Machine Learning, 23, 5–32 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Representation of Finite State Automata in Recurrent Radial Basis Function Networks

PAOLO FRASCONI paolo@mcculloch.ing.unifi.it
MARCO GORI marco@mcculloch.ing.unifi.it
MARCO MAGGINI maggini@mcculloch.ing.unifi.it
GIOVANNI SODA giovanni@mcculloch.ing.unifi.it
Dipartimento di Sistemi e Informatica, Università di Firenze
Via di Santa Marta 3 - 50139 Firenze - Italy

Editor: C. Lee Giles

Abstract. In this paper, we propose some techniques for injecting finite state automata into Recurrent Radial Basis Function networks (R^{2}BF). When providing proper hints and constraining the weight space properly, we show that these networks behave as automata. A technique is suggested for forcing the learning process to develop automata representations that is based on adding a proper penalty function to the ordinary cost. Successful experimental results are shown for inductive inference of regular grammars.

Keywords: Automata, backpropagation through time, high-order neural networks, inductive inference, learning from hints, radial basis functions, recurrent radial basis functions, recurrent networks

1. Introduction

The ability of learning from examples is certainly the most appealing feature of neural networks. In the last few years, several researchers have used connectionist models for solving different kinds of problems ranging from robot control to pattern recognition. Coping with optimization of functions with several thousands of variables is quite common. Surprisingly, in many practical cases, global or near-global optimization is attained also with non sophisticated numerical methods. For example, successful applications of neural nets for recognition of handwritten characters (le Cun, 1989) and for phoneme discrimination (Waibel et al., 1989) have been proposed which do not report serious convergence problems.

Some attempts to understand the theoretical reasons for the successes and failures of supervised learning schemes have been carried out which explain when such schemes are likely to succeed in discovering optimal solutions (Bianchini et al., 1994; Gori & Tesi, 1992; Yu, 1992), and to generalize to new examples (Baum & Haussler, 1989). These results give some theoretical foundations to learning from tabula rasa configurations, but unfortunately, the conditions they provide for optimal convergence and for generalization are quite limited in practice. Although these theoretical results open the doors to deal with more interesting practical problems, we believe that they also give a warning about the limitations of learning from tabula rasa in artificial neural networks.

As stated by Minsky and Papert, *"...significant learning at significant rate presupposes some significant prior structure. Simple learning schemes based on adjusting coefficients can indeed be practical and valuable when the partial functions are reasonably matched on the task,..."* (Minsky and Papert, 1988, p. 16).

Recently, Geman et al. (1992) have used the statistical viewpoint to highlight strengths and weaknesses of neural models. In that framework the use of nonparametric statistical inference leads them to formulate the *bias/variance dilemma.* They conclude that *"Important properties must be built-in or hard-wired, perhaps to be tuned later by experience, but not learned in any statistical meaningful way".*

In the attempt to relate the optimal convergence of learning algorithms using no prior knowledge with the generalization, Frasconi & Gori (1993) have found an *uncertainty principle* of learning stating formally that one should not expect both good convergence and generalization. When learning from tabula rasa in feedforward networks, measures can be given of both convergence and generalization that lead to conclude that they are like conjugate variables in Quantum Mechanics.

The ideas reported in this paper can be traced back to our belief in "biased models" for solving most meaningful practical problems. When assuming this viewpoint, neural modeling is no longer focused exclusively on learning, but also on the identification of significant architecture and weight constraints. In order to conceive models of this kind, one has to specify the kind of prior knowledge to be used. Quite a common situation is that this knowledge is more or less affected by uncertainty and, therefore, it should be embedded carefully into a connectionist model in order not to force artificial behavior. The uncertainty can be approached naturally by neural network learning schemes. Instead of developing representations from tabula rasa, the role of learning becomes that of refining prior knowledge. In this way, learning algorithms become tightly related to rule embedding.

Bearing in mind these general ideas, in this paper we consider prior knowledge given in terms of automata. This is related to our previous work on the subject, that is briefly reviewed in the next section together with similar approaches existing in literature. In order to deal with automata, we introduce particular networks, referred to as recurrent radial basis function networks, that are composed of two layers: a locally-tuned processing unit layer and a sigmoidal unit layer. Feedback connections are assumed from the sigmoidal unit layer (*state layer*) to the radial basis function layer. We show that, like second-order recurrent networks (Giles et al., 1992a), these networks are very well suited for dealing with automata. A theoretical analysis is given for proving that under certain weight constraints, the R^2BF networks behave exactly as automata.

A technique is also suggested for forcing the outputs of the state neurons to be as close as possible to boolean values. The learning scheme that we propose is based on adding a proper penalty function to the ordinary cost that favors the development of symbolic representations.

We evaluate the proposed architecture for inductive inference of regular grammars. Unlike tasks of isolated word recognition we had previously dealt with (Frasconi et al., 1991, 1995) we want the network to learn an automaton without giving any prior information on the state transition rules. It is worth mentioning that, also for this task, the role of the prior knowledge is of remarkable importance. Using our approach, we force the network to learn automata, thus restricting significantly the wide range of complex dynamic behavior that the recurrent networks can exhibit. Basically, the only prior knowledge we want to introduce into the network is a strong bias towards a *finite state* behavior of the network dynamics. This is achieved using the particular architecture and a penalty function on the state neurons. Unlike other approaches to grammatical inference using recurrent networks (Cleeremans et al., 1989; Elman, 1991; Pollack, 1991; Giles et al., 1992a), following our proposal the representations developed by the network are intrinsically of symbolic nature, and the automata states are actually associated with very "small" clusters in the network state space. This favors the successful extraction of automata from the network state space. Notice that it is also possible to embed some hints on state transitions into the network, thus adding explicit prior knowledge on a particular automaton.

The paper is organized as follows. In the next section, we briefly review related work and give some insights on the relationships of the topics discussed in this paper with our previous research. In section 3, we introduce the recurrent radial basis functions, while in section 4 we show that these networks behave as automata under some weight constraints. In section 5, a technique for forcing automata behavior is described and in section 6, successful results are presented concerning the application of the proposed networks and learning scheme to inductive inference of regular grammars.

2. Related Work

In the last few years many researchers have proposed different ways of introducing prior knowledge into artificial neural networks for specific tasks with successful results.

For example, le Cun (1989) has proposed the use of receptive fields in feedforward nets for applications to handwritten character recognition. The implementation of that concept results in special equality constraints on the weights. Perantonis & Lisboa (1992) have introduced a method for reducing the number of weights of a third-order network used for pattern recognition by imposing invariance under translation, rotation, and scaling. For incorporating the temporal properties of speech signals, Waibel et al. (1989) have proposed the Time Delay Neural Networks (TDNN), where proper weight constraints make it possible to deal effectively with the temporal nature of speech. In order to deal with phoneme recognition Bengio et al. (1992) have used special recurrent architectures, with only self-loop recurrent connections, to introduce a "forgetting behavior" into the network. This turns out to be useful, since only recent information is likely to affect phoneme recognition.

These papers, for specific problems and network architectures, report significant attempts to incorporate domain knowledge in terms of weight constraints into neural networks, which, in turn, leads to reduce the number of free parameters to learn.

A natural evolution of proposals based on specific tasks is that of looking for general methods to describe prior knowledge that can be integrated with artificial neural networks, and particularly with their powerful learning schemes. A general framework for proposing integration of prior knowledge with learning from examples has been suggested in (Towell et al., 1990; Towell and Shavlik, 1993; Shavlik, 1994).

They conceive such integration as a three-step process: insertion of the knowledge into the network, refinement by learning from examples, and extraction of the knowledge after learning refinement. In many cases the learning refinement may destroy the prior knowledge injected into the network. Rather than reviewing most significant approaches that have recently appeared in the literature, we focus our attention on methods for injecting automata into recurrent networks that are strictly related to what is proposed in this paper.

2.1. Finite automata and recurrent networks

Neural networks have been suggested for implementing automata a long time ago. The research in this field can be traced back to early works by McCulloch & Pitts (1943). The recent renewal of interest in Neural Networks has led several researchers to carry out detailed analyses on this and related subjects (Fogelman-Soulie et al., 1987; Goles & Martinez, 1990). The learning capability of many recently proposed models has been particularly stressed. The availability of new connectionist procedures and of more powerful computers have allowed many researchers to investigate the possibility of learning interesting sequential tasks from examples.

Instead of learning from tabula rasa, however, one should exploit the partial information available on the task. Learning from partial information is referred to as "learning from hints". Although the availability of that information does not overcome the theoretical limitations arising from computational complexity arguments (Abu-Mostafa, 1990), there is no doubt about the actual role of learning from hints in practice. Interesting "hints" for solving special problems have been proposed in (Al-Mashouq & Reed, 1991; Omlin & Giles, 1992). In particular, Omlin and Giles (1992) have proposed the learning of Finite State Automata using hints placed in a very elegant way into second-order recurrent networks. They have shown how to set up some of the weights to larger values rather than starting with small random initial values for all the weights.

Other researchers (Das & Mozer, 1994; Zeng et al., 1993) have devised appropriate architectures and learning schemes, for biasing the behavior of recurrent neural networks towards a finite state dynamics. These approaches are motivated by the difficulty of selecting a stable state dynamics (useful for automata) in the very rich set of complex dynamics that are possible in unconstrained neural networks.

