	$56.29\% \leqslant \mu \leqslant 59.64\%$
1000 random strings of length 100	1000	4.61%	$3.57\% \leqslant \mu \leqslant 5.66\%$
	700	7.07%	$5.67\% \leqslant \mu \leqslant 8.47\%$
	300	26.23%	$23.38\% \leqslant \mu \leqslant 29.07\%$
	100	48.50%	$47.47\% \leqslant \mu \leqslant 49.52\%$
	50	61.17%	$59.36\% \leqslant \mu \leqslant 62.98\%$
1000 random strings of length 1000	1000	4.59%	$3.58\% \leqslant \mu \leqslant 5.59\%$
	700	7.68%	$6.08\% \leqslant \mu \leqslant 9.28\%$
	300	25.89%	$23.02\% \leqslant \mu \leqslant 28.76\%$
	100	47.47%	$46.43\% \leqslant \mu \leqslant 48.50\%$
	50	60.24%	$58.05\% \leqslant \mu \leqslant 62.43\%$

The table shows some statistics (t-student) on the generalization performance of networks with eight recurrent state neurons on two different test sets. The training sets contained the first 1000, 700, 300, 100, and 50 strings in alphabetical order; starting with strings of length 1 the strings alternated between positive and negative examples. The average performance and the 90% confidence intervals clearly show how the network performances improve with increasing training set size—and thus string lengths—particularly for the test set containing strings of lengths 100 and 1000. The networks generally perform poorer on longer strings—maximum string length 15 compared to string length 100 and 1000—due to the increasing instability of the internal DFA representation with increasing string length, leading to misclassification of strings.

for each character in the alphabet of the relevant language. This condition might be too restrictive for grammars with large alphabets.

3.2. Training Procedure

A special recurrent neuron is selected as the network's output neuron. A network learns by comparing its actual output with the desired classification at the end of a string. Weight updates are computed using a second-order form of the RTRL net of Williams and Zipser (1989) which computes the partial derivative of a quadratic error function with respect to all weights W_{ijk}.

3.3. Network Performance

We consider a regular language over the alphabet $\{0, 1\}$ that is recognized by a randomly generated minimal 10-state DFA. From a data set consisting of 1000 alternating positive and negative example strings starting with strings of length 0, we formed five different training sets. The first training set consisted of all 1000 example strings; the remaining training sets consisted of the first 700, 300, 100 and 50 strings, respectively, of the 1000 example strings. This was done in order to achieve a variety of generalization performances for networks of fixed size.

All the weights were initialized to random values in the interval $[-1.0, 1.0]$. The learning rate α and the momentum η were both set to 0.5.

We used an incremental learning strategy where a network is first trained on a small subset of all training strings ("working set"); this subset only contains the shortest strings. When a network has either learned to correctly classify all the strings of a working set or a maximum number of epochs has expired for the current working set, new strings from the training set are added to the working set and the network is trained on this expanded working set. The working set is expanded until a network correctly classifies all the strings of the training set. This incremental training heuristic aims at overcoming the problems that arise from training on long strings with a gradient descent learning algorithm (Bengio et al., 1994).

We trained 25 networks with eight state neurons on each of the five training sets and performed a statistical analysis on the networks' average generalization performances (Table 1) on three test sets consisting of all strings of length up to 15 ($65k$ strings), 1000 random strings of length 100, and 1000 random strings of length 1000. We observe that the average generalization performance improves with increasing training set size which contain strings of increasing length; the 90% confidence intervals are non-overlapping. This result is not surprising since networks trained on smaller data sets have less information and thus learn a poorer model of the source grammar. Networks generally perform poorer when tested on strings that are longer compared to the training sets; a network's internal DFA representation becomes increasingly unstable with increasing string length which leads to string misclassification.

The next section addresses the issue of how DFA extraction can be made computationally feasible and the quality of the extracted rules, i.e., how well do extracted DFAs generalize compared to the network's generalization performance; we also compare the performance of DFAs that are extracted with different quantization levels from the same network.

4. RULE EXTRACTION

4.1. Algorithm

We describe our heuristic for extracting rules from recurrent networks in the form of DFAs (Giles et al., 1992). Different approaches are described in Cleeremans et al. (1989), Watrous and Kuhn (1992) and Tino and Sajda (1995). The heuristic is not restricted to second-order networks and can be applied to any

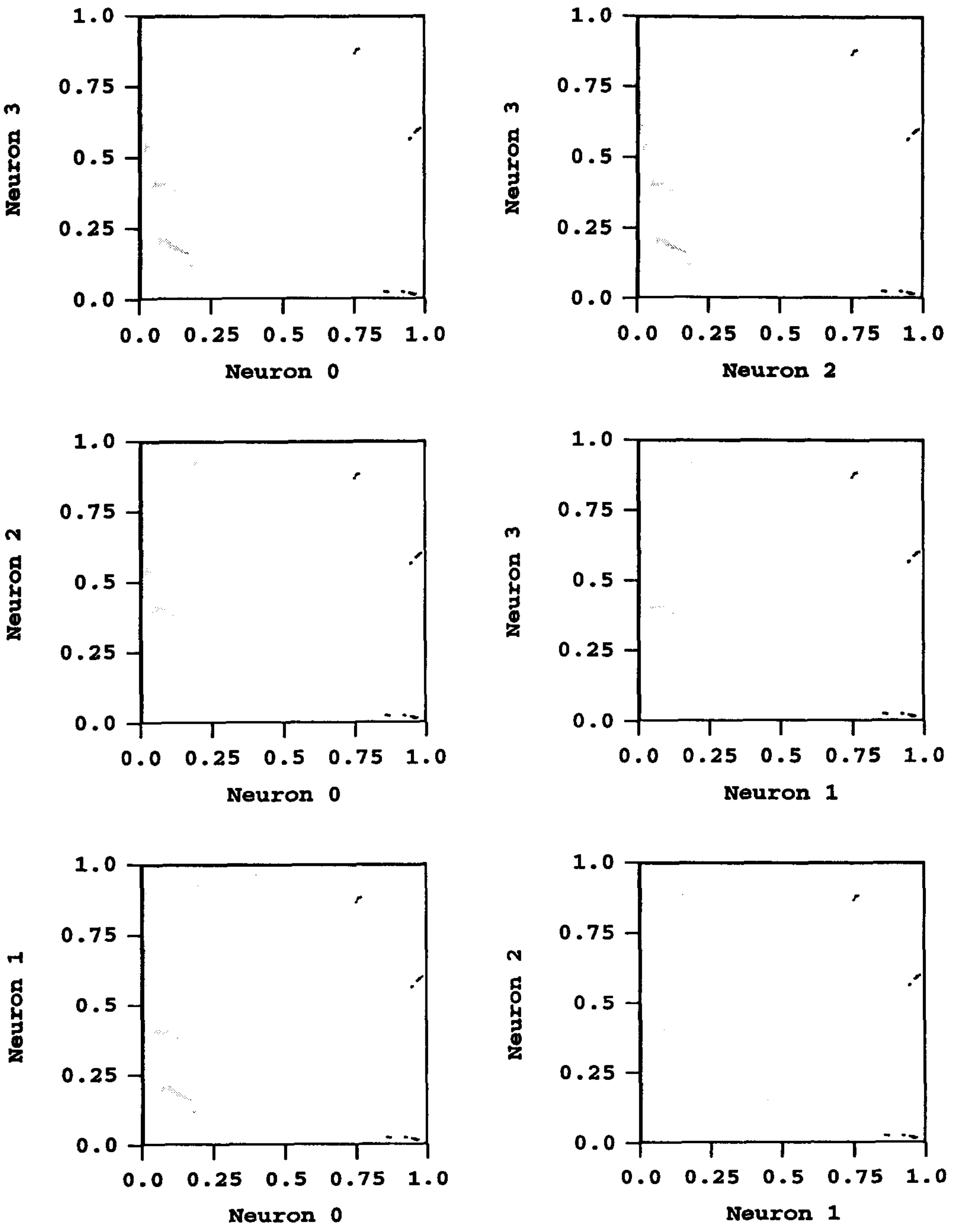

FIGURE 2. Clustering in network state space. A recurrent network with four state neurons was trained to accept only strings which do not contain three consecutive 0s. The network builds an internal representation of the learned language in the space of its output neurons. The two-dimensional projection of the state space $[0, 1]^4$ into the (S_i, S_j)-plane for all possible pairs (S_i, S_j) are shown (six projections). The figures show that the state vectors reached for the strings of the test set tend to form clusters. We hypothesize that these clusters correspond to states of the learned DFA and transitions between clusters correspond to state transitions in the DFA.

discrete-time neural network. The algorithm used to extract a finite-state automaton from a network is based on the observation that the outputs of the recurrent state neurons of a trained network tend to cluster as shown in Figure 2. The outputs of a trained four-neuron recurrent network which is tested on a small test set are shown as two-dimensional projections into the (S_i, S_j)-plane, for all possible pairs (S_i, S_j) (six projections). Our hypothesis is that collections of these clusters correspond to the states of the finite-state automaton the network has learned. Furthermore, these clusters tend to be well separated in networks which learned a given grammar well as measured by their generalization performance on unseen strings. For the DFA extractions, we consider network states where the output of the response neuron S_0 is larger than 0.5 to correspond to accepting DFA states; otherwise, the current network state corresponds to a rejecting DFA state. The problem of DFA extraction is thus reduced to identifying clusters in the output space $[0, 1]^N$ of all state neurons. We use a dynamical state space exploration which identifies the DFA states and at the same time avoids exploration of the entire space which is computationally not feasible.