Our first approach to introduce prior knowledge into recurrent networks is described in (Frasconi et al., 1991, 1995; Gori & Soda, 1993). It was inspired by problems of automatic speech recognition. Finite state automata were used for modeling the word lexical knowledge. These automata were nondeterministic (Hopcroft and Ullman, 1979) in that each state transition took place in an unspecified number of steps. The learning was responsible for discovering the optimal duration associated with these state transitions and for dealing with the uncertainty of the given prior knowledge. The main limitation of this model concerns the automaton injected into the network for which all the state transitions a part from self-loops are specified. Basically, the learning process cannot develop suitable automata from examples, but can only act on the number of steps needed to perform state transitions.

A major concern that should not be neglected in speech recognition, as well as in other high level perception tasks, is that prior knowledge is uncertain. As a consequence, any appropriate model should be able to discover additional rules not included in the prior knowledge model, and also to perform a sort of nonmonotonic process by changing the prior information when "enough" evidence comes from examples. In (Gori & Soda, 1993) we propose a model for integrating a *symbolic* with a *sub-symbolic* module into recurrent networks. The symbolic module is similar to that proposed in (Frasconi et al., 1995), while the sub-symbolic one acts like a connectionist glue for dealing with the uncertain nature of the prior knowledge. Moreover, a pruning scheme is suggested that favors the development of high level representations during the learning process.

Unfortunately, as already pointed out, the automata used in (Frasconi et al., 1991, 1995) have few degrees of freedom. This has motivated the research reported in this paper, where the main efforts are in forcing automata behavior into neural networks without specifying state transition rules. One of the objects of the research reported in this paper is to develop "K" modules for enriching the K-L models suggested in (Frasconi et al., 1991, 1995). Interestingly, as shown in section 6, the analysis on forcing automata behavior into R^{2}BF networks turns out to be very useful also for purely symbolic problems like inductive inference of regular grammars.

3. Recurrent Radial Basis Functions

In this section, we introduce the recurrent radial basis function networks (R^{2}BF) and define the formalism adopted throughout the paper. The architecture we consider is shown in Fig. 1. Like for first- and second-order recurrent networks, in R^{2}BF networks the learning takes place according to a supervised protocol. Basically, three entities need to be defined: a network $\mathcal{N}$, a learning environment $\mathcal{L}_e$ (set of data used for learning), and an error index E_T (Rumelhart et al., 1986).

- *Network* $\mathcal{N}$.

  The layers are identified by index l. We distinguish among the input layer ($l = 0$), the radial basis function (RBF) layer ($l = 1$), the state layer ($l = 2$) and the output layer ($l = 3$). For the sequence parsing tasks we consider, the output layer consists only of one sigmoidal neuron whose output is represented by $x_o(t)$.

  The number of neurons per layer is denoted by $n(l)$. Each neuron of layer l is referred to by its index $i(l)$, $i(l) = 1, \ldots, n(l)$. As already mentioned, two different kinds of neurons are considered, namely *sigmoidal processing units* and *locally-tuned processing units*. When the input vector at time t of a given sequence is presented to the input layer of the network, for each neuron we consider:

$a_{i(l)}(t)$	neuron $i(l)$'s activation
$x_{i(l)}(t)$	neuron $i(l)$'s output.

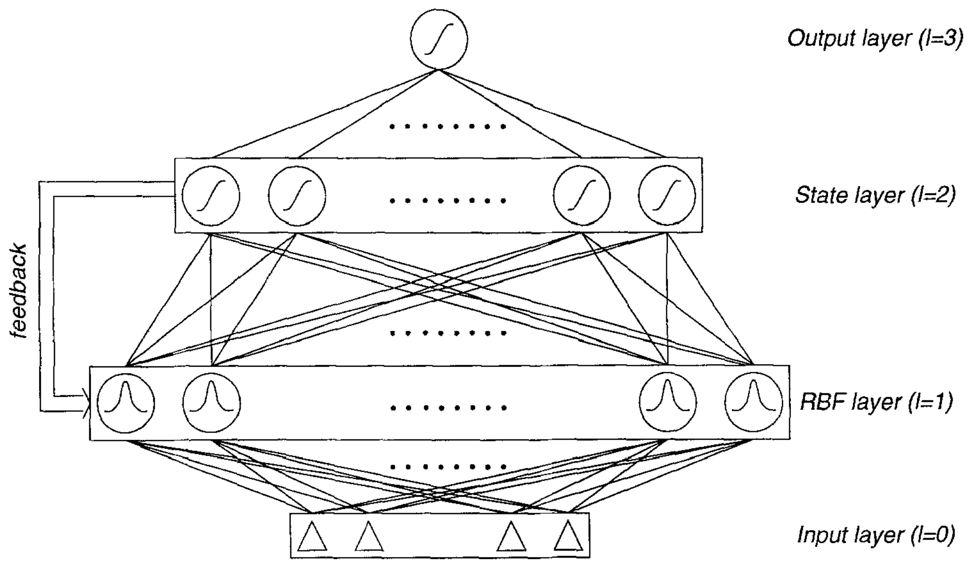


Figure 1. The Recurrent Radial Basis Function architecture.

Depending on the kind of neuron, the following processing is performed:

$$a_{i(1)}(t+1) = \frac{1}{\sigma^2_{i(1)}}\left[\sum_{j(0)=1}^{n(0)} (x_{j(0)}(t) - c_{i(1),j(0)})^2 + \sum_{j(2)=1}^{n(2)} (x_{j(2)}(t-1) - c_{i(1),j(2)})^2\right] = \frac{||[X_0(t), X_2(t-1)]' - C_{i(1)}||^2}{\sigma^2_{i(1)}} \tag{1}$$

$$a_{i(2)}(t) = w_{i(2)} + \sum_{j(1)=1}^{n(1)} w_{i(2),j(1)} x_{j(1)}(t) = W'_{i(2)} X_1(t) + w_{i(2)},$$

where $w_{i(l),j(l-1)}$ denotes the weight of the link between the neurons $i(l), j(l-1)$, and $X_l(t) \doteq [x_{1(l)}(t), \ldots, x_{n(l)}(t)]'$ represents the output vector associated with layer l. The vectors $C_{i(1)} \doteq [c_{i(1),1(0)}, \ldots, c_{i(1),n(0)}; c_{i(1),1(2)}, \ldots, c_{i(1),n(2)}]'$ are the centers of the radial basis functions and $\sigma_{i(1)}$ are the associated widths of the Gaussian functions [1].

The output of neuron $i(l)$ is related to the activation as follows:

$$x_{i(1)} = \tilde{f}(a_{i(1)}) = e^{-a_{i(1)}},$$
$$x_{i(2)} = f(a_{i(2)}) = \frac{1}{1+e^{-a_{i(2)}}}.$$

In the following, we will mainly refer to a special architecture, referred to as R^{2}BF1, which is characterized by a single feedback connection from each state unit to one RBF unit. For this particular connection scheme, the centers of the RBF units are represented as $C_{k(1)} \doteq [c_{k(1),1(0)}, \ldots, c_{k(1),n(0)}; c_{k(1),k(2)}]'$, being $k(1)$ the index of the RBF unit and $k(2)$ the index of the corresponding state neuron.

- *Learning Environment $\mathcal{L}_e$.*
The computational style considered in this paper is that of feeding the network on sequences of frames (*tokens*) beginning from a given initial state. A token $S_{T(q)}(q)$ with $q = 1, \ldots, Q$ is a sequence of $T(q)$ frames:

$$S_{T(q)}(q) \doteq \left\{ U(t,q) \in R^{n(0)}, \;\; t = 1, \ldots, T(q) \right\}.$$

The number of frames composing a given token "q" is referred to as the *token length* $T(q) \leq T_{max}$, being $T_{max} = \max_{1 \leq q \leq Q}\{T(q)\}$. The activation of each neuron is reset to its initial value after feeding the network with a token.
The learning process is based on a set of supervised tokens. They are collected into Q input/target pairs, where each token is associated with its class. Positive and negative examples are taken into account. Let d^-, $d^+ \in R$ be such that $[d^-, d^+] \subset [0,1]$ and define:

$$\mathcal{L}_e \doteq \left\{ \left(S_{T(q)}(q), d(q)\right), \; S_{T(q)}(q) \in \mathcal{S}_Q, \; d(q) \in \left\{d^-, d^+\right\}, \; q = 1, \ldots, Q \right\},$$

where $S_{T(q)}(q)$ is the input sequence, $d(q)$ is its corresponding target value for the output at $T(q)$, and $\mathcal{S}_Q$ is the token space.
According to the previous definition, the learning environment $\mathcal{L}_e$ can be partitioned into the following sets:

$$\begin{aligned} \mathcal{C}^+ &\doteq \{q \in \mathcal{L}_e \; : d(q) = d^+\}, \\ \mathcal{C}^- &\doteq \{q \in \mathcal{L}_e \; : d(q) = d^-\}. \end{aligned}$$

These sets collect the positive and the negative tokens of the learning environment.

- *Cost index.*
Given the pair $(\mathcal{N}, \mathcal{L}_e)$, the output-target data fitting is measured by means of the cost function

$$E(\mathcal{N}, \mathcal{L}_e) \doteq \sum_{q=1}^{Q} E_q = \sum_{q \in \mathcal{C}^+} \beta_+(x_o(T(q)) - d^+) + \sum_{q \in \mathcal{C}^-} \beta_-(x_o(T(q)) - d^-). \qquad (2)$$

where

$$\begin{cases} \beta_+(\alpha) = 0 & \text{if } \alpha \geq 0 \\ \beta_+(\alpha) > 0 \;\; \beta'_+(\alpha) < 0 & \text{if } \alpha < 0 \end{cases}, \quad \begin{cases} \beta_-(\alpha) = 0 & \text{if } \alpha \leq 0 \\ \beta_-(\alpha) > 0 \;\; \beta'_-(\alpha) > 0 & \text{if } \alpha > 0 \end{cases},$$

and " $\prime$ " stands for differentiation with respect to α. This threshold-LMS error has been introduced by Sontag & Sussman (1989). This cost does not penalize outputs "beyond" the target values. It is very well suited both for theoretical analyses and practical applications (Bianchini et al., 1994).