The extraction algorithm divides the output of each of the N state neurons into q intervals of equal size, yielding q^N partitions in the space of outputs of the state neurons. We also refer to q as the *quantization level.*

Starting in some defined initial network state, the trained weights of the network will cause the current network state to follow a continuous state trajectory as symbols are presented as input. The algorithm considers all strings in alphabetical order starting with length 1. This procedure defines a search tree with the initial state as its root; the number of successors of each node is equal to the number of symbols in the input alphabet. Links between nodes correspond to transitions between DFA states. The search is performed in breadth-first order. When a transition is made into another (not necessarily different) partition, then all paths from that new network state are followed for subsequent input symbols creating new states in the DFA for each new input symbol, subject to the following three conditions. (1) When a previously visited partition is reached, then only the new transition is defined between the previous and the current partition; no new DFA state is created. This corresponds to pruning the search tree at that node. (2) In general, many different state vectors belong to the same partition. We choose the state vector which first reached a partition as the new initial state for all subsequent symbols. This condition only applies when two or more transitions from a partition to another partition are extracted. (3) When a transition leads from a partition immediately to the same partition, then a loop is created and the search tree is also pruned at that node. The algorithm terminates when no new partitions are visited for the first time — hence no new DFA states are created — and all possible transitions for all DFA states have been extracted.

The extraction would seem computationally infeasible if all q^N partitions had to be visited. However, the clusters corresponding to DFA states are often local and can be covered with fewer partitions. Also, we aim to extract DFAs with the smallest possible quantization level which explains the training data. These properties along with the pruning of the search tree makes DFA extraction feasible even for DFAs with larger alphabets.

Clearly, the extracted DFA depends on the quantization level q chosen, i.e., in general, different DFAs will be extracted for different values of q. Furthermore, because of condition 2, different DFAs may be extracted depending on the order in which the successors of a node in the search tree are visited. In general, however, these distinctions are not significant because of the subsequent standard minimization algorithm (Hopcroft & Ullmann, 1979). This algorithm yields a unique, minimal representation of the extracted DFA. Many different DFA extracted thus collapse into *equivalence classes.*

4.2. Example

We will illustrate in detail how the extraction algorithm builds the corresponding DFA using a simple example. Assume a recurrent network with two state neurons and two input neurons is trained on some data set. The network state vector has dimension 2 and the range of possible values of S_0 and S_1 can be represented as a unit square in the (S_1, S_2)-plane. For reasons of simplicity, we choose quantization level $q=3$, i.e., the output of each of the two state neurons is divided into three equal length intervals, defining $3^2=9$ discrete partitions. Each of these partitions corresponds to a hypothetical state in the unknown DFA. We will assign labels *1, 2, 3*, . . . to the partitions in the order in which they are visited for the first time.

The start state is defined by the initial network state vector used for training the network; it lies in partition *1* (Figure 3a). The state vector reached in partition *1* falls within the accepting region (the output of the response neuron is larger than 0.5); thus, this initial state is marked as an accepting state (shaded circle). The start state represents the totally unknown DFA. On input "0" and "1", the network makes a transition into partitions *1* and *2*, respectively. This causes the creation of a transition to a new accepting DFA state 2 and a transition from state 1 to itself. In the next step,