As for other recurrent networks (e.g.: first and second-order recurrent networks), the learning is based on the optimization of that function. The gradient of the cost can be computed by following approximate schemes like the one proposed by Elman (1990) [2], or exact schemes based on *time-unfolding* (see, e.g.: Rumelhart et al., 1986; Williams & Peng, 1990) or on the *forward propagation scheme* (Williams & Zipser, 1989; Kuhn et al., 1990). However, because of the constraints we impose on our architecture, we can relate the used learning procedure to that proposed by Moody for static radial basis function networks. Moody's hybrid learning scheme for static RBF (Moody & Darken, 1989) is significantly less demanding than Backpropagation from a computational point of view. The main reason of the success of such scheme is that the RBF centers are adjusted very efficiently with self-organization algorithms, and the upper level weights are subsequently adjusted simply by using LMS. In section 4, we will show that if we are interested in learning automata then the radial basis function centers in R^{2}BF networks can be fixed on the vertices of the boolean hypercube, and the learning can be restricted to the weights of the neurons of the state layer. This is somewhat related to Moody's hybrid scheme for static networks.

4. R^{2}BF Working as Automata

In this section, we explore the relationships between R^{2}BF and automata. We show that the R^{2}BF are very well-suited for expressing the automata next-state function, and give a set of weight constraints guaranteeing automata behavior.

4.1. *R^{2}BF as automata canonical form*

The recurrent network introduced in the previous paragraph is very well-suited for implementing automata. Let us assume that the automaton is given in terms of its next-state function in the following canonical form (*sum of products*) [3]:

$$s_i|_{t+1} = \sum_{j,k} m(S_j, I_k)|_t \quad \forall\ i = 1, \ldots, n \ , \tag{3}$$

where s_i is the i-th bit of the state binary code, S_j is the binary code of the j-th state, I_k is the code for the k-th input symbol, and $m(S_j, I_k)$ is the *minterm* associated with the pair (S_j, I_k) [4].

In order to implement automata by R^{2}BF networks, let us consider the equation for the activation of any state neuron. From equations (1) it follows that

$$a_{i(2)}(t+1) = \sum_{i(1)=1}^{n(1)} w_{i(2)i(1)} \prod_{i(2)=1}^{n(2)} \mathrm{e}^{-\frac{(x_{i(2)}(t) - c_{i(1)i(2)})^2}{\sigma^2_{i(1)}}} \prod_{i(0)=1}^{n(0)} \mathrm{e}^{-\frac{(x_{i(0)}(t) - c_{i(1)i(0)})^2}{\sigma^2_{i(1)}}} + w_{i(2)}. \tag{4}$$

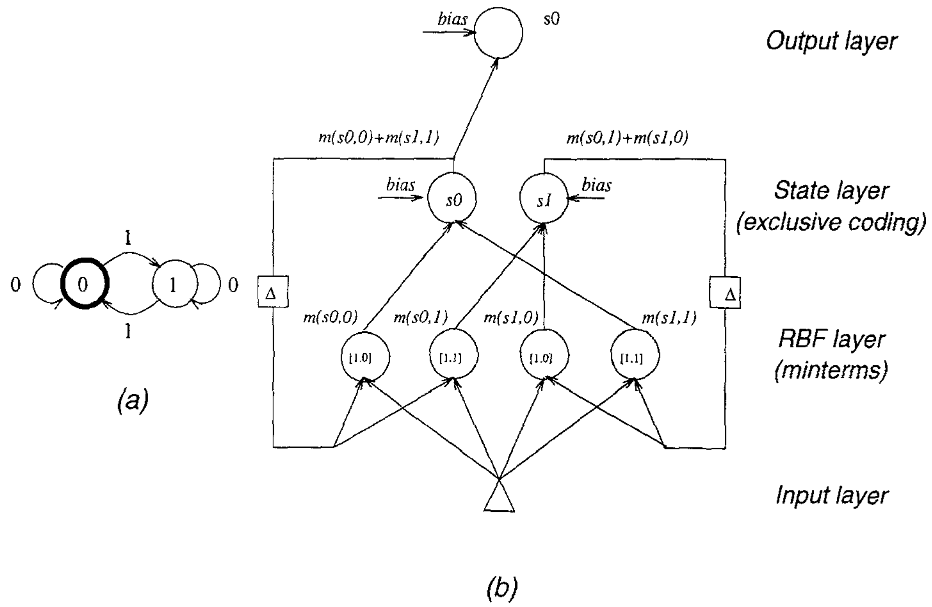


Figure 2. Implementation of a given automaton by means of a R^{2}BF network. The units of the RBF layer extract the minterms, while the sigmoidal neurons of the state layer perform the "or" of the connected inputs. (a) Parity automaton. (b) R^{2}BF network that recognizes parity; the corresponding center is shown inside each RBF unit.

We can associate a minterm $m(S_j, I_k)$ with each $i(1)$ and, therefore, with each radial basis function neuron. The product in equation (4) is close to 1 provided that both the state $x_{i(2)}(t)$ and the inputs $x_{i(0)}(t)$ are close to the centers $c_{i(1)i(2)}$ and $c_{i(1)i(0)}$. On the other hand, extracting minterms with locally-tuned processing units is a very simple task, since it suffices to locate their centers on the associated hypercube vertices and to assume "arbitrarily small" values for σ_i. Finally, the sigmoidal neurons must perform the "or" of the minterms using the $w_{i(2)i(1)}$ weights. It turns out that the R^{2}BF networks give a straightforward realization of the automata canonical form (3).

In Fig. 2, we give an example to show how a given automaton can be injected into a R^{2}BF architecture. We assume an exclusive coding for the two states of the parity automaton, so that two state neurons are required. The four radial basis function units realize the minterms required to detect all the possible state-input pairs, and, because of the exclusive coding of the states, the radial basis function units receive feedback from one state unit only. As motivated in section 4.2 and shown with experimental results in section 6, in many cases, this kind of architecture may be preferable to fully connected R^{2}BF. Obviously, we do not want to stress the capability of these networks to implement automata, but the natural way they incorporate the automata next-state function. When using the R^{2}BF for learning automata from examples, the previous analyses on the translation of the canonical form turns out to be very useful. In this case, of course, if no further information is available on the automaton, the minterms are unknown but, they are certainly located on the vertices of the hypercube. When choosing an architecture with a locally-tuned processing unit for each hypercube's vertex, we provide a very significant hint on the problem by forcing the network towards automata behavior. When considering automata with "many" states and "many" symbols, the number of radial basis functions, that is equal to the number of automata minterms, may become prohibitive. However, with additional prior knowledge on the automata, one can limit significantly the number of radial basis functions. The most general automata can have a number of minterms equal to the number of the states times the number of symbols. This number can be significantly reduced when considering that the number of radial basis functions that are really required is related to the number of arcs of the automata state diagram. Many interesting problems can be modeled by automata having *don't care conditions* in the next-state function and, consequently, the number of RBF units can be reduced significantly. For example this happens when we want to build a model for a particular word in speech recognition on the basis of a phoneme coder that performs prediction at the frame level. Because of the word lexical structure, we can limit the number of state transitions, which in turn reduces the complexity of the network without penalizing its capability to solve the task. Obviously, this is related to the insertion of knowledge at a higher level and is outside the scope of this paper (Frasconi et al., 1995; Frasconi et al., in press).

Remark 1. (R^2BF and high-order recurrent nets.) The "order" of a neural network refers to the dimensionality of the product terms in the weighted sum (Cover, 1965; Giles & Maxwell, 1987; Rumelhart et al., 1986; Minsky & Papert, 1988). Unlike feedforward networks where the order has a geometrical nature, in recurrent networks it could be also of temporal nature. The activation of a neuron can in fact depend on product terms of outputs at different times. When looking at equation (4), it turns out that the R^2BF have a very intriguing relationship with high-order [5] recurrent networks. Basically, the locally-tuned processing units act like *sigma-pi* neurons (Rumelhart et al., 1986) with inputs filtered by a Gaussian function. When the input to the network is a single boolean $\{0, 1\}$ and each RBF neuron receives one feedback connection only (R^2BF1), the recurrent radial basis function networks turn out to be strictly related to second-order recurrent networks (Giles et al., 1992a). Moreover, the R^2BF networks have an additional feature that makes them very suitable for dealing with automata: if we use small values of σ (Gaussian width), and locate the RBF centers on the hypercube vertices, then these networks react to boolean values only.

Remark 2. (The R^2BF output neuron.) In the R^2BF architecture proposed in the previous section, the state layer is connected to an output neuron by forward links with no delay (see Fig. 1). Since an automaton has commonly more than one accepting state, the network state layer is not suitable for a direct output computation, unless one neuron of the layer is reserved for that task. In order to reach the state coded by this neuron, a possible solution is that of adding one more *end* symbol to each string (Giles et al., 1992b; Watrous & Kuhn, 1992).

4.2. The automata weight space

In the previous section, we have discussed of automata realization by considering boolean values. A straightforward consequence of this theoretical assumption is that the weights connecting the radial basis functions to the sigmoidal neurons must be infinite. However, when considering logical values with thresholds, the R^{2}BF automata behavior is gained for a larger weight domain. Also in this case, it is possible to choose the widths of the Gaussian functions and the weights of the sigmoidal neurons in order to guarantee an automata behavior. A more formal statement of this fact can be given by defining hyperboxes where the state of the network is forced to evolve. Let $\rho^-, \rho^+ \in (0,1)$ be the logical thresholds. When dealing with a network having n states, an automata behavior is characterized by a state evolution that is forced in $\{[0,\rho^-) \cup (\rho^+,1]\}^n$.

Definition. [Boolean quantization function] Let

$$\rho^-, \rho^+ \in (0,1)$$

be two threshold values. The boolean-valued function Γ defined as

$$\begin{aligned} &\Gamma : [0,1] \longmapsto \{0,1\} : \\ &\Gamma(x;\rho^-,\rho^+) = \begin{cases} 0 & \text{if } x < \rho^- \\ 1 & \text{if } x > \rho^+ \end{cases} \end{aligned} \tag{5}$$

is called *a boolean quantization function of x with thresholds ρ^-, ρ^+.*

A boolean quantization function is not defined in $[\rho^-,\rho^+]$. For values belonging to that interval no boolean interpretation is given.

Definition. [Boolean-like AND functions] Let $\rho^-, \rho^+ \in [0,1)$ be the quantization thresholds. A real function

$$\mathcal{B}_\wedge : \{[0,\rho^-) \cup (\rho^+,1]\}^n \longmapsto [0,\rho^-) \cup (\rho^+,1] :$$

$$y = \mathcal{B}_\wedge(x_1, x_2, \ldots, x_n; \rho^-.\rho^+)$$

is called a boolean-like AND function if

$$\Gamma(y;\rho^-,\rho^+) = \bigwedge_{i=1}^{n} \Gamma(x_i;\rho^-,\rho^+). \tag{6}$$

A similar definition is given for OR functions:

Definition. [Boolean-like OR functions] Let ρ^+, $\rho^- \in [0,1)$ be the quantization thresholds. A real function

$$\mathcal{B}_\vee : \{[0,\rho^-) \cup (\rho^+,1]\}^n \longmapsto [0,\rho^-) \cup (\rho^+,1] :$$

$$y = \mathcal{B}_\vee(x_1, x_2, \ldots, x_n; \rho^-, \rho^+)$$

is called a boolean-like OR function if

$$\Gamma(y; \rho^-, \rho^+) = \bigvee_{i=1}^{n} \Gamma(x_i; \rho^-, \rho^+). \tag{7}$$

Similar definitions can be given for the NOT operator and for any minterm. These boolean-like functions can be computed both by single sigmoidal neuron and radial basis functions. As shown in the previous section, a natural solution turns out to be that of using the sigmoidal neurons for implementing the "or" and the radial basis functions for the minterms.

Let us begin considering the realization of the boolean-like OR function. Denote with $w_0, w_1, \ldots, w_n$ the weights of this neuron, being w_0 the bias. The following theorem gives a set of weight constraints guaranteeing the realization of the OR operator. See (Watrous et al., 1993) for a related result.

THEOREM 1 *Assume a vector of weights $w_i > 0, i = 1 \ldots n$. A sigmoidal neuron implements a boolean-like OR function with thresholds ρ^-, ρ^+ if the weights satisfy the following $n+1$ linear constraints:*

$$\begin{aligned} & w_0 + \sum_{i=1}^{n} w_i \rho^- \leq f^{-1}(\rho^-) \\ & w_0 + w_j \rho^+ \geq f^{-1}(\rho^+) \quad \forall j = 1 \ldots n. \end{aligned} \tag{8}$$

In particular, the weight vector defined as follows

$$\begin{aligned} w_1 &= w_2 = \cdots = w_n = \frac{f^{-1}(\rho^+) - f^{-1}(\rho^-)}{\rho^+ - n\rho^-} \\ w_0 &= -\frac{\rho^+ f^{-1}(\rho^-) - n\rho^- f^{-1}(\rho^+)}{\rho^+ - n\rho^-} \end{aligned} \tag{9}$$

satisfies inequalities (8).

Proof: We begin by observing that the relationship

$$\Gamma(y; \rho^-, \rho^+) = \bigwedge_{i=1}^{n} \Gamma(x_i; \rho^-, \rho^+) \tag{10}$$

is implied by the following two conditions:

$$\begin{cases} y < \rho^- & \text{if } x_i < \rho^- \quad \forall i, 1 \leq i \leq n \\ y > \rho^+ & \text{if } \exists i, 1 \leq i \leq n \quad : x_i > \rho^+. \end{cases} \tag{11}$$

Therefore, it suffices to prove that the hypotheses of the theorem imply (11). Since all the weights $w_i, i = 1 \ldots n$ are positive, if $x_i < \rho^-, \forall i = 1 \ldots n$, we have

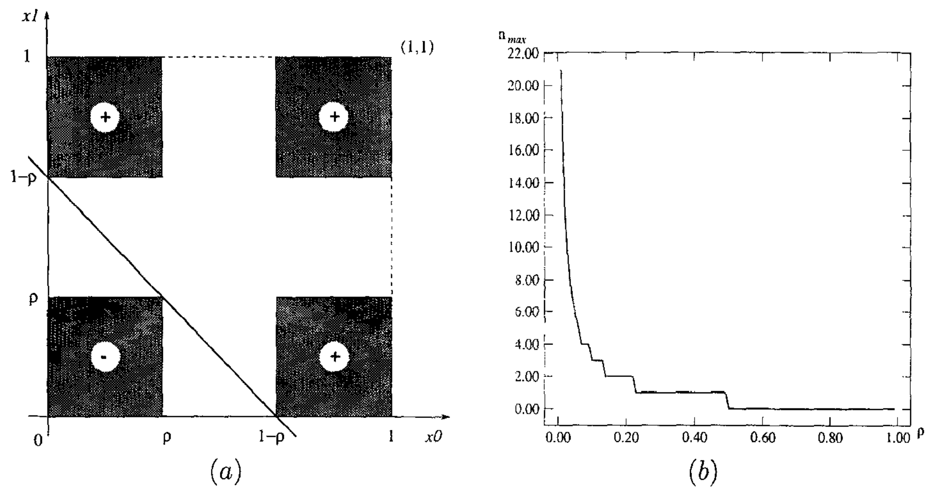


Figure 3. (a) Boolean quantization on a 2-dimensional input space for the OR function implemented by a sigmoidal neuron. The diagram is drawn for $\rho = 1/3$ (limit threshold value). (b) Maximum fan-in to a RBF unit for a given threshold value. Notice that the RBF units with more than two inputs turn out to be very saturated, thus compromising seriously the learning process.

$$f^{-1}(y) < w_0 + \sum_{i=1}^{n} w_i \rho^- \leq f^{-1}(\rho^-) \tag{12}$$

where the last inequality comes from the first of (8).

Similarly, if $\exists i : x_i > \rho^+$ then

$$f^{-1}(y) > w_0 + w_j \rho^+ \geq f^{-1}(\rho^+), \tag{13}$$

where once again the last inequality comes from (8). Finally, equations (9) can be easily verified by substitution with equal weights $(w_1 = w_2 = \cdots = w_n)$. ■

Remark 3. (Large networks and complex dynamics.) For the sake of simplicity, let us assume [6] $\rho^+ = 1 - \rho$ and $\rho^- = \rho$. It is easy to see that since the linear constraints (8) are satisfied with $w_i > 0$, for $i = 1 \ldots n$ (as required by Theorem 1), then the input threshold ρ must belong to the following interval:

$$0 < \rho < \frac{1}{n+1}. \tag{14}$$

This concept is clearly explained in Fig. 3a for $n = 2$. This condition arises from the domain we have chosen defining our boolean-like functions. Actually, it imposes a limitation on the fan-in of the sigmoidal neuron when the input threshold ρ is fixed. From a practical point of view, it also indicates that the automata behavior *becomes more and more unlikely for "large" networks.* This restriction could be overcome by splitting each neuron into more units, thus increasing the number of layers. In so doing,

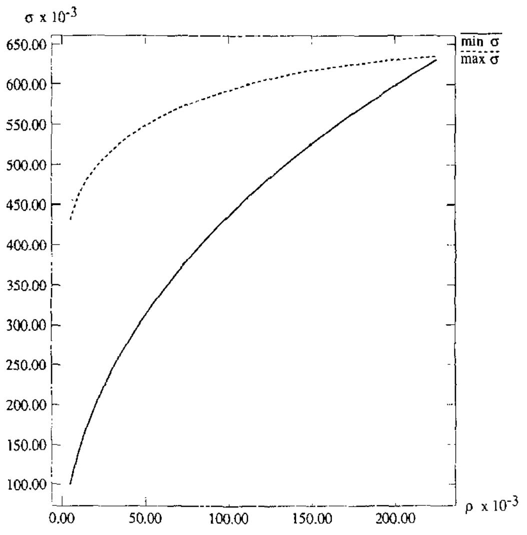


Figure 4. Maximum and minimum σ_i values of a RBF unit with two inputs (used in R^2BF1 networks) for a given threshold ρ. These bounds were found using Theorem 2.

however, the training could become more difficult because of the increased depth of the network. Results similar to those stated by Theorem 1 can be obtained for the radial basis function neurons devoted to implementing the automata minterms. The following theorem gives the constraints for the Gaussian width σ needed to meet the thresholding logical conditions.