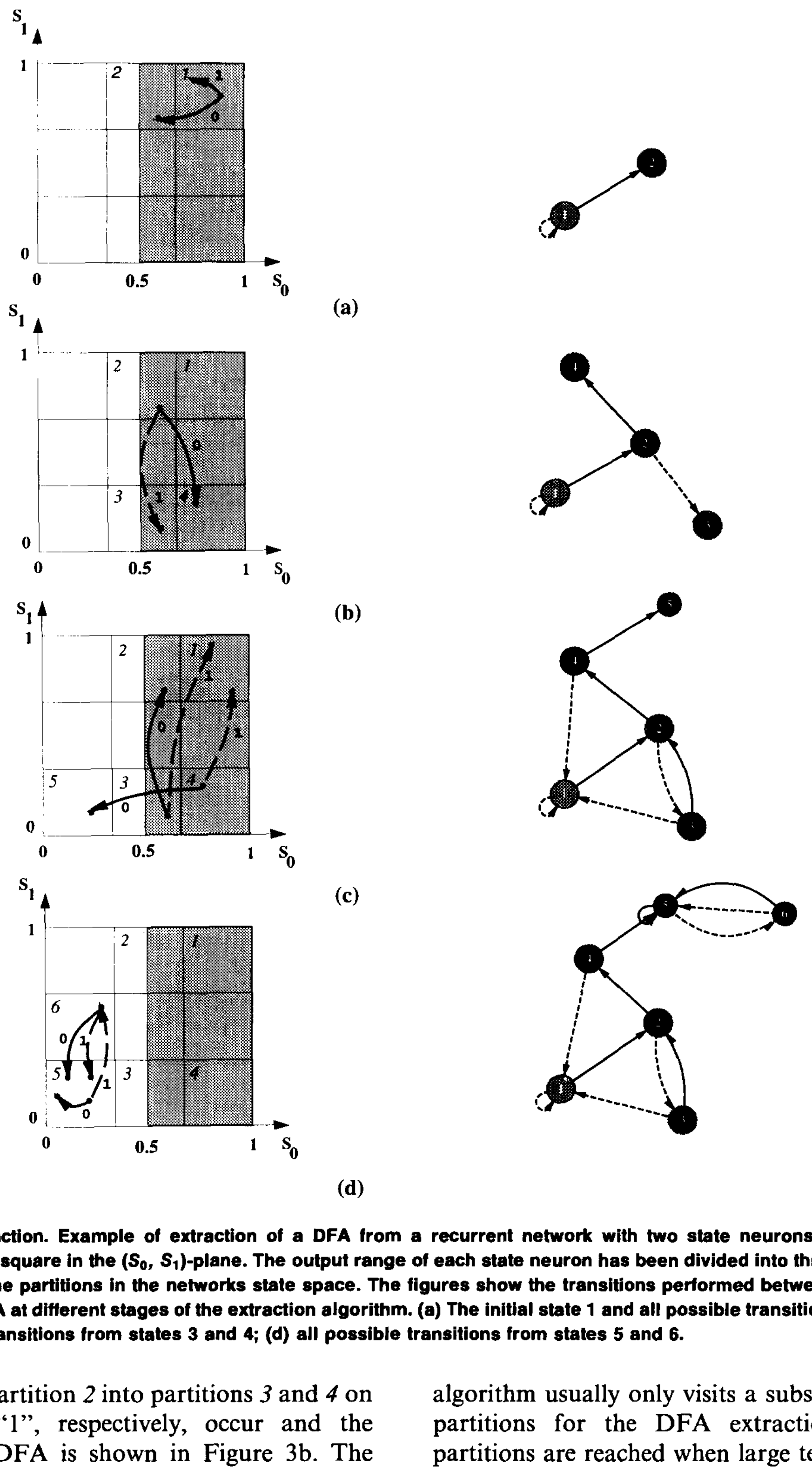

FIGURE 3. DFA extraction. Example of extraction of a DFA from a recurrent network with two state neurons. The state space is represented as a unit square in the (S_0, S_1)-plane. The output range of each state neuron has been divided into three intervals of equal length resulting in nine partitions in the networks state space. The figures show the transitions performed between partitions and the (partial) extracted DFA at different stages of the extraction algorithm. (a) The initial state 1 and all possible transitions; (b) all transitions from state 2; (c) all transitions from states 3 and 4; (d) all possible transitions from states 5 and 6.

transitions from partition *2* into partitions *3* and *4* on input "0" and "1", respectively, occur and the resulting partial DFA is shown in Figure 3b. The DFA in Figure 3c shows the current knowledge about the DFA after all state transitions from states 3 and 4 have been extracted from the network. In the last step, only one more new state is created (Figure 3d). Since no known state has left any undefined transitions, the extraction algorithm terminates. Notice that not all partitions have been assigned to DFA states. The algorithm usually only visits a subset of all available partitions for the DFA extraction. Many more partitions are reached when large test sets (especially when they contain many long strings) are used (e.g., when measuring the generalization performance on a large test set). The extracted DFA and its unique, minimized representation are shown in Figure 4. The DFAs accept all strings which do not contain three consecutive 0s (notice that both DFAs accept exactly the same regular language).

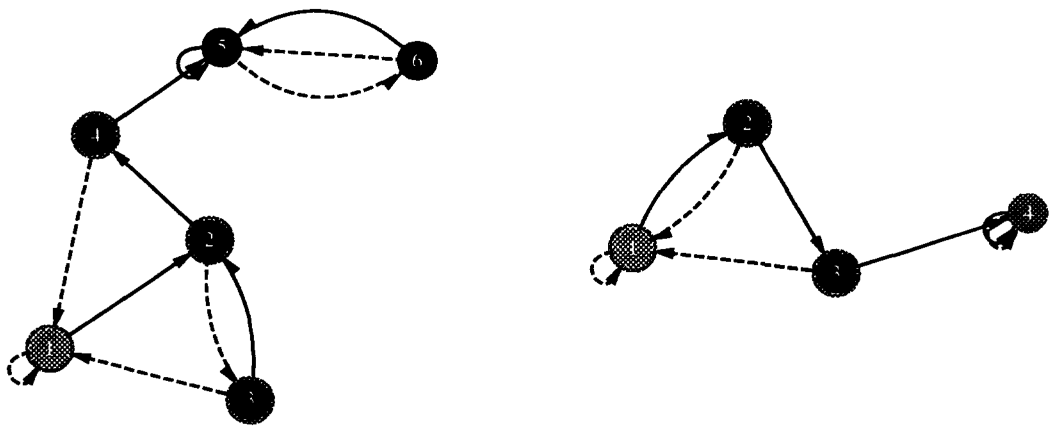

FIGURE 4. DFA minimization. (a) Extracted DFA and (b) its unique representation obtained through minimization. Both DFAs accept the same language, consisting of all strings which do not contain the substring "000".

If several DFAs are extracted with different quantization levels q_i, then one or more of the extracted DFA M_{q_i} may be consistent with the given training set, i.e., several DFA M_{q_i} may correctly classify the training set. Clearly, we need to make a choice between different consistent DFA. The heuristic for choosing a DFA will be discussed next.

4.3. Model Selection

Let M denote the unknown DFA and $L(M)$ the language accepted by M. By choosing a particular quantization level q_i, we extract a minimized finite-state automaton, which we consider to be a *hypothesis* M_{q_i}, for the grammar to be inferred. A DFA M is called a *consistent DFA* if it correctly classifies all strings of the training set; otherwise, it is called an *inconsistent* model of the unknown source grammar. Given a set of consistent hypotheses M_{q_1}, $M_{q_2}, \ldots,$ M_{q_Q}, we need criteria that allow us to choose the hypothesis that best approximates the unknown language $L(M)$. We refer to the process of choosing a DFA M_{q_i} *as model selection.* A possible heuristic for model selection would be to split a given data set into two disjoint sets (training and testing set), to train the network on the training set and to test the network's generalization performance on the test set. However, by disregarding a subset of the original data set for training, we may be eliminating

TABLE 2
DFA Generalization Performance

		Smallest Consistent DFA		Other Consistent DFAs	
Test Set	Training Set Size	Average μ	90% Confidence Interval	Average μ	90% Confidence Interval
Strings of length 1–15	1000	0.01%	0.00% $\leqslant \mu \leqslant$ 0.03%	0.18%	0.13% $\leqslant \mu \leqslant$ 0.23%
	700	0.10%	0.03% $\leqslant \mu \leqslant$ 0.16%	0.52%	0.45% $\leqslant \mu \leqslant$ 0.60%
	300	3.21%	2.00% $\leqslant \mu \leqslant$ 4.42%	4.93%	4.44% $\leqslant \mu \leqslant$ 5.43%
	100	38.19%	36.77% $\leqslant \mu \leqslant$ 39.62%	38.63%	38.19% $\leqslant \mu \leqslant$ 39.07%
	50	39.53%	38.68% $\leqslant \mu \leqslant$ 40.39%	38.54%	38.10% $\leqslant \mu \leqslant$ 38.99%
1000 random strings of length 100	1000	0.02%	0.00% $\leqslant \mu \leqslant$ 0.04%	2.00%	1.50% $\leqslant \mu \leqslant$ 2.49%
	700	1.35%	0.54% $\leqslant \mu \leqslant$ 2.16%	3.24%	2.81% $\leqslant \mu \leqslant$ 3.50%
	300	12.00%	8.47% $\leqslant \mu \leqslant$ 15.48%	19.84%	18.51% $\leqslant \mu \leqslant$ 21.18%
	100	38.52%	37.30% $\leqslant \mu \leqslant$ 39.74%	38.63%	38.17% $\leqslant \mu \leqslant$ 39.09%
	50	42.91%	41.73% $\leqslant \mu \leqslant$ 44.09%	42.23%	41.72% $\leqslant \mu \leqslant$ 42.75%
1000 random strings of length 1000	1000	0.02%	0.00% $\leqslant \mu \leqslant$ 0.06%	4.41%	3.25% $\leqslant \mu \leqslant$ 5.57%
	700	0.92%	0.22% $\leqslant \mu \leqslant$ 1.62%	6.79%	5.52% $\leqslant \mu \leqslant$ 8.06%
	300	12.39%	8.84% $\leqslant \mu \leqslant$ 15.94%	24.93%	22.42% $\leqslant \mu \leqslant$ 27.43%
	100	37.38%	36.47% $\leqslant \mu \leqslant$ 38.30%	40.82%	39.11% $\leqslant \mu \leqslant$ 42.53%
	50	41.70%	40.48% $\leqslant \mu \leqslant$ 42.93%	40.97%	40.42% $\leqslant \mu \leqslant$ 41.53%

The table shows some statistics (t-student) on the generalization performance of consistent DFAs on test sets containing strings of varying lengths (DFAs which were inconsistent with the training set were disregarded); the DFAs were extracted from trained networks with quantization levels $q = 2,\ 3, \ldots, 10$. The average performance and 90% confidence shows that the DFAs extracted with the smallest quantization levels have increasingly superior performance over the remaining consistent DFAs as the length of the test strings increases. This becomes the case even for networks that were trained on small data sets (100 strings) and thus necessarily learned a poorer model of the random 10-state DFA than the networks that were trained on larger data sets.