THEOREM 2 *A locally-tuned processing unit "i" implements a minterm associated with the hypercube vertex C_i, provided that the center is located in C_i and the Gaussian width σ_i satisfies the conditions*

$$\frac{\rho\sqrt{n}}{\sqrt{ln(\frac{1}{1-\rho})}} \leq \sigma_i \leq \frac{1-\rho}{\sqrt{ln(\frac{1}{\rho})}} \tag{15}$$

Proof: The proof can be given exactly as for Theorem 1 by imposing the thresholding limit conditions. ■

Remark 4. (Large networks and complex dynamics.) Like for sigmoidal neuron, the previous theorem gives some interesting conclusions about the neuron fan-in (minterm order). Basically, the choice of σ_i becomes more and more restricted as the fan-in increases. Only one choice (*limit case*) is possible when the upper and lower bound in equation (15) becomes equal. For each ρ, in Fig. 3b, we can see what is the maximum value of the fan-in that is compatible with the minterm realization. Fig. 4 reports the diagram of maximum and the minimum σ_i values of a RBF unit with two inputs (used in R^{2}BF1 networks) as a function of the threshold ρ. These bounds (see equation (15)), ensure the minterm implementation and are very useful for setting up the $\sigma_{i(1)}$ parameters (see section 6).

For any R^{2}BF network, we can force the weights of sigmoidal and RBF neurons to satisfy the constraints (8) and (15), respectively. Let Ω_a be the admissible weight domain deriving from imposing such constraints. The theorems concerning the implementation of the OR function with sigmoidal neurons and any minterm with radial basis functions offer conditions for guaranteeing the automata realization. This is stated formally by the following theorem.

THEOREM 3 *Let us assume that a recurrent radial basis function network $\mathcal{N}$ performs a processing of symbolic inputs under the following conditions*

- *The initial state of $\mathcal{N}$ is in $\mathcal{D} \doteq \{[0, \rho^-) \cup (\rho^+, 1]\}^n$*
- *The network $\mathcal{N}$ has weights in the admissible domain Ω_a.*

Under these assumptions, the network evolution is guaranteed to kept the state in $\mathcal{D}$, that is $\forall t = 1, \ldots, \infty \Rightarrow X_2(t) \in \mathcal{D}$.

Proof: The proof can be given by induction on t. Let us assume that $X_2(t) \in \mathcal{D}$. Because of the network architecture, this vector is processed by the radial basis function layer. Once the centers of the radial basis functions are on the proper vertices of the boolean hypercube and the constraints (15) hold for all the neurons of the layer, Theorem 2 guarantees that $X_1(t) \in \mathcal{D}$, since the inputs are strings. Finally, if the weight constraints (8) are satisfied, because of Theorem 1, $X_1(t+1) \in \mathcal{D}$ follows. ∎

Under the hypotheses of Theorem 3, the network evolves from one "cluster of points" to another one. *Once these clusters have been labeled, this complex evolution can be associated with the next-state function of an automaton.*

The Remarks 3 and 4 give some very interesting insights about such behavior. It seems that the automata behavior becomes more unlikely with the increase of the network size. As a result, "large" networks exhibit naturally very complex dynamics, and it could be very difficult to force them to behave as automata. For this reason, the R^{2}BF1 networks seem more adequate for learning automata since the RBF units have a lower fan-in. In Fig. 3b we can see that RBF units with more than two inputs turn out to be very saturated, thus compromising seriously the learning process. Although these conclusions strictly hold for R^{2}BF networks, they are likely to apply to other recurrent networks as well. Based on experimental results obtained with second-order recurrent networks, Miller & Giles (1993) have recently observed that the drift from the automata behavior experimented for long sequences "seems strongly correlated with larger number of weights". For R^{2}BF networks, the results of this section give similar intuitions a theoretical foundation.

5. Forcing Automata Behavior into R^{2}BF

The difficulties of learning automata exactly have already been shown experimentally by numerous researchers. For most experiments of inductive inference of regular grammars (see the next section), the main problem is not that of learning the examples, but that of generalizing the automata behavior to other sequences of the grammar, particularly if they are very long. On the other hand, learning long sequences is very difficult because of the vanishing of the gradient during the Backpropagation through time (Bengio et al., 1994; Gori et al., 1994). Kolen (1994) has recently observed that "... significant state information can be buried deep within the system's initial conditions." Basically, the apparent automata behavior arising from the learning of sequences of relatively small length, may change to more complex dynamics for longer sequences.

To some extent, the techniques for extracting automata after learning (Cleeremans et al., 1989; Servan-Schreiber et al., 1991; Giles et al., 1992b; Watrous and Kuhn, 1992) are interesting attempts to overcome this problem. For example, Giles et al. (1992b) report explicitly that the extracted automaton can exhibit better performance than the recurrent network from which it has been extracted. However, an implicit assumption for a successful extraction of automata with clustering techniques is that the network state space is fairly well-separated in clusters. Unfortunately the network dynamics deriving from learning by example can be very complex and hardly approximable with automata.

Zeng et al. (1993) have recently overcome the problem pointed out by Kolen using a heuristic technique for learning in second-order recurrent networks with hard-limiting threshold activation functions. In so doing, there is no need to use clustering algorithms for extracting automata subsequently. However, as the authors point out, there is an increased computational burden associated with the proposed heuristic algorithm. Das & Mozer (1994) propose to include a clustering algorithm directly in the learning procedure and make it possible to put constraints on the number of state clusters developed by the network.

In this paper, we propose a very effective technique for approximating automata behavior that is based on forcing the network state space to be fairly well-separated in clusters. In addition to the hint associated with the automata minterms that lead to fix up the centers of the radial basis function, we suggest using a constraint that forces the outputs of the state neurons to be "high" and "low" (e.g.: 1,0).
In order to force these outputs during the input processing, we introduce the following *penalty function* to be added to the cost function (2)

$$P(\mathcal{N},\mathcal{L}) = \sum_{q=1}^{Q} \sum_{t=1}^{T(q)} \sum_{i(2)=1}^{n(2)} \max\left\{0, (x_{i(2)}(t,q) - \rho^-) \cdot (\rho^+ - x_{i(2)}(t,q))\right\} \qquad (16)$$

Basically, this function is null when the outputs of the network state neurons fall outside the interval $[\rho^-, \rho^+]$, during the input processing. This is a sort of *teacher forcing* [7] of the state neurons and leads to develop state representations clustered on the vertices of the hypercube.

The learning process can be carried out by optimizing $E + \lambda_p P$, where λ_p may affect significantly the rate of the learning process. We found remarkable advantages in optimizing the following index

$$V(\mathcal{N},\mathcal{L}) = E(\mathcal{N},\mathcal{L}) + \lambda_p \frac{P(\mathcal{N},\mathcal{L})}{1 + \exp(-(E_0 - E)/KT)}. \tag{17}$$

The aim of the learning is to find weights for which both the error E and the penalty function P become null. Using the index (17) with proper values for the constants E_0, K, and T, at the beginning the learning process decreases mainly the error E until it becomes very small. As long as the error decreases, the penalty function becomes more and more significant in the optimization. At the end, the learning is in fact focused on forcing the penalty to zero, thus favoring automata representations.

In order to understand the effect of optimizing the function (17), let us assume that $V = 0$ holds at the end of the learning process; consequently also the penalty function gets null ($P = 0$). From equation (16) we obtain $x_{i(2)}(t,q) \leq \rho^-$ or $x_{i(2)}(t,q) \geq \rho^+$, $\forall \ i(2) = 1,\ldots,n(2)$, $\forall \ t = 1,\ldots,T(q)$, $\forall q = 1,\ldots,Q$, that is $X_2(t) \in \mathcal{D}$. Notice that, unlike Theorem 3, the condition $V = 0$ does not imply automata behavior $\forall \ t = 1,\ldots,\infty$, but only for $t < T_{max}$. However, the optimization of equation (17) is a very straightforward method for forcing automata behavior.

6. R^{2}BF Networks for Inductive Inference of Regular Grammars

In this section, we report the experimental results obtained using R^{2}BF1 networks for inductive inference of regular grammars. The term "inductive inference" (Angluin & Smith 1983) denotes the process of hypothesizing a general rule from examples and seems to be a fundamental component of intelligent behavior. Many researchers have recently experimented the use of connectionist models for approaching simple problems of inductive inference (Cleeremans et al., 1989; Pollack, 1991; Giles et al., 1992a; Omlin & Giles, 1992; Watrous & Kuhn, 1992; Zeng et al., 1993). Following these researchers, we have tested extensively the R^{2}BF1 architecture on several tasks and in particular on all Tomita languages (Tomita, 1982) (see Table 1). These languages are based on the input symbols {0,1} that were coded as the real values 0.0 and 1.0.

6.1. Experimental set up

For each language we trained four R^{2}BF1 networks with an increasing number of state neurons (5,6,7,8) using Tomita's training set (Tomita, 1982). The R^{2}BF1 networks with n state neurons ($n(2) = n$) had $2n$ radial basis function units ($n(1) = 2n$), and one first order sigmoidal output neuron ($n(3) = 1$). The use of R^{2}BF1 networks has already been motivated in section 4.2. Architecture with full feedback from the state layer to the radial basis functions layer were also experimented. According to the theoretical expectations, we found their training more expensive and, therefore, we focused our attention on the R^{2}BF1 architecture only.

Table 1. Tomita grammars. The "Examples" column reports the number of examples contained in the original learning set (positive and negative strings, respectively).