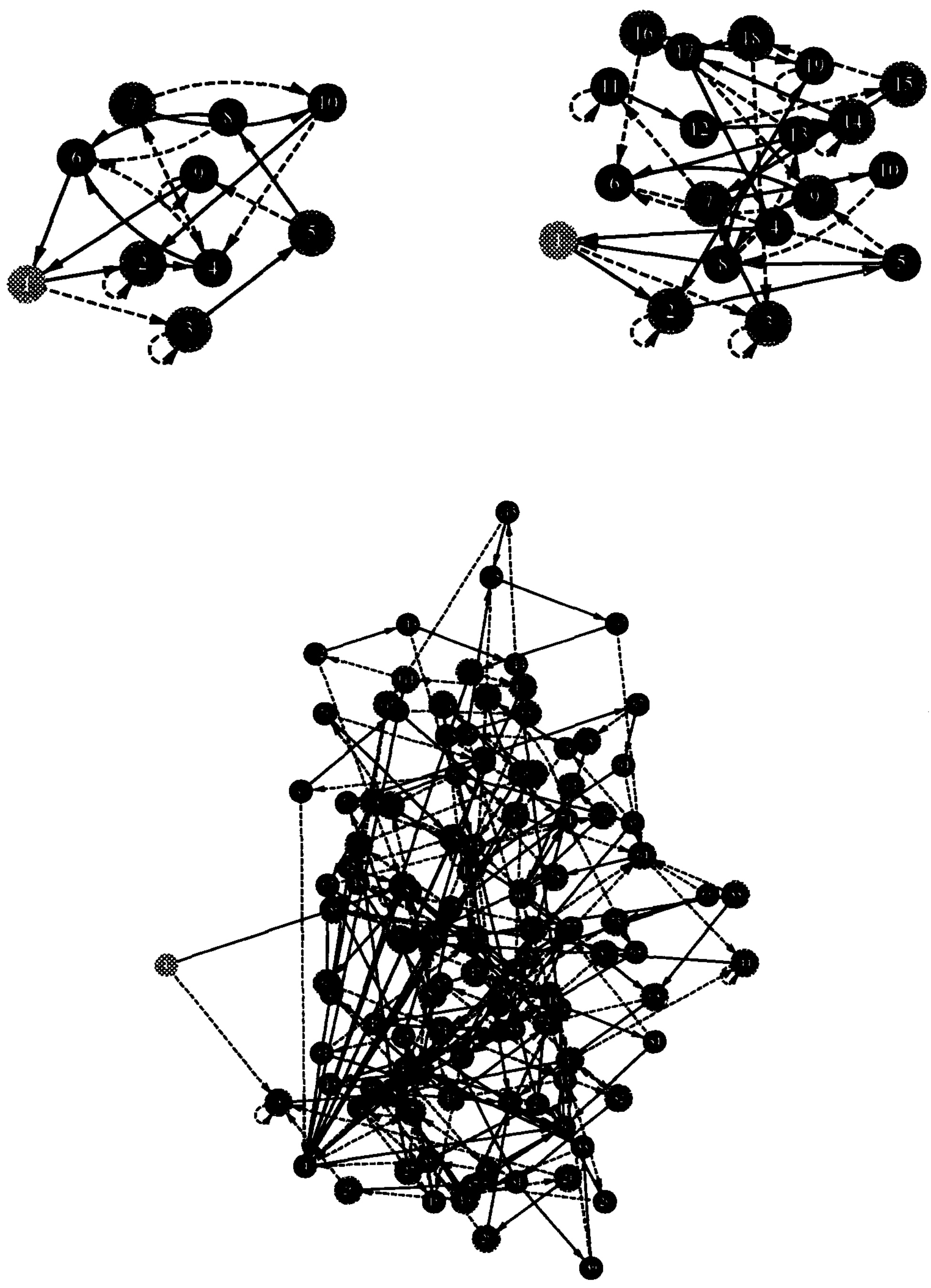

FIGURE 5. Model selection. Minimized DFAs extracted from a trained network with quantization levels (a) $q=3$, (b) $q=6$ and (c) $q=8$. The smallest DFA was used to generate the training set consisting of the first 1000 positive and negative example strings in alphabetical order. Thus, it correctly classifies all strings. DFAs M_6 and M_8 make generalization errors with M_6 showing a better generalization performance than M_8.

valuable data from the training set which would improve the network's generalization performance if the entire data set were used for training. Clearly, we wish to make a model selection based solely on simple properties of the extracted DFAs and not resort to a test set.

The minimized DFAs which were extracted from a trained network for quantization levels $q=3$, $q=6$ and $q=8$ are shown in Figure 5. All three DFAs were consistent with the training set, i.e., they correctly classified all strings in the training set. Which DFA best models the unknown source M?

To answer this question, we extracted DFAs with quantization levels $q=2,\ldots,10$ from the 25 trained networks for each of the five training sets and measured the DFAs generalization performances on the same three tests as before. In some cases, DFAs that were extracted with quantization levels 2 or 3 were inconsistent with a network's training set; these DFAs were discarded.

In all our experiments, the consistent, minimized DFAs that were extracted with the smallest quantization levels also had the fewest number of states. Following Occam's razor, we measured the generalization performances of these smallest consistent DFAs and compared them to the performances of the remaining DFAs. The results of a statistical analysis are shown in Table 2. We observe that the smallest consistent DFAs generally outperform the remaining consistent DFAs. In the case where DFAs were extracted from networks trained on only 50 strings, the smallest consistent DFAs are outperformed by the remaining consistent DFAs. However, this is statistically not significant since the confidence intervals for the DFAs average performance overlap.

These simulation results lead us to conjecture that the DFA M_q which best models the unknown DFA M is the smallest consistent DFA; M_q can be found by extracting DFAs $M_2, \ldots, M_{q-1}$, M_q in that order. The best model is the first consistent DFA M_q. We further hypothesize that there always exists a value q such that M_q is the smallest consistent DFA. Assuming that we succeed in extracting a DFA that is consistent with the training set, then such a smallest q exists by the well-ordering principle. Our intuition leads us to believe that large DFAs tend to overfit the given training data set. They do not capture the structure of the unknown source grammar well leading to poorer generalization performance.

5. CONCLUSIONS

A heuristic has been introduced that permits the inference of good grammars from trained networks. The rules of learned grammars are extracted in the form of deterministic finite-state automata (DFAs). The problem of rule extraction is reduced to a dynamical state exploration and cluster analysis in the N-dimensional output space of recurrent state neurons. Clusters of state neuron activations are found through a simple partitioning procedure, parameterized by a *partitioning* or *quantization level* q. Other methods such as hierarchical clustering or unsupervised learning algorithms could be used instead. Although different DFAs may be extracted for different values of the quantization level q, a standard minimization algorithm (Hopcroft & Ullman, 1979) can be applied yielding unique, minimal representations of the DFAs. Furthermore, these DFAs may be identical over ranges of the quantization level q.

We have observed that different DFAs of minimal representation can be extracted from a trained network depending on the parameter q. It is quite common that several extracted DFAs are consistent with the training set, i.e., several DFAs correctly classify the training set. Generalization tests on the different minimized DFAs have shown that there exists among the consistent DFAs one which shows the best generalization performance. Thus, it becomes necessary to perform a *model selection* among several DFA candidates. In our experiments, the smallest consistent DFA always outperformed the trained network and all other extracted DFAs. It was also always the DFA that was extracted with the smallest quantization level and that was consistent with the training set.

Our rule extraction algorithm depends on dividing the continuous state space of recurrent networks with sigmoidal discriminant functions into discrete partitions. Different partitionings may cause different, minimal DFAs to be extracted from a network which makes a model selection necessary. Two recently proposed methods avoid that problem.

Zeng et al. (1993) define a discrete state space prior to training rather than prior to the rule extraction. The sigmoidal discriminant function is still used for computing the weight updates. For string classification, the sigmoidal discriminant function $g(x)$ becomes a step function $h(x) = g(x_i)$ for $x_i \leqslant x \leqslant x_{i+1}$ where $g(x_i)$ is the original sigmoidal discriminant function. This yields q^N discrete state vectors in $[0, 1]^N$, each of which potentially corresponds to a DFA state.

Das and Mozer (1994) have proposed a self-clustering learning method which assumes that the unknown source grammar requires a finite number of discrete internal states that are corrupted due to inaccuracies in the weights. Their networks learn to recover these states with maximum a posteriori probability. These clusters change during the course of learning.

Both proposed methods report improved network generalization performance based on the stability of the internal DFA representation. However, they fail to address the issue of training performance, i.e., DFAs which are readily learned by networks without state space clustering may not be learned as easily by either of the two clustering methods.

Our results raise the issue improved performance be obtained from some recursive combination of rule extraction and rule insertion? We have developed a single algorithm which maps a (partial) description of a DFA into a *second-order* recurrent neural network by programming some of the weights accordingly (Omlin & Giles, 1992). The architecture of the network is defined by the amount of prior information about the otherwise unknown DFA. Significant learning time improvements were achieved by training networks with prior knowledge. By continuously inserting and extracting knowledge from a network, starting with little or no prior information, the size of the network would change after each rule insertion/training/rule extraction cycle and be determined by the current partial knowledge of the DFA, i.e., the extracted symbolic knowledge would control the growth and decay of the network architecture. Thus, this symbolically-guided training procedure could lead to faster training and better generalization performance.

REFERENCES

Angluin, D., & Smith, C. H. (1983). Inductive inference: theory and methods. *ACM Computing Surveys*, **15**, 236–269.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, **1**, 157–166.

Cleeremans, A., Servan-Schreiber, D., & McClelland, J. (1989). Finite state automata and simple recurrent networks. *Neural Computation*, **1**, 372–381.

Clouse, D. S., Giles, C. L., Horne, B. G., & Cottrell, G. W. (1994). Learning large DeBruijn automata with feed-forward neural networks. Technical Report CS94-398, Department of Computer Science and Engineering, University of California at San Diego, La Jolla.

Crutchfield, J. P., & Young, K. (1991). Computation at the onset of chaos. In W. H. Zurek (Ed.), *Proceedings of the 1988 Workshop on Complexity, Entropy and the Physics of Information* (pp. 223–269). Redwood City: Addison-Wesley.

Das, S., & Mozer, M. C. (1994). A united gradient-descent/clustering architecture for finite state machine induction. In J. W. Cowan, G. Tesauro & J. Alspector (Eds.), *Advances in Neural Information Processing Systems 6* (pp. 19–26). San Mateo: Morgan Kaufmann Publishers.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, **14**, 179–211.

Frasconi, P., Gori, M., Maggini, M., & Soda, G. (1991). A unified approach for integrating explicit knowledge and learning by example in recurrent networks. In *1991 IEEE INNS International Joint Conference on Neural Networks*. Seattle (pp. 811–816). Piscataway: IEEE Press.

Frasconi, P., Gori, M., Maggini, M., & Soda, G. (1995). Representation of finite state automata in recurrent radial basis function networks. *Machine Learning*, in press.

Fu, L. (1994). Rule generation from neural networks. *IEEE Transactions on Neural Networks*, **24**, 1114–1124.

Giles, C. L., Sun, G. Z., Chen, H. H., Lee, Y. C., & Chen, D. (1990). Higher order recurrent networks and grammatical inference. In D. S. Touretzky (Ed.), *Advances in Neural Information Systems 2* (pp. 380–387). San Mateo: Morgan Kaufmann Publishers.