Grammar	Rule	States	Examples
G1	1*	2	8-8
G2	(10)*	3	5-10
G3	no odd substring of 0s after odd substrings of 1s	5	12-12
G4	no 000s	4	10-9
G5	#01+#10 = 0 mod 2	4	9-11
G6	abs(#1-#0) = 0 mod 3	3	9-12
G7	0*1*0*1*	5	12-8

As described in section 4.1, the centers were set up in such a way that the locally-tuned processing units extracted minterms. As a consequence, these units were partitioned in two sets of n units with centers fixed in $[0.0, 0.9]$ and $[1.0, 0.9]$, respectively. Each unit in a set received the feedback connection from a different state neuron. The widths of the radial basis functions were initialized to 0.3 ($\sigma_{i(1)} = 0.3$) by considerations based on Fig. 4. Both the centers and the width of these units were kept fixed during training. The state layer and the radial basis function layer were fully-connected. These connections were trainable and randomly initialized in $[0.0, 1.0]$. Each sigmoidal unit had also a trainable bias initialized in $[-1.0, 0.0]$.
All the state units were connected to the output neuron with trainable weights randomly initialized in $[0.0, 1.0]$. The bias of this unit was initialized in $[-1.0, 0.0]$.
Before feeding the network with a sequence, all the state neurons were initialized to 0.0, apart from neuron 0 that was set up to 1.0.

Each network was trained by forcing automata representations as shown in section 5. The gradient of the error $E(\mathcal{N}, \mathcal{L})$ was computed by using the BPTT algorithm (Williams & Peng, 1990). The target values were 0.9 for grammatical and 0.1 for ungrammatical strings. A LMS-threshold function was used for the error computation (Sontag & Sussman, 1989).

The penalty function $P(\mathcal{N}, \mathcal{L})$ was chosen as shown in section 5 in order to penalize output values in $[0.1, 0.9]$. These constraints were managed as additional supervisions on the state neurons and, therefore, the gradient of P was still computed by BPTT. In all the experiments we found no problems in reaching $E(\mathcal{N}, \mathcal{L}) \simeq 0$, whereas sometimes it was hard to satisfy all the constraints. In these cases, after a fixed maximum number of epochs, the training was stopped without reaching the optimal solution also for the constraints.

6.2. *Automata extraction*

In the previous section we have seen that if we find a global minimum of the function (17) then the network state trajectory associated with the training set is constrained in $\mathcal{D}$. Said another way, the outputs of the network state layer can be quantized so that they can

Table 2. Automata Extraction Algorithm.

```
1. Compute all the state vectors for each sequence of the learning set.
2. $K \leftarrow 2$
3. Use the $k$-mean algorithm for partitioning the set of state vectors into K subsets;
4. If the distance of two centers is less than $d_c$ then $K \leftarrow K - 1$ else $K \leftarrow K + 1$ and go to
   step 3;
5. Compute the transition table:
   (A) Since each cluster corresponds with a state of the automaton, for each cluster use its
       center as network state at $t$ (or as initial condition), and from this state feed the network
       with one symbol.
   (B) Get the resulting state vector and find the cluster which contains this vector.
   (C) Repeat these two steps for all symbols and states.
6. Compute the initial state by finding the cluster containing the initial state of the network.
7. Compute the accepting states:
   (A) Initialize the state neurons with each cluster center (i.e. state of the automaton) and get
       the value of the output neuron.
   (B) If it is greater than 0.5 then the current cluster corresponds with an accepting state. [8]
```

be associated with the states of a finite state automaton. The extraction of the automaton becomes more difficult when the learning algorithm gets stuck in a local minimum of the function V. In these cases the points of the state trajectory are not necessarily clustered round the hypercube vertices, and consequently, the automata extraction is more involved.

Following other researchers (Cleeremans et al., 1989; Servan-Schreiber et al., 1991; Giles et al., 1992b; Watrous & Kuhn, 1992), an automaton was extracted from each trained network using the clustering algorithm based on k-mean reported in Table 2.

The parameter d_c represents the minimum tolerated distance between two cluster centers. For all the experiments on Tomita's languages d_c was set up to 0.6. As already pointed out in section 5, if the learning process ends with $V = 0$, then the points of the state trajectory are clusters round the hypercube vertices. Hence, the clustering by k-mean (step 3) is not strictly required. When ending the learning with sub-optimal solutions in function V, typical case is that in which $E = 0$ but some constraints are violated (i.e. the networks develops a non boolean state coding). In that case, it is possible to extract an automaton using a trial and error application of the previous algorithm. We can reduce the cluster distance d_c until we obtain an automaton that makes no errors on the learning set.

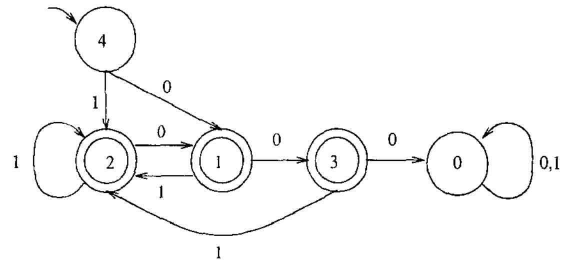
A state reduction algorithm can then be applied to the extracted automaton to get an equivalent machine with the minimum number of states.

Table 3. Training R^2BF1 networks on Tomita languages using Tomita training data. The results refer to the extracted automata. All the languages were learned exactly (Accuracy of 100% on Tomita's training set). The results of the table refer to all strings with length up to 12. The numbers marked with "*" indicate that the correspondent FSA was extracted from networks that did not reach the perfect constraint satisfaction ($P \simeq 0$).

Language	State neurons	FSA size	Accuracy
Tomita 1	5	3	**100.0**
	6	3	**100.0**
	7	3	**100.0**
	8	3	**100.0**
Tomita 2	5	4	79.8
	6	3	83.4
	7	4	79.8
	8	4	**99.6**
Tomita 3	5	11*	88.9
	6	9*	**93.3**
	7	9	85.1
	8	8*	79.3
Tomita 4	5	5	**100.0**
	6	5	96.0
	7	8	57.6
	8	6	58.5
Tomita 5	5	10*	**78.1**
	6	9	46.9
	7	8	58.6
	8	9	47.0
Tomita 6	5	5	**70.1**
	6	12*	47.2
	7	11*	46.4
	8	10	46.4
Tomita 7	5	5	**77.2**
	6	10	44.4
	7	8	75.4
	8	7	76.6

6.3. Experimental results

The experimental results obtained on all Tomita's languages are reported in Table 3. The second column specifies the size of the network (i.e. the number of state neurons). The third one reports the number of states of the extracted and minimized automaton (the networks for which the constraints were not satisfied are marked with "*"). The fourth column summarizes the performance of the extracted automaton (recognition rate) on a test set containing all the strings with length up to 12 (8190 strings). The extracted automaton always performs perfectly on the learning set. In the cases where the constraints



State	Code	Standard dev.
0	00000	0.068
1	11111	0.067
2	11100	0.077
3	01011	0.009
4	10000	0

(*a*) (*b*)

Figure 5. Learning Tomita 4 language with a R^{2}BF1 network of size 5. Perfect generalization was attained using Tomita's training set. (a) Extracted automaton. (b) State coding with the corresponding standard deviation of the output distribution of the state layer neurons.

were not satisfied ($P > 0$), the learning process was clearly trapped in a local minimum, but the extraction of the FSA was still possible.

The experimental results show that the task of learning these grammars from the small learning sets proposed by Tomita is quite hard. Perfect generalization was obtained for Tomita 1 and 4 languages. For the other languages, all strings of the learning set were correctly recognized but no perfect generalization was obtained. For a comparison with second-order networks see (Watrous & Kuhn, 1992).

Fig. 5a shows the automaton extracted from the network of size 5 that recognized successfully Tomita 4 language. Fig. 5b reports the state coding of the extracted automaton and the standard deviation of the associated clusters. The extracted automaton is minimal apart from state 4, that acts as the initial state and can easily be removed.

For Tomita 2 and Tomita 7 languages we extracted two automata that had the same dimension as the minimal machines which recognize the languages (see Fig. 6). These automata succeeded in classifying the learning set but did not reach perfect generalization. As pointed out in (Angluin & Smith, 1983), this is quite common in problems of inductive inference.

6.4. Clustering in the state space of R^{2}BF

Some more experiments were performed on Tomita 4 language in order to investigate in more detail the capabilities of R^{2}BF networks and the associated learning technique. We were interested in comparing R^{2}BF and second-order recurrent networks in terms of the developed state space representations. For this reason, we also trained second-order recurrent networks with the architecture described in (Pollack, 1991; Giles et al., 1992b; Watrous & Kuhn, 1992), on Tomita 4 language. The state space trajectory obtained

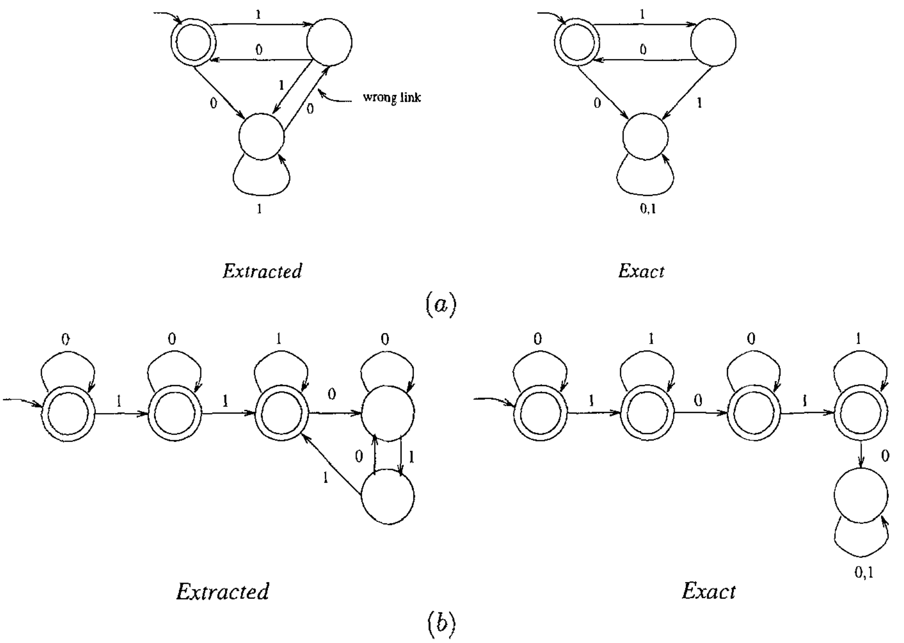


Figure 6. Examples of minimal extracted machines from Tomita's training set which do not attain perfect generalization: (a) Tomita 2 language, automaton extracted from size 6 R^2BF1 network. (b) Tomita 7 language, automaton extracted from size 5 R^2BF1 network.

while feeding this network with the input sequences is reported in Fig. 7. The network dynamics is quite complex with respect to the simple automaton associated with Tomita 4 grammar. We extracted an automaton from this network using our automata extraction technique. We obtained an automaton having 12 states, using a cluster distance threshold d_c equal to 0.2. The neural network had a slightly worse performance (57.1%) than the extracted automaton (58.3%).