Giles, C. C., & Omlin, C. W. (1992). Inserting rules into recurrent neural networks. In S. Kung, F. Fallside, J. A. Sorenson & C. Kamm (Eds.), *Neural networks for signal processing II, Proceedings of the 1992 IEEE Workshop* (pp. 13–32). Piscataway: IEEE Press.

Giles, C. L., & Omlin, C. W. (1994). Pruning recurrent neural networks for improved generalization performance. *IEEE Transactions on Neural Networks*, **5**(5), 848–851.

Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., & Lee, Y. C. (1992). Learning and extracting finite state automata with second-order recurrent neural network. *Neural Computation*, **4**, 393–405.

Giles, C. L., Horne, B. G., & Lin, T. (1995a). Learning a class of large finite state machines with a recurrent neural network. *Neural Networks*, **8**(9), 1359–1365.

Giles, C. L., Chen, D., Sun, G. Z., Chen, H. H., Lee, Y. C., & Goudreau, M. W. (1995b). Constructive learning of recurrent neural networks: limitations of recurrent cascade correlation and a simple solution. *IEEE Transactions on Neural Networks*, **6**(4), 829–836.

Gold, E. M. (1978). Complexity of automaton identification from given data. *Information and Control*, **37**, 302–320.

Goudreau, M. W., Giles, C. L., Chakradhar, S. T., & Chen, D. (1994). First-order vs second-order single layer recurrent networks. *IEEE Transactions on Neural Networks*, **5**(3), 511–513.

Hanson, S., & Burr, D. (1991). What connectionist models learn: learning and representation in connectionist networks. In R. Mammone & Y. Zeevi (Eds.), *Neural networks: theory and applications*. Boston: Academic Press.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation*. Redwood City: Addison-Wesley.

Hopcroft, J. E., & Ullmann, J. F. (1979). *Introduction to automata theory, languages and computation*. Reading, MA: Addison-Wesley.

Jim, K., Horne, B. G., & Giles, C. L. (1995). Effects of noise on convergence and generalization in recurrent networks. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems 7*, (pp. 649–656). Cambridge, MA: MIT Press.

Lee, Y. C., Doolen, G., Chen, H. H., Sun, G. Z., Maxwell, T., Lee, H. Y., & Giles, C. L. (1986). Machine learning using a higher order correlational network. *Physica D*, **22**, 276–306.

Manolios, P., & Fanelli, R. (1994). First order recurrent neural networks and deterministic finite state automata. *Neural Computation*, **6**(6), 1154–1172.

Omlin, C. W., & Giles, C. L. (1992). Training second order recurrent neural networks using hints. In D. Sleeman & P. Edwards (Eds.), *Proceedings of the Ninth International Conference on Machine Learning* (pp. 363–368). San Mateo: Morgan Kaufmann Publishers.

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning*, **7**, 227–252.

Siegelmann, H. T., & Sontag, E. D. (1992). On the computational power of neural nets. In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory* (pp. 440–449). New York: ACM.

Sun, G. Z., Chen, H. H., Giles, C. L., Lee, Y. C. & Chen, D. (1990). Connectionist push-down automata that learn context-free grammars. *Proceedings of the International Joint Conference on Neural Networks* (pp. 577–580). Hillsdale, NJ: Lawrence Erlbaum.

Tino, P., & Sajda, J. (1995). Learning and extracting initial mealy machines with a modular neural network model. *Neural Computation*, **7**(4), 882–844.

Towell, G. G., Craven, M. W., & Shavlik, J. W. (1990). Constructive induction using knowledge-based neural networks. In L. A. Birnbaum, & G. C. Collins (Eds.), *Proceedings of the Eighth International Machine Learning Workshop* (p. 213). San Mateo: Morgan Kaufmann Publishers.

Watrous, R. L., & Kuhn, G. M. (1992). Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, **4**, 406–414.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**, 270–280.

Zeng, Z., Goodman, R., & Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, **5**, 976–990.

NOMENCLATURE

$S_j^{(t)}$	output of state neuron j after time step t
$S_0^{(f)}$	output of the response neuron at the end of an input string
$\mathbf{S}$	network state vector
$I_k^{(t)}$	output of input neuron k at time step t
W_{ijk}	second-order weight from state neuron j and input neuron k to state neuron i
N, L	number of state and input neurons
Ξ_i	total input to state neuron i
$g()$	sigmoidal discriminant of each neuron
	A regular grammar $G = \langle S, N, T, P\rangle$ consists of:
	• start symbol S
	• set of non-terminal symbols N
	• set of terminal symbols T
	• set of production rules P
$L(G)$	the language generated by the regular grammar G
	A deterministic finite-state automaton (DFA) $M = \langle \sum, Q, R, F, \delta\rangle$ consists of:
	• alphabet $\sum$
	• set of DFA states Q
	• start state R
	• set of accepting F
	• state transition functions δ: $Q \times \sum \rightarrow Q$
$L(M)$	the (regular) language accepted by DFA M
q	quantization level
M_q	DFA extracted with quantization level q