Fig. 8a shows the state space trajectories for a R^2BF1 network of size 2 trained without constraints on Tomita 4 language. The diagram reveals the presence of six quite large clusters.

The diagram of Fig. 8b shows the effect of the constraints on the previous network. Only 4 small clusters appear which correspond with the 4 states of the automaton extracted using our automata extraction technique. The performance of the constrained network was better than that of the unconstrained one (100% v.s. 96.0%), while the accuracy we found with a second-order network in the same conditions was significantly lower (57.1%). However, we believe that automata extraction techniques on second-order networks, like that reported in (Giles et al., 1992b), may improve the performance significantly, while the extraction of automata from our networks can only change the performance slightly. This is due to a more accurate approximation of automata behavior of the R^2BF networks that is clearly shown in Fig. 8.

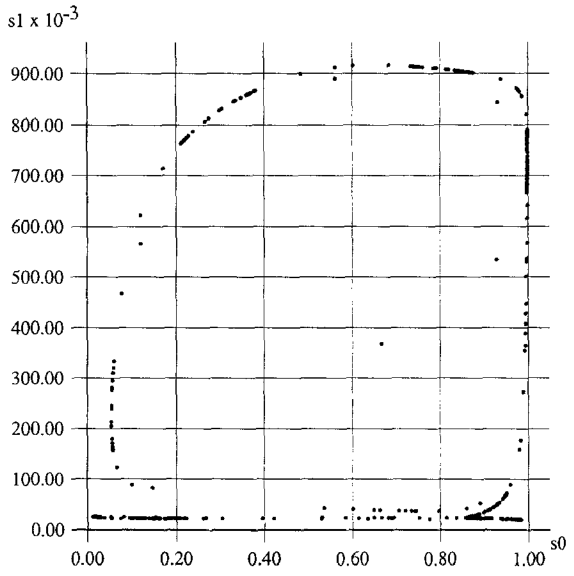


Figure 7. Space state trajectory for a second-order network trained on Tomita 4 language.

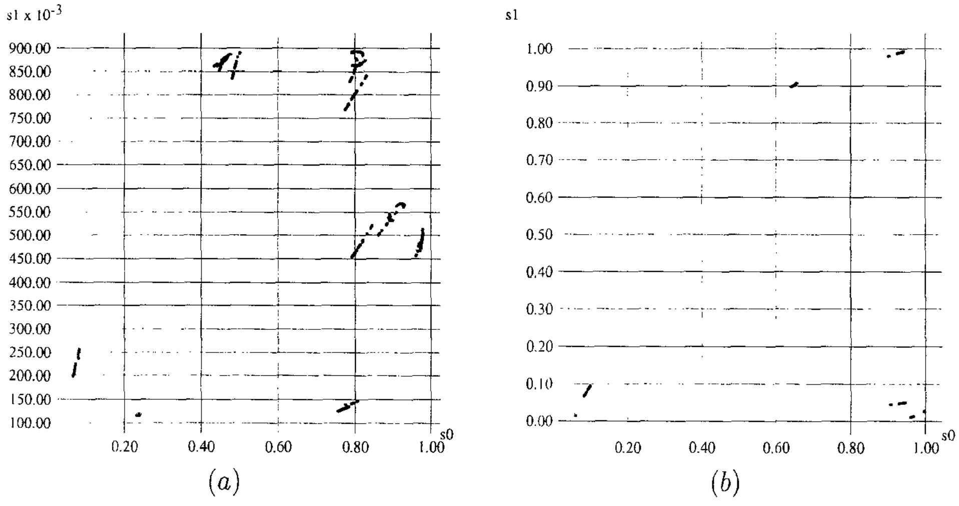


Figure 8. State space trajectories for a R^{2}BF1 network of size 2 trained on Tomita 4 language. (*a*) Unconstrained network. (*b*) Constrained network. In this experiment, although the automata constraints on the training set were not satisfied completely (see the "spurious" cluster with coordinates (0.64,0.90)), forcing automata with the penalty function had the remarkable effect of reducing significantly the variance of the distribution.

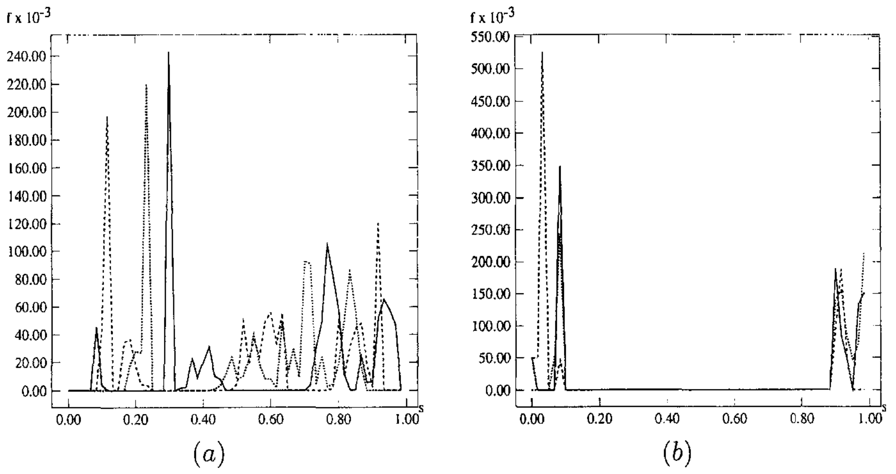


Figure 9. Distribution of the outputs of the state neurons for a R^2BF1 network of size 3 trained on Tomita 4 language. (a) Unconstrained network. (b) Constrained network. In this case, forcing automata behavior has not only the effect of reducing the variance of the distribution, but also that of clustering all the outputs on the vertices of the hypercube (global optimization was performed on V).

Another example of well separated regions created using R^2BF networks for learning Tomita 4 language is given in Fig. 9. It reports the distribution of the outputs of the state neurons for a network with $n(1) = 3$. According to the theoretical expectations, in the first case (unconstrained network) the distribution spans the whole interval, whereas in the second one (constrained network) no value falls into the "forbidden" area ($[\rho^-, \rho^+]$). This is because with this network the learning ended with $V = 0$.

6.5. *Experiments with a larger learning set*

All the experiments reported so far refer to learning with Tomita's training set. The evaluation of techniques for inductive inference however, must be very careful. Even human beings might be very annoyed to find that their supposed rule, working perfectly on the examples, was not the one the test maker had intended. As pointed out in (Gold, 1967; Angluin & Smith, 1983), the basis of this annoyance is that there are infinitely many more or less plausible rules for generating different sequences, and without more constraints it is impossible to establish whether or not the rule found on the learning set is the one the test maker had intended. For this reason, we trained R^2BF1 networks with 5 state neurons using an extended learning set for all the languages for which we did not obtain a perfect generalization. Tomita's training sets were incremented with the 5% of all the strings with lengths from 1 up to 10. These strings were randomly chosen. We adopted an incremental learning strategy, so that new strings were added to the learning set only when the previous ones had been exactly learned.

Using these learning sets we were able to learn exactly all the languages and to attain perfect generalization on all Tomita's languages.

7. Conclusions

In this paper, we have proposed some techniques for forcing automata behavior into recurrent radial basis function networks. This research is strictly related to our previous work on injecting prior knowledge into recurrent networks for automatic speech recognition (Frasconi et al., 1991, 1995; Gori & Soda, 1993). The paper has proposed some novel results that can be summarized as follows.

First, we have shown that the R^{2}BF networks are very well-suited for dealing with automata and we have given a very useful hint on the location of the radial basis function centers. The centers are in fact related to the minterms of the canonical form of the next-state function. This initialization of the centers is somewhat related to the hybrid learning scheme suggested by Moody & Darken (1989) for static radial basis functions. Interestingly, we have also shown that the R^{2}BF are closely related to high-order recurrent networks.

Second, in addition to the hint on the center location, we have proven that, under proper linear weight constraints, the R^{2}BF's dynamics can be described in terms of automata by associating symbolic states to clusters in the network state space. An implication of this theoretical analysis is that the admissible weight space becomes increasingly "small" with the size of the network, thus making the learning process very difficult for "large" networks. This conclusion, derived for R^{2}BF networks, is likely to hold also for other recurrent networks, and suggests that "large" networks have complex dynamics that may be difficult to approximate with automata. This also motivates the choice of R^{2}BF1 networks for the experimental application to inductive inference, since these networks are based on RBF units with two inputs only.

Third, we have given an approximate technique for forcing automata behavior into R^{2}BF networks, that is based on imposing that the outputs of the state neurons are "close" to boolean values. This constraint can be implemented by a proper penalty function that has the effect of changing the error function to optimize. In so doing, the network acts exactly as an automaton, at least on the training set.

Finally, we have experimented the effectiveness of the proposed theory for problems of inductive inference of regular grammars. There were two main reasons for choosing similar problems. First, they allow us to asses the capability of learning any finite state automaton, that was the main limitation of our previous approach (Frasconi et al., 1995, Frasconi et al., 1991). Second, many researchers have recently faced this problem using connectionist models (Cleeremans et al., 1989; Elman, 1991; Pollack, 1991; Servan-Schreiber et al., 1991; Giles et al., 1992b; Omlin & Giles, 1994; Watrous & Kuhn, 1992). In our experiments, the training set of Tomita's languages (Tomita, 1982) was always learned exactly and the results on a test composed of strings with length up to 12 (using Tomita's training set for learning) were very promising. For Tomita 1 and Tomita 4 languages we obtained a perfect generalization using the small Tomita sets only. When slightly incrementing these training sets, the trained networks attained perfect generalization for all Tomita's languages. The basic feature of the proposed network and learning scheme, that is the development of state representations distributed in "small" clusters, seems to be the main reason of these successful results.

Acknowledgments

We thank Yoshua Bengio, Raymond L. Watrous and the anonymous reviewers for their fruitful comments and suggestions on an earlier draft of this paper. This research was supported by MURST 40% and CNR under grant no. 93.052.40.74.

Notes

1. The radial basis functions considered in this paper are based on Gaussian functions as proposed in (Moody & Darken, 1989).
2. The approximation arises from the truncation of the true gradient to the previous time step. This may prevent the learning process from finding long-term dependencies.
3. We deal with automata acting as recognizers and, therefore, their complete specification can be given by the next-state function and the set of accepting states.
4. This minterm produces a *high* value only when its inputs match the codes of S_j and I_k, i.e. it decodes the particular binary configuration produced by the concatenation of the codes of S_j and I_k (Mano, 1988).
5. The order refers to product terms of outputs and inputs at the same time (geometrical concept).
6. We will keep this assumption also in the remainder of the paper.
7. Notice that in literature, the term "teacher forcing" has been used to refer to a different technique (Williams & Zipser, 1989).
8. This step must be modified if the output neuron receives connections from the network inputs. This happens with the second-order architectures proposed in (Pollack, 1991; Giles et al., 1992b; Watrous & Kuhn, 1992). In this case the output neuron must be considered as an additional component of the state vector and the accepting states are characterized by the fact that this component is greater than 0.5.

References

Abu-Mostafa, Y. S. (1990). Learning from hints in neural networks. *Journal of Complexity*, 6:192–198.

Al-Mashouq, K. A. and Reed, I. S. (1991). Including hints in training neural nets. *Neural Computation*, 3(4):418.

Angluin, D. and Smith, C. H. (1983). Inductive inference: Theory and methods. *ACM Computing Surveys*, 15(3):237–269.

Baum, E. B. and Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1(1):151–160.

Bengio, Y., Gori, M., and Mori, R. D. (1992). Learning the dynamic nature of speech with back-propagation for sequences. *Pattern Recognition Letters*, 13(5):375–385. Special issue on Artificial Neural Networks.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. Special Issue on Dynamic Recurrent Neural Networks.

Bianchini, M., Gori, M., and Maggini, M. (1994). On the problem of local minima in recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(2):167–177. Special Issue on Dynamic Recurrent Neural Networks.

Cleeremans, A., Servan-Schreiber, D., and McClelland, J. L. (1989). Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381.

Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14:326–334.

Das, S. and Mozer, M. C. (1994). A unified gradient-descent/clustering architecture for finite state machine induction. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Neural Information Processing Systems 6*, pages 19–26.

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

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2/3):195–226. Special issue on Connectionist Approaches to Language Learning.

Fogelman-Soulie, F., Robert, Y., and Tchuente, M. (1987). *Automata networks in computers science.* Manchester University Press.

Frasconi, P. and Gori, M. (1993). Multilayered networks and the C-G uncertainty principle. In *SPIE International Conference, Science of Artificial Neural Networks*, pages 396–401, Orlando, Florida.

Frasconi, P., Gori, M., Maggini, M., and Soda, G. (1991). A unified approach for integrating explicit knowledge and learning by examples in recurrent networks. In *Proceedings of IEEE-IJCNN91*, volume I, pages 811–816, Seattle WA.

Frasconi, P., Gori, M., Maggini, M., and Soda, G. (1995). Unified integration of explicit rules and learning by example in recurrent networks. *IEEE Transactions on Knowledge and Data Engineering, 7.*

Frasconi, P., Gori, M., and Soda, G. (in press). Recurrent neural networks and prior knowledge for sequence processing: a constrained nondeterministic approach. *Knowledge-based Systems.*

Geman, S., Bienenstock, E., and Dourstat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58.

Giles, C. L. and Maxwell, T. (1987). Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, 26(23):4972–4978.

Giles, C. L., Miller, C. B., Chen, D., Chen, H.H., Sun, G.Z, and Lee, Y. C. (1992a). Extracting and learning an unknown grammar with recurrent neural networks. In Moody, J., Hanson, S., and Lippmann, R., editors, *Advances in Neural Information Processing Systems 4*, pages 317–324, San Mateo CA. Morgan Kauffman Publishers.

Giles, C. L., Miller, C. B., Chen, D., Sun, G. Z., Chen, H. H., Sun, G.Z., and Lee, Y. C. (1992b). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10:447–474.

Goles, E. and Martinez, S. (1990). *Neural and Automata Networks.* Kluwer Academic Publishers, Dordrecht, Boston, London.

Gori, M., Maggini, M., and Soda, G. (1994). Scheduling of modular architectures for inductive inference of regular grammars. In *Proceedings of the workshop on Combining Symbolic and Connectionist Processing, ECAI '94*, pages 78–87, Amsterdam.

Gori, M. and Soda, G. (1993). Projecting sub-symbolic onto symbolic representations in artificial neural networks. In Torasso, P., editor, *Lecture Notes in Artificial Intelligence, Advances in Artificial Intelligence*, pages 84–89.

Gori, M. and Tesi, A. (1992). On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):76–86.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading MA.

Kolen, J. F. (1994). Recurrent networks: State machines or iterated function systems? In Mozer, M. C., Smolensky, P., Touretzky, D. S., Elman, J. L., and Weigend, A. S., editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 203–210, Hillsdale NJ. Erlbaum.

Kuhn, G., Watrous, R. L., and Ladendorf, B. (1990). Connected recognition with a recurrent network. *Speech Communication*, 9:41–49.

le Cun, Y. (1989). Generalization and network design strategies. In Pfeifer, R., Schreter, Z., Fogelman, F., and Steels, L., editors, *Connectionism in Perspective*, pages 143–155, Amsterdam. Elsevier. Proceedings of the International Conference Connectionism in Perspective, University of Zürich, 10. – 13. October 1988.

Mano, M. M. (1988). *Computer Engineering, Hardware Design.* Prentice-Hall.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Miller, C. B. and Giles, C. L. (1993). Experimental comparison of the effect of order in recurrent neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):849–872. Special Issue on Applications of Neural Networks to Pattern Recognition.

Minsky, M. L. and Papert, S. A. (1988). *Perceptrons - Expanded Edition.* MIT Press, Cambridge.

Moody, J. and Darken, C. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294.

Omlin, C. W. and Giles, C. L. (1992). Training second-order recurrent neural networks using hints. In Sleeman, D. and Edwards, P., editors, *Proceedings of the Ninth International Conference on Machine Learning*, pages 363–368, San Mateo CA. Morgan Kaufman Publishers.

Omlin, C. W. and Giles, C. L. (1994). Constructing deterministic finite-state automata in sparse recurrent neural networks. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'94)*, pages 1732–1737.

Perantonis, S. J. and Lisboa, P. J. G. (1992). Translation, rotation, and scale invariant pattern recognition by high-order neural networks and moment classifiers. *IEEE Transactions on Neural Networks*, 3(2):241–251.

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning*, 7(2/3):196–227. Special issue on Connectionist Approaches to Language Learning.

Rumelhart, D. E., Hinton, G. E., , and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, volume 1: Foundations, chapter 8, pages 318–362. MIT Press, Cambridge.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. (1991). Graded state machines: the representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 7(2/3):161–194. Special issue on Connectionist Approaches to Language Learning.

Shavlik, J. W. (1994). Combining symbolic and neural training. *Machine Learning*, 14(3):321–331.

Sontag, E. D. and Sussman, H. J. (1989). Backpropagation separates when perceptrons do. In *Proceedings of the International Joint Conference on Neural Networks*, volume I, pages 639–642, Washington DC. IEEE Press.

Tomita, M. (1982). Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference*, pages 105–108, Ann Arbor MI.

Towell, G. G. and Shavlik, J. W. (1993). The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101.

Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 861–866, Boston MA.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339.

Watrous, R. L. and Kuhn, G. M. (1992). Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4(3):406–414.

Watrous, R. L., Towell, G. G., Glassman, M. S., Shahraray, M., and Theivanayagam, D. (1993). Synthesize, optimize, analyze, repeat (SOAR): Application of neural network tools to ECG patient monitoring. In *Proceedings of the 1993 International Symposium on Nonlinear Theory and Its Applications*, Honolulu.

Williams, R. J. and Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Yu, X. H. (1992). Can backpropagation error surface not have local minima? *IEEE Transactions on Neural Networks*, 3(6):1019–1020.

Zeng, Z., Goodman, R., and Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990.

Received November 18, 1993
Accepted January 23, 1995
Final Manuscript February 11, 1995